CHAPTER 6

PARALLEL ARCHITECTURES FOR GTFDs

6.1. INTRODUCTION

The advent of Fourier transform has revolutionized the implementation of many signal processing algorithms in real-time; The implementation of generalized time-frequency representations is no exception. The interpretation of GTFDs as the Fourier transform of the generalized autocorrelation function has motivated many researchers to consider applying the FFT techniques, e.g., the butterfly structure ,to compute the GTFDs without paying any attention to the computation of the GACF (Qian *et al*, 1990). In this Chapter, we shall deviate from this viewpoint and propose a method that directly operates on the signal and requires no knowledge of the kernel in the Fourier domain. We use time-recursive approach to compute the GTFDs and show that this approach is very suitable for computing running-windowed GTFDS. Time-recursive approach to compute discrete Sine/ Cosine/ Hartley transform has gained much attention because its hardware for serial data has less complexity, as many applications involve the computation on sequential data (Liu *et al*, 1994). The butterfly structures have less latency compared to time-recursive approaches but they are more complex in terms of hardware and need global interconnections. Some of the reasons to use time-recursive approach instead of the butterfly structure are because it:

- Has local communications.
- Is highly parallel, modular and regular.

- Is less complex in terms of hardware.

The time-recursive approach is similar to the multiply and accumulate (MAC) operation as in many digital signal processors. It does not decimate the sequence in either time or frequency but directly operates on the sequence and acts as a mere summation involving multiplications. We first review the time-recursive approach to compute the discrete Fourier transform and then deal with STFT and GTFDs.

6.2. DISCRETE FOURIER TRANSFORM

The discrete time Fourier transform of a discrete time sequence *x(n)* can be related to continuous time Fourier transform, given by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n}, \tag{6.1}$$

where ω is a continuous variable denoting the frequency. In general, we operate on finite size sequences and hence we compute the discrete time Fourier transform of an N-point sequence as:

$$X(\omega) = \sum_{n=0}^{N-1} x(n) e^{-j\omega n}. \tag{6.2}$$

The discrete Fourier transform is obtained by sampling the frequency spectrum as:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nk}{N}}, \; k = 0,1...,N-1. \tag{6.3}$$

To compute the above equation in a time recursive fashion, we start with the induction

step as:

- Induction: $X(-1,k) = 0$.
- Compute: $X(t,k) = \left[X(t-1,k) + x(t)\right] e^{j\frac{2\pi k}{N}}$.
- Stop at $t = N-1$, assign $X(k) = X(N,k)$.

We get the DFT of the sequence after *N* such iterations. The architecture for the above equation is shown in Fig. 6.1. Using only real multiplications, the above equation can be rewritten as:

Let $X(t,k) = X_r(t,k) + jX_i(t,k)$ and similarly $x(n) = x_r(n) + jx_i(n)$, then

$X_r(t+1,k) = X_r(t,k)\cos(\omega_k) - X_i(t,k)\sin(\omega_k)$, where $\omega_k = 2\pi k / N$ and

$$X_i(t+1,k) = X_i(t,k)\cos(\omega_k) + X_r(t,k)\sin(\omega_k), \tag{6.4}$$

which can be implemented as shown in Fig. 6.2. The total number of complex multipliers required are *(N-1)* and the number of complex adders needed are *(N+1)*. A radix-2 FFT algorithm requires $\frac{N}{2}\log_2(N)$ complex multipliers. However, the number of complex multplications that ripple through the stages from input to the output are $\log_2(N)$ in the case of radix-2 FFT whereas they are *N* in time-recursive approach. Throughout our discussion we use this measure to compare the throughput rates of different schemes in terms of complex multiplications defined as:

