[F-14 18-869] On VLSI Complexity Theory

Thomas Jackson, Haewon Jeong

I. Introduction

Soon after they were developed, very-large-scale integration circuits became an important part of the forefront of computing. The development of reliable interconnect directly on silicon wafers enabled circuits consisting of many devices to be built very easily. As the design space exploded in size with the flexibility of VLSI, an important question arose: is there any way to provide a good lower-bound on the time and energy it takes for a given operation in a computing network? Such a lower bound, if tight enough, can indicate when a circuit has been optimized as much as is theoretically possible.

All computing operations, at their core, involve the manipulation of information to achieve a desired result. Across any implementation of a given function, the fundamental flow of information must be the same. Therefore, it is logical to borrow concepts from information theory to analyze a given problem. Information theoretical techniques are well suited to providing lower bounds since they strip away overheads that are implementation dependant.

These notes will begin by outlining the general model of VLSI circuits used by Thompson ([3]) to analyze these types of problems. Then some examples will be given of calculating lower bounds on computation problems. In this theory, upper bounds can conveniently be shown by considering real implementations. Examples of where Thompson's theory provides a relatively tight lower bound are given, as well as a theoretical example where the lower bound given by his method is clearly loose. This provides some insight as to which problems are well-suited to this analysis, and which are not. Finally, a short discussion of a few extensions to Thompson's work is given.

II. Modeling of VLSI Circuits

The foundational work for considering on-chip computing networks can be found in C.D. Thompson's thesis from 1980. Thompson's thesis [Thompson, 1980] starts by laying out the assumptions used to model a VLSI circuit. This includes the modeling of the physical components, which are abstracted as nodes and wires. Additionally it includes a measurement of energy consumption in VLSI circuits that is amenable to constructing bounds, as well as defining what exactly is meant by a VLSI computation problem.

A. *VLSI Circuit Definition*

For the purpose of an initial exploration into this theoretical circuit analysis, a VLSI circuit is defined as a graph with a set of constraints that are derived from the physical constraints of creating VLSI circuits. In VLSI circuit fabrication, the active devices (Transistors) are all located on a two dimensional plane. Therefore, in this model the VLSI circuit graph is considered to be on a two-dimensional grid.

Each square of the grid can contain a node or a wire crossover, and each edge of the square can contain at most one wire. This means at most 4 wires can be associated with one grid square. Using this model, the area of the circuit is defined as the number of squares occupied by wires or nodes. The length of the sides of each square is given by a circuit parameter λ.

In this model, the wires are used to carry information from node to node, where some sort of elemental computation is done. To make conclusions about the time it takes for the circuit to complete its computation, the bandwidth of these wires must be defined. The wires in this model are assumed to have, at most, unit bandwidth in either direction.

The inputs to the computation problem are defined as N input variables that each take on one of M different values. Each node can take a combination of up to three of these values and produce outputs. Any delay involved in this computation is added to the delay of the wires associated with the node. The initial state of the system is (i.e. the input to the computation problem) is defined at a subset of the nodes called the source nodes, and this information is distributed throughout the graph as computation proceeds. Similarly, there is a set of sink nodes which represent the outputs of the computation problem. The problem is considered solved by time T when all sink nodes take on correct values for the input for all t>T.

III. Lower Bound Results

In this section, we will first derive the lower bound for the circuit area, and then the lower bound on the computation time. Finally we will show how to link two different lower bound results to compute area-time lower bound with some example functions.

A. *Area Lower Bounds*

Area of circuit can be calculated by counting the number of unit squares occupied by wires and nodes. Lower bound on the circuit area can be given in terms of minimum bisection width of a VLSI circuit graph. Here, we will show that a circuit graph with minimum bisection width w occupies at least $w^2/4$ unit squares. To do this, we will first introduce concepts of circuit biesection and minimum bisection width.

Definition III.1. *For undirected graph $G = (V, E)$, a subset of edges $E_S \subseteq E$ is said to "biesect" a subset of vertices $S \subseteq V$ when it satisfies the following conditions.*

1) $V_1 \cup V_2 = V, S_1 \cup S_2 = S$
2) $S_1 \subseteq V_1, S_2 \subseteq V_2$
3) $|S_1| \leq |S_2| \leq |S_1| + 1$
4) *Every path from a vertex in V_1 to a vertex in V_2 contains an edge in E_S.*

Note that bisection here doesn't bisect all vertices into equal halves, but it only bisects a specific subset of vertices S into equal halves. This is stated in the item 3) in Defintion III.1. In our VLSI circuit graph, a subset S will be a set of input nodes.

