Correspondence

Corrections to "A General Theory of Phase Noise in Electrical Oscillators"

Ali Hajimiri and Thomas H. Lee

The authors of the above paper[1] have found an error in (19) on p. 185. The factor of 8 in the denominator should be 4; therefore (19) should read

$$\mathcal{L}\{\Delta\omega\} = 10 \cdot \log \left(\frac{\dfrac{\overline{i_n^2}}{\Delta f} \displaystyle\sum_{n=0}^{\infty} c_n^2}{4 q_{\max}^2 \Delta\omega^2} \right).$$

Noise power around the frequency $n\omega_0 + \Delta\omega$ causes two equal sidebands at $\omega_0 \pm \Delta\omega$. However, the noise power at $n\omega_0 - \Delta\omega$ has a similar effect as mentioned in the paper. Therefore, twice the power of noise at $n\omega_0 + \Delta\omega$ should be taken into account. This will also change the 4 in the denominator of (21) to 2 to read

$$\mathcal{L}\{\Delta\omega\} = 10 \cdot \log \left(\frac{\Gamma_{\text{rms}}^2}{q_{\max}^2} \cdot \frac{\overline{i_n^2}/\Delta f}{2 \cdot \Delta\omega^2} \right).$$

Similarly, (24) must change, and its correct form is

$$\Delta\omega_{1/f^3} = \omega_{1/f} \cdot \left(\frac{c_0}{2\Gamma_{\text{rms}}} \right)^2 \approx \omega_{1/f} \cdot \frac{1}{2} \left(\frac{c_0}{c_1} \right)^2.$$

This will result in the factor of 1/2 becoming redundant in (29), i.e.,

$$\mathcal{L}\{\Delta\omega\} = 10 \cdot \log \left(\frac{kT}{V_{\max}^2} \cdot \frac{1}{R_p \cdot (C\omega_0)^2} \cdot \left(\frac{\omega_0}{\Delta\omega} \right)^2 \right).$$

However, note that the discussion following (29) is still valid.

The factor $c_0^2/2\Gamma_{\text{rms}}^2$ should be changed to $(c_0/2\Gamma_{\text{rms}})^2$ in the following instances:

1) p. 185, second column, last paragraph;
2) p. 190, second column, first paragraph;
3) p. 190, second column, second paragraph.

Nevertheless, the expression used to calculate the Γ_{rms} to predict phase noise of ring oscillators is based on a simulation that takes this effect into account automatically, and therefore the predictions are still valid. The authors regret any confusion this error may have caused.

Manuscript received February 27, 1998.

The authors are with the Center for Integrated Systems, Stanford University, Stanford, CA 94305-4070 USA.

Publisher Item Identifier S 0018-9200(98)03730-5.

[1]A. Hajimiri and T. H. Lee, *IEEE J. Solid-State Circuits,* vol. 33, pp. 179–194, Feb. 1998.

Comments on "A 64-Point Fourier Transform Chip for Video Motion Compensation Using Phase Correlation"[1]

Kevin J. McGee

Abstract—**The fast Fourier transform (FFT) processor of the above paper,[1] contains many interesting and novel features. However, bit reversed input/output FFT algorithms, matrix transposers, and bit reversers have been noted in the literature. In addition, lower radix algorithms can be modified to be made computationally equivalent to higher radix algorithms. Many FFT ideas, including those of the above paper,[1] can also be applied to other important algorithms and architectures.**

I. Introduction

In the above paper,[1] the authors present a fast Fourier transform (FFT) processor that contains many interesting and novel features. The mathematics in the above paper,[1] describe a matrix computation where both time inputs and frequency outputs are in bit-reversed order. Bit-reversed input/output FFT algorithms, while not widely known, are not new, having been previously described in [3]. Fig. 1, for example, is a 16-point, radix-4, undecimated, bit reversed input/output, constant output geometry graph based on [3].

The algorithm[1] is also described as a decimation-in-time-and-frequency (DITF) type, but the architecture appears to be based on decimation-in-time (DIT). In the above paper,[1] Figs. 4 and 10 show a first calculation stage with unity twiddles before the butterfly and a second and third calculation stage with prebutterfly twiddles. Although the butterfly implementation of Fig. 5[1] may be unique, the use of prebutterfly twiddles in all three stages, along with unity twiddles in the first, would seem to indicate DIT. The architecture[1] is also a pipeline and contains many elements common to this type of processor, such as matrix transposers and bit reversers, as will be described below.

