SRC TR 85-8

VLSI ARCHITECTURES BASED ON THE SMALL N ALGORITHMS

by

Joseph Ja' Ja'

and

Robert Michael Owens

Dept. of Electrical Engineering
University of Maryland
College Park, MD 20742

Abstract

Digital convolution and the discrete Fourier transform are basic operations whose computational requirements are of great importance in many applications. In this paper, we propose a new type of VLSI architectures which are shown to be quite suitable to handle these operations. These architectures will result in fully pipelined bit-serial arrays which require no control units. Some preliminary implementations indicate a substantial speed-up gain over other existing designs.

1. *Introduction*

Digital convolution and the discrete Fourier transform are basic operations whose computational requirements are of great importance to many applications such as signal processing and image recognition. Many researchers have studied these operations and have come up with different schemes to implement them. The only hope for a substantial speed up lies in the efficient use of parallelism and pipelining. Today with the advent of VLSI, hundreds of thousands of transistors can be laid out on a single chip. This opens up the door to designing fast special-purpose hardware to perform these operations. Several special architectures have been proposed and implemented for various digital operations. Recently the systolic approach ([KL],[KS]) has been advocated by many researchers mainly because of the amount of parallelism used and the regularity and the simplicity of the overall structure. We see a couple of disadvantages with this approach: I/O interfacing and the amount of hardware required.

In this paper, we consider classes of algorithms based on the so called *small n algorithms* for convolution and discrete Fourier transform and show how to implement them on a regular structure using the maximum amount of parallelism in a certain sense. The basic cells are bit-serial adders and multipliers that can be laid out quite compactly. Two types of building blocks will be sufficient to implement any class of algorithms to be discussed in the next two sections. The overall structure of these algorithms lends itself well to parallel and pipeline implementation. Moreover, the overall arithmetic requirements can be shown to be significantly smaller than those of the alternative algorithms.

2. *Fast Convolution Algorithms*

The *cyclic convolution* $\{y_j\}$ of two length n sequences $\{x_i\}$ and $\{h_i\}$, $CYC(n)$,

is defined by:

$$y_j = \sum_{i=0}^{n-1} h_i \; x_{j-i}, \quad j = 0, 1, \cdots, n-1,$$

where $j-i$ is computed modulo n. We start by discussing a class of algorithms which view the above problem as multiplying two polynomials modulo a third polynomial as follows. Let $Y(z)$, $X(z)$ and $H(z)$ be three polynomials defined by:

$$Y(z) = \sum_{i=0}^{n-1} y_i \; z^i,$$

$$X(z) = \sum_{i=0}^{n-1} x_i \; z^i,$$

$$H(z) = \sum_{i=0}^{n-1} h_i \; z^i.$$

Then it is easy to verify that

$$Y(z) = (X(z)\, H(z)) \; mod \, (z^n - 1).$$

The computational complexity of this problem has been studied by Winograd ([W2]) and algorithms using the fewest number of nonscalar multiplications have been developed. However these algorithms become impractical for large values of n and only the optimal algorithms with small n are used. These algorithms (for short convolution) are based on the Chinese Remainder Theorem for polynomials ([AC], [W1]) and, whenever the data is real, do not use sines or cosines or complex arithmetic. Moreover, these algorithms use fewer arithmetic operations than the radix 2 fast Fourier transform, *FFT* based algorithms. For example, the convolution of $(x_0\, x_1\, x_2)^T$ and $(h_0\, h_1\, h_2)^T$ can be computed as follows:

$$a_0 = x_0 + x_1 + x_2, \quad a_1 = x_0 - x_2, \quad a_2 = x_1 - x_2, \quad a_3 = a_1 + a_2,$$

$$b_0 = (h_0 + h_1 + h_2)/3,\ \ b_1 = h_0 - h_2,\ \ b_2 = h_1 - h_2,\ \ b_3 = (h_1 + h_2)/3,$$

$$m_0 = a_0\, b_0,\ \ m_1 = a_1\, b_1,\ \ m_2 = a_2\, b_2,\ \ m_3 = a_3\, b_3,$$

$$u_0 = m_1 - m_3,\ \ u_1 = m_2 - m_3,$$

$$y_0 = m_0 + u_0,\ \ y_1 = m_0 - u_0 - u_1,\ \ y_2 = m_0 + u_1.$$

Before we proceed, we discuss one possible representation of each of these small n algorithms. Each of these algorithms consists of three steps: a set of input additions/subtractions, a set of elementwise multiplications, and a set of output additions/subtractions. These operations can be represented by three matrices **A**, **B**, and **C** such that:

Input Additions: $\hat{\mathbf{h}} = \mathbf{A}\,\mathbf{h} \qquad \hat{\mathbf{x}} = \mathbf{B}\,\mathbf{x}$

Elementwise Multiplications: $\hat{\mathbf{y}} = \hat{\mathbf{h}} \circledast \hat{\mathbf{x}}$

Output Additions: $\mathbf{y} = \mathbf{C}\,\hat{\mathbf{y}}$

where $\circledast$ is elementwise multiplication. Notice that if the algorithm uses δ multiplications, then **A** and **B** are of size $\delta \times n$ and **C** is of size $n \times \delta$. Optimal algorithms have been developed for $2 \leq n \leq 9$ ([AC]). Note that in many practical cases, **h** can be assumed fixed and all the input additions/subtractions involving the h_i's can be precomputed. For larger values of n, other schemes have been found. One such scheme is the Agarwal-Cooley algorithm ([AC]) which can be described as follows.

