Proceedings of the 1980 IEEE International Conference on Circuits and Computers

FOURIER TRANSFORMS IN VLSI

C. D. Thompson
Division of Computer Science
U. C. Berkeley
Berkeley, CA

1. INTRODUCTION

One of the difficulties of VLSI design is the sheer 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. Analytical techniques are more often applicable to very large systems than to moderately-sized ones.

This paper shows that asymptotic analysis can be used to guide 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 which portions are truly important to optimize and which designs are inherently flawed. 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 can provide criteria for judging optimality. The performance of an optimal circuit can only be matched, not beaten. Several of the implementations of this paper are demonstrably optimal. They achieve the limiting area*(time)^2 performance of $\Omega(N^2 \log^2 N)$ for the N-element Fourier transform [14]. (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 by some constant times $N^2 \log^2 N$.) Similar performance limits have been proved for the problems of sorting, matrix multiplication, and integer multiplication [1,3,10,14].

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, ...) and to the bipolar transistors of I^2L.

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

Section 3 describes seven 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 performance 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 from 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 at any one point in the plane.

The unit of time in the model is proportional 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 proportional to the size of a simple circuit. 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 [8]. The amplifier stages are individually clocked, so that a driver has O(log k) delay but unit bandwidth.

The notion of "self-timed regions" [11] is incorporated into the model to account for the difficult of obtaining chip-wide synchronization. The nodes in a self-timed region are in synchrony: all signal transistions 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 [14].

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.
d. 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 receiver 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 state of the nodes in an "input register" may be modified from outside when a computation is initiated. See Assumption 6.

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 = Θ(log N). (A word length of $\lceil \log M \rceil = c*(\log N)$ bits is necessary and sufficient to describe the value of an input variable.)
d. The output variables $\tilde{y}$ are related to the input variables $\tilde{x}$ by the equation $\tilde{y} = A\tilde{x}$. 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.

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_0 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 [14]. 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.

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 are designed as a rectangle of O(1) width which can be folded into any rectangular shape.

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].

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

The input vector $\tilde{x}$ 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 vector emerges on the bottom edge. Thus N input registers are required on the upper left region of the chip, and N output registers must be located at the bottom. Additionally, N shift registers of O(N log N) bits each must be placed on the upper right to store the matrix A. 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 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" [13] or a "butterfly" [4], depending upon how the cells are ordered.

A computation in the FFT network may be visulaized 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. Cell i in the top row is connected to the cell immediately below it and to a cell in column (i+N/4) mode N/2. The second connection must be laid out carefully: you can probably convince yourself that N/2 horizontal channelys 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 N drivers of O(N) area that are needed for the length-N wires. It also allows plenty of room to store the constant ω^k required by each cell.

The connections between the second and third rows oocupy 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 ($0 \leq i < N/4$) connects to cell i and to cell (i+N/8) mod 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 suffice.

Similarly, the connections between 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 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.

The SE Network

A shuffle-exchange (SE) network with N/2 multiply-add cells can perform an FFT in log N steps of computation [13]. 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.

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 construction applies only to N of the form 2^{2^n}. In the general case of $N=2^n$, the best embedding is $O(N^2/\sqrt{\log N})$ [14]. (As little as $N^2/\log^2 N$ area might be enough, for this would lead to an optimal area*(time)^2 performance for the SE network.) These area results are a bit surprising, for there are only 4 connections to each node in the SE network.

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 down 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. The pipelined time of the SE network is thus $O(\log^2 N)$, since there are log N stages of computation.

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. The CC pattern is very similar to the FFT network of $(K/2)*(\log K) = N$ cells.

The computation of an FFT on the CCC network is a little more complicated than on the SE, although the approach is similar. Each of the $O(\log N)$ multiply-add steps is preceded and followed by a routing step. These routing steps take $O(\log N)$ time each, for they move one word over each intercellular connection. The time performance of the CCC is thus $O(\log^2 N)$.

The CCC can be embedded in $O(N^2/\log^2 N)$ area [9]. This is an optimal result since the area*(time)^2 performance is within a constant factor of the limit, $\Omega(N^2 \log^2 N)$.

The Mesh

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 it 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 [12,14]. 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 computatation.

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 a mesh. (Mesh cells are 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 $(i+N/2) \bmod N$. 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 step, 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 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 from 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.