II. Matrix Transposers and Bit Reversers

Block serial/parallel or parallel/serial converters, sometimes called matrix transpose or corner turn buffers, are used in many systems. They perform a matrix transpose on data blocks by exchanging rows and columns. Fig. 2 (from [7]) shows, from upper left to lower right, the flow of data through a 4 × 4 shift-based transposer. The rotator lines show where data will be routed on the next clock cycle and the output is the transpose of the input. The switching action was noted in [7] and [8] and rotator designs can be found in [4], [7], and [8]. Although Fig. 6(b)[1] is also an 8 × 8 transposer, it is being used in a somewhat unusual way. By providing a complex (real and

Manuscript received January 31, 1997; revised March 5, 1998.

The author was with the Naval Undersea Warfare Center, Newport, RI 02841 USA. He is now at 33 Everett Street, Newport, RI 02840 USA.

Publisher Item Identifier S 0018-9200(98)03731-7.

[1]C. C. W. Hui, T. J. Ding, J. V. McCanny, and R. F. Woods, *IEEE J. Solid-State Circuits,* vol. 31, pp. 1751–1761, Nov. 1996.

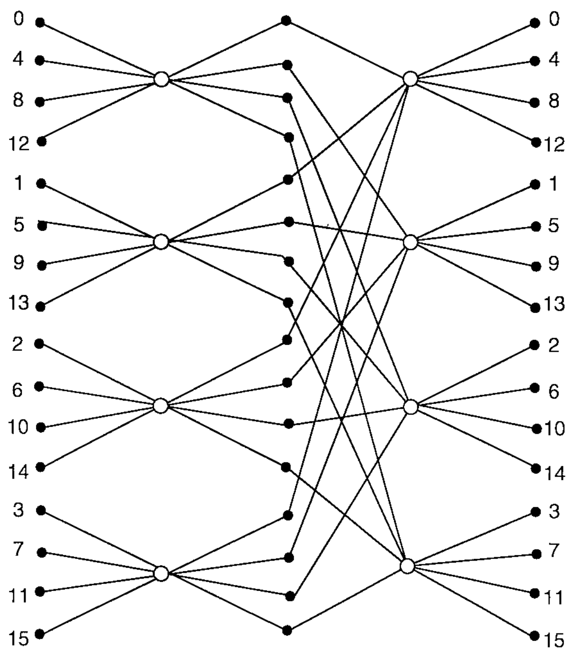

Fig. 1. Sixteen-point, radix-4, bit reverse input/output, undecimated, constant output geometry graph.

imaginary) input data stream and then combining real and imaginary serial outputs, the 8 × 8 in the above paper[1] is being used, in effect, as a 4 × 4 for complex valued data (four real and four imaginary parallel inputs, four complex serial outputs).

Interestingly, both the 4 × 4 and 8 × 8 architectures are known [1], but only as FFT pipeline data routers; they do not appear to have been recognized as transposers. Although there are more than a dozen alternative transposer implementations [9], none of them use the ideas depicted in Fig. 2 or Fig. 6(b),[1] and all of the alternatives require at least twice (sometimes four times) as much storage and much more wiring capacity. This is especially surprising, since circuits like Fig. 2 are often found in pipelines. No matter what type of transposer, 3 × 3, 4 × 4, etc., a circuit of the type in Fig. 6(b)[1] or Fig. 2 in very large scale integration should be smaller in area and faster than any of the alternatives.

Transpose is closely related to bit reverse [7]. The FFT pipeline in Fig. 3 (from [5, Fig. 5]) may be helpful in describing the difference. Fig. 3 is a 64-point sequential input, line sequential output processor. As noted in [5], line sequential output means that outputs are in sequential order per output line. Section A in Fig. 3 is a radix-4 processor for bit-reversed inputs and line sequential outputs. Section B takes 64-point sequentially ordered inputs, presented four at a time, and converts them into radix-4 based bit-reversed order [7]. The switching action is the same as Fig. 2, but at one-fourth the rate. Section B can also change radix-4 based bit reversed inputs into sequential outputs. Fig. 6(c)[1] is also a bit reverser, but for a serial, complex valued data stream.

Although section B bit reverses 4-row/16-column input data, it produces outputs in the same 4-row/16-column format. In contrast, a true mathematical transpose would require that the data be output in a 16-row/4-column format. Such a transposer could be used at the output of Fig. 3 so that outputs will be in true sequential order. Since a transposer of the type needed can be built using the same amount of storage as a 64-point bit reverser, the total memory required for a fully parallel processor of Fig. 3 with the output modified to provide sequential outputs is 156 (48+12+48+48).