Let n be such that $n = n_1\, n_2 \cdots n_l$, where the n_i's are relatively prime. Then $CYC(n)$ can be computed by using algorithms for $CYC(n_i)$, $1 \leq i \leq l$. Each of these algorithms is represented by $\mathbf{A}^{\mathbf{i}}$, $\mathbf{B}^{\mathbf{i}}$, and $\mathbf{C}^{\mathbf{i}}$, where $\mathbf{A}^{\mathbf{i}}$ and $\mathbf{B}^{\mathbf{i}}$ are of size $\delta_i \times n_i$ and $\mathbf{C}^{\mathbf{i}}$ is of size $n_i \times \delta_i$. Then one can check that:

$$\mathbf{y} = (\mathbf{C}^1 \otimes \mathbf{C}^2 \otimes \cdots \otimes \mathbf{C}^l)((\mathbf{A}^l \otimes \cdots \otimes \mathbf{A}^2 \otimes \mathbf{A}^1)\,\mathbf{h} \circledast (\mathbf{B}^l \otimes \cdots \otimes \mathbf{B}^2 \otimes \mathbf{B}^1)\,\mathbf{x}) \quad (*)$$

where $\otimes$ is the kronecker product. We now derive a slightly different version which is more suitable for implementation. Assume that a j, $1 < j \leq l$ exists such that $n_1 n_2 \cdots n_{j-1} \approx n_j n_{j+1} \cdots n_l$. Then equation $(*)$ can be rewritten as follows:

$$\mathbf{y} = (\mathbf{C}^1 \otimes \cdots \otimes \mathbf{C}^{j-1}) \otimes (\mathbf{C}^j \otimes \cdots \otimes \mathbf{C}^l) ((\mathbf{A}^l \otimes \cdots \otimes \mathbf{A}^j) \otimes (\mathbf{A}^{j-1} \otimes \cdots \otimes \mathbf{A}^1)\,\mathbf{h} \circledast (\mathbf{B}^l \otimes \cdots \otimes \mathbf{B}^j) \otimes (\mathbf{B}^{j-1} \otimes \cdots \otimes \mathbf{B}^1)\,\mathbf{x}) \quad .$$

Let $\mathbf{X}$, $\mathbf{H}$, and $\mathbf{Y}$ be two l-dimensional arrays of size $n_l \cdots n_j \times n_{j-1} \cdots n_1$ holding the elements of $\mathbf{x}$, $\mathbf{h}$, and $\mathbf{y}$, respectively, in row-major order form. Then one can show that:

$$\mathbf{Y} = \overline{\mathbf{C}}^2(\overline{\mathbf{C}}^1[\overline{\mathbf{A}}^1(\overline{\mathbf{A}}^2\mathbf{H})^T \circledast \overline{\mathbf{B}}^1(\overline{\mathbf{B}}^2\mathbf{X})^T])^T \quad (2.1)$$

where

$$\overline{\mathbf{C}}^1 = \mathbf{C}^1 \otimes \cdots \otimes \mathbf{C}^{j-1} \qquad \overline{\mathbf{C}}^2 = \mathbf{C}^j \otimes \cdots \otimes \mathbf{C}^l$$
$$\overline{\mathbf{A}}^1 = \mathbf{A}^{j-1} \otimes \cdots \otimes \mathbf{A}^1 \qquad \overline{\mathbf{A}}^2 = \mathbf{A}^l \otimes \cdots \otimes \mathbf{A}^j$$
$$\overline{\mathbf{B}}^1 = \mathbf{B}^{j-1} \otimes \cdots \otimes \mathbf{B}^1 \qquad \overline{\mathbf{B}}^2 = \mathbf{B}^l \otimes \cdots \otimes \mathbf{B}^j$$

Notice that the basic operations involved in computing $\mathbf{Y}$ are of three types: 1) input/output additions of the form $\mathbf{A}\,\mathbf{X}$, such that the elements of $\mathbf{A}$ are either -1, 0, and 1; 2) elementwise multiplication of the form $\mathbf{X} \circledast \mathbf{Y}$; and 3) the transpose operation of the form $\mathbf{X}^T$.