Definition III.2 (*Minimum Bisection Width*)**.** *The minimum bisection width w is defined as*
$w = min\{|E_S|$ *s.t.* E_S *bisects* S *in* $G\}$

To see the link between minimum bisection width and circuit area, let's look into a preliminary result before going into the area lower bound theorem.

Lemma III.1. *If a circuit graph G fits in a rectangle area of $(w-1)^2\lambda^2$, then its minimum bisection width is at most w.*

Proof: Let the height of a rectangle to be bigger than its width. Then the height of the rectangle is at most $w - 1$. Now, think about bisecting a graph with one vertical line. If we can succesfuly do this, then the minimum bisection width will be at most $w - 1$, so the theorem holds.

Only thing left to be proven is a case where we can't bisect a graph with a single verticle line. This happens only when more than one nodes are lying on this bisecting vertical line. In this case, we can divide nodes lying on the vertical line into two parts with a unit-length horizontal line and then do vertical bisections like a zig-zag line. Figure 1 shows an example of such zig-zag bisection. In this case, total bisection width will be at most $(w - 1) + 1$, which is w. ■

Now, let's move onto our main result, the area lower bound of VSLI graph. Zig-zag bisection technique we used in the proof of Lemma III.1 will again be used to prove the lower bound theorem, but we will not go into details of this proof.

Theorem III.2. *If the minimum bisection width of the source nodes in a communication graph is w, then the area occupied by the graph is lower bounded by $\frac{w^2}{4}$.*

Proof: We will only show an idea of the proof briefly. First, start with the vertical zig-zag bisecting line as in the proof of Lemma III.1 which achives minimum bisection width w. Then, we will construct more zig-zag bisecting lines by moving this vertical line by one unit left or right. As we move the vertical line farther from the original center, more horizontal zig-zags are needed. By keep doing this zig-zag step, is is possible to construct $\lfloor \frac{w}{2} \rfloor$ different bisections. Counting the unit squares occupied by wires crossing thses $\lfloor \frac{w}{2} \rfloor$ bisecting zig-zags, we can derive the area lower bound $w - 1 + \sum\limits_{1<k\leq\lfloor w/2 \rfloor} (w - 2k + 2) \geq \frac{w^2}{4}$. ■

B. *Time Lower Bounds*

To derive the lower bound on the time, we will again use a minimum bisection argument to link it with area lower bound. If the information needed to be transmitted across the minimum bisection is b bits, then we need at least $\frac{b}{w}$ time units to compelete the computation (assuming that each wire can transmit 1 bit/time unit). However, it is hard to calculate the exact value of b, so we will instead calculate the minimum information to be communicated across any bisection. This may lead to a looser lower bound, but we will see that for some computations like FFT or sorting, this lower bound is tight.

As you can see in figure 1, minimum bisection will divide the circuit into two halves, S and R. By this bisection, input nodes will be divided into equal halves. We will denote these nodes as $\tilde{x}_R$ and $\tilde{x}_S$, and output nodes can also be denoted as as $\tilde{y}_R$ and $\tilde{y}_S$. Output nodes are not necessarily divided into equal halves, so let $|\tilde{y}_R| = k$ and $|\tilde{y}_S = N - k$. Without loss of generality, we can assume that $k > \lceil N/2 \rceil$.

A goal function we want to compute is F, but each half only have to compute bisected subfunction F_R or F_S. F_R can be written as follow.

$$\tilde{y_R} = F_R(x_R, *) \tag{1}$$

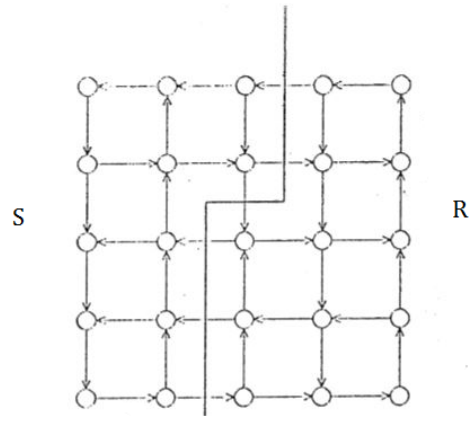

Fig. 1. A bisection of a circuit which bisects the circuit into equal havles, S and R.

For specific assignment of $\tilde{x}_R$ and $\tilde{x}_S$, it can be written as:

$$\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S)|\tilde{x}_R \tag{2}$$

where $|\tilde{y}_R| = k, |\tilde{x}_R| = \lceil N/2 \rceil$, and $|\tilde{x}_S| = \lfloor N/2 \rfloor$.