Among the other ways that the architecture of Fig. 3 can provide outputs in sequential order is to add a section B followed by a 4 × 4 transposer (i.e., Fig. 3 followed by section B followed by Fig. 2). This would seem to be a fully parallel version of the above paper,[1] with a total memory requirement of 168 (48+12+48+48+12). The serial processor in the above paper,[1] in contrast, would seem to be an 8 × 8 transposer (being used as a 4 × 4 to serialize the inputs), followed by a serial Fig. 3, then a serial section B, and then another 8 × 8 (which is also being used as a 4 ×4). The bit reverser/transposer output modification to Fig. 3 was not known to the author until after having read the above paper.[1] The bit reverser/transposer combination in the above paper[1] is an interesting and clever circuit in its own right. Although it requires slightly more memory than the true mathematical transpose, it appears to be an elegant way to provide outputs in a sequential format. The same or similar bit reverser/transposer or transposer/bit reverser combinations could be quite useful in many other architectures and not just for FFT.

It might also be noted that section C is a processor for calculating a 16-point bit reversed input graph. The outputs will be in line sequential order, since every butterfly of the rightmost column of the graph is processed in sequence, from top to bottom. The sequence of butterfly processing is the key determinant of the output sequence, prior to any output post-processing. It might also be pointed out that the processor works for all types of 16-point bit reversed input graphs, no matter what the output sequence or geometry. This is due to having only two calculation stages as higher stage pipelines are more sensitive to the graph type [5].

III. Equivalence of Higher and Lower Radix Graphs

It is often stated, as in the above paper,[1] that higher radix FFT's are more computationally efficient than lower radix. However, modified lower radix graphs can be made computationally equivalent to higher radix graphs. This equivalence can be demonstrated through a simple example. Fig. 4 (from [5]) is a 16-point, radix-2, DIT, in-place geometry FFT graph. Butterfly A has exponents 2 and 6 to its right. Since exponents are additive (e.g., e to the 6 equals e to the 2 times e to the 4), the exponent 6 can be rewritten as 2+4. To the right of butterfly B are exponents 1 and 5 (the latter can be expressed as 1+4). Butterfly C has 3 and 7 (the 7 can be split into 3+4). Consequently, butterfly A has common exponents of 2 on each of its output lines, butterfly B has common exponents of 1 on each output, and butterfly C has common exponents of 3 on both outputs. The common rightside exponents of 2, 1, and 3 can all be moved left, across the linear butterflies A, B, and C, respectively, and added to the existing exponents on the input side to yield Fig. 5 (from [5]). Although Fig. 5 has a radix-2 structure, it has the same number and type of adds, subtracts and multiplies as the 16-point radix-4 graph of Fig. 6. One way to see this is to compress each group of four radix-2 butterflies in Fig. 5 into its radix-4 equivalent. Alternately, if one starts with Fig. 6, expands each radix-4 butterfly into its radix-2 equivalent, and rearranges nodes, then one can also obtain Fig. 5. Figs. 5 and 6 are computationally equivalent. There are many other examples, such as the 64 point radix-8 graph and radix 3/9 graphs mentioned in [5] and the comprehensive 32-point examples shown in [7].

The computational equivalence between higher and modified lower radix FFT algorithms was developed in 1981 and described at the presentation for [5] and reiterated in [7]; there are no real computational savings in higher radix FFT algorithms; lower radix graphs can be modified to obtain the same number and type of adds, subtracts,