3. *Fast Algorithms for Computing the Discrete Fourier Transform*

Recall that the discrete Fourier transform $\{y_j\}$ of a length n sequence $\{x_i\}$,

$DFT(n)$, is given by:

$$y_k = \sum_{j=0}^{n-1} \omega^{kj} x_j, \quad 0 \leq k \leq n-1,$$

where $\omega = e^{-2\pi i/n}$. We start by discussing a class of algorithms that are different than the FFT. For large values of n, these algorithms use substantially fewer multiplications and about the same number of additions/subtractions as those of the radix 2 FFT. The basic idea behind these algorithms is the reduction of $DFT(n)$ to a cyclic convolution which, as described in the previous section, can be reduced to computing the product of two polynomials modulo a third polynomial.

Based on the reduction to cyclic convolution method, several authors have developed algorithms to compute $DFT(n)$ for several values of n and in particular for $n = 2, 3, 4, 5, 7, 8, 9, 16$. The general structure of such algorithms, like those for cyclic convolution, consists of a sequence of input additions/subtractions followed by multiplication steps followed by output additions/subtractions. These operations can be represented by three matrices **S**, **C**, and **T** such that the input/output relationship for $DFT(n)$ can be expressed by:

$$\mathbf{y} = \mathbf{S}\,\mathbf{C}\,\mathbf{T}\,\mathbf{x}$$

where **C** is a $\delta \times \delta$ diagonal matrix representing the δ multiplications, **T** is a $\delta \times n$ matrix of 0, 1 –1 representing the input additions/subtractions, and **S** is an $n \times \delta$ matrix of 0, 1, –1 representing the output additions/subtractions. Note that $n \leq 16$ and $\delta \leq 18$ for small n.

For larger values of n, $DFT(n)$ can be computed by combining an appropriate set of these small n algorithms. Let $n = n_1 n_2 \cdots n_l$, where the n_i's are relatively prime. We will briefly outline how to compute $DFT(n)$ by using the algorithms for $DFT(n_i)$, $1 \leq i \leq l$. Each of these algorithms is represented by $\mathbf{S^i}\,\mathbf{C^i}\,\mathbf{T^i} = \mathbf{D^i}$, where

$\mathbf{C}^i$ is $\delta_i \times \delta_i$, $1 \leq i \leq l$. It is possible to reorder the entries of $\mathbf{x}$ and $\mathbf{y}$ so that

$$\bar{\mathbf{y}} = \mathbf{D}^l \otimes \cdots \otimes \mathbf{D}^2 \otimes \mathbf{D}^1 \bar{\mathbf{x}},$$

where $\otimes$ is the Kronecker product operation, and $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are the reordered input and output vectors. We thus obtain

$$\bar{\mathbf{y}} = (\mathbf{S}^l \mathbf{C}^l \mathbf{T}^l) \otimes \cdots \otimes (\mathbf{S}^2 \mathbf{C}^2 \mathbf{T}^2) \otimes (\mathbf{S}^1 \mathbf{C}^1 \mathbf{T}^1) \bar{\mathbf{x}}$$

Good's algorithm (also called the prime factor algorithm) and Winograd's algorithm (WFTA) differ on how to compute the above equation. Let $\mathbf{X}$ and $\mathbf{Y}$ be two l-dimensional arrays of sizes $n_l \times \cdots \times n_2 \times \times n_1$ such that $X_{j_l, \cdots, j_2, j_1} = \bar{x}_j$ and $Y_{j_l, \cdots, j_2, j_1} = \bar{y}_j$, where $(j_l, \cdots, j_2, j_1)$ is the Chinese Remainder representation of j.

Using the above notation and some elementary properties of the Kronecker product, we can express $\mathbf{Y}$ in terms of $\mathbf{X}$ as follows:

$$\mathbf{Y} = (\mathbf{S}^1 \mathbf{C}^1 \mathbf{T}^1 (\mathbf{S}^2 \mathbf{C}^2 \mathbf{T}^2 (\cdots (\mathbf{S}^l \mathbf{C}^l \mathbf{T}^l \mathbf{X})^T)^T \cdots)^T .$$

The above equation is essentially Good's algorithm based on the small n DFT's. Winograd's algorithm is slightly harder to describe. Using elementary properties of the Kronecker product, we have

$$\bar{\mathbf{y}} = (\mathbf{S}^l \otimes \cdots \otimes \mathbf{S}^2 \otimes \mathbf{S}^1)(\mathbf{C}^l \otimes \cdots \otimes \mathbf{C}^2 \otimes \mathbf{C}^1)(\mathbf{T}^l \otimes \cdots \otimes \mathbf{T}^2 \otimes \mathbf{T}^1) \bar{\mathbf{x}}, \quad (*)$$

which is essentially Winograd's algorithm (WFTA). For our purposes we need to rewrite it in a different form using the multidimensional arrays $\mathbf{Y}$ and $\mathbf{X}$. It is not hard to see that the WFTA algorithm can be expressed as follows:

$$\mathbf{Y} = \mathbf{S}^l (\cdots (\mathbf{S}^2 (\mathbf{S}^1 \mathbf{C} \circledast \mathbf{T}^1 (\mathbf{T}^2 (\cdots (\mathbf{T}^l \mathbf{X})^T \cdots)^T \cdots)^T)^{T^{inv}})^{T^{inv}} \cdots)^{T^{inv}} ,$$

where $\mathbf{C}$ is an l-dimensional array of size $\delta_1 \times \delta_2 \times \cdots \times \delta_l$ such that

$$C_{i_1, i_2, \cdots, i_l} = C^1(i_1,i_1)C^2(i_2,i_2) \cdots C^l(i_l,i_l).$$

and T and T^{inv} denote the generalized transpose and the inverse transpose operations respectively ([ER]).