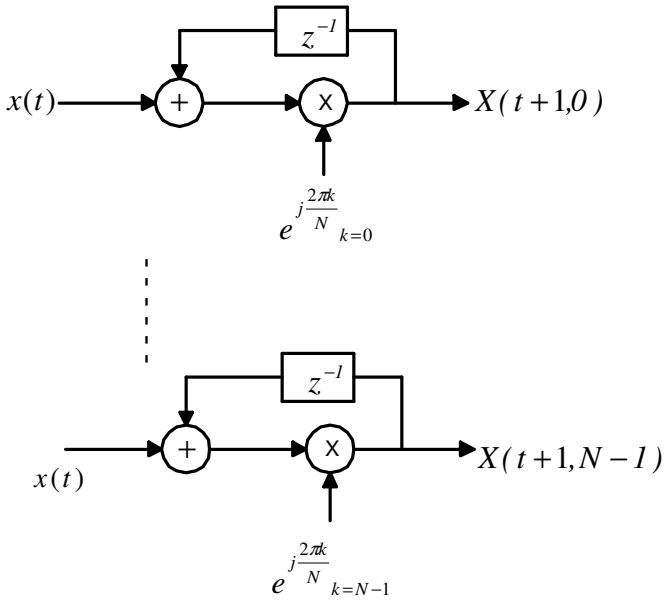

Fig.6.1. Architecture for DFT

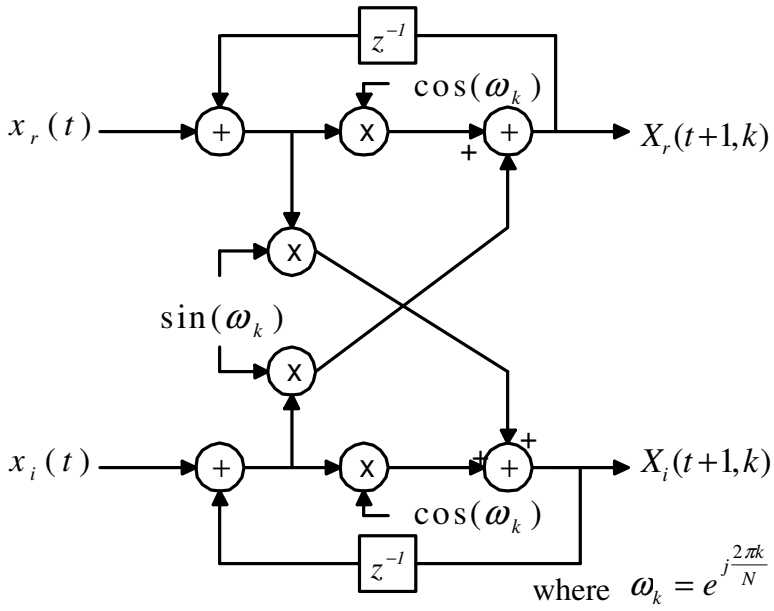

Fig. 6.2. Implementation of Eqn. (6.4) using real multiplications

OPs = Number of complex multiplications whose output ripples through the succeeding stages. (6.5)

While defining the *OPs,* we have not considered the complex multiplications that take place across parallel stages as long as they do not effect the remaining parallel stages. We are considering the computational complexity in terms of complex multiplications only and not complex additions since multiplications are more cumbersome than additions. However, as a special, our comparison of various algorithms in terms of *OPs* may not be valid when the complexity of additions exceeds the complexity of multiplications. We observe that the data flow the architecture assumes is sequential and serial. When we mean sequential, the data is quite regular, i.e., $x(0), x(1), x(2) ..., x(N-1)$ and if the same order is not followed e.g., $x(2), x(N-1), x(0) ..., x(1)$, we call it as nonsequential data. In many cases the data can be nonsequential and we need to wait until all samples arrive and then start pumping them serially in to the architecture in which case we need this extra buffering time when compared to the radix-2 algorithms that operate on parallel data. To circumvent the problem, we have proposed a pre-processor that avoids this buffering time. The pre-processor design is just trivial but of theoretical importance when the throughput rate is important. We consider this issue in the following Section.