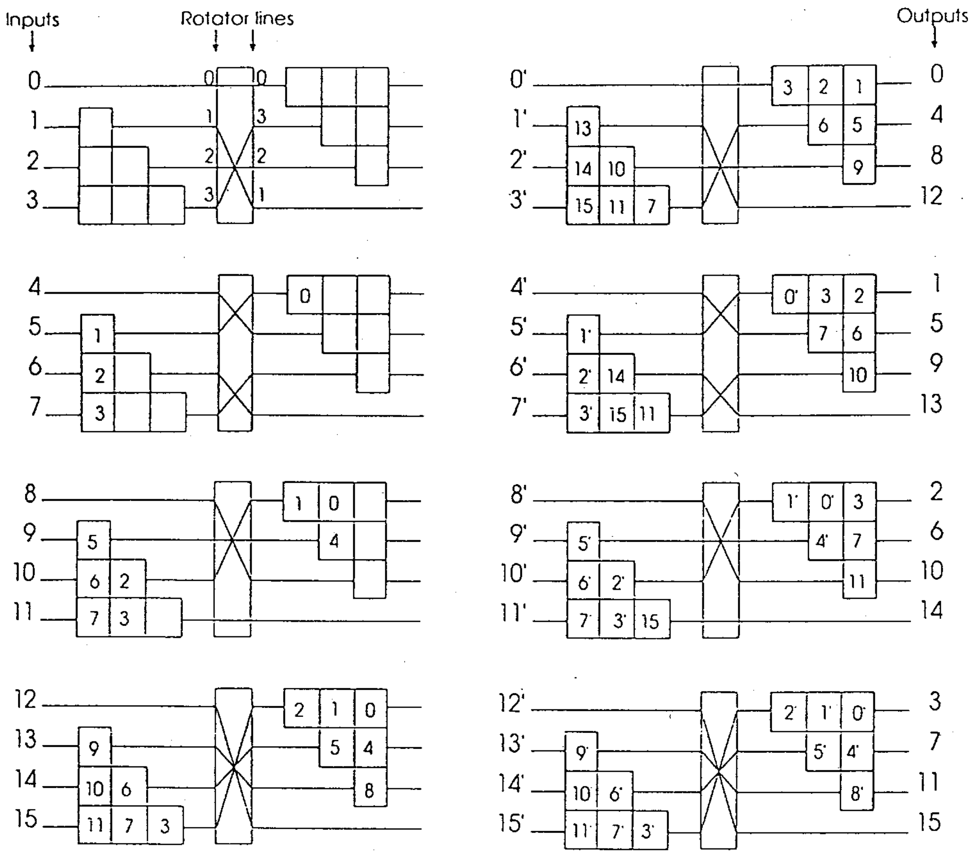

Fig. 2. A 4 × 4 transpose.

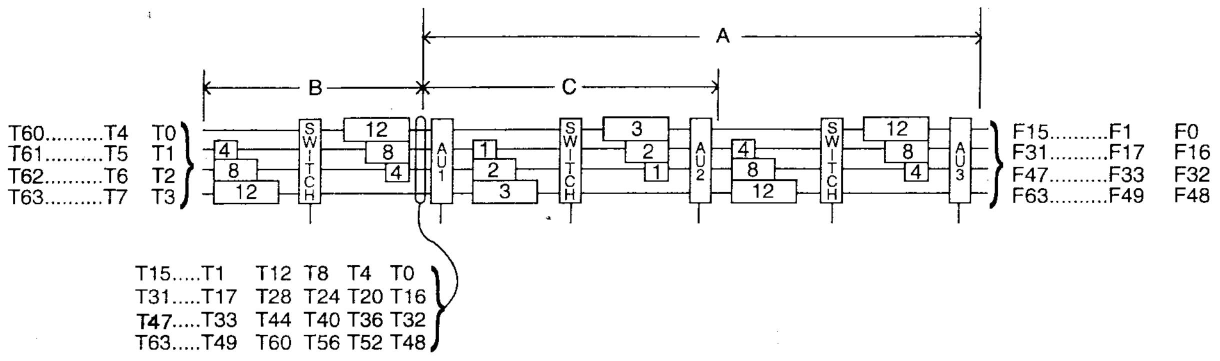

Fig. 3. A 100% efficient 64-point radix-4 pipeline.

and multiplies as in any higher radix graph. Higher radix graphs are really nothing more than lower radix graphs in disguise—just expand the higher radix butterflies into their lower radix equivalents and rearrange the nodes to get the desired input/output sequence and geometry. As noted in [5], one may expand any higher radix butterfly into either its DIT or DIF equivalent, as well as moving common bits on one side of a butterfly to the other. Although these types of manipulations can make it difficult to categorize the modified graph, the changes can sometimes be very important for hardware or software implementations.

It is also interesting to consider further expansion of butterflies to the bit, Boolean, or gate level (e.g., for a radix-2 butterfly with 8-bit inputs, expand the butterfly until it becomes eight butterflies with single bit inputs and outputs, borrow/carry, etc.). Subsequent manipulations of larger graphs could then result in the development of new insights, graphs, or processing schemes. This kind of manipulation can be particularly intriguing when working with algorithms other than FFT.

