AN EFFICIENT FFT ALGORITHM FOR REAL-SYMMETRIC DATA

Jianping Chen *Henrik Sorensen*
Department of Electrical Engineering,
University of Pennsylvania,
Philadelphia, PA 19104

ABSTRACT

A very efficient algorithm for computing the DFT of real-symmetric input is presented. The algorithm is based on Bruun's algorithm where, except for the last stage, all twiddle factors are purely real. It is well-known that about half of the arithmetic operations and memory requirements can be removed when the input is real-valued. It may be assumed that another half of the computational and memory requirements can be similarly eliminated when the input is real and symmetric. This is, however, impossible with a standard radix-2 FFT, but can be achieved by the Bruun algorithm. The symmetries within the algorithm with for real-symmetric input are exploited to remove about three fourth of the butterflies and memory locations. The algorithm presented here achieves the same low arithmetic as the split-radix FFT for real-symmetric data, but has a structure that is as simple as the radix-2. The implementation on the TMS320C30 shows that the new algorithm fits a DSP processor very well. The presented program requires 0.51-0.60 ms to compute a length 1024 FFT with real-symmetric data.

1. INTRODUCTION

The fast Fourier transform (FFT) algorithm has been of interest for many years [1]. FFT algorithms have traditionally tried to minimizing the number of arithmetic operations, but with the advanced special purpose digital signal processors (DSPs) developed, recently it has become equally important that the algorithm has a simple data-flow and matches the DSP architectures [11,12,13]. It is well-known that the split-radix FFT has the lowest arithmetic complexity of any published FFT [7,9], but its irregular indexing scheme probably makes it less suitable for DSP implementations [11,12,14]. For many applications, the input sequence is real and symmetric and efforts have gone into developing more efficient algorithms for these cases [7,8]. The Bruun algorithm, when applied to real-symmetric data, has not only a low arithmetic complexity, but also a very simple structure and the implementation on a TMS320C30 shows that it exploits the DSP architecture well.

2. THE BRUUN FFT ALGORITHM

The DFT is normally defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \qquad k = 0, 1, \ldots, N-1 \tag{1}$$

where $W_N = e^{-j2\pi/N}$. This can be rewritten as a polynomial reduction

$$X[k] = X(z) \bmod (z - W_N^k) \tag{2}$$

where

$$X(z) = \sum_{n=0}^{N-1} x[n] z^n \bmod (z^N - 1) \tag{3}$$

To develop the Bruun FFT [3,4,5], factor the $Z^N - 1$ polynomial recursively using the identity:

$$z^{-4i} + az^{-2i} + 1 = (z^{-2i} + \sqrt{2-a}\,z^{-i} + 1)(z^{-2i} - \sqrt{2-a}\,z^{-i} + 1) \tag{4}$$

and reduce $X(z)$ modulo each of these factors. To determine the output blocks, the following equality is used.

$$z^{-2} - 2cos(2\pi k/N)z^{-1} + 1 = (z^{-1} - W^k)(z^{-1} - W^{-k}) \tag{5}$$

The signal flow-graph of the Bruun FFT can hence be obtained by translating the filter coefficient blocks into FFT butterflies. The resulting three basic butterflies are shown in Fig. 1.

The Bruun FFT changes the complex twiddle factors of the radix-2 FFT into real ones except for the last stage, and basically has the same simple structure as the radix-2 FFT. A direct advantage obtained is that the number of multiplications are nearly halved. More significant, however, is that the Bruun algorithm provides a convenient way of reducing the computational complexity when computing the DFT of real-symmetric data.

3. DEVELOPING THE REAL-SYMMETRIC ALGORITHM

Scrutinizing the Bruun FFT with real-symmetric data shows that at every stage the output of each group has an even or odd symmetry. Fig. 2 shows the signal flow-graph of the length-16 Bruun FFT for real-symmetric data. At each stage, the first group consisting of the butterflies without twiddle factors which transforms a N-point even-symmetric sequence into a $(N/2)$-point even-symmetric sequence and a $(N/2)$-point odd-symmetric one. The other groups of butterflies with twiddle factor F_i respectively transform an N-point odd-symmetric sequence into two $(N/2)$-point odd-symmetric sequences. In addition, the data at the symmetry point of every odd-symmetric sequence is zero. Due to these symmetries, the last stage becomes unnecessary since all the butterflies, except for the first one, only involves multiplication by one.