6.3. COMPUTATION OF DFT FOR NONSEQUENTIAL DATA

Most of the algorithms that compute DFT consider either sequential serial or parallel data. The DFT/ inverse DFT implemented using time-recursive approach where the data is of nonsequential nature, necessitates buffering of data to make it sequential. We

propose a pre-processor, which takes care of the nonsequential nature of the data. The pre-processor has to introduce a pre-multiplication factor of $e^{j\frac{2\pi k\tau}{N}}$, where τ is determined by the occurrence of the bit position, e.g., a bit delayed by L samples has to undergo a phase offset of $e^{j\frac{2\pi k(N-L)}{N}}$ and for L bit positions in advance it is $e^{j\frac{2\pi kL}{N}}$. Selection of proper value of the pre-multiplication factor is determined jointly by *modulo-N* counters which increment their content by one every time and a two-dimensional kernel that holds the multiplication factor, the address of which is determined by the content in the counter. The architecture is shown in Fig. 6.3.

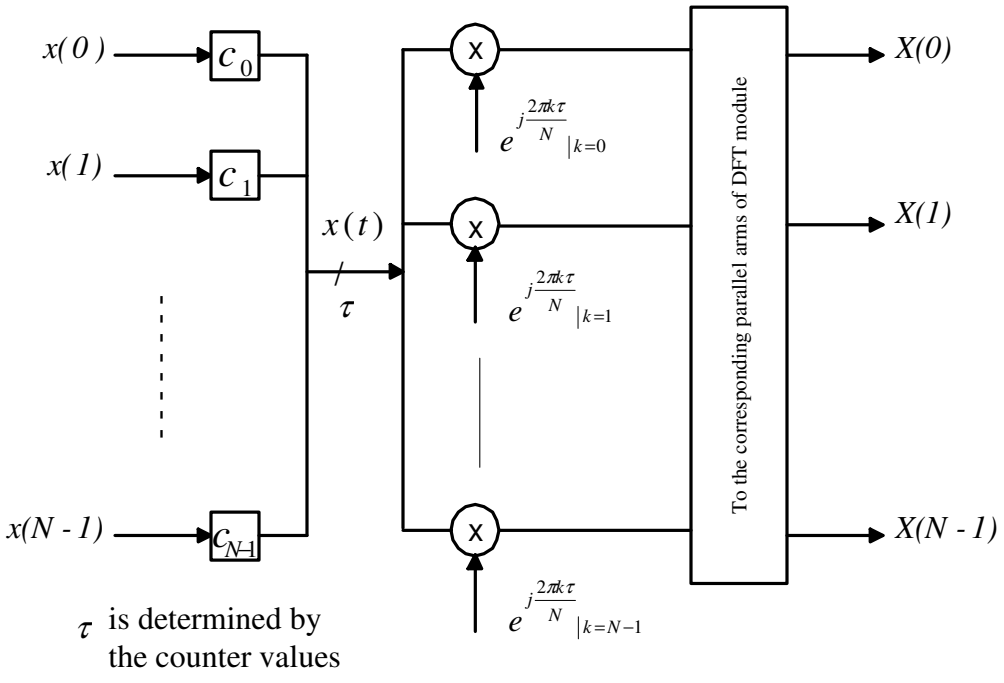

Fig. 6.3. Architecture of the pre-processor for computing DFT of nonsequential data

The algorithm is described below:

- Initialize each counter C_i as $C_i = (N - i - 1)$ for $i = 0,1,...,N - 1$.
- Increment the counters unless and until that particular bit position has occurred.
- Determine which column of the two dimensional kernel, defined as $\phi(\mathrm{k}, \tau) = \mathrm{e}^{\mathrm{j}\frac{2\pi \mathrm{k}\tau}{\mathrm{N}}}$, has to be selected using the final count in the *modulo-N* counter corresponding to bit-position whose data is available.
- Feed the pre-multiplied data to the corresponding parallel arms of the DFT module.