Minimum information flow we need is now thought as information that needs to be communicated from S to compute $F_R(\tilde{x}_R,)$. We can see that the amount of information needed depends on F. For example, a simple multiplication function $\tilde{y} = a\tilde{x}$ doesn't need any information flow over the bisection as it can be perfectly partitioned into $\tilde{y}_R = a\tilde{x}_R$ and $\tilde{y}_S = a\tilde{x}_S$. However, if function F is injective, we need the whole $\tilde{x}_S$ to compute F_R. In this case, following lemma gives the simple time lower bound.

Lemma III.3. *If the minimum bisection R of a communication graph of width w induces an injective function $\tilde{y}_R = F_R(\tilde{x}_R, \tilde{x}_S)$, for each $\tilde{x}_R$, and if all $|\{\tilde{x}_S\}|$ values of $\tilde{x}_S$ are equally likely, then the average time to compute F is at least* $(\log|\tilde{x}_S/w|)$.

If a function is not injective, it is more complicated to compute the amount of information. For non-injective functions, the amount of information flow does not only depend on the function F, but also depends on the input $\tilde{x}_R$. Think of a function which outputs the smallest value among the input x. If any of x_R input is 0, then we don't need any information from S. For other cases, the smallest value among $\tilde{x}_S$ should be transmitted to R to compute the output.

We will first examine the average case time lower bound which is averaged over all inputs. Then, we will investigate the worst case input lower bound beause in many cases, it is hard to get the average of all inputs. For this calculation, we will introduce a new concept, information complexity of a function.

Definition III.3 (*Information Complexity of a Function*)**.** *Information complexity of a function F, $H(F)$ is defined as the minimum information flow across any bisection R. It satisfies the following inequality.*

$$H(F) \geq \min_R [\sum_{\tilde{x_R}} p(\tilde{x_R}) \sum_{\tilde{y_R}} -p(\tilde{y_R}|\tilde{x_R}) \log p(\tilde{y_R}|\tilde{x_R})] \tag{3}$$

where $|\tilde{x_R}| = \lceil N/2 \rceil, |\tilde{y_R}| = \lceil K/2 \rceil, p(\tilde{x_R}) = 1/|\tilde{x_R}|$, *and*
$p(\tilde{y_R}|\tilde{x_R}) = |\{\tilde{x_s} s.t. F_R(\tilde{x_R}, \tilde{x_S}) = \tilde{y_R}/|\{\tilde{x_S}\}|$

When you see equation (3), you can note that the right hand side of inequality is conditional entropy of $\tilde{y}_R$ given $\tilde{x}_R$, which is information needed to decide $\tilde{y}_R$ given $\tilde{x}_R$. Conditional entropy value is averaged over all input $\tilde{x}_R$'s, so it can be thought as average information to be communcated across the bisection to compute a function $y_R = F(x_R, x_S)$. This is explicitly stated in the following lemma.

Lemma III.4. *The average time to compute a function F on a communication graph of width w is at least $\frac{H(F)}{w}$.*

Proof: Since $H(F)$ is minimum information flow over any bisection, minimum bisection also needds at least $H(F)$ bits of communication. The bandwidth here is limited by the minimum bisection width w, so the average time to evaluate function F will be at least $\frac{H(F)}{w}$. ■

Definition III.4. *The worst-case information complexity of a function $H_{worst}(F)$ is the minimum information across any bisection R and the worst input. It follows the inequality below:*

$$H_{worst}(F) \geq \min_R \max_{\tilde{x}_R} \sum_{\tilde{y}_R} -p(\tilde{y}_R|\tilde{x}_R) \log p(\tilde{y}_R|\tilde{x}_R) \tag{4}$$

Using the defintion of the worst case inforamtion complexity, the worst-case time lower bound can be given similar to Lemma III.4.

Lemma III.5. *The worst-case time to compute a function F on a communication graph of width w is at least $\frac{H_{worst}(F)}{w}$.*

Proof: Same argument as Lemma III.4. ■

Definitions and theorems we introduced here can be better understood with examples. In the following subsection, we will give some examples where we actually compute the lower bounds.

C. Lower Bound Examples

We will first examine information complexity of equality and compare functions. By comparing the information complexity of two functions, we can see somewhat counter-intuitive result that equality function is easier to compute than compare function. Then we will examine more comlicated functions, N-point discrete Fourier transform and sorting.

For all following examples, we will assume that inputs are uniformly distributed, i.e, $P(\tilde{x}_R) = 1/|\tilde{x}_R| forall \tilde{x}_R$.

