APPENDIX

At steady-state, the expected value of $R^{-1}(i)$ is given by

$$\lim_{m \to \infty} E\{R^{-1}(i)\} = (1 - \lambda)R_e^{-1}$$

(21)

where $R_e$ is the true input autocorrelation matrix (14). The inverse autocorrelation estimate $R^{-1}(i)$ (7) behaves like a quasideterministic quantity in the steady-state [7], [16], [20] when the forgetting factor and the filter order $N$ satisfy the condition $N(1-\lambda) \ll 1$, which is discussed in Section III. In [5], this is shown for Gaussian input samples by the averaging principle of [15] for $N(1-\lambda) \ll 1$ and is also checked by simulations. Therefore, at steady-state, we can replace $R^{-1}(i)$ by its expected value (21) and evaluate the following expectations:

$$E\{k(i)x^T(i)\} = E\{R^{-1}(i)x(i)x^T(i)\}$$

$$= (1 - \lambda)R_e^{-1} E\{x(i)x^T(i)\}$$

$$= (1 - \lambda)I$$

(22)

and

$$E\{k(i)k^T(i)\} = E\{R^{-1}(i)x(i)x^T(i)R^{-1}(i)\}$$

$$= (1 - \lambda)^2 R_e^{-1} E\{x(i)x^T(i)\}R_e^{-1}$$

$$= (1 - \lambda)^2 R_e^{-1}$$

(23)

where we have also used the definitions of Kalman gain (6) and the input autocorrelation matrix $R_e$ (14).

By following the same argument and definitions, we can also evaluate the expectation

$$E\{k(i)x^T(i)\theta(i-1)\theta^T(i-1)x(i)k^T(i)\}$$

$$= (1 - \lambda)^2 \text{trace}(R_eR_e(i-1)R_e^{-1})$$

$$+ 2R_e(i-1)).$$

(24)

Here, we have used the assumption that the input sequence is a Gaussian process and have employed the expansion of a fourth-order jointly Gaussian process in terms of its second-order moments. The derivation of this expectation follows a procedure similar to the one in [5].

REFERENCES


New FFT Bit-Reversal Algorithm
Juan M. Rius and R. De Porrat-Döría

Abstract—This correspondence presents a very short, simple, easy to understand bit-reversal algorithm for radix-2 fast Fourier transform (FFT), which is, furthermore, easily extensible to radix-M. In addition, when implemented together with Yong's technique, the computing time is comparable to that of the fastest algorithms.

I. INTRODUCTION

A number of bit-reversal algorithms have been published in recent years in order to improve the classical and commonly used Gold–Rader's algorithm [1]. In 1989, Rodriguez [2] achieved a small improvement by determining the largest array index that must be swapped. More recently, in 1991, Yong [3] presented a very clever idea to reduce the number of loop iterations from $N-1$ to $N/4$.

Manuscript received May 15, 1992; revised September 12, 1994. The associate editor coordinating the review of this paper and approving it for publication was Prof. Henrik V. Sorensen.

The authors are with the Antennas, Microwave and Radar Group, Department Teoria del Senyal i Comunicacions, Universitat Politècnica de Catalunya, Barcelona, Spain.

IEEE Log Number 9409778.
 TABLE I  
INTERGRS FROM 0 TO 15 (N = 16) AND THEIR BIT-REVERSALS.  
THE GENERAL RULE FOR ORDINARY OR BIT-REVERSAL COUNTING IS  
BASED ON COMPUTING THE NUMBERS FROM 2^E TO 2^{E+1}-1 BY  
ADDDING THE BOLD-TYPEDBITS TO THE NUMBERS FROM 0 TO 2^E-1

<table>
<thead>
<tr>
<th>2^E to 2^{E+1-1}</th>
<th>Original</th>
<th>Bit-reversal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000</td>
<td>0000</td>
</tr>
<tr>
<td>E=0</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>E=1</td>
<td>2</td>
<td>0100</td>
</tr>
<tr>
<td>E=2</td>
<td>3</td>
<td>0110</td>
</tr>
<tr>
<td>E=3</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1101</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>1011</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