The above algorithm has a nested structure and starts by computing all the input additions, then computing all multiplications and finally applying the output additions. Compare this with Good's algorithm which alternates between DFT's computations, each of which involving input additions, a set of multiplications, and output additions.

If we examine the small n algorithms, we find that $\delta_i \approx n_i$ (as a matter of fact $\delta_i \leq n_i + 2$ for all i). Since $n = n_l\ n_{l-1} \cdots n_1$, the number of multiplications used by Winograd's algorithm is approximately n. Compare this with the FFT algorithm which will use about $2n \log(n/2)$ real multiplications.

We now derive a slightly different version of Good's algorithm which is more suitable for implementation. Assume that a j, $1 \leq j \leq l$, exists such that $n_1 n_2 \cdots n_{j-1} \approx n_j\ n_{j+1} \cdots n_l$. Then equation $(*)$ can be rewritten as follows:

$$\bar{\mathbf{y}} = (\mathbf{S}^l \otimes \cdots \otimes \mathbf{S}^j) \otimes (\mathbf{S}^{j-1} \otimes \cdots \otimes \mathbf{S}^1)(\mathbf{C}^l \otimes \cdots \otimes \mathbf{S}^j) \otimes (\mathbf{S}^{j-1} \otimes \cdots \otimes \mathbf{C}^1)$$
$$(\mathbf{T}^l \otimes \cdots \otimes \mathbf{T}^j) \otimes (\mathbf{T}^{j-1} \otimes \cdots \otimes \mathbf{T}^1)\bar{\mathbf{x}} \ .$$

Let $\mathbf{X}$ and $\mathbf{Y}$ be two arrays of sizes $n_1 n_2 \cdots n_{j-1} \times n_j\ n_{j+1} \cdots n_l$ holding the elements of $\mathbf{x}$ and $\mathbf{y}$, respectively, in row major order form. Then one can show that:

$$\mathbf{Y} = (\bar{\mathbf{S}}^1\bar{\mathbf{C}}^1\bar{\mathbf{T}}^1(\bar{\mathbf{S}}^2\bar{\mathbf{C}}^2\bar{\mathbf{T}}^2\mathbf{X})^T)^T \quad (3.1).$$

where

$$\bar{\mathbf{S}}^1 = \mathbf{S}^{j-1} \otimes \cdots \otimes \mathbf{S}^1 \qquad \bar{\mathbf{S}}^2 = \mathbf{S}^1 \otimes \cdots \otimes \mathbf{S}^j$$
$$\bar{\mathbf{C}}^1 = \mathbf{C}^{j-1} \otimes \cdots \otimes \mathbf{C}^1 \qquad \bar{\mathbf{C}}^2 = \mathbf{C}^1 \otimes \cdots \otimes \mathbf{C}^j$$
$$\bar{\mathbf{T}}^1 = \mathbf{T}^{j-1} \otimes \cdots \otimes \mathbf{T}^1 \qquad \bar{\mathbf{T}}^2 = \mathbf{T}^1 \otimes \cdots \otimes \mathbf{T}^j$$

Likewise using a variation of Winograd's scheme, we can derive:

$$\mathbf{Y} = (\bar{\mathbf{S}}^2(\bar{\mathbf{S}}^1\, \mathbf{C} \circledast \bar{\mathbf{T}}^1(\bar{\mathbf{T}}^2\mathbf{X})^T)^T \quad (3.2).$$

So far we have seen two different algorithms which are based on the small n fast algorithms: the first has the structure of the prime factor algorithm, and the second has a nested structure introduced by Winograd. We can also use the general structure of the FFT algorithm to obtain another type of algorithms based on the small n fast algorithms. We now outline such an approach. Let $n = n_1 n_2$, where $DFT(n_1)$ and $DFT(n_2)$ are given by two Winograd-type algorithms $(\mathbf{S}_1, \mathbf{C}_1, \mathbf{T}_1)$ and $(\mathbf{S}_2, \mathbf{C}_2, \mathbf{T}_2)$. For any $0 \leq i \leq n-1$ and $0 \leq j \leq n-1$ we can write i and j as $i = i_1 + i_2 n_1$ and $j = j_1 n_2 + j_2$. Then $y = DFT(x)$ is given by

$$y_{i_1+i_2n_1} = \sum_{j_2=0}^{n_2-1} (\omega^{n_1})^{i_2 j_2} (\omega^{i_1 j_2} \sum_{j_1=0}^{n_1-1} (\omega^{n_2})^{i_1 j_1} x_{j_1 n_2 + j_2}).$$

