FOURIER TRANSFORMS IN VLSI

by

C. D. Thompson

Memorandum No. UCB/ERL M80/51

15 October 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Fourier Transforms in VLSI*

C. D. Thompson

Division of Computer Science
U. C. Berkeley
Berkeley, CA 94720

ABSTRACT

Eight designs are proposed for the computation of an N-element Fourier transform. The largest of the designs requires $O(N^2 \log N)$ units of silicon area and operates in $O(\log N)$ time. The smallest designs occupy only $O(N \log N)$ area, but take $O(N \log N)$ time to perform their calculations.

The designs exhibit an area-time tradeoff: the smaller ones are slower, for two reasons. First, they may have fewer functional units and thus less parallelism. Second, their functional units may be interconnected in a pattern that is less efficient but more compact.

The optimality of several of the designs is immediate, since they achieve the limiting area*time^2 performance of $\Omega(N^2 \log^2 N)$.

Key words and phrases: parallel algorithms, area-time complexity, VLSI, Fourier transform, FFT, shuffle-exchange network, mesh-connected computers.

Fourier Transforms in VLSI*

C. D. Thompson

Division of Computer Science
U. C. Berkeley
Berkeley, CA 94720

1. Introduction

One of the difficulties of VLSI design is the magnitude of the task. It is not easy to lay out one hundred thousand transistors, let alone ten million of them. Yet there is a sense in which the scale of VLSI is advantageous. The complexity of a VLSI chip is so great that asymptotic approximations can give insight into performance evaluation and design.

This paper shows how asymptotic analysis can aid in the design of Fourier transform circuits in VLSI. Approaching chip design in this way has three advantages. First of all, the analysis is simple: the calculations are easy to perform and thus easy to believe. Second, the analysis points out the bottlenecks in a design, indicating the portions that should be optimized. It is impossible to "miss the forest for the trees" when one is thinking of asymptotic performance.

A third advantage of the analytic approach is that it provides a simple framework for the evaluation and explanation of various designs. In the case of the N-element Fourier transform, it is known that no circuit can have a better area*time^2 performance than $\Omega(N^2\log^2 N)$ [13]†. Similar performance limits have been proved for the problems of sorting, matrix multiplication and integer multiplication [1,3,9,13].

The fact that there is a theoretical limit to area*time^2 performance suggests that designs be evaluated in terms of how closely they approach this limit. Any design that achieves this limit must be optimal in some sense and thus deserves careful study. This paper presents a number of nearly-optimal designs, corresponding to different tradeoffs of area for time. For example, Section 3's "FFT network" takes only $O(\log N)$ time but quadratic area to perform its Fourier transform. Thus it is a faster but larger circuit than, say, the "Mesh implementation" which solves an N-element problem in approximately $\sqrt{N}$ time

* This work was supported in part by the U.S. Army Research Office under Grant DAAG29-78-G-0167.

† The omega notation means "grows at least as fast as": as N increases, the product of area with the square of the solution time for these circuits is bounded from below by some constant times $N^2\log^2 N$. The more familiar "big-O" notation is used for upper bounds. A circuit occupies area $A=O(N)$ if there is some constant c for which $A \le cN$ for all but a finite number of problem sizes N. Finally, all logarithms in this paper are base two.

and linear area.

Section 2 of this paper develops a simple model for VLSI, laying the groundwork for the implementations and the analyses. The model is based on a small number of assumptions that are valid for any currently envisioned transistor-based technology. Thus the results apply equally well to the field-effect transistors of the MOS technologies (CMOS, HMOS, VMOS, . . .), to the bipolar transistors of I^2L, and to any GaAs process.

Section 3 describes eight implementations of Fourier transform-solving circuits in VLSI. Most of these circuits are highly parallel in nature.

Section 4 concludes the paper with a summary of the perormance figures of the designs.

2. The Model

Briefly, a VLSI circuit is modeled as a collection of nodes and wires. A node represents a wire junction, a transistor, or a gate. A wire represents the conductor that carries signals form one node to another.

In keeping with the planar nature of VLSI, nodes are laid out in a non-overlapping fashion. Only a constant number of wires (say 2 or 4) can cross over any point in the plane.

The unit of time in the model is equal to the response time of a simple circuit. In particular, a wire can carry one bit of information in one unit of time. This bit is typically used to change the state of the transistor at the other end of the wire.

The unit of area in the model is determined by the "minimum feature width" of the processing technology. Wires have unit width and nodes occupy $O(1)$ area, that is, a node is some constant number of wire-widths on a side. The area of a node also includes an allowance for power and clock wires, which are not represented explicitly in the model.

The problem of long-distance communication receives special attention. Most nodes can drive only short wires. A specialized "driver node" of $O(k)$ area is required to send a signal down a wire of length k. A driver has $O(\log k)$ stages of amplification, the last stage of which has gate (or junction) area proportional to k. This structure is consistent with the assumption that the load presented by a long wire is capacitive in nature and proportional to its length [7]. The amplifier stages are individually clocked, so that a driver has $O(\log k)$ delay but unit bandwidth.