The remaining operations are similar to that of DFT described in Section 6.3. Various stages of the algorithm are exemplified in the Table 6.1 shown below, for a five point sequence given in the order $x(4), x(2), x(3), x(0), x(1)$.

Table 6.1: Illustration of the algorithm for the test sequence *x*

Natural order	Non sequential	c_i	t=0	t=1	t=2	t=3	t=4	t=5
x(0)	x(4)	c_0	4	0	1	2	**3**	X
x(1)	x(2)	c_1	3	4	0	1	2	**3**
x(2)	x(3)	c_2	2	3	**4**	X	X	X
x(3)	x(0)	c_3	1	2	3	**4**	X	X
x(4)	x(1)	c_4	0	**1**	X	X	X	X

The 'bold faced' entries show the counter values in use to select that particular column of the kernel $\phi(k,\tau) = e^{j\frac{2\pi k\tau}{N}}$, and X are 'don't care' entries.

After pre-multiplication, the modified data would be $x(4)e^{j\frac{2\pi k}{5}}, x(2)e^{j\frac{2\pi 4k}{5}}, x(3)e^{j\frac{2\pi 4k}{5}}, x(0)e^{j\frac{2\pi 3k}{5}}$ *and* $x(1)e^{j\frac{2\pi 3k}{5}}$. By substituting the above modified data in Eqn. (6.4), it easily follows that DFT of the sequence in natural order can be obtained. Comparisons of our approach to different methods for serial data are tabulated in Table 6.2

Table 6.2: Comparison of various approaches for computing DFT of serial data

	Butterfly structure (Radix-2)	Time-recursive approach	Proposed method
complex multipliers	$\frac{N}{2}log_2(N)$	$(N-1)$	$2(N-1)$
complex multiplications	$\frac{N}{2}log_2(N)$	$(N-1)$	$(N-1)(D+1)$
Throughput	$B+log_2(N)$	$(N+B)$	$(N+1)$

B: Number of *OPs* that can be performed during the buffering time

D: Number of bit positions that differ from natural order

The proposed architecture requires (N-1) complex multipliers besides (N-1) needed for computing DFT. The throughput of the structure is *(N+1),* since the modules can be operated in a two stage pipelined fashion, as compared to *(N+B)* for time-recursive approach and $B+log_2(N)$ for butterfly structure, where B is the number of *OPs* that can be performed in the buffering time. It can be observed that the order of computations is directly dependent on the number of bit-positions that differ by. For D differences in bit-

positions, the total complex multiplications required are $N(N-1)+D(N-1)$, which indicates that, for data in natural order the computational complexity is the same for the proposed architecture and time-recursive approach without pre-processor. The algorithm can be easily extended to compute DCT/ DST/ DHT on similar lines with the same pre-processing. Eventhough we have proposed this algorithm for the DFT, we have not considered GTFD implementation of nonsequential data.

6.4. SHORT-TIME FOURIER TRANSFORM

The discrete time STFT can be related to DFT given in Eqn. (6.2). The STFT of *x(n)* is a set of such DFTs corresponding to different time sections of *x(n)*. The time section for time *i* is obtained by multiplying *x(n)* with a time shifted sequence *w(i-n)*,where *w(n)* represents a window function, and for simplicity we assume the window to be a rectangular one.

$$X(i,\omega)=\sum_{n=i}^{i+N-1}x(n)e^{-j\omega(n-i)} \quad . \tag{6.6}$$

The relation between $X(i+1,\omega)$ and $X(i,\omega)$ can be shown to be (Liu *et al*, 1994):

$$X(i+1,\omega)=e^{j\omega}[X(i,\omega)-x(i)+x(i+N)e^{-j\omega N}] \, . \tag{6.7}$$

Discrete STFT can be obtained from the discrete time STFT through the following relation:

$$X(i+1,k)=X(i+1,\omega)\Big|_{\omega=\frac{2\pi k}{N}\ k=0,1,..,N-1}$$

$$= e^{j\frac{2\pi k}{N}} [\, X(i,k) + x(i+N) - x(i)]_{k=0,1,\ldots N-1}. \qquad (6.8)$$