The above facts allow us to construct a real-symmetric valued FFT by skipping the lower half data and butterflies of every group and the last stage except for its first butterfly. Thus, about half of the arithmetic and memory requirements are removed. The data and butterflies that actually need to be computed are drawn in boldface and solid line respectively in Fig. 2.

With a proper rearrangement to the remaining data and butterflies, an in-place computation is possible. For the

0-7803-0532-9/92 $3.00

group of butterflies without twiddle factors, two butterflies, the kth and the $(N/4+1-k)$th butterflies, are computed simultaneous, where N is the group length and $k = 1, 2, ..., N/8$. The lower input of each of them is fetched from the location of its symmetry point. Then this location is used to receive the lower output of the other butterfly. In this way, the two butterflies can be computed in-place. An example of the combined butterfly is given in Fig. 3 (e), which corresponds to the first and the fourth butterflies in the first stage of the signal flow-graph in Fig. 2. The remaining $(N/4+1)$th butterfly is computed separately, using only one addition since its lower output is zero.

To compute the main butterflies in-place, the lower input of each butterfly can still be fetched from the location of its symmetry point, with the negative sign incorporated into the butterfly. The corresponding lower output must be placed into the location of the middle input. However, the data at the location of the middle input of the kth butterfly is also used by the $(N/4+2-k)$th butterfly, because that location is the symmetry point of the lower input of the latter butterfly. This problem can be nicely solved by computing the two butterflies together, since, as we find, the location of middle input of the $(N/4+2-k)$th butterfly is just the symmetry point of the lower input of the kth butterfly regularly. Here, N is still the group length and $k = 2, 3, ..., N/8$. To illustrate this point, a length-16 main butterfly group is given in Fig. 3 (a), where $k = 2$ and $N/4+2-k = 4$. As can be seen, the 5th location containing $c5$ is not only the middle input of the 2nd butterfly but also the symmetry point of the lower input $(-c5)$ of the 4th butterfly. Similarly the 7th location with $c7$ is both the middle input of the 4th butterfly and the symmetry point of the lower input $(-c7)$ of the 2nd butterfly. The lower inputs of the 2nd and 4th butterflies are fetched respectively from the 7th and the 5th locations. After computation, the 5th and 7th location are used to store the lower outputs of the 2nd and the 4th butterflies respectively. The resulting combined butterfly is shown in Fig. 3 (b). The first and the $((N/8)+1)$th butterflies are computed separately. The first butterfly can be simplified to save one addition since its lower input is zero. The middle input of the $((N/8)+1)$th butterfly is just the symmetry point of the lower input of the butterfly itself and this butterfly hence can be computed singly. The two butterflies are shown respectively in Fig. 3 (c) and (d).

The arithmetic complexity to compute the length N real-symmetric Bruun FFT is $(1/4)N(M-3)+1$ multiplications and $(1/4)N(3M-7)+2M+2$ additions, where $M = log_2 N$. This count is the same as the split-radix FFT for real-symmetric data [7] (notice the counts in [7] exclude multiplications by 2), and is obviously lower than that of Ziegler's real-symmetric valued radix-2 algorithm [8]. Moreover, the structure of the new algorithm is still simple, since it is formed by simply cutting all groups of butterflies into halves. It is even simpler than that of the radix-2 FFT for real-symmetric data [8]. Furthermore, the storage requirement is reduced. Only $(N/2)+1$ memory locations are needed for a length-N transform and the computation is in-place.

4. THE UNSCRAMBLING OF THE OUTPUT ORDER

The major disadvantage of the Bruun algorithm lies in the irregularity of its output order. Although the Bruun algorithm has been researched extensively [3, 4, 5], this problem remains unsolved.

We have found two ways of unscrambling the output order of the Bruun FFT. One is recovering it directly into the regular order by using an unscrambling table. The correct position of each output is put in the table in advance, and the storing of the output is guided with the table. This method is simple and needs very few operations, but requires a second memory section for the output data. The other method is to first change the output order into the bit-reversed one, and then from the bit-reversed one into the regular order. In this latter method, the first step can be done in-place in a swapping pattern, and for the second step the most of today's DSPs have a bit-reversed addressing mode to allow the bit-reversal being done for free during the retrieval of the data. To change the output order into a bit-reversed order, certain pairs of data must be swapped in a stage-by-stage fashion. The swapping pattern is quite regular, and is easily programmed.