1) Equality and Compare: Equality function is a function that takes two inputs and outputs whether they are equal or not. Compare function is similar, but it outputs whether the first argument is greater than the second input or not. For both functions, we will constrain input arguments to take values from 0 to $M-1$. More formal definition of equality function F_{eq} and F_{comp} is given below.

$$F_{eq}(x_R, x_S) = \begin{cases} 0, & \text{if } x_R \neq x_S \\ 1, & \text{if } x_R = x_S \end{cases} \tag{5}$$

$$F_{comp}(x_R, x_S) = \begin{cases} 0, & \text{if } x_R > x_S \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

To compute information complexity of thse functions, we can take the side which has an outptut node as a side R.

$$\begin{aligned} H(F_{eq}) &= H(\tilde{y}_R|\tilde{x}_R) \\ &= \sum_{\tilde{x}_R} \frac{1}{M}\left(\frac{1}{M}\log M + \frac{M-1}{M}\log\frac{M}{M-1}\right) \end{aligned} \tag{7}$$

$$\begin{aligned} H(F_{comp}) &= H(\tilde{y}_R|\tilde{x}_R) \\ &= \sum_{\tilde{x}_R} \frac{1}{M}\left(\frac{\tilde{x}_R}{M}\log\frac{M}{\tilde{x}_R} + \frac{M-\tilde{x}_R}{M}\log\frac{M}{M-\tilde{x}_R}\right) \\ &\geq 2\sum_{\tilde{x}_R} \frac{1}{M}\frac{\tilde{x}_R}{M}\log\frac{M}{\tilde{x}_R} \\ &\geq 2\sum_{\tilde{x}_R \leq \frac{M}{2}} \frac{1}{M}\frac{\tilde{x}_R}{M} \\ &> \frac{1}{2} \end{aligned} \tag{8}$$

The last inequality in equation (8) states that $H(F_{comp})$ is always greater than $\frac{1}{2}$ whereas $H(F_{eq})$ that keep decreasing with increasing M. Thus, for small M, equality function has greater complexity, but as M increases, equality computation gets easier than compare function.

2) Discrete Fourier Transform: N-point DFT can be thought as a matrix multiplication $y = F \cdot x$ where

$$F = \begin{bmatrix} \alpha^{0\cdot 0} & \alpha^{0\cdot 1} & \cdots & \alpha^{0\cdot(N-1)} \\ \alpha^{1\cdot 0} & \alpha^{1\cdot 1} & \cdots & \alpha^{1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(N-1)\cdot 0} & \alpha^{(N-1)\cdot 1} & \cdots & \alpha^{(N-1)\cdot(N-1)} \end{bmatrix} \tag{9}$$

α here is a primitive Nth root of unity, i.e, $\alpha^N = 1, \alpha \neq 1$. Matrix F is always invertible and inverted matrix F^{-1} represnets inverse Fourier transform.

To apply bisection argument, we will here consider reduced DFT. In reduced DFT, we only compute the first $N/2$ elements of the output. By restricting ourselves to reduced DFT, function we compute can be remained as a bijection after bisecting. As the function is bijective, we can easily compute the average information complexity.

Lemma III.6. $H(rDFT) > \frac{N}{4}\log N$

Proof: Let's denote reduced DFT as rDFT, and the output of rDFT residing on side R as $\tilde{y}_{rR}$. As rDFT is bijective given $\tilde{x}_R$, the number of possible values of $\tilde{y}_{rR}$ given $\tilde{x}_R$ is $N^{N/4}$. Hence, the amount of information needed from the side S is at least $\log\left(N^{N/4}\right) = \frac{N}{4}\log N$. ■

It is shown that any communication graph that solves DFT solves rDFT in the same amount of time (Lemma 7 in [3]), so we can just use information complexity of rDFT to compute the lower bound for a full DFT. With this time lower bound result, we can give area-time lower bound for DFT as follows.

Theorem III.7. *AT^2 complexity for DFT is bounded by $\Omega(N^2\log^2 N)$.*

Proof: The average time lower bound for DFT is bounded by

$$\begin{aligned} T &\geq \frac{H(\text{DFT})}{w} = \frac{H(\text{rDFT})}{w} \\ &> \frac{\frac{N}{4}\log N}{w} \end{aligned} \tag{10}$$

By combining this with Theorem III.2, we can get

$$
\begin{aligned}
AT^2 &\geq \frac{w^2}{4} \cdot \left(\frac{\frac{N}{4} \log N}{w} \right)^2 \\
&= \left(\frac{N}{8} \right)^2 \log^2 N \\
&\sim \Omega(N^2 \log^2 N)
\end{aligned} \tag{11}
$$

■

Note that this area-time complexity is greater than the optimal computational complexity of Fourier transform which is known to be $\Theta(N \log N)$. It suggests that when we think about the energy complexity, we can't simply use conventional computational complexity since they differ in order sense.