The architecture for implementing the above equation is the same as the one for the DFT except that instead of x(n) used for DFT computation, $x(i+N) - x(i)$ has to be used. Using only real multiplications, the above equation can be rewritten as

$$X_r(i+1,k) = [X_r(i,k) + x_r(i+N) - x_r(i)]\cos(\omega_k) - [X_j(i,k) + x_j(i+N) - x_j(i)]\sin(\omega_k),$$

where $\omega_k = 2\pi k / N$ and

$$X_r(i+1,k) = [X_r(i,k) + x_r(i+N) - x_r(i)]\sin(\omega_k) + [X_j(i,k) + x_j(i+N) - x_j(i)]\cos(\omega_k), \qquad (6.9)$$

which can be implemented using the same architecture used for computing DFT as shown in Fig. 6.2. The total number of complex multipliers required are *(N-1)* and the number of complex adders needed are *(N+1)*. The throughput rate is the one *OP* to obtain $X(n_0 + 1, \omega)$ from $X(n_0, \omega)$ and the throughput rate of obtaining nonoverlapped STFT is *N OPs* as given in (Liu, 1993).

6.5. WIGNER-VILLE DISTRIBUTION

We consider the WVD, which is a special case of GTFDs, in this Section. The WVD can be represented in GTFD form by setting the kernel to one (i.e., $\phi(\theta, \tau) = 1$), which is given by

$$C(t, \omega) = \int x(t + \frac{\tau}{2})\, x^*(t - \frac{\tau}{2})\, e^{-j\omega\tau} d\tau \,. \qquad (6.10)$$

As discussed in Section 3.5, after sampling using the half-outer product sampling scheme, the discrete WVD can be given by

$$C(n,k) = 2 \sum_{\tau=-\frac{(N-1)}{2}}^{\frac{(N-1)}{2}} x(n+\tau)\, x^*(n-\tau)\, e^{-j\frac{2\pi k\tau}{N}} . \qquad (6.11)$$

Throughout the discussion, we make the following assumptions:

- The signal length (window length) is always odd.
- L, which represents the total number of samples of $\phi_\tau(\theta)$ is also odd.

A direct interpretation of Eqn. (6.11) as Fourier transform results in excessive computations. A closer look at the data flow reveals that there zero-valued samples exist in an orderly fashion (Barry, 1992). To compute Eqn. (6.11), simple DFT module can be used with proper data shuffling, which is illustrated Fig. 6.4.

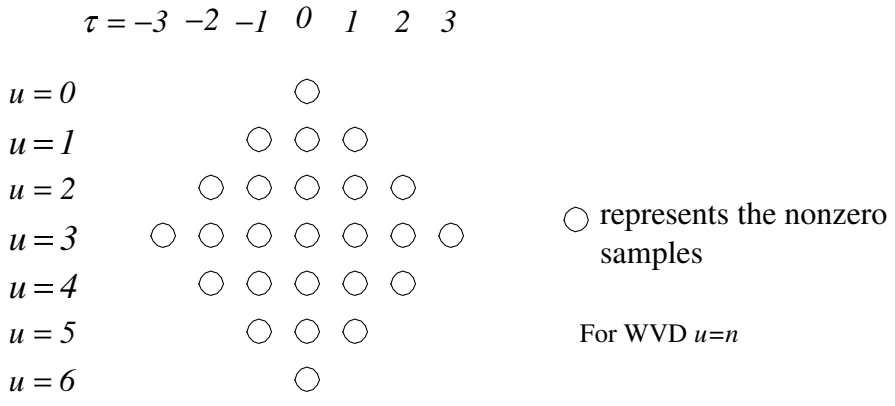

Fig. 6.4. Data flow for *N=7* in WVD and GTFDs