On the other hand, Evans [4] presented in 1987 and improved in  
1989 [6] an algorithm that obtains directly the array indexes for the  
swapping operation, without doing the count from 0 to N-1 and  
without comparisons between an index and its bit reversal. In 1990,  
Walker [5] improved the algorithm for N=16 to an odd power of  
2 and presented a very efficient implementation. These algorithms  
are faster than the classic Gold-Rader's or Rodriguez's, but they require  
the use of a seed table on the order of the square root of  
N. Recently, Orchard [7] presented a different counter procedure  
that, applied with the Gold-Rader or Evans algorithms, reduces  
dramatically the computing time in RISC machines.

II. BIT-REVERSAL PROCEDURE

The algorithm presented here is extremely simple and generates a  
swap table quite efficiently. When implemented together with Yong's  
technique, it becomes one of the faster algorithms known to the  
authors. In addition, the table generated can be used in subsequent  
FFT's when the number of points is the same.

The rule for the bit-reversal procedure can be seen in Table I, which  
shows the integers from 0 to 15 (N = 16) and their bit-reversals.  

The rule for the ordinary count from 0 to N-1 is the following:  
the numbers from 2^E to 2^{E+1-1} are equal to the numbers from  
0 to 2^E-1 plus 2^E. In the same way, in the bit-reversal count,  
we must add the bit-reverse of 2^E, which is 2^E N^{E-1}, so that  
the general rule becomes "The bit-reversals of numbers from 2^E to  
2^{E+1-1} are equal to the bit-reversal of numbers from 0 to 2^E-1  
plus 2^{E+1-1} N^{E-1} - 1 + 2^E."

An implementation of this routine in C language is

```c
void rius(N, J, X) int N; int J[N]; complex X[N]; { int k, l, i, j, N2, N4, N21; N2=N/2; N4=N2/2; N21=N2+1; j=0; J[0]=0; swap(i, N2, X); for(i=2, l=2, r=N/4; r>l; l*=2, r/=2) { J[i]=J[k]+r; if(i<J[i]) swap(i, J[i], X); } }
```  

The operation count for this algorithm is, without including index  
calculations  
N- 2 integer additions (+r)  
2(N- 2) integer increments (k++, i++)  
(log, N)- 1 multiplications by 2 and (log, N)- 1 divisions by 2,  
plus two more divisions N/2 and N/4, which can be efficiently  
implemented in assembly language using bit shifts.

This algorithm requires the storage of a table of N/2 index  
numbers, since it is not strictly necessary to store the bit-reversal  
of the index from N/2 to N- 1.

III. IMPROVEMENT WITH YONG'S ALGORITHM

The idea of Yong to reduce the inner loop iteration count from  
N- 2 to N/4 is the following [3]: If we know that the bit-reversal  
of even integer I < N/2 is J, then  
i) the bit-reversal of I+ 1 is J + N/2, which is always greater  
    than I+ 1,  
ii) the bit-reversal of I + N/2 is J + 1, which is never greater  
    than I + N/2, and  
iii) the bit-reversal of I + N/2 + 1 is J + N/2 + 1, which is  
    greater than I + N/2 + 1 if J is greater than I  
so that it is necessary to compute only the bit-reversals of even  
tegers less than N/2, thus reducing by a factor of 4 the number  
of inner loop iterations.