Parallel data paths are built into the mesh design in an effort to make the routing steps as efficient as possible. These paths are one word wide so that a mesh of N cells occupies a square region $O(\sqrt{N} \log N)$ on a side, for a total area of $O(N \log^2 N)$.

Each cell of the mesh has $O(\log^2 N)$ logic gates. It has an $O(\log^2 N)$-bit shift register to store log N different ω^k values, one for each step in the computation. It also has $O(\log N)$ microinstructions of $O(\log N)$ bits each, to provide local control for the shifting operations [14].

A serial-to-parallel converter is used at the interface between the bit-serial 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

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	$N^2 \log^2 \text{loglog}^2 N$
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.

unit-distance routes, then another O(log N) time units to get the data into the cells. Since the cells are O(log N) apart (due to their width and to the width of the data paths), drivers of O(log N) area and O(loglog N) delay are used on each routing path.

Total time for the FFT on the mesh is $O(\sqrt{N}$ 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.

The Cascade

It is possible to do an N-element FFT with only log N multiply-add cells, if they are organized in a "cascade" [5]. This approach uses one cell for each row of the FFT network.

The cell corresponding to the j-th row of the FFT network 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 $x_{i+N/2}$ by buffering x_i in its shift register. A similar process occurs at the other cells.

The total area of the cascade implementation 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, since there are log N cells. To keep the area of the circuit down to O(N log N), the ω^k values are computed "on the fly" by each cell. A single multiplication by ω is all that is needed to obtain the value $\omega\uparrow(i*j)$ needed in the i-th step of cell j from the value $\omega\uparrow((i-1)*j)$ it used in the previous step.

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).

The CPU Implementation

As its name suggests, this approach mimics the actions of a conventional uniprocessor (or CPU) running an FFT. The input and output registers are formed into a random-access memory of O(N log N) bits and O(N log N) area [8].

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 $O(N \log^2 N)$ time, making this the slowest design in the 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 seven implementations is summarized by the table above. Note that all the designs are nearly optimal in an area*$(\text{time})^2$ sense except for the Cascade and the CPU. (Remember that $AT^2 = \Omega(N^2 \log^2 N)$ for the solution of an N-element Fourier transform.) The problem with the Cascade and the CPU seems to be that these designs are processor-poor: the number of multiply-add cells does not grow quickly enough with problem size.

The mesh is the only design that is nearly optimal under any AT^{2x} metric for $0 \leq x \leq 1$. Here the limiting performance is $AT^{2x} = \Omega(N^{1+x} \log^{1+x} N)$ [14]. None of the other "small" designs is small enough, and the other designs are much too large.

Of course, asymptotic figures can hide significant differences in "constant factors." The model used in this paper can be said to penalize designs with simple control structures or a high ratio of memory to logic, since these designs will be somewhat smaller than the others. The MM, the FFT, the SE, and the Cascade are especially simple implementations because they have no complicated routing steps. A much more detailed model of computation (and much more detailed analysis) is required to capture this effect. The development of such models is a promising area for future research.

REFERENCES

[1] Abelson H. and Andreae P., "Information Transfer and Area-Time Tradeoffs for VLSI Multiplication," Comm. ACM, Vol. 23, pp. 27-32, (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., (1979).

[4] Cochran W., Cooley J. et al., "What Is the Fast Fourier Transform?," IEEE Trans. on Audio and Electro., Vol. AU-15, pp. 45-55, (1967).

[5] Despain A., "Very Fast Fourier Transform Algorithms for Hardware Implementation," IEEE Trans. Comput., Vol. C-28, pp. 333-341, (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] Kung H. and Leiserson C., "Systolic Arrays (for VLSI)," CMU-CS-79-103, Carnegie-Mellon Computer Science Dept., (1978).

[8] Mead C. and Rem M., "Cost and Performance of VLSI Computing Structures," IEEE Journal of Solid-State Circuits, Vol. SC-14, pp. 455-462, (1979).

[9] 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, (1979).

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

[11] Seitz C., "Self-Timed VLSI Systems," Caltech Conf. on VLSI, Caltech Computer Science Dept., pp. 345-354, (1979).

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

[13] Stone H., "Parallel Processing with the Perfect Shuffle," IEEE Trans. Comput., Vol. C-20, pp. 153-161, (1971).

[14] Thompson C., "A Complexity Theory for VLSI," Ph.D. Thesis, Carnegie-Mellon Computer Science Dept., (1980).