Since the FFT is a member of the shuffle exchange (SE) group (sorting, polynomial evaluation, matrix multiply, Viterbi, etc.), many algorithm and architecture ideas from FFT can be used for SE [7]. Although some SE graphs have nonlinear butterflies, the techniques described above can still be used favorably. It may also be possible to linearize the nonlinear SE graphs. In addition, many FFT ideas can be put to good use in filtering neural nets, error correction coding [6], and related fields. The application to coding and filtering can

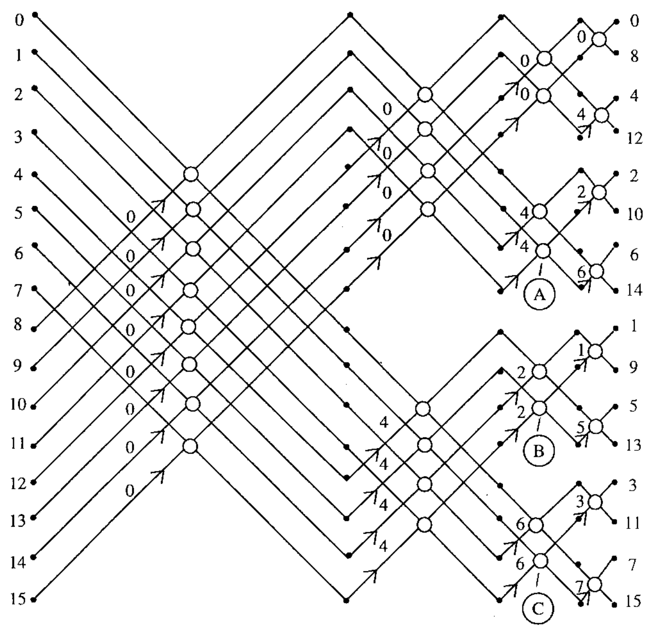
Fig. 4. A 16-point radix-2 graph.

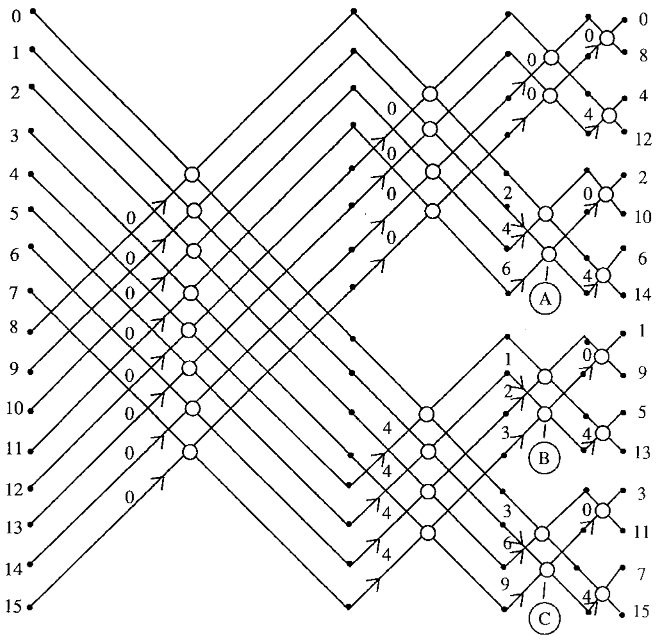
Fig. 5. Modified 16-point radix-2 graph.

be interesting, because it gets into multiple shuffle graphs. Multiple shuffle graphs include back-to-back shuffle graphs, vertically and/or horizontally extended shuffle graphs, and shuffle within shuffle graphs (i.e., expand a butterfly as a shuffle, and implement accordingly).

Processors based on multishuffle graphs offer some implementation advantages. Consider, for example, a pipeline for a shuffle graph like the 64 point, radix-8, sequential input, bit reversed output, in-place FFT in [2, Fig. 10.15]. A straightforward design would be the following sequence of pipeline stages: 8 × 8 transposer ($8 \times 8T$), 8-point butterfly ($8B$), $8 \times 8T$, $8B$, and a final $8 \times 8T$. Both inputs and outputs will be in sequence (eight at a time), and the storage required