The notion of "self-timed regions" [10] is incorporated into the model to account for the difficulty of obtaining chip-wide synchronization. The nodes in a self-timed region are in synchrony: all signal transitions occur at the same phase of a common clock. Signals originating outside the region are synchronized with this local clock by means of "receiver nodes."

The model is summarized in the list of assumptions below. A fuller explanation and defense of the model is contained in the author's thesis [13].

Assumption 1: Embedding.

a. Wires are one unit wide.

b. Two wires may cross over each other at right angles.

c. A logic node occupies $O(1)$ area. It has $O(1)$ input wires and $O(1)$ output wires, none of which are more than $O(1)$ units long.

d. Each logic node belongs to a self-timed region. All wires connecting to a logic node lie entirely within its self-timed region.

e. A self-timed region is at most $O(\log N)$ units wide or long.

f. A driver node of $O(k)$ area has an output wire that is k units long. This output wire may pass through any number of self-timed regions before it connects to the input of a receiver node.

g. A receiver node occupies $O(1)$ area. Its output wire is $O(1)$ units long.

Assumption 2: Total area.

The total area of a collection of nodes and wires is the number of unit squares in the smallest enclosing rectangle

Assumption 3: Timing.

a. Wires have unit bandwidth. They carry at most one bit of information in a unit of time.

b. Logic nodes and eceiver nodes have $O(1)$ delay.

c. The driver node for a wire of length k has $O(\log k)$ delay.

Assumption 4: Transmission functions.

a. The signals appearing on the output wires of a node are some fixed function of its current "state."

b. The state of a node is changed every time unit, according to some fixed function of the signals on its input wires.

c. Logic nodes and receiver nodes are limited to $O(1)$ bits of state.

d. Driver nodes have $O(\log k)$ bits of state, one bit for each stage in their amplification chain.

e. The states of the nodes in an "input register" are set to arbitrary values whenever a computation is initiated (see Assumption 6). No attempt is made to model off-chip I/O.

Assumption 5: Problem definition.

a. Each of N input variables takes on one of M different values with equal likelihood.

b. N is an integral power of 2.

c. $\log M = \Theta(\log N)$: a word length of $\lceil \log M \rceil = c(\log_2 N)$ bits is necessary and sufficient to describe the value of an input variable.

d. The output variables $\vec{y}$ are related to the input variables $\vec{x}$ by the equation $\vec{y} = A\vec{x}$. The (i,j)-th entry of A has the value $\omega^{((i-1)*(j-1))}$, where ω is a principal N-th root of unity in the ring of multiplication and addition mod M. (This assumption defines a number-theoretic transform; results for the more common Fourier transform over the field of complex numbers are analogous.)

Assumption 6: Input registers.

a. Each of the N input variables is associated with one input register formed of a chain of $\lceil \log M \rceil$ logic nodes.

b. A computation is initiated at time T_0 if the value of each input variable is encoded in the nodes of its input register. No other node has any information about this value.

Assumption 7: Output registers.

a. Each of the N output variables is associated with one output register formed of a chain of $\lceil \log M \rceil$ logic nodes.

b. A computation is complete at time T_c if the correct value of each output variable is determined by the current state of the nodes in its output register.

Assumption 8: Solution time.

A collection of nodes and wires operates in "pipelined time T" if it can complete a computation every T time units.

3. The Implementations

A basic building block for all of the designs is the multiply-add cell. This cell has three bit-serial inputs ω^k, x_0 and x_1. It produces two bit-serial outputs $y_0 = x_0 + \omega^k x_1$ and $y_1 = x_0 - \omega^k x_1$. The inputs and the outputs are all $\lceil \log M \rceil$ bit integers.

A multiply-add cell can be built from $O(\log N)$ logic gates [13]. The multiplication is performed by $O(\log N)$ steps of addition in a carry-save adder. The subsequent addition and subtraction can also be done in $O(\log N)$ time. Thus a complete multiply-add computation can be done in $O(\log N)$ time and $O(\log N)$ area.

Another basic building block is the shift register. A k-bit shift register is built of $O(k)$ logic nodes in $O(k)$ area. It is used to store constants, successive bits of which are available during each unit of time.

The aspect ratios of the multiply-add cell and shift register may be adjusted at will. They should be designed as a rectangle of $O(1)$ width that can be folded

into any rectangular shape.

3.1. The MM Design

Perhaps the most obvious implementation of the Fourier transform is by direct computation of the matrix-vector product of Assumption 5d. The "systolic array" of Kung and Leiserson provides an efficient means of calculating this product [7, p. 277].

The MM or Matrix Multiplication design consists of $(2N-1)^2$ multiply-add cells connected in a hexagonal mesh. These occupy a total of $O(N^2 \log N)$ area.

