Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

1991

Communication architectures for single-chip data routers.

Gopal. Panneerselvam
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

NOTICE

AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

!f pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

COMMUNICATION ARCHITECTURES
FOR SINGLE-CHIP DATA ROUTERS

by

Gopal Panneerselvam

A Dissertation Submitted to the
Faculty of Graduate Studies and Research
through the Department of Electrical Engineering
in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada

1991

Canada

To my mother and my father.

ABSTRACT

This dissertation presents a novel architectural technique for systolic architectures for applications which traditionally use high wire organizations in VLSI.

Following a review of current VLSI research and VLSI models, this dissertation argues for a particular computational model (Chazelle's model) as being appropriate for today's VLSI and ULSI technology.

Systolic arrays are particularly suited for applications where only local interprocessor communication of data is required. In areas where non local data communication is predominant, the so called "high wire organizations" are traditionally used. Such networks include sorting arrays, interconnection arrays. Using Chazelle's model, an analysis of well known interconnection networks shows that "inefficient" systolic arrays, for routing and for sorting, outperform, so far as asymptotic performance metrics are concerned, high wire organizations traditionally used for such applications.

This dissertation then proposes a new systolic architecture using the novel design philosophy of locally long but globally short connections. This involves designing arrays using large, complex cells instead of fine grained cells. This is termed "systolic architectures using cells of controllable complexity" since the latency and/or pipeline period requirement of a user determines the size and hence the interconnection complexity of the cells in a systolic array of complex cells. It turns out that many important application areas (e.g., interconnection networks, sorting networks and FFT) are suitable candidates for this approach. This class of architectures is well suited for ULSI implementation.

An experiment in designing interconnection networks show that this concept of arrays using cells of controllable complexity is useful, even in current 1.2μ VLSI technology.

ACKNOWLEDGEMENT

TABLE OF CONTENTS

CHAPTER

1

INTRODUCTION

1.1 PREAMBLE

The continuing evolution of semiconductor technology, with respect to reduction of minimum feature size, enlargement of chip area and improvement in packing efficiency has made it feasible to put millions of transistors in a single chip. The industry is emerging from the Very Large Scale Integration (VLSI) era into the Ultra Large Scale Integration (ULSI) era (feature size < 1μm) [47][46][16]. Reismen's curve in Figure 1.1 depicts this dramatic development [61][16]. It is expected that 16M DRAM (0.5μ technology) will be commercially available in 1993 [16]. 64 MByte DRAMs are currently under development and will undoubtedly be in the market place by the middle of the decade. In view of these developments, it is important to look at the change in architectures that we wish to place on high density chips as we traverse the VLSI/ULSI technology boundary. The theme that will emerge from this thesis is that solutions which are expensive in terms of number of active elements but are more efficient in terms of layout area and speed of operation are the order of the day for applications that will be in the supercomputer class. We will identify such architectures with a class of systolic arrays [34][33][35][40][41] that we expect will blossom as the technology develops. Systolic arrays are well established for arithmetic computation, where only local inter processor data movements are required. In this thesis, we will show that systolic arrays are further applicable for data routing applications, where data moves rapidly, such as sorting or interconnection networks.

c
o
m
p
o
n
e
n
t
s

p
e
r

c
h
i
p

2^{28}

2^{24}

2^{20}

2^{16}

2^{12}

2^{8}

2^{4}

2^{26}

2^{21}

2^{16}

2^{11}

MSI

LSI

VLSI

ULSI

GLSI

6

25

16μ

4μ

1μ

0.25μ

m
i
n
i
m
u
m

d
i
m
e
n
s
i
o
n

72   81   89   98   2000

year
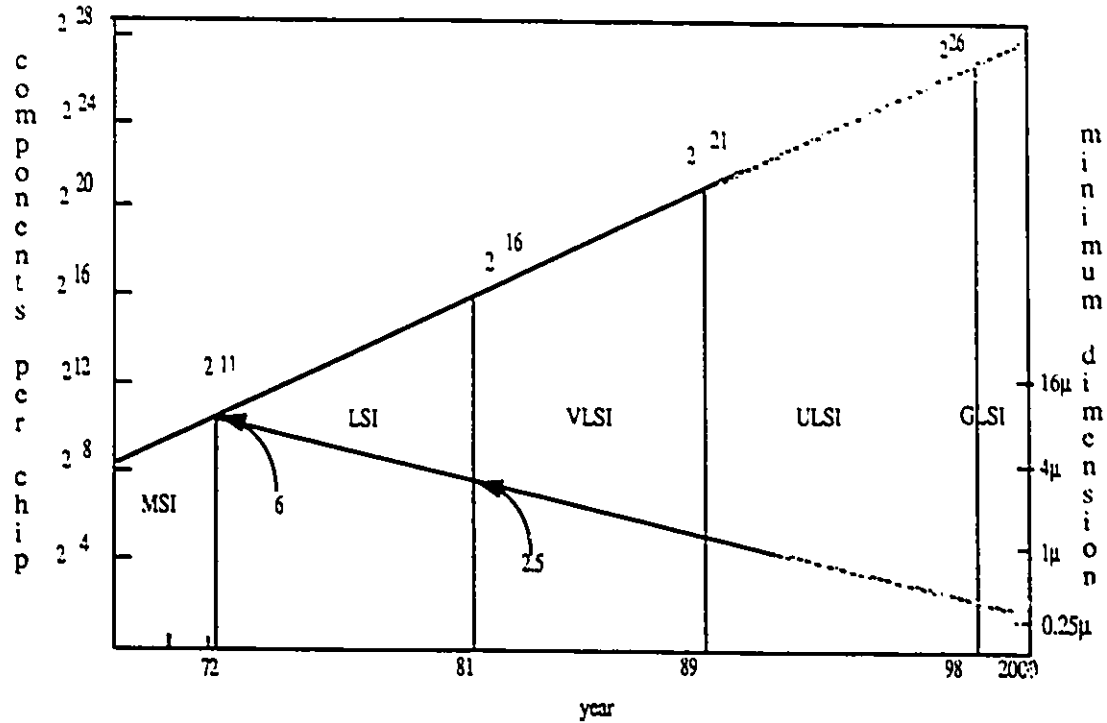
Figure 1.1 Reisman curve. Number of component density versus time.

1.2.1 COMPETING ARCHITECTURES IN THE VLSI /ULSI ENVIRONMENT

As the technology advances, it is important to examine the existing architectures and compare them from the technological point of view, as well as with the advancement of CAD tools. We can divide these architectures into two major categories using their communication structures, namely, locally connected systolic organizations and high wire organizations [82].

1.2.1 SYSTOLIC ARCHITECTURES

2

A systolic array is a set of interconnected cells, each capable of performing some simple operation [34][40]. Systolic architectures have been investigated over the last ten years for important classes of digital systems. Although their use has been somewhat limited, this is undoubtedly due to the mis-match of silicon technology available during the time that these architectures were developed. Our thesis is that certain classes of systolic arrays will become important as the technology provides greater integration density.

The main reason for our optimism is that systolic architectures provide an area efficient layout without communication overhead by exploiting the nearest neighbor connection on regular geometrical shaped processors [34][40][18]. Various circuits based on systolic architectures have been proposed for applications such as matrix computations, data base machines, digital signal processing and simulated annealing [9][35][36][48][55][77][75]. Systolic arrays have high throughput since they use extensive pipelining. The nearest neighbor connection is an important property of systolic arrays since complex interconnections are expensive in terms of silicon area in VLSI/ULSI designs.

It is convenient to test the electrical characteristics of a small number of cells thoroughly before designing a systolic array using a large number of such cells. Diagnosis of such structures is often quite straight forward [60] involving a constant number of test vectors independent of the size of the array. With this architectural simplicity, researchers have extensively investigated systolic architecture in VLSI design methodology. Many researchers recommend systolic architectures as a suitable candidate for future generation chips or integrated systems, viz., Wafer-Scale-Integration (WSI), 3-D chips [39][76].

1.2.2 HIGH WIRE ORGANIZATIONS

3

High wire organizations consist of processors or nodes connected using long communication links to achieve an optimal number of processors, with a minimum number of steps. Their ability to transmit large amounts of data across arbitrary boundaries quickly gives these organizations an essential advantage, for many important applications, over organizations like the rectangular grid. These high wire organizations are recommended for "supercomputers". The fast communication capabilities are essential for data routing applications such as interconnection networks, sorting, FFT computations, etc. The implementation aspects of high wire organizations in VLSI have been studied by many researchers [19,25,30,42,78-80,82], and their results have shown that the area required for interconnection dominates processor area.

In this dissertation we examine both high wire and systolic organizations for currently available technologies, and show that systolic organizations are not only suitable for arithmetic computation, which only require local data movement, but are also suitable for high data routing applications in the ULSI environment.

1.3 NEED FOR VLSI MODELS

In order to analyze the performance of different VLSI circuits, it is important to have an adequate model. Prior to the VLSI era, the complexity of a circuit was measured in terms of the gate or transistor count. In VLSI implementations, however, the area needed for interconnecting active devices became very important. In the so called 'high wire' organizations [82] this area is far higher than the area occupied by active devices. A number of models have been investigated in recent times [8][12][11][45][78][79]. These models allow us to estimate the asymptotic bounds on characteristics such as chip area, computational time, latency time and pipeline period. Such models are technology

independent. We have to remember that the asymptotic bounds on the performance predicted by these models will be particularly useful as we move into progressively denser ULSI circuits in the next few years where the number of cells in a chip will be at least a hundred times larger than they are today [20][16]. Asymptotic bounds show the relationship of the inputs on the parameters of area and time; however, the value of the asymptotic constants are hidden. In this thesis we propose a pragmatic design methodology for VLSI and ULSI technologies that includes the importance of these constants.

1.4 SCOPE OF THIS THESIS

In this thesis, we investigate applications of systolic arrays for routing applications. We explore novel non traditional applications of systolic arrays and show that, in terms of asymptotic bounds, simple systolic arrays have better asymptotic performance bounds than circuits based on more complex interconnections. Several such circuits have been proposed in the literature. Traditional designs have attempted to minimize the number of processing elements in digital systems. In the ULSI era, a complex digital subsystem consisting of numerous processing elements can be fabricated on a single chip. We show that the asymptotic bounds on the performance of such systems need to take the intercell propagation delays into account. Using an appropriate VLSI model, which does take this delay into account, we show that simple systolic realizations are potentially superior to solutions which attempt to minimize the number of processors using a high wire organization. However, minimizing the number of processors is also an important factor, which we considered when taking into account minimizing communication overhead. The resulting methodology yields a new class of area and time efficient systolic architectures.

5

In chapter 2, we briefly review key concepts in VLSI design that allow the determination of performance metric for VLSI circuits and argue for Chazelle's model [11] in the emerging ULSI era.

In chapter 3, we look at the applicability of systolic arrays in designing interconnection networks. In multiprocessor design, efficient data communication capabilities are extremely important and, in this context, we analyze a number of popular routing networks. We show that if the propagation delays are taken into account, the performance metric of networks which are currently popular are not attractive. We also examine a "cross bar switch" type network, that can be easily realized as a systolic array, and even though the number of 2X2 switches in such a network is large, the propagation delay between stages of the network is constant, independent of the size of the array. The performance metric of the systolic routing network are superior to routing networks which use high wire organizations. We follow this by a critical evaluation of the problem of self routing [50] in such networks. In a multiprocessor system, the routing network may have to change permutations quite frequently. Indeed, every successive communication from a given processor may specify different processors as destinations. This means that the time required to determine the switch settings in a routing network may be a very significant factor when determining the throughput rate of the network. A straight forward way to handle this problem is to use a self routing [50] scheme. A perimeter sorting network [14] is an obvious solution to achieve self routing. We show that a sorting network, based on the systolic version of Kautz' network [28], has a better performance metric compared to sorting networks requiring fewer processors and high wire organizations. The low latency time makes this *S-array* attractive in designing interconnection networks.

In chapter 4, we define a new concept of *locally long, globally short* interconnections in a systolic array; this concept is offered as a technique for handling massively parallel computational networks for ULSI densities. Using sorting as the main application area, we

6

have defined complex cells in which the *intracell* communication uses a high wire organization. The size of a cell is determined by the desired pipeline period and the fabrication technology. Using these cells as a basic building block, we then define systolic arrays for sorting where the *intercell* communication is based on local (nearest neighbors) connections alone. Using this approach it is possible to achieve a compromise between the complex interconnection of a high wire organization requiring relatively few cells and simple organization of a systolic architecture requiring more processors. We call this approach *'systolic architectures using cells of controllable complexity'* since the complexity of intracell communication, which has a direct bearing on the complexity of the cells constituting the array, may change from application to application. This complexity is determined by user requirements and the fabrication technology being used. Then we present another perimeter sorting algorithm which may be readily realized by a systolic network using 2X2 comparators. This network has the same asymptotic bounds as the S-array, so far as the area and time are concerned, but actually requires fewer cells. This approach is also amenable to the complex cell approach. We finally show that the concept of complex cells is useful in deriving globally short routing architectures; we demonstrate that this concept maps to computationally intensive routing networks using the Fast Fourier Transform (FFT) as an example.