5. THE TWIDDLE FACTORS OF THE BRUUN ALGORITHM

The Bruun algorithm involves two kinds of twiddle factors, the real factor F_i for the main butterflies and the complex factor W^k in the last stage. It is more convenient to use the derivation in [4] for the twiddle factor generation, which allows the real twiddle factors to be produced by the following formula.

$$F_i = 2cos(2\pi K(i)/N) \qquad i = 2, 3, \ldots, N/4 \qquad (6)$$

where K(i) relates to the index of the output order. $(N/4)-1$ real factors (F_i) are needed for a length-N FFT.

6. IMPLEMENTATION ON THE TMS320C30

Today's digital signal processors like the TMS320C30 have many advanced features, such as parallel instructions, pipelining and block-repeat capabilities. To achieve an efficient implementation, besides the choosing algorithm has a lower arithmetic complexity and a simple structure, adopting appropriate programing techniques to exploit those available capabilities is indispensable.

Since the butterfly computation is the core part of the FFT algorithm, minimizing the instruction count required for it can greatly speed up the program, where the parallel instruction set is a powerful tool. To maximize the occurrence of parallel instructions, techniques like computing more butterflies at a time and the task-interleaving, which means some operations of the previous or next butterfly are interleaved into the computation of the current butterfly [12, 13], can be used. For example, to compute the butterflies without twiddle factors, a direct implementation uses six instructions (but 7 clock cycles due to a memory conflict). The improved version only uses five instructions (6 cycles). The main butterfly is computed in eight instructions (cycles); down from ten.

Another effective measure of speeding up the FFT algorithm is to reduce the structural overhead. Normally a triple-nested loop is used for FFT implementations. In the standard implementation of the DIT FFT with in-order input, butterflies are computed group by group throughout all the stages. In the earlier stages, this fashion is efficient since the butterflies are distributed in only a few groups. For the later stages, however, the number of groups in a stage increases and the number of butterflies in a group goes down, which will result in much overhead. Introducing another kind of looping mode in the later stages can effectively eliminate this increase of overhead. Instead of group by group, we may compute the butterflies across groups, which means in each loop the butterflies at the same position of all groups are computed. The reduction of the overhead is drastic, especially when two butterflies are computed simultaneously. For example, to compute the fifth stage of a length-256 real FFT, 120 cycles of overhead are required if the butterflies are computed group by group. In contrast, only 32 cycles are needed in the new way. The actual saving of cycles is even more since at every loop several other instructions are executed to set up the RC register etc.

Algorithm\Length		64	256	1024
Real Symmetric Bruun	a	0.022	0.104	0.51
	b	0.027	0.124	0.60
	c	0.026	0.120	0.57
Real Bruun [17]	a	0.046	0.224	1.10
	b	0.054	0.265	1.31
	c	0.049	0.237	1.15
Real radix-2 [14]		0.075	0.354	1.67
Complex radix-2 [15]		0.090	0.473	2.43

Table 1: FFT timing on TMS320C30 DSP (in milliseconds). a: without unscrambling, b: unscrambling in-place in bit-reversed order c: unscrambling using a second array

Besides the arithmetic operations, instructions and cycles also go into memory accesses. Possible combination of certain stages or groups can reduce these operations. As was done in [12], every two groups of the butterflies without twiddle factors are combined. Two butterflies of the front stage are computed together with the linked butterfly in the rear stage. Thus, the transfers of two data elements are cancelled and the instruction (cycle) count for the three butterflies is reduced to seven. The two stages can be computed in a single loop to reduce the overhead [10,11,15]. The data transfers, however, are also reduced if the two stages are combined. The four butterflies - one group of the second last stage and the two linked butterflies in the last stage can be computed at a time. Since the output of the second last stage now need not to be stored, the memory access is reduced.

To achieve an efficient program code, avoiding pipeline conflicts can not be neglected, especially when a high parallelism is performed. With an extensive use of the register set, proper arrangement of instructions and by using the task-interleaving technique, the pipeline conflicts are prevented as far as possible in our programs.

Another purpose that we combine the last and the second last stages is to resolve pipeline conflicts. Since the outputs of the butterflies in the second last stage are kept in extended-precision registers as the inputs to the last stage they no longer need to be fetched from memory, hence causing no wait-state. About 25 % cycles (for the stage) are saved due to this arrangement.