As indicated in Figure 1, the input vector $\vec{x}$ is shifted into the upper-left-hand edge of the mesh, the constant matrix A is shifted into the upper-right-hand edge, and the result $\vec{y} = A\vec{x}$ emerges on the top edge. Figure 1 does not show the position and time that individual elements enter and leave the mesh. The interested reader is referred to [7, p. 277] for complete details of the matrix multiplication process.

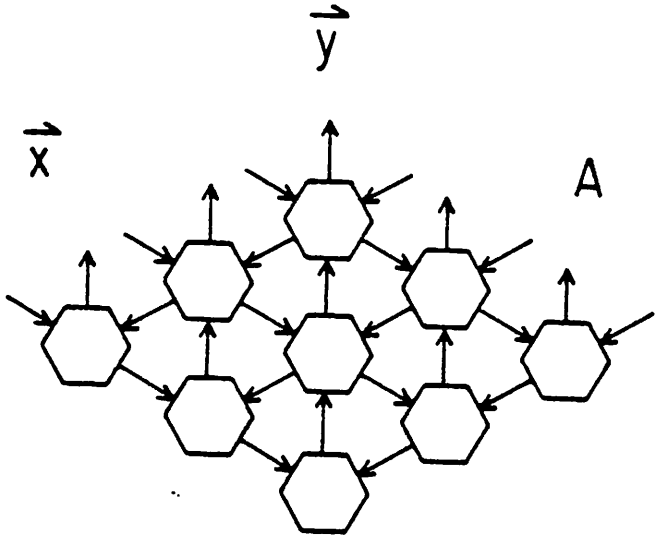

Figure 1: The MM design for $N=2$.

Shift registers must be provided on the chip for variable and constant storage. If the chip is laid out according to Figure 1, N input registers should be located in the upper left corner of the chip to hold the vector $\vec{x}$. Similarly, N output registers are needed at the top of the chip to collect the result vector $\vec{y}$. The matrix A can be stored in N shift registers of $O(N \log N)$ bits each. It is easy to see that the entire construction can fit in a rectangle of $O(N^2 \log N)$ area.

Each multiply-add step takes $O(\log N)$ time; N steps are performed during the computation of a single Fourier transform. However, N computations may proceed simultaneously if each is separated from the next by one multiply-add cell. The MM design thus operates in pipelined time $O(\log N)$.

The MM design has nearly the best possible time performance for a design of its size, since its combined area*time^2 performance is $O(N^2\log^3 N)$. As noted in the Introduction, no circuit can do better than $\Omega(N^2\log^2 N)$. The $O(\log N)$ discrepancy between these figures is due to the use of a rather slow multiply-add cell. A better asymptotic performance could be obtained through the use of larger and more complicated multiply-add cells; however, such a design would be infeasible since the "constant factors" associated with the MM design are rather large already (see Section 4).

3.2. The MV Design

The previous implementation calculated N Fourier transforms at a time. If such a large throughput is unnecessary, the MV or Matrix-Vector design can be used.

The MV design of Figure 2 consists of a row of N multiply-add cells connected in a linear fashion. This structure can perform a matrix-vector multiplication in N (parallel) multiply-add steps For the computation of a Fourier transform $\vec{y} = A\vec{x}$, the vector $\vec{x}$ is shifted into the left-most multiply-add cell, one element at a time. After N multiply-add times have elapsed, elements of the vector $\vec{y}$ emerge from the left-most cell. Each cell uses a different element of the matrix A for each multiply-add step: this may be visualized as vertical motion of the matrix A past the horizontal row of multiply-add cells. (For a more detailed description of this process, consult [7, p. 287].)

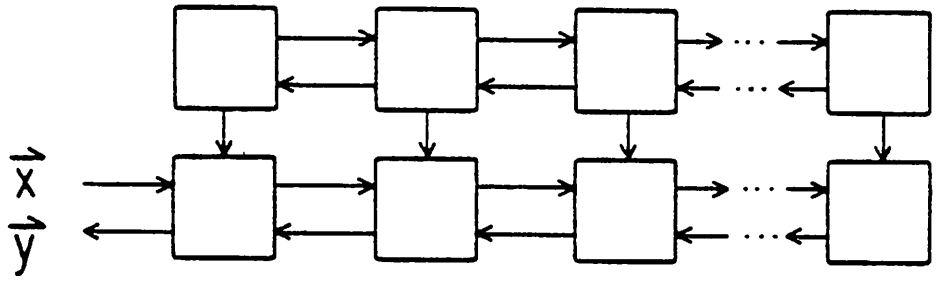

Figure 2: The MV design ($2N$ cells).

The time performance of the MV implementation is $O(N \log N)$, since it takes $O(N)$ multiply-add steps to compute a single Fourier transform. If the array A were stored explicitly, the MV design would require $O(N^2\log N)$ area – in other words, it would be just as large as the MM design, even though it operates N times more slowly. Fortunately, it is fairly simple to calculate the elements of A "on the fly" in the manner described by Kung and Leiserson [7, p. 290]. (The calculation is performed on a second row of N multiply-add cells positioned just above the original row. Each of these cells computes the value of ω^k required in the next step of the matrix multiplication $\vec{y} = A\vec{x}$. This computation may be completely overlapped with the matrix multiplication, so that there is no time overhead and only $O(N \log N)$ area overhead associated with the matrix A.)