Finally, in Chapter 5, we cement the new concepts introduced in this research work using empirical results obtained from a double metal CMOS 1.2 μ fabrication technology. We consider routing as the application area and design a systolic array and butterfly network using a layout scheme which is applicable to networks of any size. We develop a model for the area and time of a butterfly element of arbitrary size and determine that a network of such butterflies has a considerably smaller area compared to a systolic layout; however, the pipeline period of a butterfly degrades linearly with the size of the network. We use our model to examine different scenarios ranging from a very fast routing network to a network

with relatively low speed requirements. We observe that for different speed requirements, it is advantageous to use systolic arrays with cells of different sizes. This establishes the fact that, even for current technologies, our approach of systolic architectures using cells of controllable complexity offers advantages over conventional designs.

We conclude the thesis with a discussion of the findings of the research work in Chapter 6. Suggestions for future work are also offered.

CHAPTER

2

REVIEW OF VLSI/ULSI MODELS

2.1 INTRODUCTION

Since different VLSI circuits may have completely distinct characteristics, it was difficult, in the past, to come up with a standard way to compare circuits to solve a given problem. Another difficulty was to estimate how the complexity of the circuit changes if the size of the problem changes. A substantial amount of investigation into these problems has led to models of computation for VLSI. These theoretical models are essential for evaluating and comparing circuit performances and in establishing lower bounds on chip areas and computation times. In this chapter we review, briefly, how this modelling is performed and argue for a particular model which appears appropriate for ULSI technology. In subsequent chapters, we will use this model to compare existing VLSI and ULSI arrays.

The most important parameters in any VLSI computational model are the area of the circuit and its speed. The size of the circuit is extremely important even in printed circuit board design [73]; in VLSI fabrication, the cost of fabrication of a circuit is an exponential function of its area, A, [82]; the maximum die area for a single chip is on the order of $1 cm^2$. Silicon area consumed is more important than the transistor or gate count used in earlier performance metric. Thompson shows in his dissertation [79] that, in many circuits, the area of the chip is determined by the interconnection area rather than the area occupied by transistors. In other words, communication in VLSI is not free. Since

9

communication between transistors may require more silicon area than the area for the active elements themselves, it is important to consider the actual layout of active elements in the plane, along with their interconnections, when analyzing a VLSI circuit.

Two parameters are important in determining the speed of a circuit. The first parameter is the time of computation, T. We also use the term latency time, interchangeably with the term "time of computation", to describe T. One way of determining this parameter is by counting the number of elementary operations that are performed by the circuit when computing its output [78]. However, a more convenient way is to measure speed as the time between the application of the first input bit and the appearance of the last output bit [11]. We will follow the latter convention. The second parameter is the pipeline period, P. This is determined by the minimum time separating two input sets [83].

In this chapter we briefly review the graphical model for VLSI circuits, and speculate on how the trends in this decade will affect the design of next generation ULSI circuits. We review the notion of area in graph theoretic terms. To model time, we examine four popular models. We look at restrictions on I/O placement and use a conservative and practical method to place I/O pads for this investigation. We describe popular performance metric for VLSI circuits which are in the form AT, AT^2, AP and ATP and their physical significances. We have used all these metric in this thesis. Finally we look at the current trends in ULSI technology. We argue that, in view of ULSI trends, for high wire organizations, among the four models for time estimation that we have mentioned, the linear delay model for time is most applicable.

2.2 MODELLING A VLSI CIRCUIT

Pioneering work on models for VLSI computation was performed by Thompson [78][79], Brent R.P. and H.T. Kung [8], Mead [44][45], Vuillemin [83], Chazelle and Monier [12], [11], Lipton et al. [43]. Mead and Rem are among the first investigators who pointed out a need for a novel complexity theory, which involves functions of both area and time, so that, when estimating the area, the size of the processing element as well as wire area is to be included [45]. Mead and Conway also introduced VLSI design using λ, the elementary distance unit [44]. λ depends on technology and has steadily become smaller with time. Abelson was among the first to point out that communication between processors is a significant cost in the complexity of distributed systems [1].

We now describe, in some detail, the graph model we will use in this thesis. Any VLSI circuit can be modelled [79] by a directed planar graph where the nodes correspond to the active elements, e.g., transistors, or cluster of transistors, or gates (Thompson uses the term nexus [78] for nodes) and edges representing interconnections. The edges are used for transfer of information from one node to another node, to a power line or to clock generators. The computational graph is depicted as a grid of unit squares. This is the reason why the model is known as the *grid model*. Each side of unit square in the grid model represents manufacturing and physical limitations. Following the recommendations of Mead and Conway, this minimal spacing in the grid model is 4λ. This gives us the size, $16\lambda^2$, of a unit square. One unit square is just large enough to contain one transistor or one wire cross over. Wires are laid out on a grid with unit spacing. If two metal layers are available, one layer is devoted to the x direction and the other to the y direction as described in [44]. Wires meet only at a node, which are represented as squares. If d wires connect to a node, then the node is a square with side d. At most two wires may cross each other at any point in a plane. Wire may not cross over nodes, nor can nodes cross over nodes. A grid model realization is illustrated in Figure 2.1 [82].

11
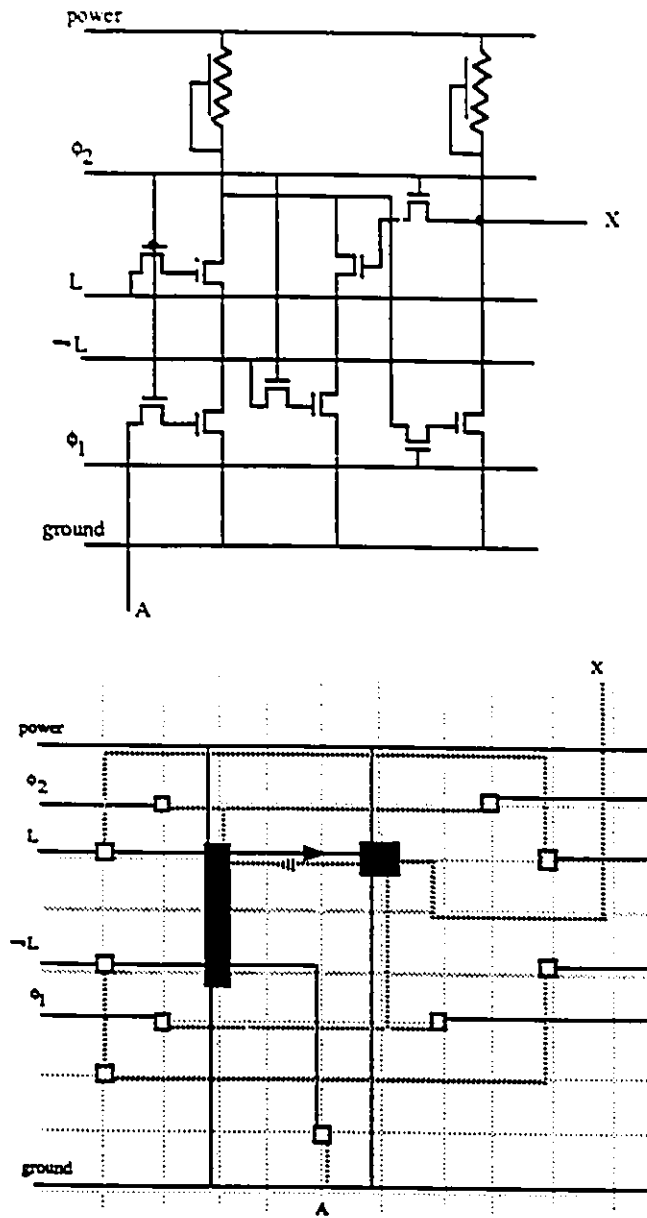
Figure 2.1 A simple register cell and its grid model representation

VLSI circuits are implemented as convex planar graphs, because of heat-dissipation and packing requirement [8]. This is a convention that we will follow in our systolic arrays.

12

2.3 TIME

A unit of time is the minimum time in which an event occurs (e.g., the time taken for a transistor to switch). In a practical circuit, for instance, using a two phase clock, the unit time is between the leading edges of the clocks driving each of the two phases [82].

Models differ radically in the mechanism they use to compute the time of computation T. There are four models that have been proposed for estimating communication time in a VLSI circuit as discussed below:

i) *Constant time model*: In this model, a bit requires constant time τ to propagate along a wire regardless of the length of the wire [8]. This has also been called the synchronous model.

ii) *Logarithmic time model*: In this model, a bit requires a time O(log l) time to propagate along a wire length l [44] and [80]. This assumes that there are log l stages of drivers to amplify the signal. Larger drivers, of course, occupy more area, but never take more than 10% of the wire they drive [79], [80].

iii) *Capacitive model*: In this model a bit requires O(l) time to propagate along a wire of length l [7].

iv) *Diffusion model*: In this model a bit requires a time O(l^2) to propagate along a wire of length l [11][68].

The ultimate justification for the linear model comes from a speed-of-light argument. No information can propagate faster than light. Moreover, in practice, parasitic effects reduce the speed of propagation several orders of magnitude below that limit and this makes the linear dependency important [11].

Based purely on asymptotic considerations, the diffusion model appears most realistic since the resistance, R, and the capacitance, C, of a wire of length l are both proportional to l. Thus the RC delay is proportional to l^2. Thompson as well as Mead and Conway indicate that the propagation delays may be made independent of the wire length by fitting larger drivers to longer wires. This is applicable for short wires typical of VLSI circuits fabricated in the early eighties. Chazelle suggested that it may be necessary to decompose long wires into wires of constant length connected by delay (computing the identity function). This gives us the linear delay model. A careful analysis of physical phenomena occurring in VLSI circuits appears in [7]. This reference considers the synchronous, capacitive and diffusion model, and defines a figure of merit $\gamma = \frac{cl}{C_0}$ where c is the capacitance of wire per unit length, l is the length of wire and C_0 is the capacitance of the active device. They conclude that

- for small γ, ($\gamma < 1$), the propagation delay is constant (synchronous model)

- for moderate γ, ($1 \leq \gamma \leq 1000$), the propagation delay is linear (capacitive model)

- for large γ, ($1000 \backsim \gamma$), the propagation delay is nonlinear (diffusion model)

In terms of the technology of the early eighties, the synchronous model was quite valid. The trend, however, is definitely to, at least, moderate values of γ and we will see, in section 2.8, Chazelle's assumption, that the time of propagation along the wire is at least proportional to the length of the wire, appears to be accurate in the foreseeable future. In our analysis we will use the linear delay model.

2.4 AREA

The lower bound on the total area, A, of the embedded communication graph occupied on a grid is equal to the number of unit squares occupied by wires or nodes. The number of squares in the smallest bounding rectangle is represented as the upper bound [79]. Baudet [6], Krishnan [32] and Kurdahi et al. [37] proposed different approaches for measurements of area; however we follow Thompson's method.
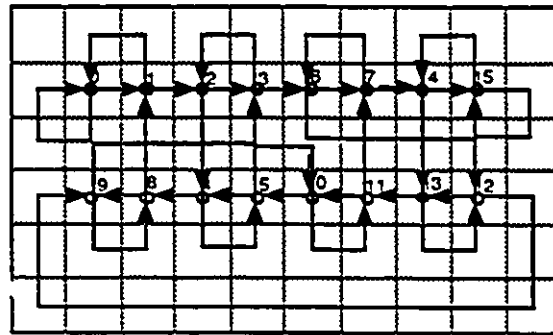
Figure 2.2   Thompson layout for n=16 Shuffle Exchange graph.

As an example, Fig 2.2 shows the communication graph for a shuffle exchange network. The lower bound here is 58 units and the upper bound is 60 units.

2.5 I/O CONSIDERATIONS

Placement of I/O pads and the possibility of replication of input pads have some importance in VLSI complexity analysis. Savage [66][67], assumed that each input variable is supplied exactly once to the chip and each input enters at one place on a chip. Kedem and Zorat [29] showed that relaxation of Savage's restrictions (i.e., input replications both in

space and in time) may be quite profitable. Baudet [6] assumed that both inputs and outputs are data independent, and input variables are read only once. Memory based storage structures are proposed by Wong and Kung [84]. Other researchers have imposed limitations on I/O port size, minimal area I/O port, fixed area I/O port and entropy constraints for internal data communication [8][11][83][86].

In our discussions, we have not looked at I/O considerations in great detail. In our designs, we will ensure that all data input to (or output from) an array is along the perimeter of the chip. This avoids the problem of considering the additional overhead of routing the input (output) lines inside the array itself.