Let

$$z_{i_1 j_2} = \sum_{j_1=0}^{n_1-1} (\omega^{n_2})^{i_1 j_1} x_{j_1 n_2 + j_2} .$$

For each fixed j_2, $z_{i_1 j_2}$ is the discrete Fourier transform of n_1 points. Hence we can apply $DFT(n_1)$ n_2 times to obtain all the $z_{i_1 j_2}$'s. We then have to scale these quantities by the factors $\omega^{i_1 j_2}$. Finally we can apply $DFT(n_2)$ n_1 times to get the desired quantities. We now translate this algorithm into matrix form. Let $\mathbf{X}$ be an $n_1 \times n_2$ array representing the input in row-major order form. Then the output $\mathbf{Y}$ is given by

$$\mathbf{Y} = \mathbf{S}^2\,\mathbf{C}^2\,\mathbf{T}^2(\mathbf{W}\circledast\mathbf{S}^1\,\mathbf{C}^1\,\mathbf{T}^1\mathbf{X})^T \quad (3.3),$$

where $\mathbf{W}$ is a *constant* $n_1 \times n_2$ array given by

$$W_{ij} = \omega^{ij} \quad ,0 \leq i \leq n_1-1,\ 0 \leq j \leq n_2-1.$$

Note that in this case we are not assuming that n_1 and n_2 are relatively prime and that there is a modest increase in the number of multiplications, namely n. However as we will see in the next section this will not affect the overall performance in any significant way. Notice that the basic operations involved in computing $\mathbf{Y}$ in each of the three algorithms given in this section are identical to those of cyclic convolution and are of three types: 1) input/output additions of the form $\mathbf{A}\,\mathbf{X}$, such that the elements of $\mathbf{A}$ are either –1, 0, and 1; 2) elementwise multiplication of the form $\mathbf{X}\circledast\mathbf{Y}$; and 3) the transpose operation of the form $\mathbf{X}^T$.

It is not hard to see that a similar class of algorithms can be designed for digital filtering.

4. *Component Structure*

Recall from the previous two sections that only three operations are involved in computing either the cyclic convolution or the discrete Fourier Transform. We propose three hardware components to compute these operations.

1) the *summation component* will be used to compute $\mathbf{AX}$, where the elements of $\mathbf{A}$ are either –1, 0, or 1. This component will therefore be used to compute the input/output additions/subtractions. It will be constructed from an array of serial adders.
2) the *scaling component* will be used to compute $\mathbf{X}\circledast\mathbf{Y}$. This component will be constructed from a number of linear array of digit online multipliers which will

perform all the corresponding multiplications.

3) the *transpose component* will be used to compute $\mathbf{X}^T$. A two dimensional array of dynamic shift registers will be used to implement this component.

Once these components are available, the cyclic convolution or the discrete Fourier transform can be be implemented for some n quite easily by interconnecting these components as shown in the next section.

To aid in the construction of the overall VLSI hardware, we propose that each of the components has whenever possible the following properties.

1) The execution rate of each component should be constant and therefore independent of the mathematical and physical characteristics of its size.
2) Each component should be constructed by interconnecting in a regular way smaller components of a few different types. The size of the subcomponents should be small and independent of the size of the problem. Furthermore, given the formal description, the design of a component should be straightforward.
3) The size of each component should be realistic and should satisfy VLSI circuit density and size constraints.
4) The logical and the physical input/output characteristics of the various components should be compatible so as to allow one component to be connected to another in a straightforward manner. Furthermore, the number of interconnections between components should be realistic.

We have shown in [OJ] that it is possible to achieve all the above goals for these basic hardware components. Below is a brief description of the summation component from [OJ].

Recall that the mathematical operation performed by the summation component

is given by $\mathbf{Z} = \mathbf{S}\,\mathbf{X}$, where

$$\mathbf{S} = \left[S_{i,j}, 0 \leq i < n_0, 0 \leq j < n_1 \right] ,$$

$$\mathbf{X} = \left[X_{i,j}, 0 \leq i < n_1, 0 \leq j < n_2 \right] ,$$

and

$$\mathbf{Z} = \left[Z_{i,j}, 0 \leq i < n_0, 0 \leq j < n_2 \right] .$$

The elements of $\mathbf{S}$ are predefined and are either -1, 0, or 1. We will make use of this fact by building them into the summation component itself rather than supplying them to the component as the operation is performed. Being able to tailor the component in this manner will decrease its area and the number of inputs. To reduce the area and number of inputs and outputs further, bit serial arithmetic is utilized in the design of the component. By using bit serial arithmetic, we will be able to reduce the perimeter required from $O(p\ (n_1 + n_0))$ to $O(n_1 + n_0)$ and the area required from $O(p\ n_1\ n_0)$ to $O(n_1\ n_0)$. We feel that serial arithmetic is necessary given the size of the problems we would like to solve (as least 1024 point DFT) and the maximum complexity of VLSI chips in the forseeable future.