The total area of the MV design is thus $O(N \log N)$, for it has $2N$ multiply-add cells of $O(\log N)$ area each. Its combined area*time^2 performance is $O(N^3 \log^3 N)$. This is a surprisingly poor result in view of the fact that the related MM design has a nearly optimal AT^2 figure. The reason that the MV design does so poorly is that an implementation with $1/N$ of the area of the MM design should only be slower by a factor of $N^{1/2}$: a linear tradeoff of time for area is not optimal. (It is possible to make a circuit with approximately linear area that has near-optimal performance, as will be demonstrated by the Mesh implementation below.)

3.3. The FFT Network

Another straightforward implementation of the Fourier transform takes advantage of the Fast Fourier Transform algorithm (the FFT). This algorithm is most naturally expressed as a computation on $(N/2)*(\log N)$ multiply-add cells, arranged in $\log N$ rows of $N/2$ cells each. The interconnections between rows can be in the form of a "perfect shuffle" [12] or a "butterfly" [4], depending upon how the cells are indexed. The butterfly arrangement is described below since it seems to give a better area bound, if only by a constant factor.

A computation in the FFT network may be visualized as flowing from top to bottom. The inputs are presented in pairs (x_{2i}, x_{2i+1}) to the top row of cells. These cells perform a multiply-add step, passing the results down to the next row of cells. The computation is complete when the data has flowed through all $\log N$ rows.

The best embedding of the FFT network is based on the butterfly organization. Please refer to Figure 3 for a sample layout. If the cells are numbered from 0 to 3 in each row, cell i in the top row is connected to cells i and $(i + N/4) \bmod N/2$ in the second row. The latter connection must be laid out carefully: you can probably convince yourself that $N/2$ horizontal channels of wiring are required between the first and second rows, one for each lateral connection. If the multiply-add cells are $O(\log N)$ units tall and $O(1)$ units wide, the first two rows occupy a region $O(N)$ units tall and $O(N)$ units wide. This layout allows room for the length-N wires. It also allows plenty of area to store the single ω^k value ($O(\log N)$ bits) required by each cell.

The connections between the second and third rows occupy just half as much room as the ones between the first two rows. In this case, cell i in the first half of the second row ($0 \le i < N/4$) connects to cell i and to cell $((i+N/8) \bmod N/4)$ in the first half of the third row. The cells in the second halves of these rows have analogous connections. Horizontal channels may be shared by corresponding cells in each half-side, so that $N/4$ channels are sufficient.

Similarly, the connections beween the third and fourth rows occupy just half as much room as the connections between the previous pair of rows. In this case, the $N/2$ cells in each row are broken into four groups. Cells within each

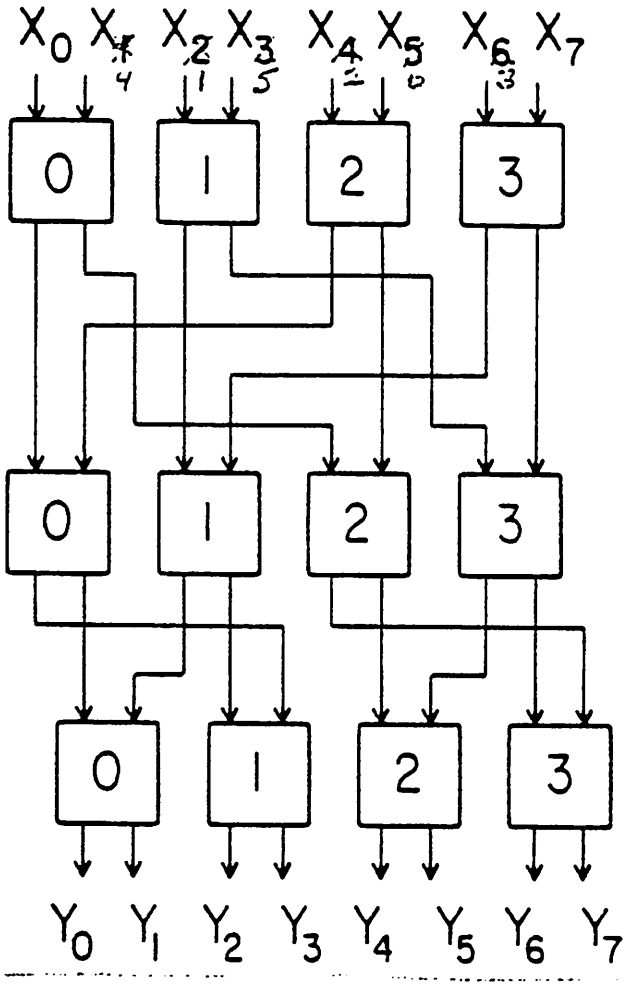

Figure 3: The FFT network for $N=8$.

group communicate solely with the cells in the group immediately beneath them.