3) Sorting: Sorting function F_{sort} takse N input arguments and each input takes a value betweeen 0 and $M - 1$. The outputs of the function is N values of inputs in a non-decreasing order. Like we did with DFT, we will reduce the sorting function to "reduced sorting", F_{rsort}, which computes only the first half of the outputs. In other words, the goal of reduced sorting is to ouput the smallest $N/2$ values among inputs in a sorted way. By reducing the sorting function, worst-case analysis becomes easy. For whichever bisection you choose, the worst-case input $\tilde{x}_R$ is a sequence where all $N/2$ arguments take value $M-1$. In this case, to compute the smallest $N/2$ values, you need all information from the other half S. The lower bound on the information complexity of reduced sorting is given in the following lemma.

Lemma III.8.

$$
H_{worst}(F_{rsort}) > \frac{N}{4} \log \left(\frac{2M}{N} \right) \tag{12}
$$

Proof: As we did earlier, without loss of generality, we can assume than the side R has more than a half of the outputs. so that $|\tilde{y}_R| > \frac{N}{4}$.

Then we will examine informaiton complexity to compute $\frac{N}{4}$ outputs of $\tilde{y}_R$. For the worst case input $\tilde{x}_R$, possible values of $\tilde{y}_R$ are length-$\frac{N}{4}$ ordered sequences consising of numbers from 0 to $M - 1$. The number of such sequences is $\binom{M + \frac{N}{4} - 1}{\frac{N}{4}}$.

Thus, to specify the value $\tilde{y}_R$ given $\tilde{x}_R$, at least $\log \binom{M + \frac{N}{4} - 1}{\frac{N}{4}}$ bits are needed from side S.

$$
\begin{aligned}
H_{worst}(F) &\geq \min_R \max_{\tilde{x}_R} \sum_{\tilde{y}_R} -p(\tilde{y}_R|\tilde{x}_R) \log p(\tilde{y}_R|\tilde{x}_R) \\
&\geq \log \binom{M + \frac{N}{4} - 1}{\frac{N}{4}} \\
&\geq \log \left(\frac{M}{N/4} \right)^{\frac{N}{4}} \\
&= \frac{N}{4} \log \left(\frac{4M}{N} \right)
\end{aligned} \tag{13}
$$

■

By combining Lemma III.8 and area lower bound, we can easily give area-time lower bound for sorting.

Theorem III.9. *AT^2 lower bound for sorting is given by $\Omega(N^2 \log^2 N)$ for $M \sim N^{1+\epsilon}$.*

Proof:

$$
\begin{aligned}
H_{worst}(F_{sort}) &\geq H_{worst}(F_{rsort}) \\
&\geq \frac{N}{4} \log \left(\frac{4M}{N} \right) \\
&\sim \Theta(N \log N) \text{ for } M \sim N^{1+\epsilon}.
\end{aligned} \tag{14}
$$

By combining this result to Theorem III.2, we can get

$$
\begin{aligned}
AT^2 &\geq \frac{w^2}{4} * \left(\frac{N \log N}{w} \right)^2 \\
&\geq N^2 \log^2 N
\end{aligned} \tag{15}
$$

■

IV. Upper Bound Examples

Upper bounds for VLSI complexity are shown constructively. By creating a communicaton graph that solves the desired computation problem, an upper bound is proven. Thompson's thesis provides a few examples of these upper bounds, which will be discussed here.

A. Upper Bound Examples

Thompson gives two upper-bound examples of graphs that solve both the DFT and the sorting problem. One of these is the "shuffle-exchange" design which is fast but large, while the other is based on a "square mesh" which is small but slow.

One key to efficiently implementing the DFT algorithm is breaking it up into a network of fairly simple cells. A DFT consists of a series of multiply-add cells. Each cell has two inputs, X_0 and X_1, two outputs Y_0 and Y_1 and a paramter k. The cell performs the function

$$
\begin{aligned}
Y_0 &= X_0 + \alpha^k X_1 \\
Y_1 &= X_0 - \alpha^k X_1
\end{aligned} \tag{16}
$$

We can graphically represent this function as shown in figure 2. Figure 3 shows an 8-bit implementation. Notice each successive level halves the size of the problem, so a size N FFT is solved by two size N/2 FFT blocks (parameterized appropriately).