An implementation of the algorithm presented in this correspon-  
dence, together with Yong's improvement is

```c
void rius-yong(N, J, X) int N, J[N]; complex X[N]; { int k, l, i, j, N2, N4, N21; N2=N/2; N4=N2/2; N21=N2+1; j=0; J[0]=0; swap(i, N2, X); for(i=2, l=2, r=N/4; r>l; l*=2, r/=2) { j=J[i]=J[k]+r; if(i<j) swap(i, j, X); swap(i+N2, j+N21, X); } }
```  

The memory requirements are a table of N/8 index numbers,  
instead of N/2. For N > 64, the N/8 memory requirement is larger  
than the sqrt(N) requirement in Evans' algorithm. However, this is not  
a problem if
i) Memory limitation is not important.  
ii) A large number of FFT with the same number of samples N  
   must be computed; The bit-reversal algorithm should be run  
   only once and the swapped indexes stored in an N entry table  
   with either algorithm.
TABLE II
CPU Timing (in Nanoseconds) Normalized by \( N \) for the Different Bit-Reversal Algorithms Mentioned in This Correspondence. Swap Operations Are Not Included. The Algorithms Have Been Programmed in C Language and Tested in a PC-486/33 MHz Personal Computer, Using a 32-Bit Protected Mode C Compiler, and a Hewlett-Packard 730 Workstation

<table>
<thead>
<tr>
<th>Architecture</th>
<th>CISC (PC-486/33 MHz)</th>
<th>RISC (HP-730)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power of 2</td>
<td>10 11</td>
<td>10 11</td>
</tr>
<tr>
<td>Gold+Rader</td>
<td>635 636</td>
<td>199 200</td>
</tr>
<tr>
<td>Gold + Orchard</td>
<td>423 421</td>
<td>112 111</td>
</tr>
<tr>
<td>Gold-Rader + Yong</td>
<td>186 185</td>
<td>54,4 54,1</td>
</tr>
<tr>
<td>Rius + Yong</td>
<td>97,6 92,4</td>
<td>47,7 46,9</td>
</tr>
<tr>
<td>Evans-89 + Table</td>
<td>145 68,2</td>
<td>51,3 25,7</td>
</tr>
<tr>
<td>Walker-90 + Table</td>
<td>122 63,8</td>
<td>28,6 14,5</td>
</tr>
<tr>
<td>Evans + Orchard</td>
<td>292 159</td>
<td>47,8 23,9</td>
</tr>
</tbody>
</table>

Further reduction in bit-reverse operation count could be achieved by recursive application of Yong’s factor 4 reduction, but it must be noted that recursive function calls in high-level programming languages are usually very slow. Actually, a fully recursive implementation of Yong’s algorithm has been programmed by the authors of this correspondence with discouraging run-time results.

IV. TIMING COMPARISON OF BIT-REVERSAL ALGORITHMS

All the bit-reversal algorithms mentioned in this correspondence are compared in Table II. The timing results do not include the swap operations and have been normalized by \( N \). Indeed, the program does the swap operation, but its computing time has been subtracted from the total time. This is more realistic than removing the swap from the program, which could result in an undesired compiler optimization. The programs were written in C language and tested on a PC-486/33 MHz personal computer and a Hewlett-Packard 730 workstation. The C compiler in the PC used 32-bit protected mode to generate a very efficient and fully optimized code.

V. EXTENSION TO RADIX-\( M \)

The Rius algorithm, which works for radix-2 FFT, can be easily extended to radix-\( M \), where \( M \) is any power of 2. When the radix is larger than 2, the reversing process involves groups of bits instead of one single bit, and, consequently, the rule changes. This is illustrated in Table III, which shows the integers from 0 to 15 and their radix-4 bit-reversals.

The extension of the Rius algorithm to radix-\( M \) is the following:

```c
porrata-rius (N, M, J, X)
int N, M;

int J[k]; complex X[i];
{
    int k, l, r, i, nm;
    J[0] = 0;
    nm = N/M;
    for (i=1; i<M; i++)
        {J[i] = i*nm;
         swap (i, J[i], X);
        }
    for (i=M, l=M, r=nm/M; r>0; i=i*M, r/=M)
        for (k=0; k<(M-1)*l; k++, i++)
            {J[i] = J[k] + r;
             if (i<J[i]) swap
                (i, J[i], X);
            }
}
```

The Yong approach to reduce the computing time is based on the structure of the radix-2 bit-reversal operation, which makes it impossible to apply when the operation changes, that is, when the radix is higher than 2.