The total area of the FFT network is $O(N^2)$, the sum of a geometrically decreasing sequence whose first term is $O(N^2)$.

The pipelined time of this implementation is $O(\log N)$, since $\log N$ computations may proceed simultaneously. Each computation must be separated from the next by at least one multiply-add cell, or by $O(\log N)$ time.

Note that the long wires between the rows do not change the asymptotic performance of the network. The drivers for these wires contribute a delay of $O(\log N)$ to each multiply-add step, but they do not affect the rate at which data can be shifted into and out of the multiply-add cells.

3.4. The SE Network

A shuffle-exchange (SE) network with $N/2$ multiply-add cells can perform an FFT in $\log N$ steps of computation [12]. Each cell uses a different ω^k value for each step. These values are obtained from an $O(\log^2 N)$-bit shift register associated with every cell.

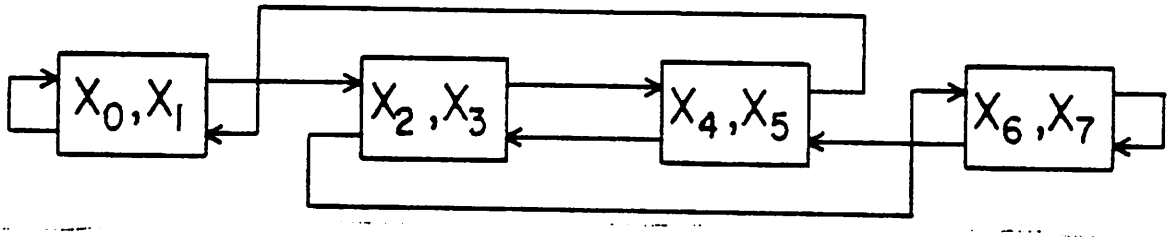

Figure 4: The SE network for $N=8$.

Figure 4 shows a 4-cell shuffle-exchange network capable of performing an 8-element transform. This layout is a condensed version of Stone's [12]. Only the shuffle connections are visible here; the exchange connections are internal to the cells.

The shuffle-exchange connections occupy much more room than the cells themselves: $O(N^2/\log N)$ area in the best embedding known [6]. This result applies only if N is of the form 2^{2^n}. In the general case of $N = 2^n$, the best embedding is $O(N^2/\sqrt{\log N})$ [13]. (An $O(N^2/\log^2 N)$-area embedding is the best possible result. Anything smaller would contradict the lower bound of $\Omega(N^2\log^2 N)$ for area*time^2 performance: the SE network operates in $O(\log^2 N)$ time, as shown below.

Each stage of computation on the SE network consists of a multiply-add step followed by a routing step. In a routing step, one word of data is sent along each intercellular connection. These connections are $O(N^2/\log N)$ in length, so that the drivers contribute $O(\log N)$ delay to the $O(\log N)$ time of a multiply-add step – an insignificant amount. The pipelined time of the SE network is thus $O(\log^2 N)$, since there are $\log N$ stages of computation.

3.5. The CCC Network

The cube-connected-cycles (CCC) interconnection for N cells is capable of performing an N-element FFT in $O(\log N)$ multiply-add steps [8]. Using the multiply-add cell of the previous constructions, the complete FFT takes $O(\log^2 N)$ time.

The CCC network is very closely related to the FFT network. In fact, a CCC network is just an FFT network with "end-around" connections between the first and last rows. For this reason, CCC networks do not exist for all N, only for those N of the form $(K/2)*(\log K)$ for some integer K. Figure 5 illustrates the CCC network for $N=8$. It is derived from the 4-element FFT network with "split cells": each cell handles one element of the input vector $\vec{x}$, instead of two as in the FFT network of Figure 3. (The reader is invited to redraw Figure 5, combining the cells that are linked by horizontal data paths. The resulting graph should be an end-around connected "butterfly.")

The CCC network is somewhat smaller than the FFT network, since it uses only N cells to solve an N-element problem instead of the $(N/2)*(\log N)$ cells used in the FFT network. Its interconnections can be embedded in $O(N^2/\log^2 N)$

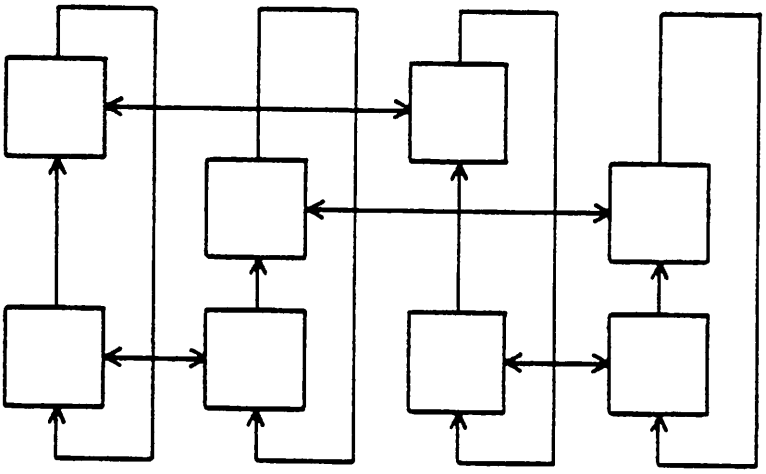

Figure 5: The CCC network for $N=8$.

area [8]. This is an optimal embedding, for the combined area*time^2 performance is within a constant factor of the limit, $\Omega(N^2\log^2 N)$.

