Block Floating Point FFT Implementation for DMT xDSL Systems

Th. Zahariadis, S. Apostolacos, I. Grammatikakis,
D. Mexis, N. Zervos and N. Vogiatzis
BLOCK FLOATING POINT FFT IMPLEMENTATION FOR DMT xDSL SYSTEMS

Th. ZAHARIADIS, S. APOSTOLACOS, I. GRAMMATIKAKIS, D. MEXIS and N. ZERVOS
Ellemedia Technologies, 223 Syggrou Avenue, 17121 N. Smirni, Athens, Greece

N. VOGIATZIS
Lucent Technologies, Forward Looking Work EMEA, Botterstraat 45, 1270 AA Huizen, The Netherlands

Received 22 October 2002
Revised 7 October 2003

The development of multiple Discrete Multitone (DMT) Digital Subscriber Line (DSL) flavors on a single platform can benefit considerably by a programmable architecture, which feature Digital Signal Processors (DSP) and Field Programmable Gate Arrays (FPGA), especially when fast prototyping is targeted. However, the flexibility assumed to be offered by algorithmic partitioning does not automatically and proportionally simplify the digital signal processing algorithms, unless the effects of overflow/saturation in intermediate processing stages are carefully studied. The effects of overflow/saturation in intermediate stages is very critical throughout the design process, since the operations involved are nonlinear in nature and affect the most significant bits of the computational process. This paper presents an efficient soft-core implementation of a Block Floating Point FFT (BLFP) algorithm, designed for a Very Illgh-speed DSL (VDSL) DMT systems and for the full variety of other xDSL DMT flavors, as the latter demand an extended dynamic range to achieve performance that may otherwise be only warranted by costly floating-point chip implementations.

Keywords: Block Floating Point FFT implementation; discrete multitone; Digital Subscriber Line (DSL); Field Programmable Gate Arrays (FPGA).

1. Introduction

Within the multitude of Digital Subscriber Line (DSL) flavors (ADSL, ADSL-lite, SDSL, HDSL, VDSL etc.), the transceiver algorithmic set needs to be adapted and optimized separately for each xDSL type. Optimization for modules that make up an xDSL transceiver have been under wide and intense study the last several years, however, the gap between theory and implementation is often difficult to bridge. Additionally, the Single-Carrier (SCM) and Multi-Carrier Modulation (MCM) schemes effectively double the long list of xDSL flavors in terms of implementation, and push the designer to come up with a way of unifying development on a common, flexible
modem platform. The flexibility of developing multiple xDSL flavors on a single platform can benefit considerably by a platform architecture that carries a mixture of DSP cores and FPGAs, especially when targeting fast prototyping. Even addressing only the multitude of MCM schemes can still yield great benefit for the designers that wish to have a common platform to develop and test all of their xDSL algorithms.

Developing a prototype platform to architect and evaluate a DMT system operating at Very-high speed DSL (VDSL) rates goes beyond selecting the right mix of DSPs and FPGAs and performing a suitable algorithmic partitioning between them.\(^2\) It has to address the thorny issue of bit-accurate modeling in data transmission and digital signal processing algorithms, where the main concern of the designer is to avoid overflow/saturation in intermediate stages of the processing since these operations are nonlinear and affect the most significant bits of the arithmetic. These nonlinear effects are expected to degrade the performance a lot more than the loss of least significant bits implied by the various scaling strategies devised to avoid overflows. This is the case with standard filters, equalizer update algorithms, but more importantly of the FFT as well.

From an implementation viewpoint, the FFT module is the most significant block of a DMT VDSL system, since its processing-intensive operations call for a FPGA realization. Numerous publications and articles have been written to establish bounds on the dynamic range of the input of an (I)FFT, so that no overflow/saturation occurs at any given stage of the algorithm. Such a task can be accomplished by either dividing the input samples by \(N\), where \(N\) is the FFT size, which would however result in a great loss of accuracy, or dividing each stage separately, thus retaining a large portion of the available dynamic range for the input.

The conditional Block Floating Point (BLFP) algorithm implementation described in this paper provides a more thorough approach and tracks the signal strength in every stage of the FFT, providing a more comprehensive scaling strategy and extended dynamic range. BLFP divides the whole stage, if and only if at least one overflow has occurred at any given sample of any given stage. As a result, the number of shifts depends on the amplitude of the input samples (in every stage), to which the dynamic range is adapted. The number of shifts that are performed is thus a random variable, depending on the frame that is being processed.