2.6 PERFORMANCE METRIC FOR VLSI

As explained earlier, the performance from VLSI models is measured by using area, A, and time, T, or some combination of these two [45]. Thompson proved his lower bounds on the performance of a VLSI circuit by considering the communication graph for a VLSI array. He defined a *minimum bisection width* 'ω' of the communication graph as the number of edges that need to be removed to disconnect one half of the vertices from the other half. The area of the graph is proved to be at least proportional to the square of its bisection width since Thompson showed that $A \geq \omega^2/4$. The time taken by the communication graph (to solve a n-point DFT problem or a sorting problem) is at least inversely proportional to its width, $(T = \Omega(n \log n)/\omega)$.

Different researchers have proposed different performance metric based on the area, A, and time, T. The most common metric is the area time product AT. Another bound AT^2 has also been used, and many 'strong' lower bounds, that do match the best circuit we can construct, are lower bounds on the product AT^2 [82]. This bound is based not on memory

requirement or input/output rate, but on the requirements for information flow within the chip [82]. One convenient way to characterize all metrics used in the literature is to use the general form AT^x, for some x, $0 \leq x \leq 2$. The time taken by the communication graph (to solve the n-point DFT or sorting problem) is at least inversely proportional to its width, ($T = \Omega(n \log n)/\omega$). The minimum value of a performance metric having the form AT^x occurs, when $\omega = \theta(n^{1/2})$. This leads to a general lower bound [79].

$$AT^x = \Omega(n^{1+x/2} \log^x n), \text{ for } 0 \leq x \leq 2 \tag{2.1}$$

For pipelined systems, Chazelle et al. introduced the ATP bound where P is the pipeline period. For many applications (e.g., digital signal processing), latency time is not very important. What is important is to have a very high throughput rate. A good circuit, in such applications should have a low area, A, and a low pipeline period, P. For such applications, the metric, AP, is an appropriate lower bound [6].

In our investigations, we will use the metrics AT, AT^2, ATP and AP to analyze the complexity of VLSI circuits.

2.7 EFFECT OF SCALING IN VLSI/ULSI

Miniaturization of MOS transistor dimensions continues to improve circuit speed and packing density. Ideal scaling theory was one of the first approaches to model the shrinking of MOS transistors [15]. When a MOS device is scaled in five dimensions -- the three physical dimensions, voltage level, and doping concentration, it leads to greater speed and density as well as lower power consumption. Scaling improves circuit performance by reducing capacitances and voltage swings and, at the same time, ensures the physical integrity of devices by keeping the electrical field constant

17

[3][16][20][24][44][47][46][71]. Voltage scaling has not yet been a major factor due to commercial considerations [3].

An MOS transistor is shown in Fig 2.3.
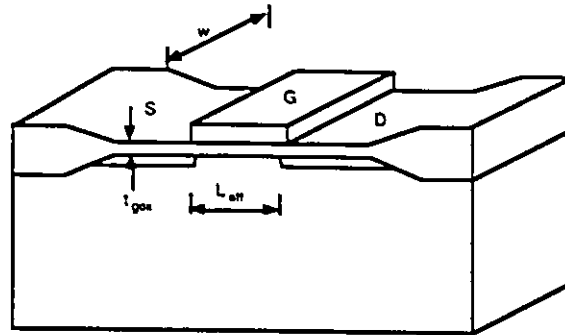
Figure 2. 3 A MOS transistor.

Table 2.1 [3] shows the effects of scaling on such devices. The table shows that, with scaling, devices get faster, power dissipation and power-delay products of the devices are reduced, packing density improves, and power dissipation density remains constant. Fast chips that integrate a large number of transistors become possible and the VLSI era has now given way to the ULSI era with continuing improvements in scaling [47][46]. Interconnection issues are the main factors that determine the number of elements in the circuit that can be integrated on a chip with a given chip performance [3][20][2][22][23][24][51][62] [63][64][65][71][87]. This is true for CMOS, bipolar, and gallium arsenide (GaAs) technologies [3].

Table 2.1 Scaling effects on Transistor

Parameter	Scaling factor S
Dimension (W, L, t_{gox}, X_j)	$1/S$
Substrate doping (N_{SUB})	S
Voltage (V_{DD}, V_{TN}, V_{TP})	$1/S$
Current per device $(I_{DS} \alpha W \varepsilon_{ox}/L \; t_{gox} (V_{DD} - V_T)^2)$	$1/S$
Gate Capacitance $(Cg = \varepsilon_{ox} WL/t_{gox})$	$1/S$
Transistor on-resistance	1
Intrinsic gate delay $(t = Cg \Delta v/I_{av} = R_{tr} C_g)$	$1/S$
Power-dissipation per gate $(P=IV)$	$1/S^2$
Power-delay product per gate $(P \ast t)$	$1/S^3$
Area per device $(A=WL)$	$1/S^2$
Power-dissipation density (P/A)	1

Scaling effects occur on both interconnection capacitance and resistance. With increasing chip dimensions, parasitic interconnection capacitance dominates the gate capacitance, and the speed improvement expected from simple scaling does not apply to circuits that drive global communication lines. Simple scaling assumes a reduction in the capacitive loading due to wires. This is true, locally, when a circuit is connected only to its neighbors, but for circuits that drive long global wires, the capacitive loading actually increases because chips get bigger with time. Large capacitive loads also increase power consumption.

Table 2.2. Scaling effects on interconnection

Parameter	Scaling factor
Cross sectional dimension ($W_{int}, H_{int}, W_{sp}, t_{ox}$)	$1/S$
Resistance per unit length ($R_{int} = r_{int}/W_{int}H_{int}$)	S^2
Capacitance per unit length($C_{int} = \epsilon_{ox}W_{int}/t_{ox}$)	1
RC constant per unit length ($R_{int} C_{int}$)	S^2
Local interconnection length (l_{loc})	$1/S$
Local interconnection RC delay($R_{int} C_{int} l^2_{loc}$)	1
Die size (D_c)	S_c
Global interconnection length (l_{int})	S_c
Global interconnection RC delay($R_{int} C_{int} l^2_{int}$)	$S^2 S_c^2$
Transmission line of flight (l_{int}/v_c)	S_c

S, S_c: Scaling factor for device dimension and chip size respectively

In addition to large capacitance loads resulting from long interconnections, the resistance of the lines also becomes a major concern. Simple scaling of local and global interconnections is summarized in Table 2.2 which uses interconnection parameters shown in Fig 2.4.
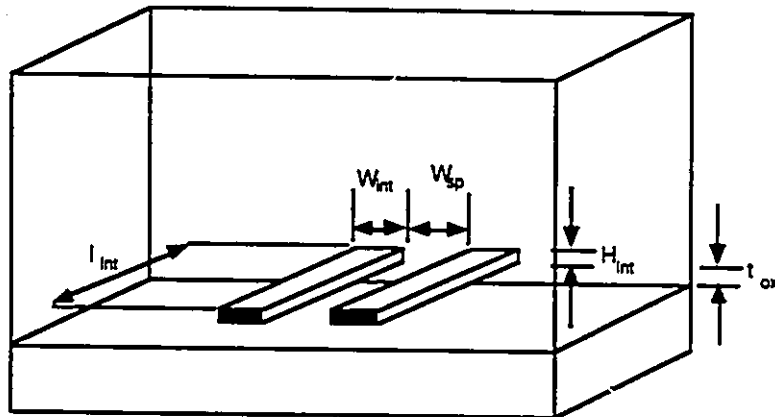
Figure 2.4   Interconnection parameters.

In addition to scaling down the size of individual devices, chip size has increased continuously throughout the evolution of ICs. This can be modelled as a chip scaling factor S_c, which represents the increase in the length of the one side of a die. As a result, global interconnection length, l_{int}, increases by S_c, and the distributed RC delay of long lines degrades by $S^2 S_c^2$.

When the width of the wire is increased, resistance drops, but the capacitance increases by the same amount. Consequently, the RC constant cannot be reduced by adjusting mask dimensions and is solely determined by the technology [4][3]. This is a very important factor in determining the limitations in using long wires in a VLSI circuit and future improvements in S and S_c will mean rapid increase in the interconnection delay.

2.8 A MODEL FOR ULSI TECHNOLOGY

As shown in Table 2.1 and 2.2, scaling reduces not only the dimensions of a device but also increases its speed. On the other hand, for interconnections, the global interconnection RC constant per unit length is increased by $S^2 S_c^2$. Thus, with the evolution of technology, device delay is decreasing but interconnection delay is increasing; the wire delay controls the speed of the computations rather than the device delay [3]. In a ULSI environment, long wires pay double penalty - both silicon area and propagation delay increase. The computational model we choose for ULSI has to reflect these scaling effects. An important paper in this area is by Zhou et al [87].

The fact that the dimensions for interconnections do not scale is important in wire delay assumption. For example in 3μ and 1.2μ technologies, the metal 1 widths (3 and 2.0 microns, respectively, obtained from Canadian Microelectronics Corporation design rules) are not even scaled by a factor of 2 as would be expected using simple scaling assumptions

21

[10]. In sub micron technologies, a minimum size transistor of 0.1μ device width can not directly drive a metal width of 1μ; this limits the driving capability of devices and also does not utilize the maximum scaling effect. Thus, as the technology evolves, the chip area is further dominated by wire area and propagation time by the wire delay.

We have mentioned earlier that Bilardi et al [7] defined a figure of merit $\gamma = \frac{cl}{C_0}$ where c is the capacitance of wire per unit length, l is the length of wire and C_0 is the capacitance of the active device. As the technology advances, the capacitance per unit length of the wires in a multilevel interconnection scheme approaches a lower limit of 2 pF/cm with SiO_2 as the dielectric material [71]. In practice approximately 3 pF/cm is achieved, whereas a minimum size gate capacitance is 2×10^{-3}pF [4]. This yields a high value for γ, in the range of 10^3, as shown in Bilardi et al. for a minimum wire length of 1 cm. For such value of γ, the wire delay of Chazelle et al. and Seitz's model is expected to be predominant [7]. Therefore, we will use Chazelle's linear model for calculations of wire delay. In fact, for values of $\gamma > 10^3$, the nonlinear effect starts to become important. We are actually being conservative when we assume the capacitive model. If we assume the diffusion effect, where the delay is proportional to the square of wire length, the effect of wire delays degrade the performances of VLSI circuits even more drastically.

Apart from the fact that ULSI circuits should have a low value for performance metric (e.g., AT^2, ATP) the regularity of interconnection in a ULSI array is very desirable from a technological point of view. As pointed out by Sasaki, the performance measure for ULSI technologies should have: function per unit area, throughput (or performance) per function, and the regularity factor of the circuit itself [65]. Systolic arrays are ideally suited for ULSI technologies, so far as the regularity factor is concerned, and in chapters 3 and 4 we show that they also have attractive performance metrics.

We summarize that in modelling VLSI/ULSI circuits, the grid model is applicable, the performance metric AT, AT^2, ATP and AP are useful and it is appropriate to use the capacitive model to estimate time.

2.9 SUMMARY

In this chapter we have introduced the subject of modelling for VLSI/ULSI circuits. An extensive review has been performed on models that have previously been proposed to predict area and time performance limits for large systems on silicon. As technology improves, devices scale, and the effect of such scaling on performance metric has also been explored. We have determined that the capacitive model is most appropriate for computing time metric, and that the grid model most suitable for computing area metric. We will use a variety of cost functions, involving both of these metric, in our following comparison studies.

CHAPTER

3

ULSI SYSTOLIC ARRAYS FOR ROUTING AND FAST COMPUTATIONS

3.1 INTRODUCTION

In a supercomputer there are many processors and many memory modules. During any computation, the processors have to communicate data to other processors and to different memory modules. It is very important that this communication be carried out as fast as possible since this is often the bottleneck in determining the throughput capabilities of a multiprocessor system. Interconnection networks are used to communicate signals from n sources (processors) to m destinations (processors and/or memories). A general model for an interconnection network for communication between processors and memory modules is shown in Figure 3.1. Typically, an interconnection network consists of several stages, where each stage consists of a number of 2x2 switches.
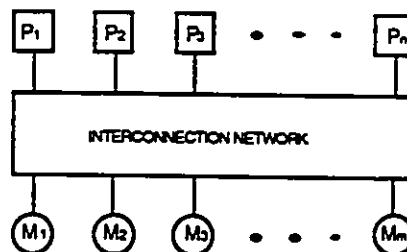
Figure 3.1. Concurrent processing systems.

When a processor wishes to communicate with another processor or memory, the communication time varies, depending on which processor is the source for this communication and which is the destination processor. In designing an interconnection network, a very important goal is to reduce the worst case communication time for any source/destination pair. In pipelined systems, the pipeline period is often more important than the latency time. In such cases, we typically connect latches to every switch in order to reduce the pipeline period.