It is rather difficult to describe the data routing pattern during the computation of a Fourier transform on a CCC, although the basic approach is similar to that taken on the SE network. Each of the log N multiply-add steps is preceded and followed by a routing step. These routing steps take $O(\log N)$ time each, for they move $O(1)$ words over each intercellular connection. Thus the total time spent in routing data does not dominate the time spent on multiply-add computations.

3.6. The Mesh Implementation

The Mesh implementation is the first example of a "small design." It requires only $O(N \log^2 N)$ area, which is to say that its area grows about linearly with problem size. The other designs have had nearly quadratic growth functions.

A square Mesh of N cells can do an N-element FFT in log N steps of computation by "simulating" the FFT network [11,13]. The action of each cell in the FFT network is implemented by one of the cells in the Mesh. Since there are N cells in the Mesh but only $N/2$ cells in each row of the FFT network, half of the Mesh cells are idle during each stage of the computation.

It turns out that a very good way to organize the computation is to put the i-th input and output registers in the i-th cell of the Mesh. (Mesh cells aare indexed in the natural, row-major ordering.) Then, following the "butterfly" form of the FFT network [4], the first multiply-add step combines the data in cell i with the data in cell $(i+N/2)$ *mod* N. This is accomplished by routing all of the data in the bottom half of the Mesh "upwards" by $\sqrt{N}/2$ rows, performing a multiply-add steps, then shifting the y_1 values back "downwards" by $\sqrt{N}/2$ rows.

The connections between the second and third rows of the FFT network can be simulated in a similar fashion, by global shifts upwards and downwards of $\sqrt{N}/4$ rows. The third butterfly is simulated by shifts of $\sqrt{N}/8$ rows, . . ., and the $(1/2 \log N)$-th butterfly corresponds to a shift by a single row. Then a series of column shifts begins, first by $\sqrt{N}/2$, then by $\sqrt{N}/4$, . . ., until the final computation of the FFT is performed with the aid of a single column shift.

Define a "unit-distance route" as a global shift of one word form each cell to its (right-, left-, up-, or down-) adjacent neighbor. There are $4(\sqrt{N}-1)$ unit-distance routes in the FFT implementation described above.

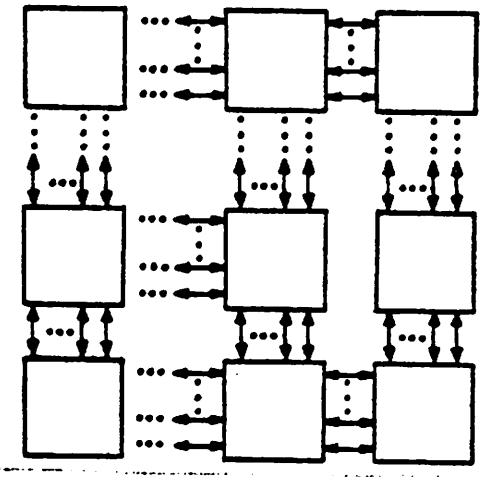

Figure 6: The Mesh of N multiply-add cells.

Parallel data paths should be provided in the Mesh design to make the routing steps as fast as possible. These paths are shown in Figure 6. The paths are one word wide; the complete Mesh of N cells occupies a square region $O(\sqrt{N} \log N)$ on a side. The total area of the Mesh is thus $O(N \log^2 N)$.

The cells in the Mesh implementation are more complicated than those in any of the previous approaches. A different distance and/or direction is used for each routing step. It is perhaps simplest to generate all routing and control signals in a local fashion, with $O(\log N)$ microinstructions of $O(\log N)$ bits each [13]. A shift register can be provided at each cell to store its microinstructions without any increase in its area, for the word-parallel data paths of the previous paragraph imply that each cell is $O(\log N)$ units on a side. This approach also allows plenty of room for the log N different ω^k values required by each cell.

A serial-to-parallel converter is used at the interface between the bit-serial multiply-add cell I/O and the word-parallel data paths. Each routing operation consists of $O(\log N)$ time periods to load this converter, some number of unit-distance routes, then another $O(\log N)$ time to get the data into the cells. Since the cells are $O(\log N)$ apart, drivers of $O(\log N)$ area and $O(\text{loglog } N)$ delay are used on each routing path.