The primary benefit of the block floating-point algorithm emanates from the fact that operations are carried out in a block fashion using a common exponent. Each value in the block can be expressed in two components — a mantissa and a common exponent. The common exponent is stored as a separate data word. This results in a minimum hardware implementation compared to that of a conventional floating-point one. As it will be shown, the BLFP FFT can yield a performance equivalent to a floating-point implementation, in a fixed-point environment. Moreover, since subcarriers are independent, all calculations after the receive FFT can
be performed using a separate exponent per sub-carrier, resulting in a near floating-point performance.

Finally, in a DMT system, even if overflows are completely avoided, the most crucial component remains undoubtedly the cascade of the transmit IFFT and receive FFT. In addition to aliasing, quantization, leakage and other sources of error inherent to discrete transform calculations that degrade the performance of a DMT system, in a fixed-point environment, data will also experience round-off and truncation error. Round-off error is introduced at the multipliers, where only the most significant bits of the product are used. Truncation error occurs after an adder overflow, where the least-significant bit is shifted out. These errors will propagate across two FFTs (modulator and demodulator), with noise and other impairments being added in-between. So, the design of the FFT with the less internal noise is of the utmost importance.

The paper is structured as follows: Sec. 2 gives a FFT evolution path and particularly the Radix-2 implementation. Section 3 describes the BLFP FFT implementation challenges, such as the gains in accuracy, twiddle factors and the issues of gains in the digital and analog domain. Section 4 presents the simulation set up and discusses the derived performance results. Finally, Sec. 5 draws some conclusions.

2. FFT Evolution

Fast Fourier Transform (FFT) algorithms are directed toward computing \( X(k) \), the Discrete Fourier Transform (DFT) of a finite-duration sequence \( x(n) \). The first major breakthrough in the implementation of FFT algorithms was the Cooley–Tukey algorithm developed in the mid-1960s, which reduced the complexity of a DFT from \( O(N^2) \) to \( O(N \log_2 N) \). Since then, a large number of FFT algorithms have been developed. The well-known Radix-2 Cooley–Tukey algorithm was shortly followed by the Radix-3, Radix-4, and Mixed Radix algorithms. Further research led to the Fast Hartley Transform (FHT) and the Split Radix (SRFFT) algorithms. FFT research was considered a fairly mature area by mid 80's, yet two new algorithms have emerged: the Quick Fourier Transform (QFT) and the Decimation-In-Time-Frequency (DITF) algorithm.

The radix Decimation-In-Time (DIT) and Decimation-In-Frequency (DIF) algorithms are obtained by using the divide-and-conquer approach to the DFT problem. The FHT differs from the other algorithms in that it uses a real kernel, as opposed to the complex exponential kernel used by the Radix algorithms. The QFT postpones the complex arithmetic to the last stage in the computation cycle by separately computing the Discrete Cosine Transform (DCT) and the Discrete Sine Transform (DST). The DITF algorithm uses both the Decimation-In-Time (DIT) and Decimation-In-Frequency (DIF) frameworks for separate parts of the computation to achieve a reduction in the computational complexity.

Throughout this paper, only radix DIT algorithms are presented. All the results are derived using a Radix-2 DIT FFT, but the same apply for Radix-4 or Split...
Radix FFT's. The number of stages may be smaller than in the Radix-2 case, but there are more numbers to be added in each butterfly resulting in a substantial loss in accuracy (because of the fixed-point arithmetic) if overflows are to be avoided.

Moreover, the DIT structure has implementation benefits. Each butterfly can be subdivided into two sections, one containing the multipliers and the other one consisting solely of adders. Such a structure facilitates the development and understanding of the scaling strategy considerably, as will become evident in the ensuing.

### 2.1. Radix-2 butterfly and magnitude growth

The basic element of the radix-2 decimation in time FFT is the butterfly depicted in Fig. 1, where $X_{in1}, X_{in2}$ are the input samples, $X_{out1}, X_{out2}$ are the output samples and $W$ is the twiddle factor. All these quantities are complex in general and they follow an appropriate indexing according to the stage of the FFT. This butterfly can be reshaped to depict the decomposition of the complex multiplication to real ones and additions.