Implementing such interconnection networks in VLSI has been a basic problem for some time and has been studied thoroughly by many researchers [19][25][30][42][78][82]. The communication architectures of most popular interconnection networks are referred to as 'high wire organizations' [82], since a significant amount of communication layout area between stages is needed for such networks.

In ULSI, these high wire organizations not only pay penalties in terms of area but also in terms of propagation delay, since long wires, as explained in chapter II, imply significant communication delays. Evolving ULSI technology permits us to design larger routing networks (i.e. a network which can handle more inputs) on a single monolithic piece of silicon. In this chapter we will prove that, based on the capacitive model chosen in chapter II, high wire organizations are no longer likely to be efficient. Further, we will show that a straight forward "cross bar" type routing network, proposed many years earlier, actually has asymptotic performance measures superior to those of routing networks requiring fewer switches.

In this chapter we also analyze the performance of a number of standard routing networks using the capacitive model. Then we present a "systolicized" version of an interconnection network which has been proposed by Kautz in 1968 [28]. We compare the performance metric of this network to those of the standard networks found in recent literature. In

25

Chapter I, we briefly reviewed the problem of self routing. In this chapter, we look at a simple sorting array which has the same intercell communication architecture as the Kautz array. This array can be readily used for self routing and has better performance, compared to other arrays for perimeter sorting, once we take the inter cell delay into account using the capacitive model.

3.2 ANALYSIS OF SOME INTERCONNECTION NETWORKS

Interconnection networks are classified either by the number of stages (i.e., *single stage* also called a *recirculating* network or *multistage* networks) or by their function (i.e., *blocking, rearrangeable* or *nonblocking* networks) [17][26].

In blocking networks, simultaneous connections of more than one path may result in conflicts in the use of network communication links. An example of a blocking network is the perfect shuffle network. A network is rearrangeable if it can perform all possible connections between inputs and outputs by rearranging its existing connections so that a connection path for a new I/O pair can always be established. An example of the rearrangeable network is the Benes network. A network that can handle all possible connections without blocking is called a nonblocking network. The Clos network is a nonblocking network.

A number of well known interconnection networks are shown in Fig 3.2. These interconnection networks have been described in [13][17][21][26][38][52][53][56][57][58][59][70][72][81][85]. Every network that we consider has n inputs and n outputs, where $n=2^k$, for some k. Each node in an interconnection network consists of a 2X2 switch and buffers to store the output of this switch.
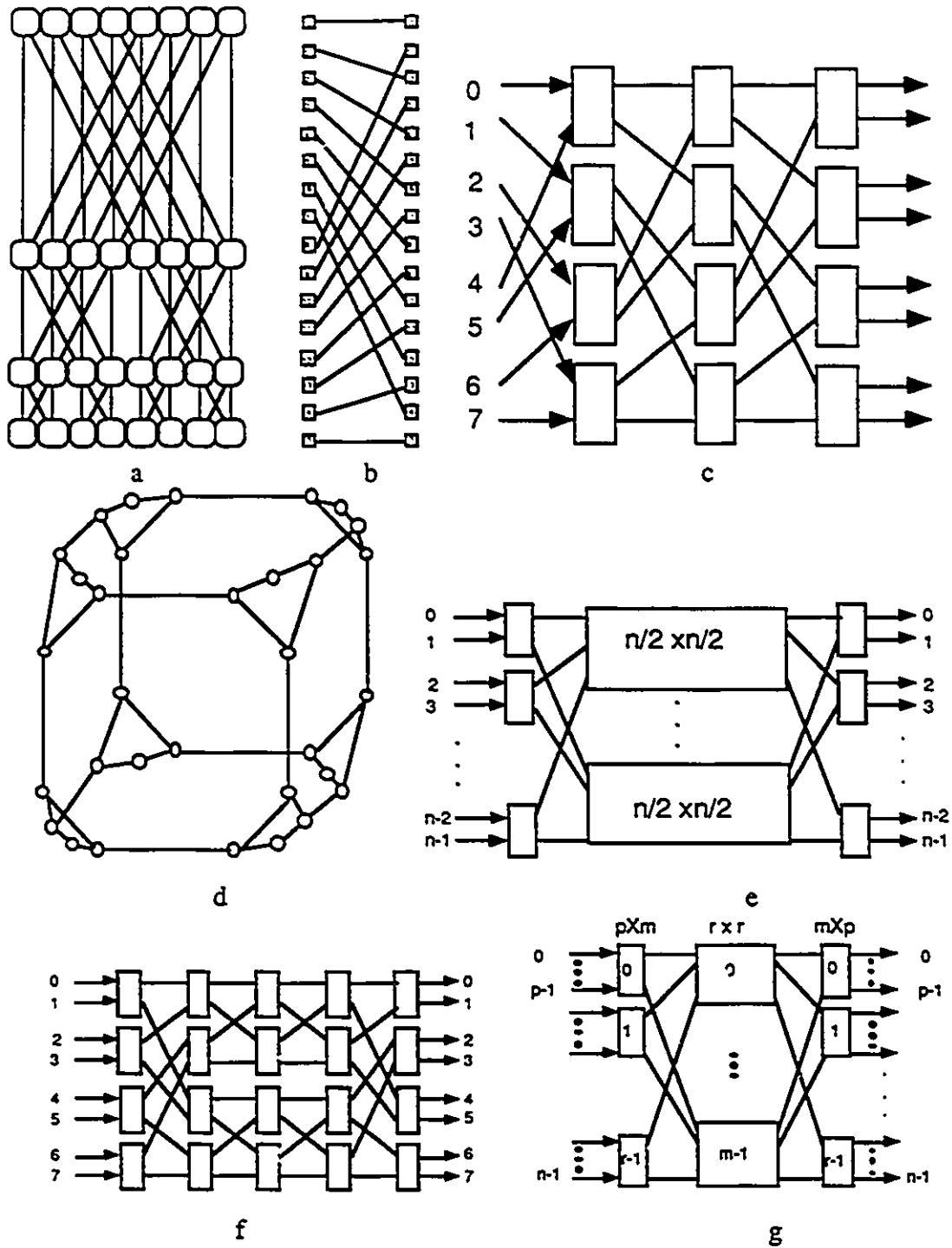
Figure 3.2   a. Butterfly network for n=8, b.Shuffle Exchange for n=16, c. Recirculating Shuffle exchange network for n=8, d. Cube connected network for n=32, e. Benes network, f. Benes network for n=8 & g. Clos network.

This ensures that these networks are fully pipelined so that, with every clock pulse, one set of data may be applied to the input of the interconnection network. After the requisite latency period, determined by the number of stages of cells each input bit has to pass through and the delay per stage, the output corresponding to a given set of inputs is available.

The delay D in a given stage is determined by the delay of the switch and the delay due to communication from the output of that stage to the input of the next stage. We assume that the system clock we use generates clock pulses at a frequency such that the time period is greater than the worst value of D. Since the worst value D_{max} of D is determined by the largest wire length, our pipeline period is taken to be the inverse of D_{max}.

In order to realize any given permutation in a rearrangeable network, the setting of each individual 2X2 switch in the network has to be determined externally. As mentioned earlier, determining the switch settings can be a complex problem in its own right. At this point we assume that a separate piece of hardware or software has correctly set the network switches for the desired permutation. When we speak of the latency period T of the network we assume that the switches are already set correctly and the time for setting the switches are not included in estimating T.

Pioneering studies of VLSI circuits and their performance metric took place in late seventies and early eighties. At that point in time, the state of the art in VLSI circuits was such that the synchronous model [8], where the time required for data communication is constant, was quite appropriate. This model is reviewed in Chapter 2 (Section 2.2). We recall that the present state of the art in VLSI/ULSI technology means that much smaller circuit elements are now technologically feasible and that the penalty due to long wires have become more pronounced. This means that the computation of VLSI/ULSI metric should be based on the capacitive model so that the worst case communication delay is

28

proportional to the maximum length of wire between active areas (Section 2.8). We now look at the ramifications of the capacitive model on the performance metric of several well known interconnection networks, namely Butterfly, Shuffle Exchange, Clos, Benes and Cube Connected Cycles [19][80][82][59][42]. The interesting point about this analysis is that the asymptotic performance metric of well known circuits, in terms of the VLSI/ULSI model we have chosen, have degraded significantly.

Table 3.1: Area and time complexities of interconnection networks.

Topology	Space/area Complexity			Time complexity		
	Gate model	Synchronous model	Capacitive model	Gate model	Synchronous model	Capacitive model
Butterfly	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Shuffle exchange	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
CCC	n	$O(n^2/\log^2 n)$	$O(n^2/\log^2 n)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Benes network	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Clos Network	$O(n^{1.5})$	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$	$O(n)$

In Table 3.1 we present the area and time complexities for the interconnection networks shown in Fig 3.2. In this table we use three of the models mentioned in chapter 2. The first is the gate count model where the number of logic gates and storage devices is taken to be a measure of the area complexity of the circuit. The second model is the synchronous model where the delay in communication is taken to be independent of the wire length. The third model is the capacitive model where the delay in propagation is linearly proportional

to the length of the wire. The reason for changes in the VLSI area/time complexities when we go from one model to another is that each model makes different cost assumptions.

In the gate model, the area of the active device is the factor which determines cost. On the other hand, in the synchronous model, the actual silicon area to fabricate the circuit determines cost. This area includes both areas for the active devices as well as the areas for the interconnections between active areas. The butterfly network, the shuffle exchange network and the Benes network require $O(n \log n)$ space complexity and $O(\log n)$ time complexity in the gate model. The VLSI implementation considerations change the area complexity of these networks from $O(n \log n)$ to $O(n^2)$ in the synchronous model. The time has not changed since this model ignores the wire delay. As explained in Chapter 2, we are now approaching the situation where this delay cannot be ignored for high wire organizations. The degradation of the time complexity, when this delay is taken into consideration, is very important and has been shown in the last column of table 3.1.

We will formally prove the correctness of the complexity values in the entries of Table 3.1 using Theorem 3.1.

Theorem 3.1:

If the capacitive model for VLSI is used, the areas (A), times (T), pipelined periods (P) and the performance metric (AT, ATP, AT^2) of the networks shown in Fig 3.2 are as shown in Tables 3.1 and 3.2.

Proof:

We will consider only the case of the Butterfly network here since the analyses for other networks are quite similar. We will use design constrains of the type described in [44]. Let us consider the geometrical structure of the butterfly network. We assume that λ is the minimum feature size allowed by the technology. We will measure all lengths in terms of

30

λ. Let us assume that, all processing elements or switches are squares which occupy at least LxL area, i.e., $O(L^2)$, where L is a side of the switch. Each side of the switch has a maximum of ω inputs or outputs (for this network ω is 2). Connections laid out in metalization layers need a minimum line width of $i\lambda$ (3λ for Mead and Conway design rules in 3μ technology) and a minimum distance between adjacent lines (minimum pitch) of $k\lambda$ units, where i and k are suitably chosen numbers ($k=3\mu$ for Mead and Conway design rules). A data path containing ω parallel communication lines requires a width of $(i+k)\lambda\omega$ units. Each cell has two inputs and outputs. Therefore, L is at least $\omega(i+k)\lambda$ units and a lower bound on the processor area is $((i+k)\lambda\omega)^2$.

By using Mead and Conway design rules, area-efficient layouts of Butterfly networks for n = 2, 4 and 8 are shown in Figure 3.3. As expected, the area for interconnection between switches increases exponentially as n increases (Figure 3.3 b).

We will use $P_{r,q}$ to denote the qth switch in the rth stage where the first and last switches in any stage are switches 0 and n-1. Let us consider the last stage s of an n input butterfly network, where $s = \log n$ (and we assume that n is a power of 2). The output of the switch $P_{s-1,j}$ has to be connected to switches $P_{s,j}$ and $P_{s,n/2+j}$ switch, for all j, $0 \le j < n$. We will term the connection from $P_{s-1,j}$ to $P_{s,j}$ ($P_{s,n/2+j}$) as the first(second) connection. Even though, the first connection from switch $P_{s-1,j}$ connects the output of $P_{s-1,j}$ to a cell in the same row, the length of the wire is determined by the other wires. The second connection passes through n/2 switches in vertical direction and in horizontal direction it has to pass through n-1 other wires. Therefore, the minimum length of this wire (in using λ as the unit) is

$$1/2 \ (L+k)n + (i+k)\omega \ (n-1)$$

The length of the array is $(L+k)n$ and the width is $(i+k)\omega \ (n-1)$. Thus the interconnection area of this stage is

$(L + k) n * (i + k)\omega (n-1) \approx O(n^2)$.