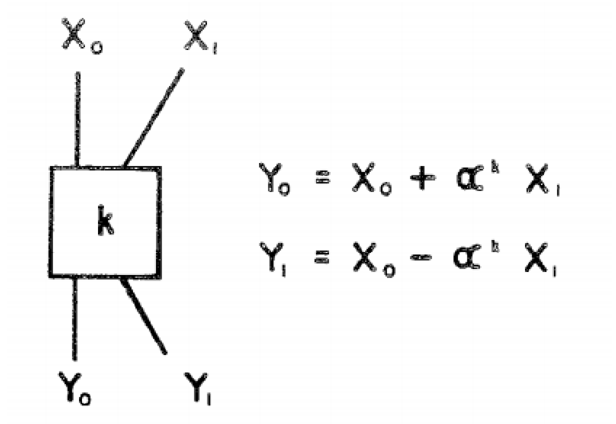

Fig. 2. The mutliply-add cell used in the FFT construction

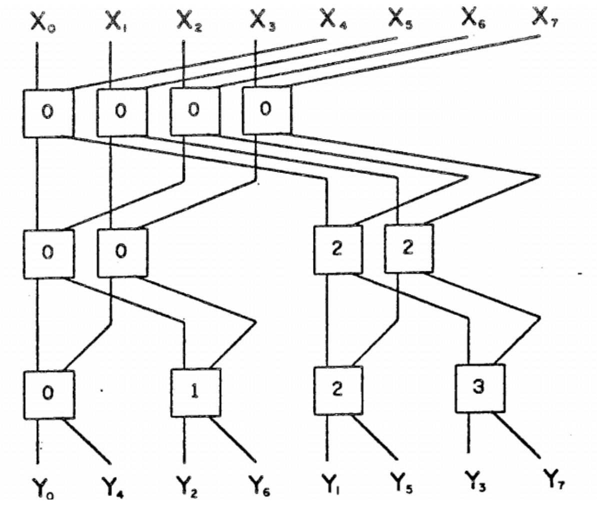
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7
0 0 0 0
0 0 2 2
0 1 2 3
Y_0 Y_4 Y_2 Y_6 Y_1 Y_5 Y_3 Y_7

Fig. 3. An 8-bit FFT implemented with the fundamental FFT cells

It is possible to implement the graph given in figure 3 directly in hardware, with a multiply-add cell for each box in the figure. This, however, would give a very loose upper bound. The key observation in producing an area- and time-efficient implementation is that the blocks on each layer of the network can be reused. Since each layer only depends on the immediately preceding row, the circuitry can be reused in each step of the algorithm, and the exact method of doing this is called a "recirculation algorithm." Thompson examines two recirculation algorithms that provide relatively tight upper bounds, the shuffle-exchange network and the mesh network.

B. Mesh Networks

The mesh network represents an extremely area-efficient way to implement the FFT. Each cell of the network corresponds to a different input/output of the FFT algorithm. At the start of the operation, the cells are loaded with the input to the FFT, and by the end of the operation each contains the correct output for that bit. The cells are connected in a mesh, as shown in figure 4

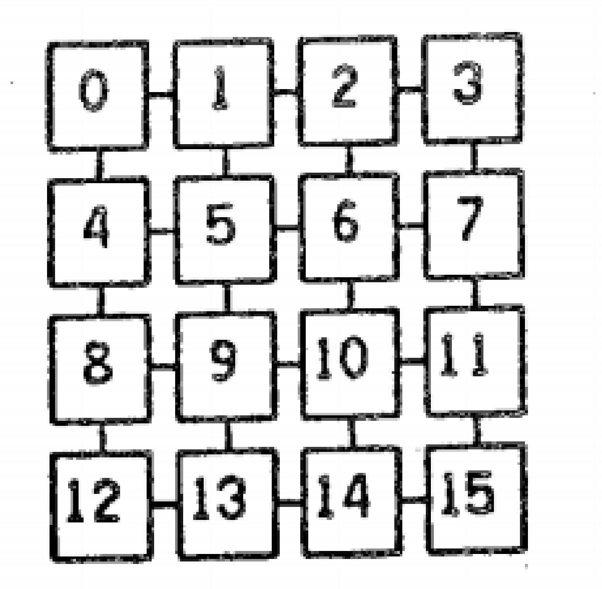

Fig. 4. The mesh network for a 16-bit FFT. The number in the cell corresponds to the bit input

Each cell has two input registers and two output registers, and at each step can create the values in the output registers based on the inputs. Each cell also has the ability to route its outputs to any other cell in the network along the most efficient path. During each stage of computation (as represented by a row in 3), the amount of time taken for communication is represented by the distance it takes to move the data into and then out of the appropriate cells. Each stage of the computation requires a smaller distance to be traveled, as the necessary operations become more local. Assuming the time to travel from one cell to another is t_r and the time to perform a multiply-add operation is t_m, Thompson shows the following:

Lemma IV.1. *An N-element FFT can be performed on an N-cell square mesh in time $T = O(N^{1/2}t_R + (log\ N)t_M)$ if a unit-distance route takes time t_R and a multiply-add step takes t_M.*

Proof: An N-element FFT consists of $log\ N$ computation rows. In a mesh network, row k is performed by making two distance-($N/2^k$) routings and one multiply-add operation. To use the topology of the mesh as efficiently as possible, vertical routings are used when $k < (log\ N)/2$. For this topology, the total time for the FFT is given by

$$T = \sum_{1 \leq k < (log\ N)/2} (2(N/2^k)/N^{1/2} t_R + t_M) + \sum_{(log\ N)/2 \leq k < log\ N} (2(N/2^k) t_R + t_M). \quad (17)$$

Therefore

$$T = 4(N^{1/2} - 1) t_R + (log N) t_M. \quad (18)$$

■

To complete the upper-bound on this operation, we must find values for t_R and t_M and for the area of each cell. The design of the multiply-add cell is guided by the result of lemma IV.1. The routing time t_R has a much larger impact on the overall time than the computation time t_M, therefore the cells will be designed to perform routing in parallel where possible, and operations in serial. Serial operation takes less area, but is slower, while parallel operation is faster and more area-hungry.

Since the routing is done in parallel, all M-bits of a FFT word can be communicated simultaneously. In the mesh network, the distance that each driver must drive is a wire of length $log\ N$. Therefore, this gives us Lemma 2, the routing time for the network.

Lemma IV.2. *The multiply-add cell has a routing time of $t_R = O(loglog\ N)$, exclusive of the time it takes to shift data into and out of the computation unit (this will be included in t_M).*

Proof: A reasonable upper-bound assumption on the delay of driving a wire of length N is $O(log\ N)$, therefore since the length in the mesh network is $log\ N$, we get that $t_R = O(loglog\ N)$. ■

The size of this driver scales as $O(log\ N)$ by $O(1)$, since it can be on the periphery of the cell, and we will show the cell is $O(log\ N)$ by $O(log\ N)$.

An upper-bound for the multiply-add circuitry can be found by imagining it is a general processing unit. Thompson shows that the actual multiply-add operation can be done with circuitry that fits in an area of $O(log\ N)$ by $O(1)$. Therefore, the area scaling comes from the instruction set passed to the multiply-add circuit.

There are $O(log\ N)$ instructions, since this is the number of computation steps needed for the FFT. The instructions consist of a value α_j needed for the FFT ($O(1)$), a control for the tranceivers and multiplexers ($O(1)$), and the number of clock cycles to persist the instruction. Since each instruction lasts for no longer than $O(N^{1/2} loglog\ N)$ time, this can be represented with an $O(log\ N)$ bit word. The bits in the instruction can be stored in $O(1)$ structures, so the total size of the memory is $O(log\ N)$ by $O(log\ N)$. The total cell area, therefore, is $O(log^2\ N)$, and this is shown in figure 5.

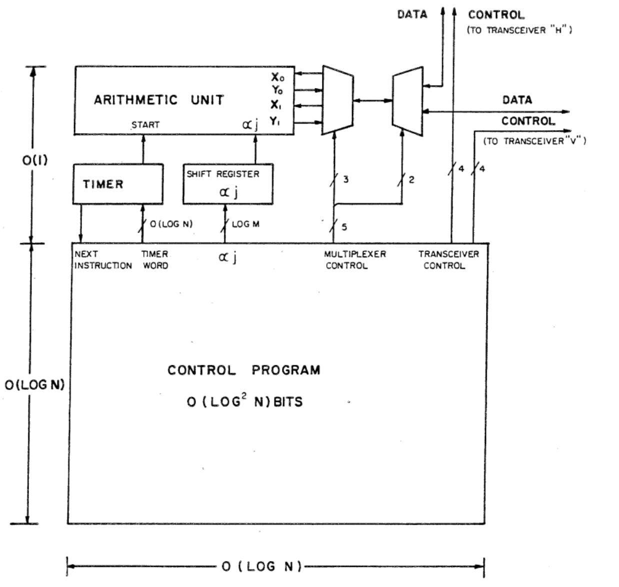

Fig. 5. The mesh network for a 16-bit FFT. The number in the cell corresponds to the bit input

The time taken to perform a multiply-add can be shown to take $O(log^2\ N)$ time. With the area and time of the cell, we are prepared to make a claim on the upper bound of the AT complexity of an FFT.