The basic processing element is shown in Fig. 2; this is the butterfly for the real part of the outputs. The same applies for the imaginary parts, except for a change of the signs of the additions. These additions and subtractions, using fixed-point arithmetic, can result in data overflow, which yields erroneous results and must be avoided during FFT calculation.

---

**Fig. 1.** A butterfly for the radix-2 DIT FFT.

**Fig. 2.** FFT butterfly with real operations.
The maximum theoretical magnitude growth possible for the radix-2 FFT butterfly shown is a factor of 2.414. Given that a radix-2 butterfly can be expressed as $X_{\text{out1}} = X_{\text{in1}} + (X_{\text{in2}} \cdot W)$, where all values are complex and $W$ is the twiddle factor, $X_{\text{out1}}$ will attain its maximum possible value when $X_{\text{in1}} = 1 + j0$; $X_{\text{in2}} = 1 - j1$; $W = 0.707 + j0.707$. This results in a maximum gain of 2.41421356. Thus the scaling based on this signal growth factor would be $1/2.414 \approx 0.4167$. Unfortunately, this approach would imply the use of one more multiplier after each stage of the FFT, resulting in more area on the FPGA chip. As a result, a scaling by $\left\lceil \log_{2} 2.41421 \right\rceil = 2$ bits per stage (where $\left\lfloor x \right\rfloor$ is the smallest integer greater than $x$) and sign-extension right after the multipliers is more realistic and easily implemented.

Unfortunately, in addition to reducing the data's dynamic range, data scaling introduces truncation error if any of the discarded bits are nonzero. As the FFT calculations proceed from stage to stage and further data shifts are required, the error can propagate and grow enough to severely distort or completely obliterate the true output signal. This effect can be alleviated by using larger adders to retain full accuracy, as well as the two extra bits for overflows. The extra bits are eliminated conditionally on the number of overflows at each stage at the input of the next stage, thereby precluding any time penalties.

As it will be shown, in such a case, the degradation in performance is negligible, when compared to a floating-point FFT. The technique will be further described in the sections that follow.

3. BLFP FFT Algorithm Implementation

The proposed soft-core block floating-point FFT implementation results in a performance equivalent to a floating-point implementation, in a fixed-point environment and minimal hardware implementation requirements as compared to a conventional floating-point one. To achieve this, the BLFP FFTI tracks the signal strength in every stage of the FFT, and divides the stage only if at least one overflow occurs at any sample. Moreover, the block operations use a common exponent, resulting in minimal hardware implementation.

In the proposed BLFP FFT implementation (Fig. 3), inputs, outputs and the twiddle factor are assumed to follow the fractional format with $K$ bits of accuracy. Initially $X_{\text{Rin2}}$ and $X_{\text{In2}}$ are multiplied with $WR$ and $WI$, respectively ($K \times K$ bits multipliers). For example, $K = 16$ and 1 sign bit, Q15 format. The products are $2 \times K$ bits long (Point A), as long as 1 left shift is performed, so as to remove the sign redundancy from the multiplication. The outputs are truncated or rounded to $L - 2$ bits, sign-extended to $L$ bits and added in an $L$-bit adder. The two most significant bits are checked for overflow (floating-point adder). $X_{\text{Rin1}}$ is placed into the $K$ upper bits, shifted right by 2 bits and sign-extended (no useful bits discarded — Point B). The adder is at least $L = K + 2$ bits wide if useful LSB accuracy is to be preserved. The results are added or subtracted to yield $X_{\text{Rout1}}$ and $X_{\text{Rout2}}$. 
which are rounded to \( L \) bits (Point C). This is done for all the butterflies in the FFT stage.

In the sequel, a common exponent for the whole output block is determined, based on the data element of the output block with the largest amplitude in this stage. The value of the exponent is the number of left shifts required for this data element to be normalized to the dynamic range of the processor (exponent detection and normalization). Based on the two most significant bits of the outputs of each butterfly, 0, 1 or 2 shifts are performed on all samples in order to be rounded or truncated to \( K \) bits to make optimal use of the available dynamic range. The common exponent of each stage is added to the sum of the previous stages in a separate memory location, in order to determine the overall exponent to be used in the next stages. The proposed process is repeated in all stages, so that maximum possible dynamic range is maintained and overflows are avoided.