We are not considering the area of the cell since this will add a linear term to the total area of this stage. The total wire area of the network is:

$$\sum_{i=1}^{(n)} O(n^2)$$

Therefore the area is $O(n^2)$. The longest wire length is

$(n/2) (L + k) + (i + k)\omega (n-1) \approx O(n)$.

The pipeline period is determined by the maximum delay requires for the longest time of the network. Thus the pipeline period is $O(n)$.

❑
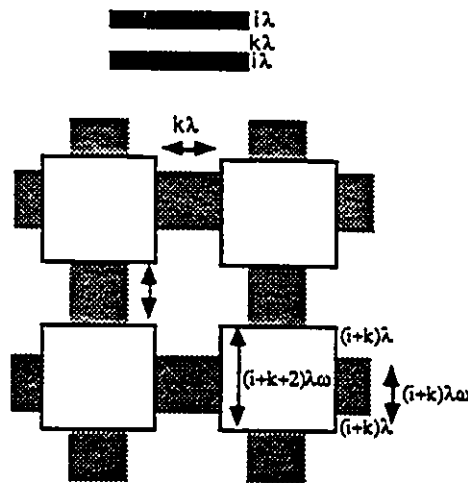
Figure 3.3 a. The layout for a mesh interconnection, in which the width between two switches has a fixed distance, each bus consists of ω wires of width (i+k)λ.
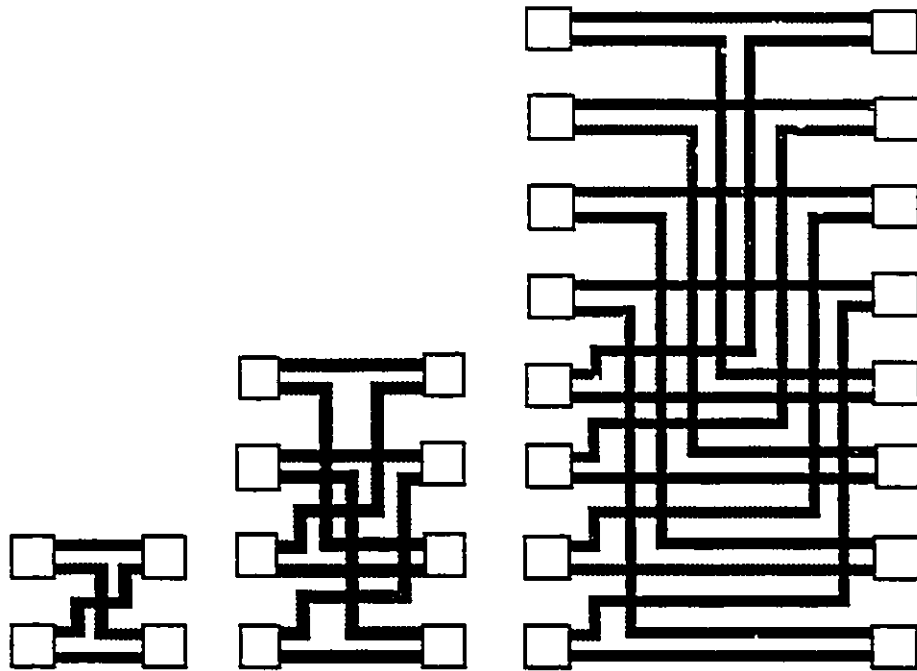
Figure 3.3 b : Layouts for the modules n=2, 4 and 8 points butterfly networks, the distance between two consecutive stages depends upon the number of inputs of the module.

3.3 A SYSTOLIC ARRAY FOR ROUTING

A cellular interconnection array was suggested by Kautz et al. [28]. In this part of the chapter, we study a very similar array which is obtained by a minor modification of the Kautz array. The "systolicized" modification of Kautz's interconnection array that we study is shown in Fig 3.4 where each 2X2 switch in a cell has been augmented by two buffers to store the two outputs of the switch. This ensures that the array is completely pipelined for maximum throughput. The setting of the switch in a cell determines whether the cell is in a 'crossing' or in a 'bending' mode. These two modes of a cell are depicted in Fig 3.4. Each shaded oval box in the diagram represents a single delay element. These

33

delay elements ensure the integrity of the wave-front at any instance of time. Kautz has suggested other realizations of interconnection arrays but we have omitted those designs since they all have identical asymptotic values of area time metric. In our discussions, we will refer to this array as the K-array. In order to realize any given permutation, the setting of each individual 2X2 switch has to be determined externally, as we did in the case of the high wire interconnection networks.

We now determine the asymptotic bounds for the area, latency time and pipeline period of this array in order to compare different performance metric of the K-array with those of interconnection networks, with high wire organization, that were discussed in Theorem 3.1.

Theorem 3.2:

The K-array with 'n' inputs requires

i. Area (A) of the network is $O(n^2)$

ii. Time (T) required for routing is $O(n)$

iii. The pipelined period (P) of this network is $O(1)$.

Proof :

The array contains $(n^2 - n)/2$ cells and the area per cell is constant. Each cell is connected to neighboring cells alone, therefore the area for interconnections per cell is constant. The area of the array is $(n^2 - n) A_{cell}/2$ where A_{cell} is the area of a single cell along with the area for its interconnections to neighboring cells. Each cell has a switch which can be set into a crossing or bending operation in a constant time $O(1)$. The array requires n clock cycles to produce the output corresponding to a given set of inputs. Each cell is connected only to four of its nearest neighbors and hence uses a constant wire length for communication. Thus the pipeline period is $O(1)$.

□

34
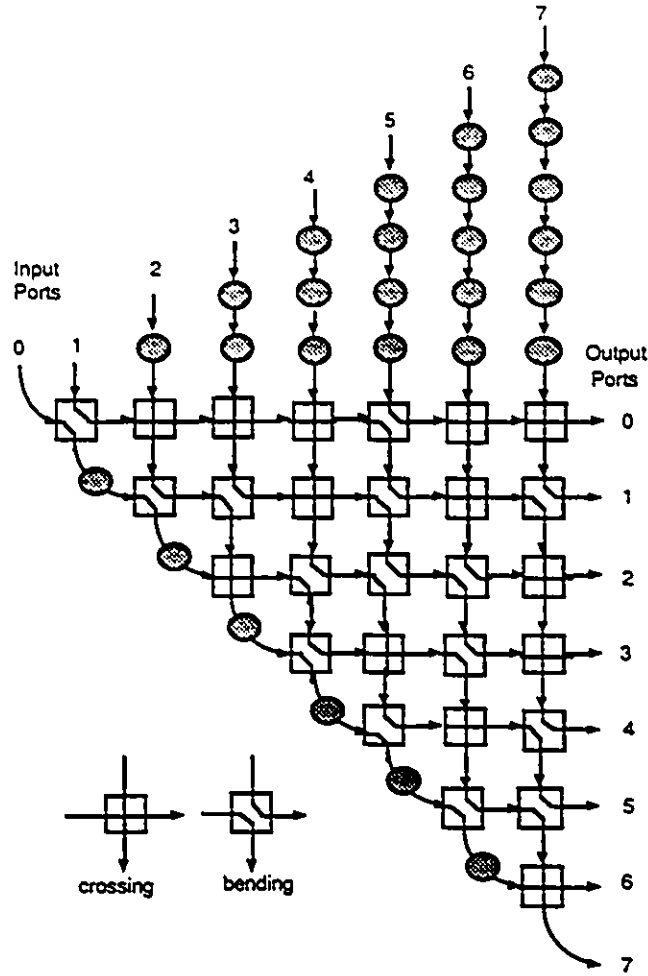
Figure 3.4. Triangular array for permutation $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 4 & 6 & 3 & 5 & 0 & 2 & 1 \end{pmatrix}$.

Table 3.2 (see the end of this chapter) shows the performance metric AT, AT^2, AP and APT of the butterfly network, the shuffle exchange network, the cube connected cycle, the Benes network and the Clos network together with the performance of the K-array. It is interesting to note that the cube connected cycle (CCC) is a very efficient structure when we use the metric of AT. The CCC, in a sense, tries to compromise between a locally connected structure and a high wire organization. As a result, the VLSI area for the CCC is very low. As pointed out in [35], the evaluation of a systolic array is quite dependent on what the user wants.

It is important to note that the k-array outperforms the others in the metric AP and APT. The Clos network also provides low values in the metrics AT, AT^2, but higher values in the AP and APT metrics.

3.4 SORTING NETWORKS FOR SELF ROUTING

A major problem in using a nonblocking interconnection network is to determine the switch settings for arbitrary permutations. It has been shown [50] that this is non trivial and takes significant time if we use a single processor. It is also shown that an SIMD machine may be used to determine the switch settings. In a mesh connected machine with $\sqrt{N}$ X $\sqrt{N}$ processors, it is possible to determine the switch settings in time $O(\sqrt{N})$. To avoid the additional overhead associated with this computation, "self routing" MINs have been suggested [50]. It is important to note that self setting MINs suggested in the literature only allow specific, rather than general, routing.

As mentioned above, we will discuss a straight forward method for implementing nonblocking interconnection networks using sorting networks. This approach allows us to realize a self routing interconnection network where the comparators in the sorting array determine the switch settings in individual cells of the routing array itself.

In this approach, we attach destination tags to each signal that we wish to communicate. An example is shown in Fig 3.5. To obtain the permutation $\begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 2 & 0 & 1 \end{pmatrix}$, $input_0$ has to be $3x_0$ where x_0 is the bit to be communicated to $output3$ and so on. As shown in Fig 3.5, the sorting network generates outputs so that x_2 is obtained from $output_0$, x_3 is obtained from $output1$ etc.
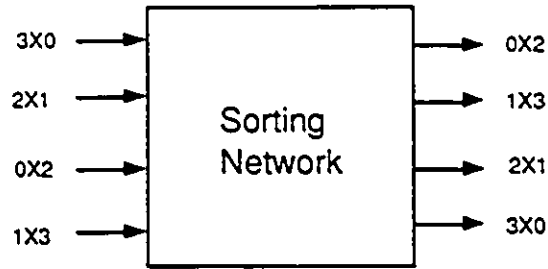
Fig 3.5 Sorting network for self routing.

A number of schemes for sorting have been proposed for implementation in VLSI technologies [14][69][78][82]. Each cell in these sorting arrays has a comparator and a switch. Thus each cell in any of these schemes is identical to the cell used in Fig 3.4 except that there is a comparator in each cell that compares the two inputs to the cell. If we use the convention followed in Figure 3.4 and, for a given cell at a particular instance of time, the input to the cell from the left is less than (greater than or equal to) the input from the top, the switch in the cell is set in the bending (crossing) mode.

The architecture we investigate is adopted from [69] and is shown in Fig 3.6. We will term this the *S-array*. Each cell in the array receives two inputs - one vertical (or from the top) and one horizontal (i.e., from the left). Each cell compares the horizontal and the vertical inputs and routes the larger (smaller) input to a latch associated with the vertical (horizontal) output. It is important to note the similarity of this array to that described in 3.3.

Theorem 3.3:

In an S-array for sorting n elements,

i. Area (A) of the network is $O(n^2 \log n)$

ii. Time (T) required for computation is $O(n \log n)$

iii. The pipelined period (P) of this network is $O(\log n)$

37

Proof :

The array contains $(n^2 - n)/2$ cells (same as the K-array). Each cell has to compare two numbers each containing log n bits. Thus the size of each cell is O(log n) bits. The array requires n clock cycles (same as the K-array). Each clock should be long enough to compare 2 numbers each containing log n bits. Thus the pipelined period is O(log n).

<div align="right">❏</div>

As discussed by Nassimi [50], the computational complexity in setting the switches in a multistage interconnection network is very high. To our knowledge, there is no analysis of the VLSI area required to communicate the switch settings to individual switches. Intuitively, this area is likely to be considerable. Our solution, using a sorting network to determine the switch settings, completely avoids this problem. It is interesting to note that the S-array, for the metric *AP* and *ATP*, outperforms standard arrays. For the metric *AT* and AT^2, the S-array has complexity measures comparable to those of the Ω network, the butterfly network and the Benes network. The S-array is superior, so far as the asymptotic bounds are concerned, to existing sorting networks (e.g., the bitonic sorting network) [5][31]. The self routing property of the S-array, therefore, is not "costly" in terms of performance metric. A comparison between Batchers sorting network with a S-array is shown in Table 3.3.