Total time for the FFT on the Mesh is thus $O(\sqrt{N} \text{ loglog } N)$. The $O(\sqrt{N})$ unit-distance routes take the majority of the time; the $O(\log N)$ multiply-add

steps are asymptotically insignificant.

3.7. The Cascade Implementation

It is possible to do an N-element FFT with many fewer multiply-add cells than used in the previous implementations. The Cascade is an approach that uses only log N cells, one cell for each row of the FFT network [5]. See Figure 7.

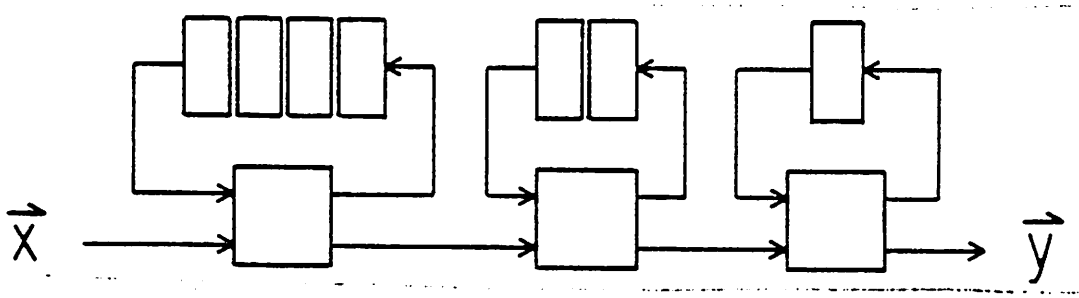

Figure 7: The Cascade implementation of the 8-element FFT.

The cell corresponding to the j-th row of the FFT network ($1 \le j \le \log N$) buffers its data in a shift register of $N/2^j$ words. The data streams through the cascade in a serial fashion: the first cell is able to combine x_i with $x_{i+N/2^j}$ by buffering x_i in its shift register. A similar process occurs at the other cells.

The total area of the Cascade is $O(N \log N)$, due mostly to the shift registers. The area of the multiply-add cells is unimportant in an asymptotic sense.

Each cell uses $N/2$ different ω^k values of $O(\log N)$ bits each. These would occupy $O(N \log^2 N)$ area if stored explicitly. Thus they should be computed "on the fly" by each cell, in much the same way as in the MV implementation. Each cell can compute a new value in a single multiplication time, obtaining $\omega^{i*N/2^j}$ for its i-th multiply-add step from the product of $\omega^{N/2^j}$ with the value it used in its $(i-1)$-st multiply-add step, $\omega^{(i-1)*N/2^j}$.

The time performance of the Cascade is $O(N \log N)$. A second computation may be started as soon as the first one has cleared the first cell, which takes time $O(N \log N)$.

3.8. The CPU Implementation

As its name suggests, this approach mimics the actions of a conventional CPU or uniprocessor as it runs an FFT. The input and output registers are realized by a random-access memory of $O(N \log N)$ bits and $O(N \log N)$ area [7, p. 321].

The CPU portion of the design is a glorified multiply-add cell that does a step of computation in $O(\log N)$ time. This is just sufficient time to fetch a word that might be as much as $O(\sqrt{N \log N})$ units distant. There is thus no asymptotic incentive to build a super-fast multiplication unit.

The $(N/2)*(\log N)$ multiplication steps in an FFT take a total of $O(N \log^2 N)$ time, making this the slowest design of this paper. Total area is $O(N \log N)$, due

mostly to variable storage. The ω^k values must be generated "on the fly" to obtain this area bound.

4. Conclusion

The area and time performance of the eight implementations is summarized by Table 1. Note that all the designs are nearly optimal in an area*time^2 sense except for the MV, the Cascade and the CPU. (Remember that $AT^2 = \Omega(N^2 \log^2 N)$ for the solution of the N-element Fourier transform.) The problem with these nonoptimal designs is that they are processor-poor: the number of multiply-add cells does not grow quickly enough with problem size.

Design	Area	Time	Area*Time^2
MM	$N^2 \log N$	$\log N$	$N^2 \log^3 N$
FFT	N^2	$\log N$	$N^2 \log^2 N$
SE	$N^2 / \log N$	$\log^2 N$	$N^2 \log^3 N$
CCC	$N^2 / \log^2 N$	$\log^2 N$	$N^2 \log^2 N$
MESH	$N \log^2 N$	$\sqrt{N}$ loglog N	$N^2 \log^2 N \text{loglog}^2 N$
MV	$N \log N$	$N \log N$	$N^3 \log^3 N$
Cascade	$N \log N$	$N \log N$	$N^3 \log^3 N$
CPU	$N \log N$	$N \log^2 N$	$N^3 \log^5 N$

Table 1: Area-time performance of the Fourier transform-solving circuits.

The Mesh is the only design that is nearly optimal under any AT^{2x} metric for $0 \le x \le 1$. Here the limiting performance is $AT^{2x} = \Omega(N^{1+x} \log^{2x} N)$ [13]. None of the other designs with $O(N)$ or fewer multiply-add cells is fast enough, while the other designs are much too large.