The block floating-point representation does provide an advantage over both, fixed and floating-point formats. Scaling each value up by the common exponent increases the dynamic range of data elements in comparison to that of a fixed-point implementation. At the same time, having a common exponent for all data values preserves the precision of a fixed-point processor. Therefore, the block floating-point algorithm is more efficient than a conventional floating-point implementation, but equally accurate.

3.1. Gain in accuracy

The proposed BLFT FFT implementation is repeated throughout all the FFT stages, averting the possibility of an overflow and gaining maximum accuracy.

The maximum possible bit growth occurs when the input data are equal and at full-scale. If \( x(n) = [2^{(b-1)} - 1, 2^{(b-1)} - 1, \ldots, 2^{(b-1)} - 1] \) or \([-2^{(b-1)}, -2^{(b-1)}, \ldots, -2^{(b-1)}]\), where \( b \) is the processor's word-length, the FFT of a constant input is an impulse whose amplitude is given by the following equation:
\[ X(k) = \begin{cases} 
\sum_{n=0}^{N-1} x(n) = N \cdot x, & k = 0, \\
0, & k \neq 0. 
\end{cases} \quad (1) \]

This result is intuitive, as a constant time-domain signal concentrates all its power into the DC component of its transform. This equation shows that the data can grow by a factor of \( N \), or \( \log_2 N \) bits, so when \( x \) is at full-scale, \( \log_2 N \) shifts are necessary to prevent overflow. We note that the inverse transform would likewise produce \( \log_2 N \) bits of overflow given the same maximum DC input sequence. However, the inverse transform requires that we divide by \( N \), equivalent to subtracting \( \log_2 N \) from the data's common exponent.

\[ x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) = N \cdot X, \quad n = 0. \quad (2) \]

From Eq. (2), it is shown that the data, overall, does not experience any growth, although it experiences maximum data scaling.

Given that a sequence can grow by 1 bit during the first two stages and by 2 bits in all subsequent stages, unconditional fixed-point scaling techniques would shift the data a total of \( 2 \cdot \log_2 N - 2 \) bits, almost twice the maximum possible bit growth. For this reason, block floating-point schemes can generally be expected to perform better than unconditional schemes. For example, for a 2048-point FFT, the number of FFT stages is \( \log_2(2048) = 11 \) and the maximum number of shifts is 20. However, during simulations, no more than 8 shifts were performed for a variety of input signals and powers. This is justified since the scaling strategy divides the stage by 4 instead of the maximum gain derived above, so the peak amplitudes per stage are significantly smaller resulting in a smaller number of shifts in total.

This is a benefit derived by scaling up all values in the input such that they fully occupy the dynamic range available. As hypothesized, this case indicates a good result, since shifting all values to the left allows for increased precision. Since bit growth of values between butterfly stages is a possibility but not a necessity, scaling down automatically during each stage can compromise the precision of results. Scaling down only makes sense if there is evidence of bit growth during butterfly computations. However, when bit growth is not expected, the results of that stage can be directly fed to the next stage while yielding full benefit of the available dynamic range. Automatically scaling down in this last instance would not use the full available dynamic range. In this manner, optimal scaling from stage-to-stage is used to prevent overflow while at the same time, accuracy of the overall system is improved.

### 3.2. Twiddle factors

It has been shown\(^9\) that even in a floating-point environment the accuracy of the FFT is greatly impaired by inaccurate sine/cosine tables. When an accurate
sine/cosine table is used, the FFT is remarkably stable. Inaccuracy in the sine/cosine table is common and results from inaccurate underlying software library functions and ill-advised recursions for look-up table construction. In the simulations presented in this section, the complex roots of unity used in the FFT are calculated using 64-bit IEEE floating-point numbers and then rounded to 16 bits, so maximum accuracy is preserved.

In an actual implementation, the sine/cosine tables could be either calculated in the DSP and then downloaded to the FPGA, or stored in binary format on the EPROM or Flash Memory used to program the FPGA at boot, if speed in initialization is required, depending on the machine architecture. In the first case, the twiddle factors could be computed to high accuracy, either with high-precision arithmetic or Kahan's summation method (if Kahan's method works effectively on the given hardware). Truncated Taylor's series also work very well for this purpose. A faster approach is the use of quickly converging algorithms, such as rational or continued fraction approximations, at the cost of a loss in accuracy. However, the designer must keep in mind that speed is generally irrelevant for this application (initialization of the FFT).