Theorem IV.3. *An N-element FFT can be performed in $O(Nlog^2\ N)$ area and $O(N^{1/2} loglog\ N)$ time on N multiply-add cells arranged in a square mesh.*

Proof: By lemma IV.1, an N-element FFT can be performed in time $T = O(N^{1/2} t_R + (log\ N) t_M)$ on an N-cell mesh. As discussed above, $t_R = O(loglog\ N)$ and $t_m = O(log^2\ N)$, therefore $T = O(N^{1/2} loglog\ N)$.

The area is $O(Nlog^2\ N)$ since the mesh consists of N cells of area $O(log^2\ N)$. ■

This result is very interesting, because if we take the AT^2 value, we get that the system is upper-bounded by $O(N^2 log^2\ N loglog^2\ N)$. As shown in the previous section, the lower bound using Thompson's method for the FFT is $AT^2 = \Omega(N^2 log^2\ N)$. This shows that the lower-bound for this circuit is fairly tight, since a conservative (although well-designed) upper-bound is only off by $O(loglog^2\ N)$. Interestingly, this type of analysis can help guide in further circuit design. For example, since this system time is dominated by routing time, there is little incentive to improve the time of the multiply-add circuitry.

We can also discuss the fact that Thompson's technique is very loose in some cases, especially when there can be a lot of "local" computation.

C. *Shuffle-Exchange Network*

In the mesh network discussed above, the communication time drastically overshadows the computation time, mostly due to the fact that each step of the FFT might require a multi-stage communication. Therefore, it makes sense to consider a network that requires only a unit-length communication each time period. The shuffle-exchange network is a topology that achieves this, and Thompson shows that the time it takes to compute an FFT on a shuffle-exchange network is $T = O((log\ N)t_R + (log\ N)t_M)$. Unfortunately, this increase in speed comes with an increase in area.

The upper-bound on time for the shuffle-exchange FFT is shown to be $T = O(log^2\ N)$, while the area degrades to $A = O(N^2/log^{1/2}\ N)$. This makes $AT^2 = O(N^2 log^{7/2}\ N)$. Interestingly, this circuit is also very close to the lower bound, being off by only $O(log^{3/2} N)$ from the lower-bound given above.

V. Related Topics

There are a few interesting related topics that spring directly from Thompson's work in VLSI. One such topic is the question of producing good lower-bounds on the area of the VLSI implementation of a given computational graph. This is a subtly different problem than the lower-bound found by Thompson. Using his method, a lower-bound is generated for a particular problem, but the only information used to set this bound is the minimum bisection width of the problem. It is not difficult to come up with computing graphs that have a small minimum bisection with, but still potentially large area. This topic is explored by F. Leighton in "New Lower Bound Techniques for VLSI," ([2]) which provides tight bounds for both planar graphs and a subset of non-planar graphs using Thompson's grid formalism.

Additionally, Thompson's work has been used to show complexity bounds for additional functions. For example, in "The VLSI Complexity of Boolean Functions," ([1]) M.R. Kramer and J. van Leeowen prove bounds on general Boolean functions of N variables. Specifically, they show that all Boolean functions of N variables can be computed by a VLSI circuit of area $O(2^n)$ area in time $O(1)$. Interestingly, they also show that there exist Boolean functions for which every VLSI chip implementing them must have $\Omega(2^n)$ area.

VI. Conclusion

By combing information theory with concepts from graph theory and circuit design, we are able to generate lower bounds on the area and time it takes to compute a function in hardware. For many functions, this lower-bound is quite tight and this has been demonstrated through the generation of close upper-bounds on the same problems. Through the generation of these upper and lower bounds, guidelines can develop for circuit designers to help them optimize their circuits. Additionally, a tight lower bound can indicate whether or not it is worthwhile to pursue a more area and time efficient solution. Another key impact of this work is providing a translation between the physical world of hardware design and the theoretical world of information theory, giving a framework through which hardware networks can be studied.

References

[1] M.R. Kramer and J. van Leeuwen. The vlsi complexity of boolean functions. In E. Brger, G. Hasenjaeger, and D. Rdding, editors, *Logic and Machines: Decision Problems and Complexity*, volume 171 of *Lecture Notes in Computer Science*, pages 397–407. Springer Berlin Heidelberg, 1984.

[2] Frank Thomson Leighton. New lower bound techniques for VLSI. In *Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on*, pages 1–12, Oct 1981.

[3] C. D. Thompson. A complexity theory for VLSI, 1980.