VI. CONCLUSIONS

A new bit-reversal algorithm has been presented. The fact that it is extraordinarily simple, easy to understand, short, and very easy to extend to radix-\( M \) makes it the most practical algorithm known to the authors. In addition, when implemented together with Yong’s technique, the computing time is comparable to that of the fastest algorithms.

REFERENCES

Abstract—This work is concerned with the boundary conditions involved in processing a finite discrete-time signal with a critically sampled perfect reconstruction filter bank. It is desirable that the boundary conditions represent edge effects and define a transformation into a space having the same dimensionality as the original signal. The complication that arises is in the computation of the inverse transform: Although it is straightforward to reconstruct the signal values that were not influenced by the boundary conditions, recovering those values on the boundaries is nontrivial. The solution of this problem is discussed for general linear boundary conditions. No symmetry assumptions are made on the boundary conditions or on the impulse responses of the analysis filters. A low-rank linear transform is derived that expresses the boundary conditions in terms of the transform coefficients. This in turn provides a method for inverting the subband decomposition. In contrast to the result in [5], the inverse problem is not overdetermined. 

Some comments are in order as to the notation used here. Matrices are denoted by uppercase letters and vectors by boldface lowercase letters. $R$ denotes the set of real numbers, $R^n$ the set of all $n$-dimensional vectors over the reals, $Z$ the set of integers, and $Z^+$ the set of positive integers. $\delta_{n,m}$ is the Kronecker delta function, which is defined for $n, m \in Z$ and is equal to one for $n = m$ and is zero otherwise. All vectors and matrices are indexed beginning at zero. $J_n$ is the $n \times n$ matrix with $i$th element $h_{i+1,n-1}$. Since premultiplication of a matrix by $J_n$ reverses the order of the rows, it is commonly referred to as the reflection matrix. The support of a discrete-time filter $h$, denoted as $\text{supp}(h)$, is defined as the largest set \{ $n \in Z$ $|$ $0 \leq n \leq n_1$ \} such that $h(n_0) \neq 0$ and $h(n_1) \neq 0$ are both nonzero.

II. PERFECT RECONSTRUCTION MULTIRATE FILTER BANKS

An $M$-channel perfect reconstruction multirate filter bank (PRMBF) is depicted in Fig. 1 [7]. Analysis consists of convolution of the discrete-time signal $u$ with each $h_i$ (for $i = 1, 2, \ldots, M$) followed by $M$-fold decimation in each channel. The $M$ subband sequences are denoted by $u_i$. Synthesis is achieved by an $M$-fold upsampling of each subband, followed by convolution with the appropriate $g_i$ and then summing these filter outputs. The $h_i$ and $g_i$ are designed such that in the absence of numerical roundoff, $\hat{u}$ is equal to $u$, i.e., the filter bank exhibits perfect reconstruction. Since the downsample factor in each channel is equal to $M$, the overall rate of data in the subband decomposition is the same as for $u$, and the system is said to be critically sampled. In general, PRMBF's are designed such that $\hat{u}$ is equal to $u$ up to delay. Such delays are undesirable in image coding applications since this would result in a shift in the decoded image. In this work, the term "perfect reconstruction" is not assumed to include a delay.

In the analysis that follows, it is convenient to describe the actions of the system in Fig. 1 in terms of a block matrix representation along the lines of that presented in [8]. The $M \times M$ matrices $A_i$ and $B_i$ are defined for all $i \in Z$ as

\[(A_i)_{mn} = h_{i+1}(-Mn - m),\]

and

\[(B_i^T)_{mn} = h_{i+1}(-Mi - m).\]