It is important to remember, however, that each cell in the S-array contains a comparator which compares (log N) bit numbers. Therefore, the cells are larger than those used in networks where no facility for self routing exists. This means that the superior performance of the S-array will be evident only for larger values of N compared to the value of N for which the K-array becomes superior to structures involving high wire organizations.
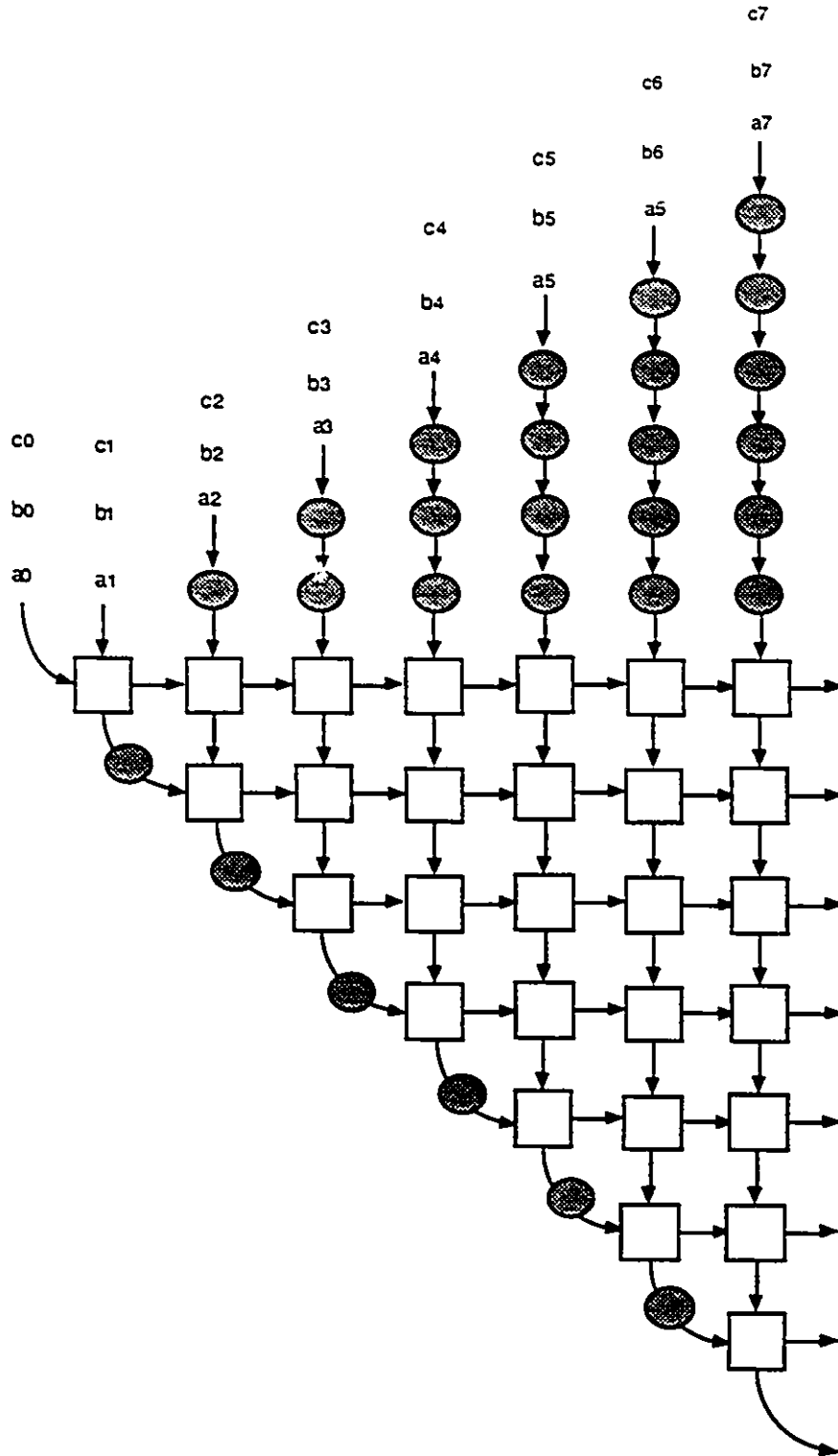
Fig. 3.6. Triangular array for sorting. Shaded areas represent delay elements.

Table 3.2a: Area & Time Complexities of interconnection networks.

Topology	Space/area Complexity			Time complexity		
	LSI	VLSI	ULSI	LSI	VLSI	ULSI
Butterfly	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Shuffle exchange	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
CCC	$O(n)$	$O\left(\dfrac{n^2}{\log^2 n}\right)$	$O\left(\dfrac{n^2}{\log^2 n}\right)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Benes network	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$
Clos Network	$O(n^{1.5})$	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$	$O(n)$
Systolic k-array	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$

Table 3.2b: Performance measures of interconnection networks.

Topology	Area A	Time T	Pipeline Period P	AT	AT^2	AP	ATP
Butterfly	$O(n^2)$	$O(n \log n)$	$O(n)$	$O(n^3 \log n)$	$O(n^4 \log^2 n)$	$O(n^3)$	$O(n^4 \log n)$
Shuffle exchange	$O(n^2)$	$O(n \log n)$	$O(n)$	$O(n^3 \log n)$	$O(n^4 \log^2 n)$	$O(n^3)$	$O(n^4 \log n)$
CCC	$O\left(\dfrac{n^2}{\log^2 n}\right)$	$O(n \log n)$	$O(n)$	$O\left(\dfrac{n^3}{\log n}\right)$	$O(n^4)$	$O\left(\dfrac{n^3}{\log^2 n}\right)$	$O\left(\dfrac{n^4}{\log n}\right)$
Benes network	$O(n^2)$	$O(n \log n)$	$O(n)$	$O(n^3 \log n)$	$O(n^4 \log^2 n)$	$O(n^3)$	$O(n^4 \log n)$
Clos Network	$O(n^2)$	$O(n)$	$O(n)$	$O(n^3)$	$O(n^4)$	$O(n^3)$	$O(n^4)$
Systolic k-array	$O(n^2)$	$O(n)$	$O(1)$	$O(n^3)$	$O(n^4)$	$O(n^2)$	$O(n^3)$

40

Table 3.3: Performance measures of sorting networks.

Topology	Area A	Time T	Pipeline Period P	AT
Batcher sorting network	$O(n^2 \log n)$	$O(n \log^2 n)$	$O(n \log n)$	$O(n^3 \log^3 n)$
Systolic S-array	$O(n^2 \log n)$	$O(n \log n)$	$O(\log n)$	$O(n^3 \log^2 n)$
Topology	$AT2$	AP	ATP	
Batcher sorting network	$O(n^4 \log^5 n)$	$O(n^3 \log^2 n)$	$O(n^4 \log^4 n)$	
Systolic S-array	$O(n^4 \log^3 n)$	$O(n^2 \log^2 n)$	$O(n^3 \log^3 n)$	

3.5 SUMMARY

In this chapter we have analyzed a variety of interconnection networks and sorting networks with the goal of providing a comprehensive comparison study for implementation in the ULSI medium.

A table has been generated that compares several 'area efficient' routing networks with area and time complexities using the three models: gate model, synchronous model, capacitive model. A systolic routing array, modified from an existing array, has been introduced. We have shown that this array outperforms the comparison arrays in metric AP and APT.

We have also introduced a simple technique to solve the major problem of determining switch settings for nonblocking interconnection networks. This approach has allowed us to realize a self routing interconnection network, rather than the more usual globally controlled network. This S-array is shown to outperform existing sorting networks in a detailed comparison study.

CHAPTER

4

SYSTOLIC ARCHITECTURES WITH CELLS OF CONTROLLABLE COMPLEXITY

4.1 INTRODUCTION

We have discussed, in Chapters 2 and 3, why, in ULSI technologies, short and regular interconnections in systolic architectures realize an area efficient layout with little communication overhead. One problem with the type of architectures we have examined in chapter 3 is that the number of cells in such networks are relatively large. In fact, this is the reason why high wire organizations, where the number of processors is attractively low, were proposed in the first place. However, as we also saw in chapter 3, the overhead due to complex interconnections renders the high wire organizations inefficient as the number of inputs to the network increases. In a sense, there is a trade-off between complex interconnection, $O(n \log n)$ cells in high wire organizations and local interconnections $O(n^2)$ cells in systolic arrays. In this chapter we introduce a novel communication architecture which possesses advantages of the both design families; we refer to this as the *locally long but globally short (LLGS) connection methodology*. We show, in this chapter, that systolic arrays based on local intercell communication may be designed where the cells use complex intracell communication. It is important to note, in this approach, that the number of inputs to a cell in the array, and hence the overhead due to communication within the cell, will depend on the technology we are using and the speed requirements of

the user. As the technology evolves, our recommendations for optimum cell size will also change. Our approach is to design a large (coarse grained) cell using a high wire organization within the cell instead of using a small (fine grain) cell as the basic building block .

In order to provide an example base to appropriately introduce and explore the concept, we use perimeter sorting as the application area. We first explore the concept of LLGS connection methodologies using the data communication architecture of the K-array and the S-array from chapter 3. This sorting network may be based on bitonic sorting or on odd-even merge [5][31] and, to our knowledge, represents a novel approach to designing sorting networks. Then we look at another systolic array for perimeter sorting. This is a new structure which has the same asymptotic performance metric as the S-array but it is attractive because it requires fewer comparators. We show that this new array may also utilize the LLGS concept. We also show that the approach of using complex intracell communication is useful in a number of other applications where high wire organizations are often used. This includes an array for routing and an array for computing the FFT. An important point about all these applications is the fact that they are widely used and require many inputs. Therefore our observations regarding the overhead due to high wire organization are particularly applicable for all these arrays.

4.2 THE LLGS PHILOSOPHY

To explain this concept, let us consider an application of sorting using N data elements. One way to design a systolic array for sorting is to use elementary 2 input cells as building blocks. The sorting network (S-array) of chapter 3 is an example of such a network. Let us consider a new systolic array which uses complex cells with r inputs in each cell. A

cell, in this systolic array, is an array of subcells and uses a high wire organization for communication between subcells. We are claiming that, in some important application areas, the performance of the systolic array will improve by a factor f(r) where f is a function determined by the nature of the application we are considering. Of course, so far as the asymptotic array bounds are concerned, the performance does not improve since f(r) is a constant for a given complex cell; however, in terms of VLSI area and time this improvement is likely to be significant. Another way to look at this concept is to say that we consider data communication in two levels - inter cell communication and intra cell communication. If the communication graph corresponding to a systolic array has a width of ω, our approach uses complex cells such that the intercell communication graph has a width of $\dfrac{\omega}{f(r)}$.

When designing a systolic array, it is likely that a fastest clock speed can be achieved with a small cell. In a network for sorting, the smallest cell will correspond to a cell with a comparator and a 2×2 switch (as used in the S-network in chapter 3). When implementing a digital system, the clock rate is determined by many factors, not just the throughput rate of one subsystem (the network for sorting in this example). Therefore, it does not make sense to design a circuit which operates at a clock rate considerably faster than what the user requires (Δ). What the user requires is an array with the smallest VLSI layout area which will work at the specified speed. We therefore propose to look at systolic arrays where each cell is fairly complex and is composed of a large number of less complex subcells. The communication between subcells may use a high wire organization provided that the total delay within the cell is acceptable.

At this point we give, as an example, the array for merge sorting that we will examine later in detail. Fig 4.1 gives the block diagram of an array to sort N numbers. For simplicity, we are now describing the array simply as a combinational circuit. Later we will introduce

data latches and design a fully pipelined systolic array. Each cell in the array has r horizontal inputs and r vertical inputs. The r horizontal (vertical) inputs to a cell are sorted.