7. RESULTS

As a result, a very efficient assembly language program is generated. The execution time required for various FFT lengths is given in Table 1. For comparison, the manufacturer's benchmark for the real-valued radix-2 FFT program [14] is also shown in the table. As can be seen, for a length-1024 real FFT the presented program needs only 0.60 milliseconds, which is less than one quarter of the time required by the most efficient complex radix-2 program [15] (2.43 ms) and about half the time for the real Bruun (1.31 ms).

This shows that the Bruun FFT is just as efficient as the best radix-2 programs, but where the radix-2 FFT can not easily be modified to efficiently handle real-symmetric data, the Bruun algorithms does that very well. In other words, the efficient programming techniques that lead to the results in [15] can also be applied to achieve a real-data program, but not a real-symmetric program, because the radix-2 FFT does not have any direct redundancies that can be exploited to reduce the computational and memory requirements.

8. CONCLUSION

The good performance of the Bruun FFT algorithm for real-symmetric data is demonstrated. Because of the inherent real twiddle factors and inherent symmetries within the Bruun FFT, a special purpose algorithm with one quarter of the operations and memory requirements of a complex FFT can be obtained, while a simple structure is always preserved. The high efficiency of the resulting algorithms has been proved by the implementation on the DSP processor. The unscrambling of the irregular output order costs the Bruun algorithm more operations or memory locations. Nevertheless, the presented real and real-symmetric Bruun algorithms seem to be the best for real and real-symmetric data, especially when an unscrambling is not required or a second memory section is available for the unscrambling.

9. ACKNOWLEDGEMENTS

We gratefully acknowledge the support of Texas Instruments under contract MSC481-W16 to pursue this work. The programs are available on the TI BBS system and a more detailed description of the work can be found in [17].

10. REFERENCES

1. J. W. Cooly and J. W. Tukey, "An algorithm for the machine calculation of the complex Fourier series", Math. Comput., vol. 19, pp. 297-301, Apr. 1965.
2. A. V. Oppenheim and R. W. Schafer, "Digital Signal Processing", Englewood Cliffs, NJ: Printice-Hall, 1975.
3. G. Bruun, "Z-Transform DFT Filters and FFT's", IEEE Trans. ASSP, vol. ASSP-26, pp. 56-63, 1978.
4. R. Storn, "A Novel Radix-2 Pipeline Architecture for the Computation of the DFT", ISCAS'88, pp.1899-1902, 1988.
5. H. J. Nussbaumer, "Fast Fourier Transform and Convolution Algorithm", New York: Springer-Verlag, 1982.
6. H. V. Sorensen, et al., "Real-Valued Fast Fourier Transform Algorithms", IEEE Trans. ASSP, vol. ASSP-35, pp. 849-863, 1987.
7. P. Duhamel, "Implementation of 'Split-Radix' FFT Algorithms for Complex, Real, and Real-Symmetric Data", IEEE Trans. ASSP, vol. ASSP-34, pp. 285-295, 1986.
8. H. Ziegler, "A Fast Fourier Transform Algorithm for Symmetric Real-valued Series", IEEE Trans. Audio Electroacoust., vol. AU-20, pp. 353-356, 1972.
9. H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On Computing the Split-Radix FFT", IEEE Trans. ASSP, vol. ASSP-34, pp. 152-156, 1986.
10. C. S. Burrus and T. W. Parks, "DFT/FFT and Convolution Algorithms", New York: Wiley, 1984.
11. R. Meyer, et al., "FFT Implementation on DSP-Chips - Theory and Practice", Proc. IEEE ICASSP, New Mexico (1990), pp. 1503-1506.
12. R. Meyer and K. Schwarz, "FFT Implementation on DSP-Chips", preprint.
13. H. V. Sorensen, et al., "Efficient FFT algorithms for DSP processors using tenser product decompositions", Proc. IEEE ICASSP, New Mexico (1990), pp. 1507-1510.
14. P. E. Papamichalis, "FFT Implementation on the TMS320C30", Proc. IEEE ICASSP New York (1988), pp. 1399-1402.
15. R. Meyer and K. Schwarz, "Complex, radix-2 FFT: R2DIT.ASM", from Texas Instruments TMS320 software library.
16. Third-Generation TMS320 User's Guide. Texas Instruments, Jan., 1989.
17. J.P. Chen and H.V. Sorensen, "Implementation of the Bruun FFT algorithm for Real and Real-Symmetric Data", To be submitted, 1991