3.3. Gains in the digital and analog domain

All the preceding analysis would work fine in an all-digital environment, but the situation is quite different in the case of a DMT transmission system. In a DMT transceiver, carrier generation at the transmitter is performed by use of an IFFT that maps complex symbols that have Hermitian symmetry to a real-valued baseband signal that is then transmitted. As it is well known, the output of the IFFT must be divided by $N$ in order for Parseval's theorem to hold true. Suppose that the IFFT has already performed $M$ shifts during its execution. In order for all the transmitted frames to have the same power, the output samples must be further divided by $2^{(N - M)}$. The remaining $N - M$ shifts should not be performed in the digital domain, since this would imply a great loss of LSBs and all the gain in dynamic range would have vanished even before the D/A conversion. On the other hand, the transmitter must maintain a constant transmit power. The best way to work around both of these problems is to use a Programmable Gain Amplifier (PGA) after the D/A converter, in order to perform the remaining divisions by 2 in the analog domain. For a PGA to be used in such a way, it must have a settling time much less than the sampling interval and subsequently a bandwidth much greater than the bandwidth occupied by the DMT signal (at least 10 times greater). There exist many such components from various manufacturers.

The best transmitter in terms of dynamic range would therefore be a block floating-point IFFT followed by a D/A converter and a PGA to maintain a constant transmit power.

On the other side, the use of a PGA (analog domain) and a gain in the digital domain (both are contained in the Automatic Gain Control mechanism at the
receiver) is imperative in order to drive the A/D converter efficiently and yield appreciably large samples to the time-domain equalizer that creates the equivalent channel for the DMT signal. If this has been achieved, the received FFT performs a number of shifts that is taken into account at the FEQ, since as it has been mentioned, floating-point arithmetic per carrier is used. The total necessary gain at the receiver is also factored into the digital and analog domain. It is clear that the implementation of the transmitter IFFT is more important in terms of dynamic range.

4. Simulation Set-Up Performance

Several statistical models\(^1\) place an upper bound on the output noise-to-signal ratio for FFTs employing unconditional BLFP scaling (i.e., per-stage scaling). Most published articles in the field assume that 1 bit of overflow occurs during each stage, and obtain the same result, that the output noise-to-signal ratio increases by average \(1/2\) bit per stage. However, the output noise-to-signal ratio for conditional BLFP scaling depends strongly on the number of overflows, and at which FFT stage they occur. As the positions and timing of the overflows are correlated with the input signal, one needs to know the signal statistics in order to analyze the output noise-to-signal ratio. Instead of analyzing the problem theoretically, most authors perform experiments using white-noise inputs and compare the results to the upper bound obtained for unconditional BLFP scaling. As expected, the experiments show that conditional BLFP performs better than unconditional BLFP, as fewer shifts of the data occur, and hence less error propagates to the outputs.

In this paper, we determine the performance of a FFT using conditional BLFP scaling in the context of multi-carrier transmission. In a DMT system, the IFFT modulator in tandem with the FFT de-modulator, provide a way to create the carriers necessary for multi-carrier transmission and to retrieve the symbols at the receiver. On a very basic level, the transmitted quantities are just a reversible linear transformation of the input symbols. However, if finite precision arithmetic is used,
round off and truncation noise as well as the A/D and D/A inherent nonlinear behavior, render the IFFT of the transmitter partially irreversible by the FFT of the receiver.

The simulations are based on the structure of Fig. 4. The input symbols are uniformly distributed on the point-grids defined for different constellation sizes in the ANSI VDSL Technical Specification. All bins are loaded with the same constellation size and all the symbols are normalized with the average power of the constellation, so that a time series of such symbols has unit variance. Only the DC and Nyquist carriers are not modulated since the 2048-point FFT used has even length and the output signal is confined to be real. For the same reason, its Hermitian symmetric part is then appended. Then, the IFFT modulation and D/A conversion with $Kc1$ bits of accuracy are performed. At the receiver, the signal is amplified by a gain $G$, which is a fixed gain depending on the variance of the DMT signal and does not vary from frame to frame. The A/D and D/A are uniform quantizers in the range of $[-1, 1]$. This interval includes $-1$ because of the 2's complement arithmetic used. All values above 1 are clipped to the largest representable number $(2^{A/D \ \text{resolution}} - 1)$ and those below $-1$ are clipped to $-1$ (hard limiting). The signal is then quantized to $Kc2$ bits of accuracy and the receiver FFT is performed. Finally, the received symbol is multiplied in floating point with a gain $1/G$ and subtracted from the transmitted one to yield a noise estimate.