For reasons given in [Ow], we propose left directed (least to most significant digit) bit serial arithmetic if the input is integer data and right directed (most to least significant bit) if the input is fractional data. The following is the left directed design since it is the most complex of the two. The summation component can be constructed by interconnecting smaller subcomponents of only three different types (addition, subtraction, and delay), which share the same rectangular shape. Each of these subcomponents can be represented as shown in figure 4.1.

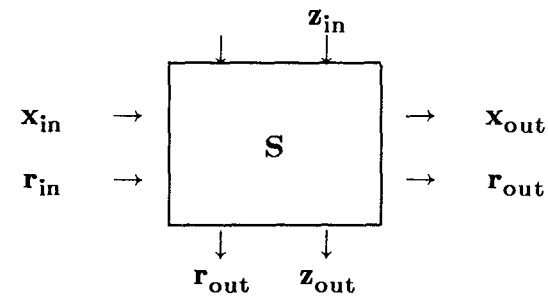

Figure 4.1. Summation Subcomponent.

The input $\mathbf{r_{in}}$ is used to indicate that the first digit of a element is being supplied to the subcomponent. The functional description of the digit online addition subcomponent is given by figure 4.2.

Digit Online Addition Subcomponent

$x_{out} = x_{in}$

$r_{out} = r_{in}$

if $r_{in} \equiv 0$ then

$s = x_{in} + z_{in} - b\ c$ where c is chosen such that

$$-b+2 \leq s \leq b-2$$

$z_{out} = t + c$

else

$s = x_{in} + z_{in}$

$z_{out} = t$

ifend

$t = s$

Figure 4.2. Subcomponent Functional Definition.

where x_{in}, r_{in}, and z_{in} are the values supplied to inputs $\mathbf{x_{in}}$, $\mathbf{r_{in}}$, and $\mathbf{z_{in}}$ respectively in the time interval prior to some clocking, x_{out}, r_{out}, and z_{out} are the values generated at

outputs x_{out}, r_{out}, and z_{out} respectively in the interval after that clocking, and b is the base of the number system being used. The functional description of the subtraction and delay subcomponents follows from the description of the addition subcomponent.

Digit online subcomponents of the three different types are interconnected as illustrated by figure 4.3 to form a digit online summation component.

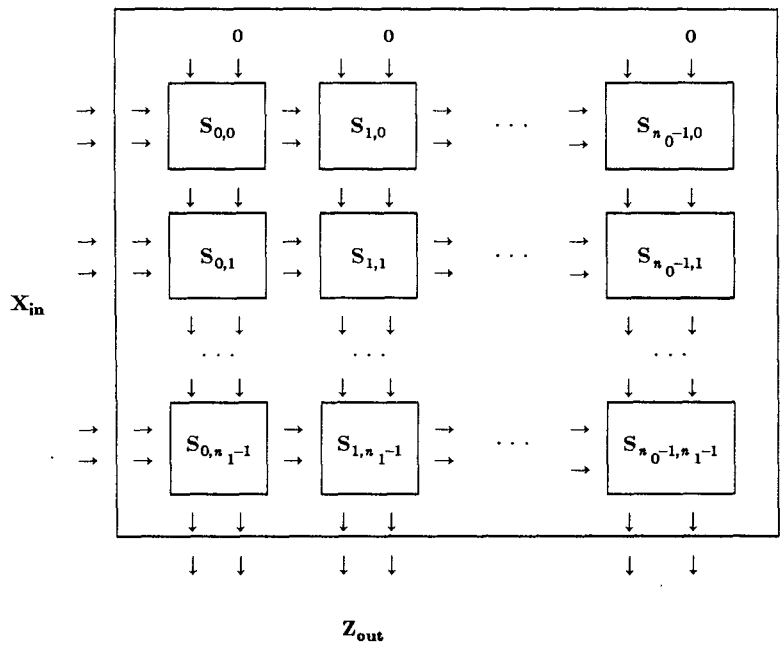

Figure 4.3. Summation Component.

Preliminary implementions of our design show that each of the subcomponents can be laid out on an area of 170 microns by 170 microns and that the subcomponents have a delay of 40 nanoseconds. Hence, given the current restrictions imposed by MOSIS, we will be able to faburicate an array of 24 $\times$ 24 subcomponents on a single chip. This subcomponent has been submitted to MOSIS for fabrication.

5. *Component Interconnection*

Consider the cyclic convolution of two sequences where one of the input se-

quences is predefined. Then, $CYC(n)$ can be computed by an arithmetic unit whose overall hardware structure, as suggested by equation 2.1, is indicated in figure 5.1.