INPUT TO SORTING ARRAY
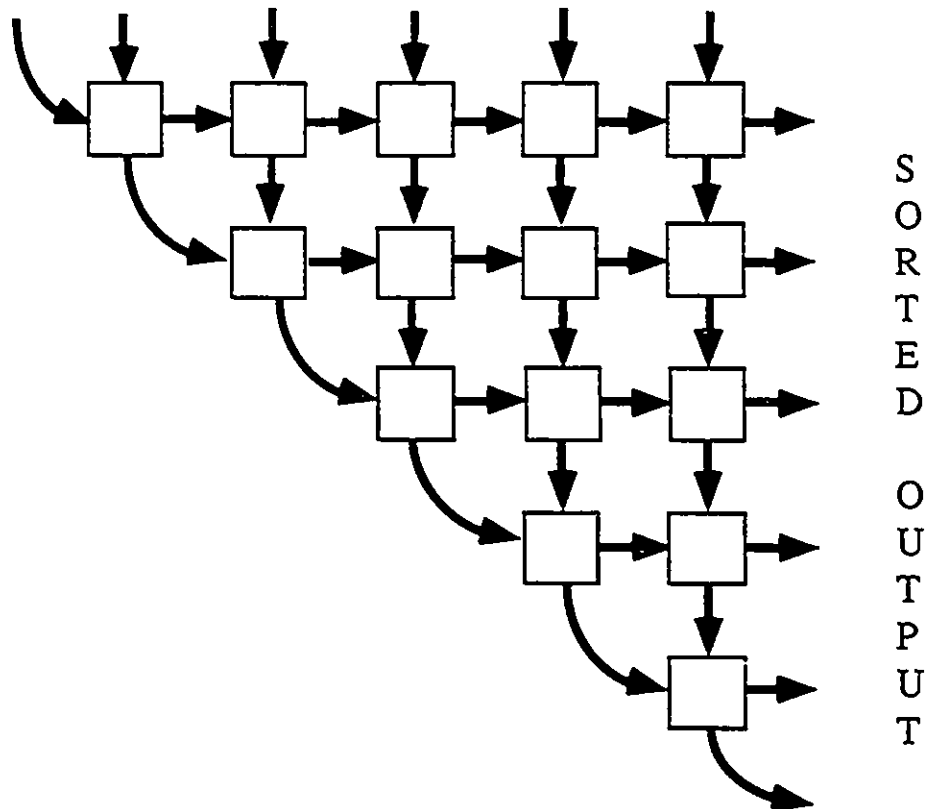
Figure 4.1 Array for merge sorting using cells of controllable complexity

Each thick arrow in Fig 4.1 represents a bus carrying r numbers. Each cell sorts the 2r inputs it receives and, using merge sort, generates a sorted output consisting of 2r numbers. As mentioned before, we use a high wire organization within the cell. The value of r may change from application to application. In this architecture, one extreme is to have r = 1. This gives us the S-array presented in Chapter 3. The other extreme is to use r = N/2 so that we need only one cell to sort N numbers. The latter choice gives us a conventional high wire organization for merge-sorting [31]. The first alternative requires $O(N^2)$ comparators with a fixed overhead for interconnections for each comparator. The second

alternative requires $O(N \log N)$ comparators but the overhead for interconnection increases exponentially with successive stages. The first alternative needs more comparators (active area) but the second requires more area for interconnection. The higher the value of r (in other words, the bigger the size of the cell), the greater the interconnection overhead per cell; however, for a given value of N, the area occupied by the comparators is smaller. This architecture provides flexibility by allowing a compromise in determining the size of a cell by balancing the overhead for interconnection area against the overhead for active area.

To realize the highest throughput in such an array, we have to pipeline the array. The scheme for pipelining is very simple: we include a latch with every comparator. The pipeline period is determined by the length of the longest wire, and we will see that this length is $O(r)$.

We are now ready to discuss an important question about the cell - *how large a cell should we use?* The propagation delay as well as the overhead due to interconnection area is least between rows 1 and 2 and increases by a factor of 2 with every successive stage. The total cell delay $\tau \geq \{t_1 + t_2 + ... + t_i ... + t_{\log(r)}\}$, where t_i represents delay of the ith stage. Each stage delay includes both device and wire delays $t_i = t_d + t_w$, where t_i can be controlled using a minimum time of t_w. Therefore, if we have fairly stringent requirements for pipeline period (P), optimal $\{\tau\}$ or $\Delta \geq P$, we will have to settle for an optimal $\{r\}$, since r is proportional to (τ) as well as $O(r^2)$. Alternatively, if the pipeline period is not critical, since P depends on t_d and t_w, the size of a cell should be such that any larger cell would have an unacceptable overhead due to interconnection area.

To summarize, for an user with stringent pipeline period requirements, the idea of locally long and globally short communication is as follows:

- *Determine the worst pipeline period that is acceptable to the user $(P \leq \Delta)$*

- *Design cells based on a high wire organization scheme such that the intracell delay is within the acceptable bounds ($P \geq \tau$)*

- *Design systolic arrays using this cell as a building block.*

We term this class of architecture *systolic arrays with cells of controllable complexity*. The size of the cell, in terms of the number of inputs (r), and hence the parameters of A,T,P of the high wire organization, is controlled by user requirements (Δ), and it is allowed to change from one application to another. By choosing an appropriate value of the number of inputs, r, the area and time metric of such a cell are controllable.

4.3 A SYSTOLIC ARRAY FOR SORTING

We now discuss, in detail, the design of a systolic array for sorting using the concept of locally long connections. We will use the operator II to denote the concatenation operation. Thus X II Y stands for the list obtained by appending list Y to list X. As we mentioned in section 4.2, we have a two dimensional array of complex cells arranged in the form of a triangle (Fig 4.2). Each cell in the triangular array is represented by a square and has 2 sets of inputs - inputs from the left consisting of an ordered list of r numbers and inputs from the top, again consisting of an ordered list of r numbers. In general, the array has N inputs where N = r(m+1) and the number of cells on the base of the triangle is m. Fig 4.2 shows the situation where m = 5.

In Fig. 4.2, S_0, S_1, ..., S_5 represent unordered lists of r inputs. S_0 II S_1 II ...II S_5 is the input to the array at time t_0. Following Thompson [79], we use an arrow to denote whether the data is sorted in ascending or descending sequence. In other words, the data towards

the tail of an arrow is, by our convention, smaller than the data towards the head of the arrow.

This array requires cells at the periphery of the triangular array of complex cells for some preprocessing as well as to ensure the integrity of the wave front. Preprocessing cells, represented by shaded ovals ⬭, are used for sorting so that each preprocessing cell receives a list of r inputs, sorts the list of inputs and generates this sorted list as its output in the next clock pulse.
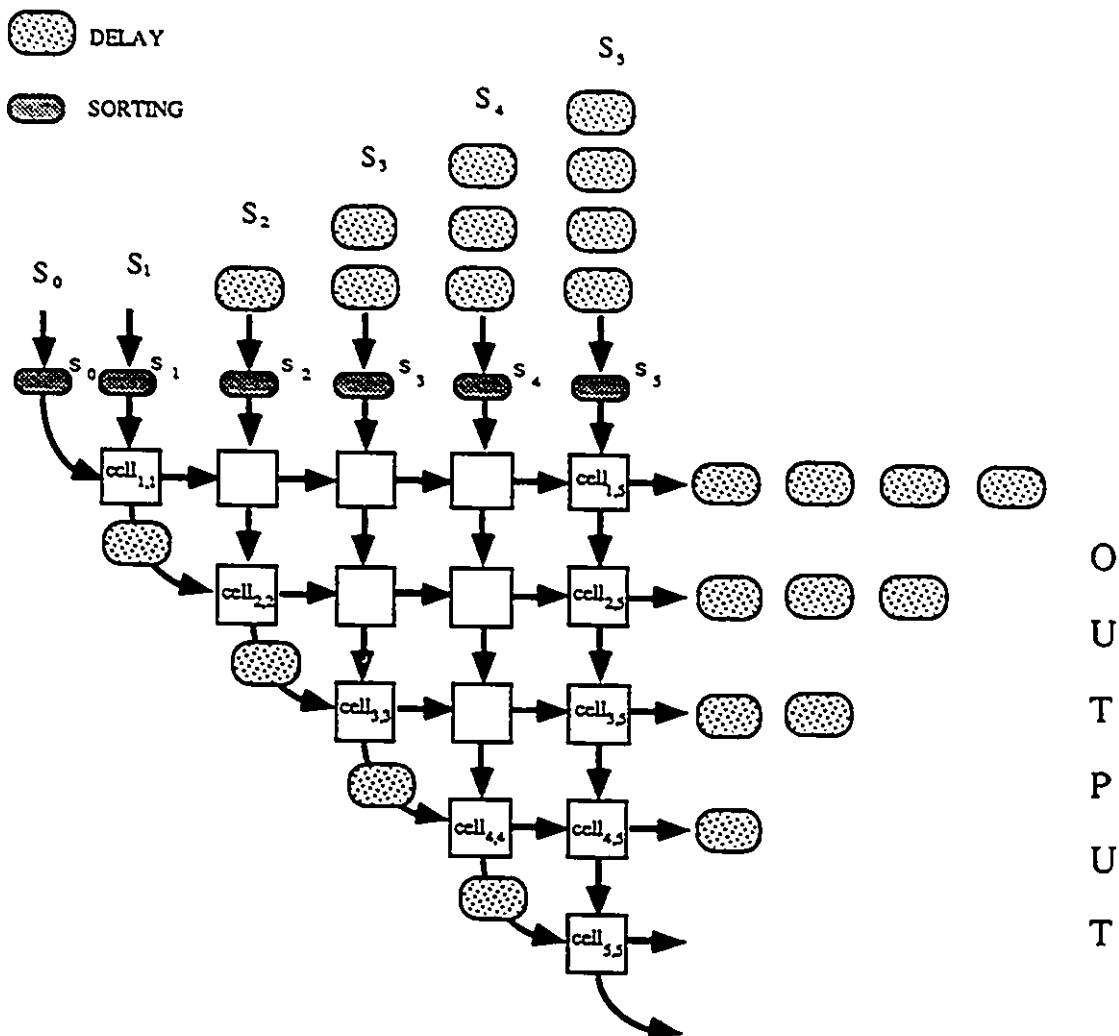
Figure 4.2   Triangular systolic parallel sorting network with multiple operands.

48

It is convenient to assign numbers 0, 1, ... , m to the preprocessing cells starting from left and moving to right. We use S_i to refer to the ith preprocessing cell. Following the style we used when discussing the array in Fig 4.1, we initially assume, for simplicity, that sorting in either the preprocessing cells or the complex cells in the triangular array itself is complete within one clock period. In other words, there is no storage device between stages of subcells but there is a set of 2r latches after the last stage of subcells within a complex cell. As mentioned in 4.2, the throughput may be further increased by pipelining the subcells within the complex cell by putting latches between subcells. However, we will not explore this option because the improvements are marginal. We represent a delay element by an oval ⬭.
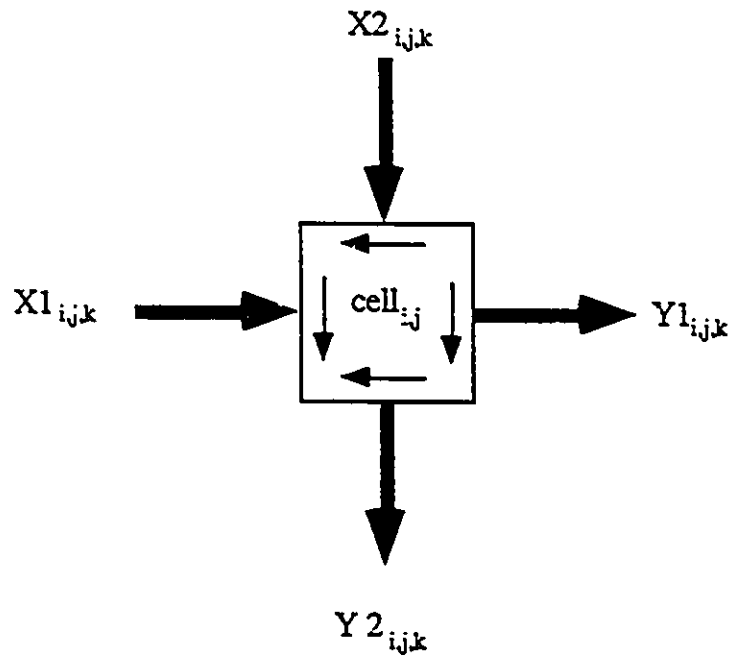
Figure 4.3 A comparator cell. Each thick line represents an ordered list of r numbers.