Both the IFFT and FFT used are of the block floating-point type. Some of the runs were also performed using a 64-bit floating point (I)FFT for comparison reasons. Even in those cases, the A/D and D/A converters are present. The width of the multipliers and adders as well as the accuracy of the A/D and D/A converters can be controlled separately to show their impact on the performance.

It must be noted that no noise is added between the transmitter and receiver and that no channel is present during the simulations. In the case of a floating-point FFT and the absence of D/A, A/D converters, the output symbols should be identical to the transmitted ones.

In order to evaluate the invertability of the Fast Fourier Transform using finite-length registers, we analyze the results of the block floating-point FFT and IFFT implementation via the Quantization Error (QE) and the Signal-to-Quantization Noise Ratio (SQNR) values. The QE is a suitable study of the results since it compares the corresponding values between two signals. Calculating the total noise power for a given pair of signals results in the quantization error. If $X_{\text{transmitted}}$ is the reference transmitted complex signal and $X_{\text{received}}$ is the received complex signal, the total QE (noise power) for these two complex signals evaluated by:

$$\text{QE} = \sum_k \{ |\text{Re}(N_n(k))|^2 + |\text{Im}(N_n(k))|^2 \}, \quad (3)$$

where $N_n(k) = X_{\text{transmitted}}(k) - X_{\text{received}}(k)$ and "Re", "Im" denote real and imaginary quantities, respectively. Similarly, the total signal power ($S$) is:
The QE is computed for the signal under test (received symbol) with respect to a signal considered to be the reference for comparison (transmitted symbol). As a result, in this analysis, the total signal power is always calculated with the reference signal — the transmitted symbol in the equations above. Armed with this knowledge, the computation of the \( SQNR \) becomes relatively simple. It is the ratio of the total signal power to the total noise power. Using the equations above:

\[
SQNR = \frac{S}{QE} = \frac{\sum_k \{[\text{Re}(X_{\text{transmitted}}(k))]^2 + [\text{Im}(X_{\text{transmitted}}(k))]^2\}}{\sum_k \{[\text{Re}(N_n(k))]^2 + [\text{Im}(N_n(k))]^2\}}.
\]

In dB, the \( SQNR \) can be expressed as \( 10 \log_{10}(SQNR) \). One useful remark is that the \( SQNR \) is also a measure of digital performance of the DMT transmission system, since it is directly related to the probability of a symbol in error at the receiver. This probability can be expressed independently for any given sub-carrier loaded with a given constellation size.

4.1. Remarks on the results

In the simulations, 10 000 frames were used. The experiments were later repeated on Altera APEX 1200 PLDs. Based on the results, we can extract the following remarks:

![Real Part](image)

![Imaginary Part](image)

Fig. 5. 2048-point BLFP FFT (32-bit adders, 16-bit multipliers, 16-bit A/D converters).
Fig. 6. 2048-point BLFP FFT (36-bit adders, 18-bit multipliers, 16-bit DAC/ADC converters).

Fig. 7. 2048-point BLFP FFT (40-bit adders, 20-bit multipliers, 16-bit DAC/ADC converters).
• For a given DAC-ADC accuracy, the size of the multiplier has to be at least 3 bits wider, or else the value of the SQNR depends heavily on the frequency and has a highly harmonic structure. It is also dependent on the constellation size used, although they all have unit variance. These are depicted on Figs. 5-7 (all constellation sizes are plotted on the same figure).

As it is shown in Fig. 5, when the multiplier accuracy is equal to the DAC/ADC converter accuracy the SQNR is quite bad. In Fig. 6 the results are improved, while in Fig. 7 (multipliers 4 bits wider), the SQNR is quite good.

• The same effect appears when the size of the adders is not sufficiently larger than that of the multipliers used (even if the multiplier width is adequate) and accuracy is lost because of the truncation of the FFT's intermediate results and parametric shifts per stage. A further increase of the width of the adders does not improve the performance. Please refer to Figs. 8 and 9.