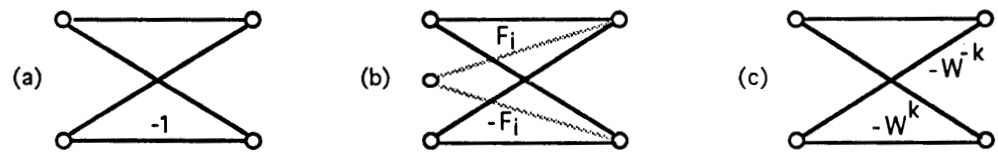

Figure 1: Three basic butterflies of the Bruun algorithm. (a) Butterfly in the first groups. (b) Main butterfly, where F is real. (c) Butterfly in the last stage.

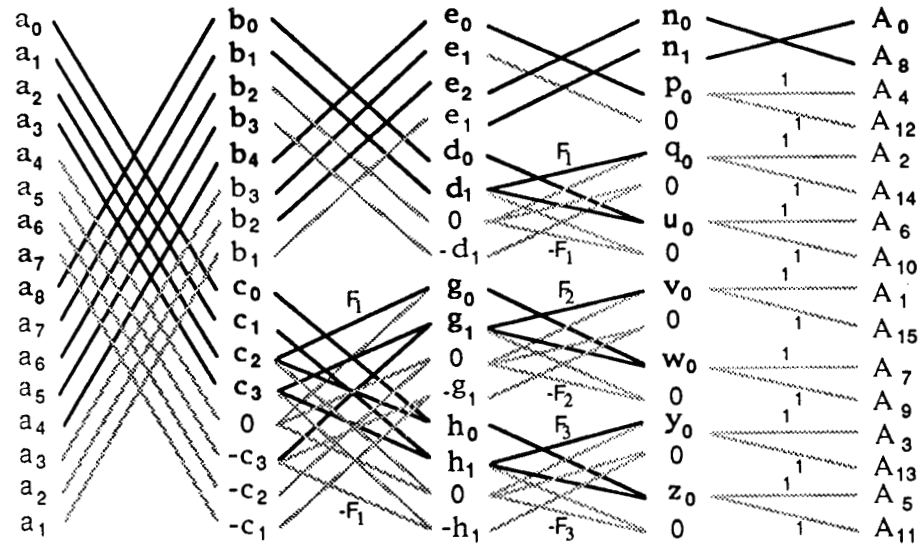

Figure 2: A length-16 Bruun FFT with real-symmetric input. The symmetries at each stage are indicated by the same letters. Only butterflies drawn in solid lines and the data elements in boldface need be computed.

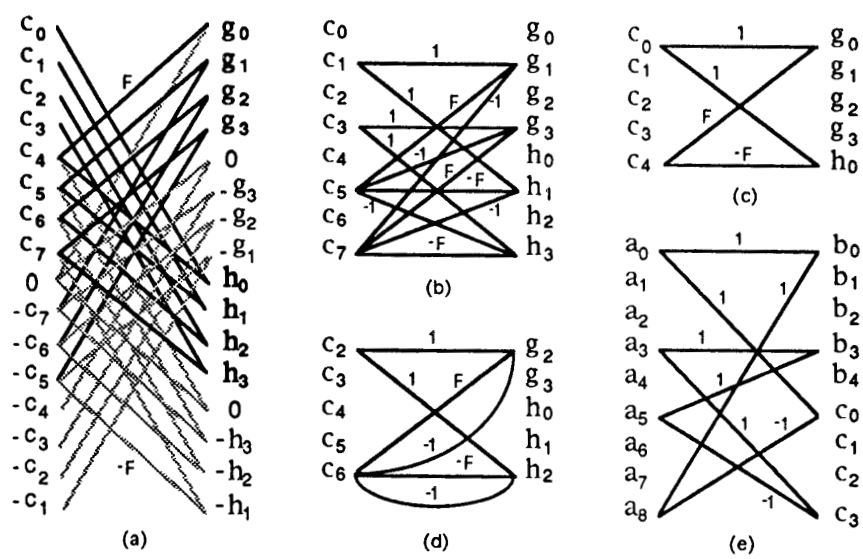

Figure 3: (a) A length-16 main butterfly group. (b) The combined two main butterflies. (c) The first main butterfly. (d) The $(N/8+1)$th main butterfly. (e) The combined two butterflies without twiddle factors.