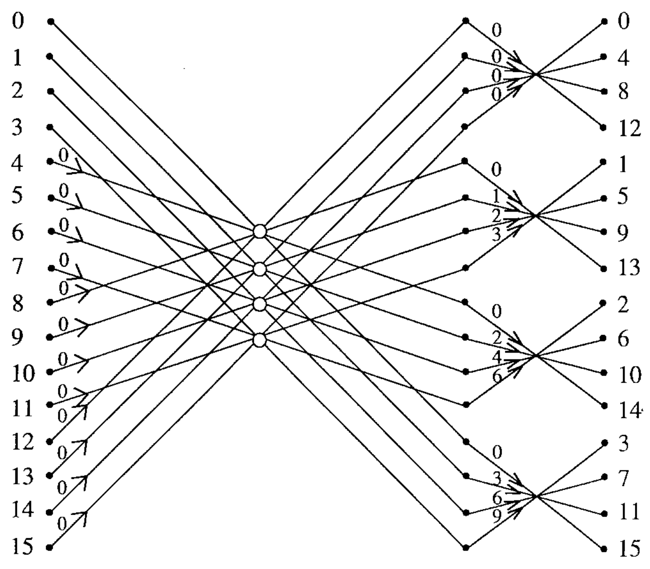
Fig. 6. A 16-point radix-4 graph.

is 168 (3 × 56). One could then apply, for example, a convolution multiplication, followed by a second pipeline identical in structure to the first to perform the inverse transform.

Starting the design from a double shuffle graph results in the following sequence of pipeline stages: $8 \times 8T$, $8B$, $8 \times 8T$, $8B$, convolve multiply, $8B$, $8 \times 8T$, $8B$, and $8 \times 8T$. Once again, the inputs and outputs will be in sequence, eight at a time. The graphs used are [2, Fig. 10.15], followed by its reverse image (i.e., take [2, Fig. 10.15] and read the graph from right to left—it becomes a bit reversed input, sequential output, in-place type, add decimation to suit). One could have also achieved the same processor by starting with the reverse graph, followed by the forward graph; it all depends on how one relates the graph elements to the architectural ones. Either way, the double shuffle processor allows one to reduce or eliminate the pipe-to-pipe staging used by two processors based on two single graphs. As a result, the memory required for the double shuffle pipeline is only 224 (4 × 56) versus 336 (2 × 168) for the single shuffle back-to-back pipes, or 350 (2 × 175) for the two chip processor mentioned in the above paper.[1] The former also allows one to seek further improvements by combining the middle calculation stages. The double shuffle processor may also be used for 64 × 64 two-dimensional image processing, but since the function requires row/column operation (like transform based beamforming), the architecture needs a 64 × 64 storage node between pipelines.

IV. Conclusion

The FFT processor of the above paper,[1] contains many interesting and novel features. Unfortunately, bit reversed input/output algorithms, matrix transposers, and bit reversers have been previously reported. This should in no way detract from the other important results and achievements of the above paper,[1] such as the array butterfly, the use of an 8 × 8 transposer as a 4 × 4 for complex valued data, implementation of a transposer in custom VLSI, a clever bit reverser/transposer combination, etc.

The equivalence of higher and lower radix graphs has also been emphasized. Lower radix graphs can be made computationally identical to higher radix graphs.

Many techniques from FFT may be applied to SE and other algorithms and their analog or digital processors. It may also be possible to linearize the nonlinear SE graphs.

References

[1] R. E. Llewellyn, “High base multiple rail Fourier transform serial stage,” U.S. Patent 3 777 131, Dec. 4, 1973.

[2] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[3] T.-Thong, “Algebraic formulation of the fast Fourier transform,” *IEEE Circuits Syst. Mag.*, vol. 3, pp. 9–19, June 1981.

[4] S. M. Kang, R. H. Krambeck, and A. Kwan, “Data shifting and rotating apparatus,” U.S. Patent 4 396 994, Aug. 2, 1983.

[5] K. J. McGee, “Pipelined FFT processors,” in *IEEE Proc. 17th Asilomar Conf.*, Oct. 1983, pp. 194–198.

[6] T. G. Marshall, Jr., “Coding of real-number sequences for error correction: A digital signal processing problem,” *IEEE J. Select. Areas Commun.*, vol. 2, pp. 381–392, Mar. 1984.

[7] K. J. McGee, “Fourier transform processors,” Naval Underwater Systems Center, Tech. Rep. NUSC TM-89-2030, Mar. 1, 1989.

[8] ——, “Comments on ‘in-place updating of path metrics in Viterbi decoders’,” *IEEE J. Solid-State Circuits*, vol. 25, pp. 1039–1040, Aug. 1990.

[9] ——, *Transpose Memories*, unpublished.