• If the size of both the multipliers and adders is adequate, the quantization noise is almost white, with a very small variation of no more than 1 dB peak-to-peak across the spectrum, in any of the cases presented (Figs. 10 and 11).

• The performance of the 64-bit floating point (I)FFT is identical for those cases (as shown on Figs. 12 and 13) indicating that all of the noise is induced at the converters and not because of the IFFT-FFT pair.

Fig. 8. 2048-point BLFP FFT (20-bit adders, 16-bit multipliers, 12-bit converters).
Fig. 9. 2048-point BLFP FFT (24-bit adders, 16-bit multipliers, 12-bit converters).

Fig. 10. 2048-point BLFP FFT (28-bit adders, 16-bit multipliers, 12-bit converters).
Fig. 11. 2048-point BLFP FFT (36-bit adders, 18-bit multipliers, 14-bit converters).

Fig. 12. 2048-point FLP FFT (64-bit floating-point internal accuracy, 12-bit converters).
Fig. 13. 2048-point FLP FFT (64-bit floating-point internal accuracy, 13-bit converters).

Fig. 14. SQNRs for various implementations of the IFFT-FFT combination versus different DAC/ADC resolutions.
Finally, it can be observed that given a white noise spectrum, there is a gain of 6 dB per bit of the A/D, D/A converter pair. More specifically, the average $SQNR$s across the spectrum as a function of bits, given the noise spectrum is white, is depicted on Fig. 14.

The best linear fit to the data points presented above yields $SQNR \, (dB) = 5.9883 \cdot b - 13.563$. Given that today's majority of A/D, D/A converter accuracy is in the range of 12 bits for the spectrum of VDSL modems, it appears that the use of 16-bit multipliers and 24-bit adders for the calculation of the (I)FFT is more than adequate yielding a $SQNR$ of 58.3 dB.

In the case that a channel is present, the received signal to received noise ratio (SNR) will be well below the 58.3 dB threshold for most of the loops because of the presence of cross-talkers and other line-related impairments. Even for short loops, where the propagation loss is not the dominant impairment and the SNR is better than 58.3 dB, even for all sub-carriers, the limiting factor is the maximum number of bits per constellation symbol that can be supported (11 bits/symbol according to the ANSI VDSL Technical Specification). In that case, the expected Symbol Error Rate for a constellation with 11 bits/symbol with a SNR of 58.3 is zero, so the DMT transceiver will convey the greatest possible rate (i.e., 11\* (Number of used carriers) bits per DMT symbol).

On the other hand, the effect of the transmit/receive filters and the channel on the Peak-to-Average ratio (PAR) cannot be neglected. The amplitude and phase characteristics will obviously alter the PAR of the received signal with regard to the transmitted one. The echoes due to impedance mismatches are also likely to increase the PAR of the received signal. This leads to the conclusion that the $SQNR$ of an actual implementation will definitely depend on the channel characteristics.

5. Conclusions: Further Improvements

From the results presented above, it is clear that the common exponent is the key characteristic of the block floating-point FFT implementation. It increases the dynamic range of data elements of a fixed-point implementation by providing a dynamic range similar to that of a floating-point implementation. By using a separate memory word for the common exponent, the precision of the mantissa quantities is preserved as that of a fixed-point processor. By the same token, the block floating-point algorithm is more economical than a conventional floating-point implementation. The majority of applications are best suited for fixed-point processors. For those that require extended dynamic range (like the DMT FFTs) but do not warrant the cost of a floating-point chip implementation, the block floating-point implementation on a fixed-point chip readily provides a cost-effective solution.

Further improvements could involve the use of nonuniform quantizers. This would lead to a significant improvement in performance since after the transmission of a DMT signal through a channel, the higher frequency sub-carriers are significantly attenuated when compared to lower frequency ones. This results in a
great loss of dynamic range for the A/D converter at the receiver, which could be overcome using nonuniform quantifiers with logarithmically related ranges. Since the main loss in A/D conversion accuracy is caused because of the high Peak-to-Average ratio of the DMT signal, coding at the transmitter to reduce PAR may be an option, resulting in less noise induced by the A/D converters. Finally, by utilizing mechanisms of algebra on finite rings, such as cyclotomic integer FFTs, could further improve invertibility with the register lengths presented above.

References