Of course, asymptotic figures can hide significant differences among designs due to "constant factors." The model used in this paper penalizes designs with simple control structures and those with a high ratio of memory to logic, since these designs will have much smaller constant factors than the others. The MM, the MV, the FFT, the SE and the Cascade are especially simple implementations because they have no complicated routing steps. They thus deserve a more detailed examination.

As indicated in Table 1, the MM is nearly optimal in its area*time^2 performance. However, it is by far the largest design considered in this paper since it uses N^2 multiply-add cells. (The others use $O(N \log N)$ or fewer cells.) Using current technology, one might place 10 multiply-add cells on a chip [13]; about 10^5 chips would be needed for a thousand-element FFT! Thus the MM design cannot be considered feasible until technology improves to the point that 100 or 1000 cells can be formed on a single wafer. Even then, the interconnections between chips will pose some difficulties, for there are 40 cells on the "edge" of a 100 cell chip.

The MV is an attractive design at present, despite its non-optimal area*time^2 performance. It uses only $2N$ cells in a linear array, so that a thousand-element Fourier transform can be implemented with only 10^2 chips of 10 multiply-add cells each. This design is of course much slower than the MM, since it produces only one element of a transform at a time rather than an entire transform.

The FFT network is also fairly attractive at present, for its $(N/2)*(\log N)$ cells can be formed on about the same number of chips as the MV, yet its performance is equal to the MM. The drawback of the FFT design is that the wiring on and between the chips is very area-consuming. This design requires $\Omega(N^2)$ area since its time performance is so fast; there is no hope of finding a clever partition of the design to reduce this value. The FFT thus has inherently long wires, whereas the MM and MV use only nearest-neighbor connections. If inter-chip wiring carries an extremely high cost, then the FFT is not a good choice.

The constant factor considerations of the SE design are very similar to those of the FFT network discussed above. The SE uses a factor of log N fewer cells than the FFT, so it is a bit smaller and slower. It suffers from the same problem of long inter-chip wires and poor partitionability.

The Cascade is another non-optimal design, like the MV, that deserves consideration because of its good "constant factors." It uses only log N multiply-add cells and N words of shift-register memory. These are arranged in a simple linear fashion. The Cascade achieves the same performance as the MV, producing one element of a Fourier transform during each multiply-add time. It is superior to the MV in that it uses many fewer multiply-add cells.

It is interesting to speculate whether the Cascade is the best way of producing one element of a Fourier transform at a time. A new metric and method of analysis is needed to answer this question, for such designs are non-optimal by definition. (If $O(N)$ time is required to complete an entire transform, there is only $O(\log^2 N)$ area remaining before $AT^2 = \Omega(N^2 \log^2 N)$ bound is reached. This is not even enough room to store the problem.)

Another interesting open problem is that of partitioning the SE network. If 100 or 1000 multiply-add cells can be placed on a single chip, what sort of off-chip connections should be provided so that these chips can be composed into a large SE design?

References

[1] Abelson H. and Andreae P., "Information Transfer and Area-Time Tradeoffs for VLSI Multiplication," *Comm. ACM*, Vol. 23, No. 1, pp. 20-23, Jan. 1980.

[2] Aho A., Hopcroft J., and Ullman J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] Brent R. and Kung H., "The Area-Time Complexity of Binary Multiplication," CMU-CS-79-136, Carnegie-Mellon Computer Science Dept., July 1979.

[4] Cochran W., Cooley J. *et al.*, "What is the Fast Fourier Transform?," *IEEE Trans. on Audio and Electro.*, Vol. AU-15, No. 2, pp. 45-55, Jun. 1967.

[5] Despain A., "Very Fast Fourier Transform Algorithms for Hardware Implementation," *IEEE Trans. Comput.*, Vol C-28, No. 5, pp. 333-341, May 1979.

[6] Hoey D. and Leiserson C., "A Layout for the Shuffle-Exchange Network," *Proc. 1980 Int'l Conf. on Parallel Processing*, IEEE Computer Society, 1980.

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

[8] Preparata F. and Vuillemin J., "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *20th Annual Symp. on Foundations of Computer Science*, IEEE Computer Society, pp. 140-147, Oct. 1979.

[9] Savage J., "Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI Models," TR-CS-50, Brown Univ. Dept. of Computer Science, Aug. 1979.

[10] Seitz C., "Self-Timed VLSI Systems," *Caltech Conf. on VLSI*, Caltech Computer Science Dept., pp. 345-354, Jan. 1979.

[11] Stevens J., "A Fast Fourier Transform Subroutine for Illiac IV," Technical Report, Center for Advanced Computation, Illinois, 1971.

[12] Stone H., "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Comput.*, Vol. C-20, No. 2, pp. 153-161, Feb. 1971.

[13] Thompson, C., *A Complexity Theory for VLSI*, Ph.D. Thesis, Carnegie-Mellon Computer Science Dept., Aug. 1980.