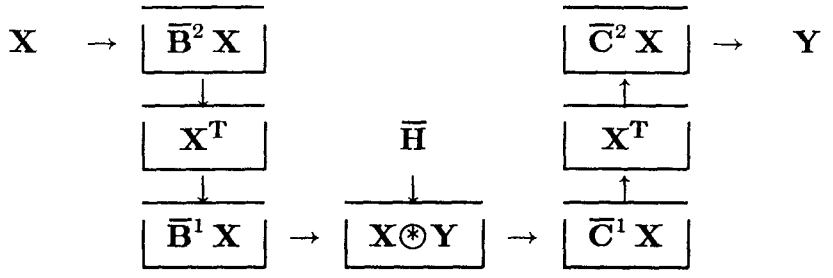

Figure 5.1 Structure of Cyclic Convolution Interconnection.

where $\overline{H} = \overline{A}^1(\overline{A}^2 H)^T$. The total chip area is approximately $O(n)$, while the time needed to compute the cyclic convolution is approximately $O(\sqrt{n})$. Hence, our design achieves the $A\ T^2$ lower bound of $\Omega(n^2)$. Also the hardware can be efficiently utilized by pipelining one problem after another through the subcomponents. Hence our design achieves pipelining at the digit, word and problem levels.

Now consider the discrete Fourier transform of a sequence. The transform $DFT(n)$ can be computed by an arithmetic unit whose overall hardware structure, as suggested by equation 3.2, is indicated in figure 5.2.

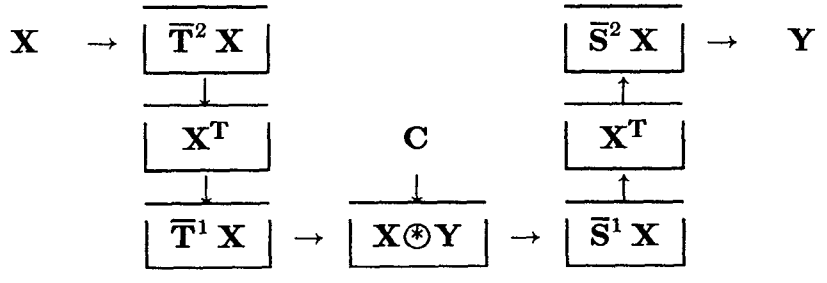

Figure 5.2 Structure of First DFT Interconnection.

The total chip area is approximately $O(n)$ and the time needed to compute $DFT(n)$ is approximately $O(\sqrt{n})$. Hence, our design achieves the $A\ T^2$ lower bound of $\Omega(n^2)$. Note that the structures for the cyclic convolution and the DFT are identical.

The DFT can also be computed by an arithmetic unit whose overall hardware structure, as suggested by equation 3.1, is indicated in figure 5.3.

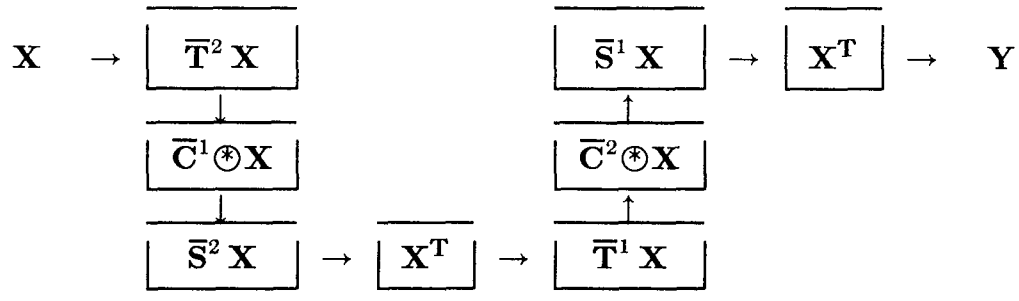

Figure 5.3 Structure of Second DFT Interconnection.

Again the total chip area is approximately $O(n)$ and the time needed to compute $DFT(n)$ is $O(\sqrt{n})$. Hence, our design achieves the $A\ T^2$ lower bound of $\Omega(n^2)$. Note that in this case, the arrays $\overline{C}^1$ and $\overline{C}^2$ need not be supplied to the scaling component. This fact can be used to reduce the size of the scaling components in this case.

The disadvantage with the second interconnection for the DFT when compared to the first interconnection is that the second has two scaling components whereas the first has only one. The advantage with the second is that each of its two scaling components is much smaller than that of the first. Hence, even though the second interconnection has more components, it may not have a larger chip area than the first. Furthermore, even if more hardware is needed by the second interconnection scheme, the throughput (number of problems which can be solved per unit of time) is not less than the throughput of the first interconnection scheme.

The transform $DFT(n)$ can also be computed by an arithmetic unit those overall hardware structure, as suggested by equation 3.3, is indicated in figure 5.4.

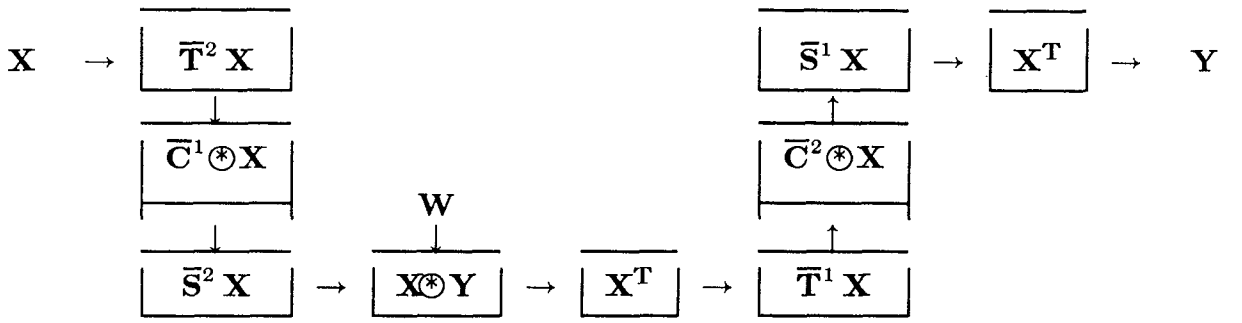

Figure 5.4 Structure of Third *DFT* Interconnection.

The total chip area is approximately $O(n)$ and the time needed to compute $DFT(n)$ is $O(\sqrt{n})$. Hence, our design achieves the $A\ T^2$ lower bound of $\Omega(n^2)$. Again note that in this case, the arrays $\overline{C}^1$ and $\overline{C}^2$ need not be supplied to the scaling component. This fact can be used to reduce the size of the scaling components.

Note, the third interconnection has one more scaling component than the second interconnection. However, as pointed out before, there is more flexibility in choosing the dimensions of the components. Furthermore, while more hardware is needed by the third interconnection scheme, the throughput (number of problems which can be solved per unit of time) does not decrease.

6) *Conclusion*

Preliminary simulation results lead us to believe that a *DFT* processor of a few chips can be constructed which is capable of performing about 64,000 pipelined 1024 point *DFT*'s per second. This rate approaches the rate needed by weighted zero-crossing techniques for pattern recognition.

Future plans will strive to reduce the size and delay of the subcomponents by considering other digit sets and other VLSI technologies and to automate the process used to design a component for a given application.

References

[AC] Agarwal, R. and J. Cooley, "New algorithms for digital convolution," IEEE Trans. Acoustics, Speech and Signal Processing, 25, pp.392-410, 1977.

[Atk] Atkins, D. E., "Introduction to the Role of Redundancy in Computer Arithmetic," *Computer,* Vol. 8, No. 6, pp. 74-76, June 1975.

[Avi] Avizienis, A., "Signed-digit Number Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers,* pp. 389, 1961.

[CT] Cooley, J. and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comp.,* 19, pp.297-301, 1965.

[D] Despain, A.M., "Very Fast Fourier Transform Algorithms for Hardware Implementation," IEEE Trans Computers C-28. pp. 333-341, (May 1979).

[Detal.] Despain, A.M. etal., "VLSI Implementation of Digital Fourier Transforms," Report No UCB/CSD82/111, Computer Science Division, University of California, Berkeley, Ca.

[ER] Elliot, D. and K. Rao, *Fast Transforms Algorithms, Analyses, Applications,* Academic Press, 1982.

[G] Good, I., "The interaction algorithm and practical Fourier analysis," *J. Royal Stat.Soc.,* ser. B, vol.20, pp.361-372, 1958, Addendum, vol.22, pp.372-375, 1960.

[JO] Ja'Ja', J. and R. Owens, "An architecture for a VLSI FFT processor," *INTEGRATION: the VLSI Journal,* 1(4), pp.305-316, 1983.

[KP] Kolba, D. and I. Parks, "A prime factor FFT algorithm using high speed convolution," *IEEE Trans. Acoust., Speech and Signal Processing,* ASSP-25, pp. 281-294, 1977.

[KL] Kung, H.T. and C.E. Leiserson, "Systolic Arrays (for VLSI)," *Sparse Matrix Proceedings* 1978, edited by J.S. Derff and G.W. Stewart, SIAM, 1979, pp. 256-282.

[MC] Mead, C. and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley, Reading, Mass., 1980.

[OJ] Owens, R.M. and J. Ja'Ja', "A VLSI Chip for the Winograd/Prime Factor Algorithm to Compute the Discrete Fourier Transform," submitted for publication to *IEEE Transactions on Acoustics Speech and Signal Processing.*

[Ow] Owens, R.M., "Techniques to Reduce the Inherent Limitations of Fully Digit On-line Arithmetic," *IEEE Transactions on Computers,* 32(4), pp. 406-411, (April, 1983).

[Th] Thompson, C., "Fourier Transform in VLSI," Technical Report, Computer Science Division, University of California, Berkeley.

[W1] Winograd, S., "On computing the discrete Fourier transform," *Proceedings of Nat. Acad. Sci. USA 73,* pp. 1005-1006, 1976.

[W2] Winograd, S., "On computing the discrete Fourier transform," *Math. Comp.* 32, pp. 175-195, 1978.

[W3] Winograd, S., "On the multiplicative complexity of the discrete Fourier transform," *Advances in Math.* 32, pp. 83-117, 1979.