A delay element has r inputs and r outputs and generates, as output at time t_{k+1}, the input it received at time t_k. We use the term $cell_{i,j}$ to refer to the complex cell in row i and column j. At time t_k, $cell_{i,j}$ receives, as its inputs, two ordered lists $X1_{i,j,k} = [x1^0_{i,j,k}, x1^1_{i,j,k}, ...$

$x_1^{r-1}{}_{i,j,k}]$ and $X_{2_{i,j,k}} = [x_{2}{}^0{}_{i,j,k}, x_{2}{}^1{}_{i,j,k}, \dots x_{2}{}^{r-1}{}_{i,j,k}]$ as shown in Fig 4.1. The complex cell produces, as its output during the next clock pulse, two sorted sets $Y_{1_{i,j,k+1}}$ and $Y_{2_{i,j,k+1}}$ containing r elements each where

$$Y_{1_{i,j,k+1}} = [y_1^0{}_{i,j,k+1}, y_1^1{}_{i,j,k+1}, \dots y_1^{r-1}{}_{i,j,k+1}]$$

and $\quad Y_{2_{i,j,k+1}} = [y_2^0{}_{i,j,k+1}, y_2^1{}_{i,j,k+1}, \dots y_2^{r-1}{}_{i,j,k+1}]$ and $y_1^{r-1}{}_{i,j,k+1} \leq y_2^0{}_{i,j,k+1}$.
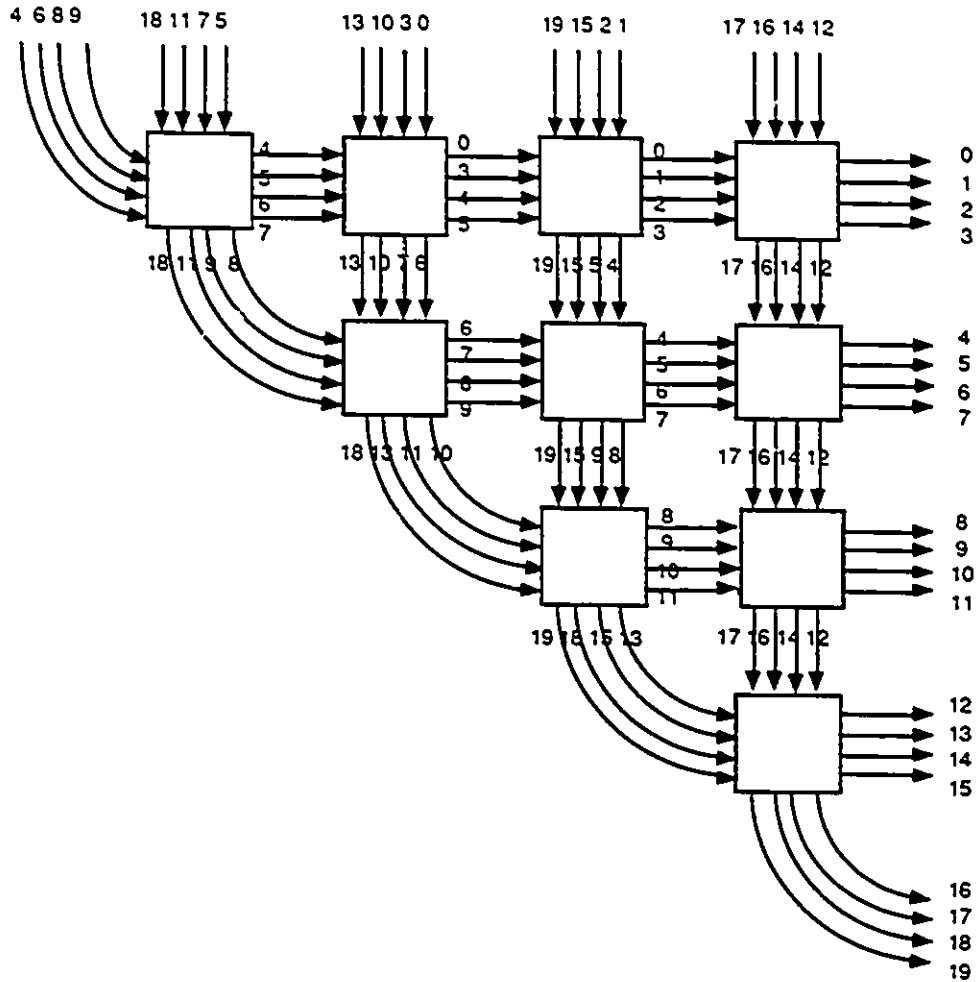
Figure 4.4 An example of merge-sorting in triangular arrays

The input $X_{1_{i,j,k}}$ is ordered in ascending sequence while $X_{2_{i,j,k}}$ is sorted in descending sequence. The output lines are in ascending sequence as we consider the output lines from

top to bottom and after we reach the lowest output, from right to left. Thus output $Y1_{i,j,k+1}$ ($Y2_{i,j,k+1}$) contains the r smallest (largest) numbers in $X1_{i,j,k} \| X2_{i,j,k}$. In other words, $cell_{i,j}$ sorts, using merging, the inputs it receives and generates, as its output, the first r of the resulting ordered list of numbers as the horizontal output and the remaining list of numbers as the vertical output.

An example of the use of such a systolic array is shown in Fig 4.4. In this figure, each complex cell accepts 4 horizontal inputs and 4 vertical inputs. For simplicity, all delay cells and latches have been removed from the network. The preprocessing of the inputs has been carried out before the top row of the array so that ordered lists of 4 inputs are fed to the array. We note that the first list is ordered in ascending sequence while all other lists are ordered in descending sequence. It will be clear, later on, why this makes it easy for us to carry out merging.

It is important to note that successive columns of complex cells generate ordered lists. For instance, the list of horizontal outputs of cells in column 3, if we scan the outputs from top to bottom, is the ordered list $L = [0,1,2,3,4,5,6,7,8,9,10,11]$. It is useful to note that if we include, at the end of L, the list of vertical outputs generated by the last cell in this column, scanning the outputs of this cell from right to left, we get an ordered list of all inputs fed to the top cells in columns 1 to 3.

Definition 4.1:

Consider an unordered list $L = [a_0, a_1, \dots, a_{rm-1}]$ of elements. Let the successive elements of this list, after we arrange the elements in order of ascending (descending) sequence, be the ordered list $[b_0, b_1, \dots, b_{rm-1}]$. The ordered list $[b_i, b_{i+1}, \dots, b_{i+m-1}]$ will be termed the *ith sorted sublist* of the list L.

Theorem 4.1:

The array shown in Fig 4.2 generates an output in the form of an ordered list after a latency period of 2m clock cycles.

Proof :

We first divide our inputs into lists of r elements each. We will use S_i, to refer to the ith list of r elements, $0 \le i \le m$. We choose our time frame so that the wave front we are considering appears as input to the array at time t_0. A total of $r(m+1)$ data elements, representing the wave front at time t_0, are input to the array at time t_0. We note that elements in S_j are delayed by $(j-1)$ delay elements so that elements in S_j are input to preprocessing cell S_j, $1 \le j \le m$ at time t_j. At time t_0, the first $2r$ elements appear as input to preprocessing cells S_0 and S_1. It can be verified immediately that $\text{cell}_{i,j}$ of the triangular array receives its input, corresponding to the wave front we are considering, at time t_{i+j-1}.

We now prove the theorem by induction. We claim that, at time t_{i+j}, $\text{cell}_{i,j}$ generates, as its horizontal output, the ith sorted sublist of $S_0 \parallel S_1 \parallel ... \parallel S_j$, for all i,j, $1 \le i,j, \le m$. Clearly, this is true for $i=1$ and $j=1$. Let this be true for all i,j, $1 \le i,j < k$. Then, at time t_k, the left input $X1_{1,k,k}$ to $\text{cell}_{1,k}$ is the first sorted sublist of $S_0 \parallel S_1 \parallel ... \parallel S_{k-1}$. The vertical input to this cell at time t_k is the ordered list corresponding to S_k. The horizontal output $Y1_{1,k,k+1}$ is the first k elements of the list $S_k \parallel X1_{1,k,k}$. Clearly $Y1_{1,k,k+1}$ is the first sorted sublist of $S_0 \parallel S_1 \parallel ... \parallel S_{k-1} \parallel S_k$. Thus our assertion is true for $\text{cell}_{1,k}$ as well. Now we use induction on j and prove, in a way similar to that for $j = 1$, that for all j, $1 \le j \le k$, $\text{cell}_{j,k}$ generates, as its horizontal output, the jth sorted sublist of $S_0 \parallel S_1 \parallel ... \parallel S_{k-1} \parallel S_k$.

$\square$

52

4.4 DESIGN OF MERGING CELLS

The next question that we consider is the design of the complex cells which will merge two sets of sorted inputs. If we follow the philosophy outlined in Chapter II, we would use a mesh type structure and obtain asymptotic performance metric for cells with $O(r)$ inputs. This would give us $O(r^2)$ area and $O(r)$ latency period per cell. As indicated earlier in this chapter, we will explore the idea of a high wire organization within the cell. There are a number of sorting networks, with a high wire area, described by Knuth [31] and to illustrate our approach, we choose Batcher's bitonic sorting network and odd-even merge sorting network.

Definition 4.2:

A list L of numbers is *monotonic* ascending (descending) if successive elements in the lists have ascending (descending) values. A list L is *bitonic* if there exists lists Ll and L2 where $L = Ll \parallel L2$ and Ll is monotonic ascending (descending) while L2 is monotonic descending (ascending).

Example :

Ll = [2,4,5,5,6,8,9] is a monotonic ascending list while L2 = [17,13,11,9,4,1] is a monotonic descending list.

List L3 = [17,13,11,9,4,1,2,4,5,5,6,8,9] and L4 = [2,4,5,5,6,8,9,17,13,11,9,4,1] are examples of bitonic lists.

4.4.1 A CELL FOR BITONIC MERGING

We now briefly review the algorithm used in Batcher's bitonic sorting network [5]. This algorithm will be used to design $cell_{i,j}$ for any i,j. To simplify our notation we are dropping the subscripts i and j. We are therefore talking about two inputs $X1_k$ and $X2_k$ where $X1_k$ ($X2_k$) is the horizontal(vertical) input to $cell_{i,j}$ at time t_k.
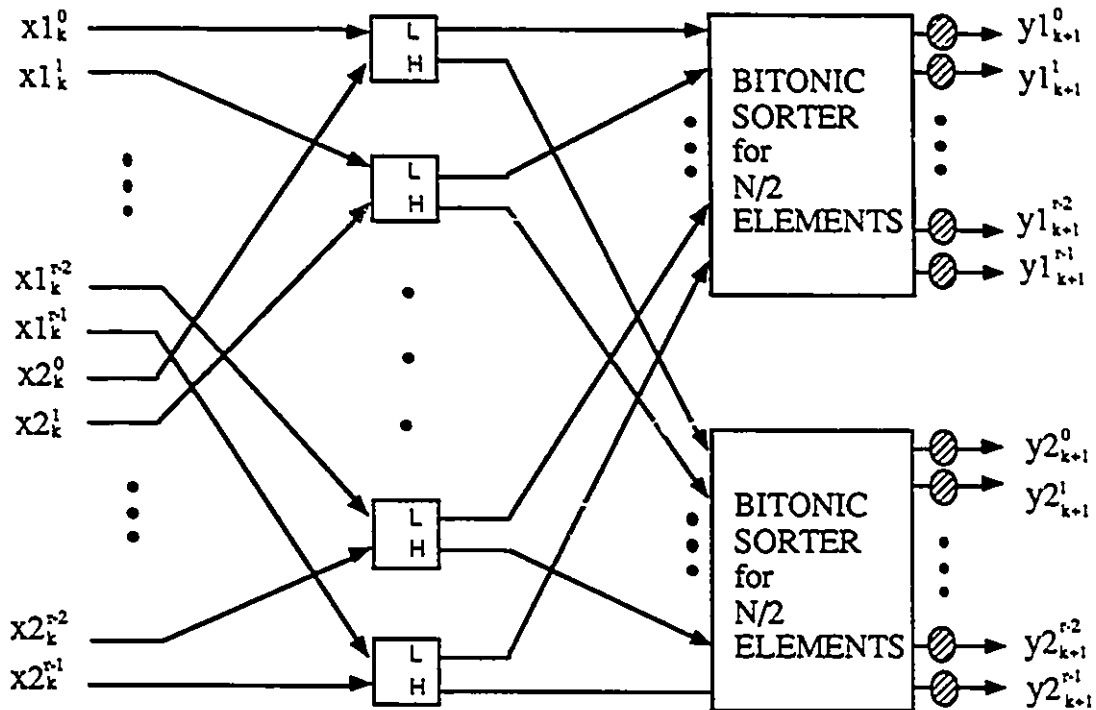
Fig 4.5 Recursive definition of the Bitonic sorting algorithm

We take a simplified situation where we have two ordered lists $X1_k$ and $X2_k$ each with r elements. $X1_k$ is sorted in ascending sequence while $X2_k$ is sorted in descending sequence. Obviously the list obtained by appending $X1_k$ after $X2_k$ is bitonic.

54

Let:

$$X1_k = [x1^0_k, x1^1_k, \ldots, x1^{r-1}_k]$$

and

$$X2_k = [x2^0_k, x2^1_k, \ldots, x2^{r-1}_k].$$

Batcher [5] shows that if we form two lists, L1 and L2, of n numbers, where L1 = $[\min(x1^0_k, x2^0_k), \min(x1^1_k, x2^1_k), \ldots, \min(x1^{r-1}_k, x2^{r-1}_k)]$ and L2 = $[\max(x1^0_k, x2^0_k), \max(x1^1_k, x2^1_k), \ldots, \max(x1^1_k, x2^1_k)]$, then both lists L1 and L2 are bitonic and no element of L1 is larger than any element of L2. This immediately suggests that we may recursively define a bitonic merge network as shown in Fig 4.5.

A network to merge two ordered lists $X1_k$ and $X2_k$ to give an ordered list as the output is shown in Fig 4.6. Here r = 4. This network may also be readily implemented using a shuffle exchange network in $O(\log^2 r)$ steps using recirculation. We omit the details of implementation of individual shuffle exchange networks since this topic has been thoroughly investigated [25][30][42][82]. The only problem that we need to consider when implementing this method using VLSI fabrication is to ensure that the data applied to the input of the merging circuit is bitonic. It may be noted that the horizontal output (going from top to bottom) is in ascending sequence while the vertical output (going from right to left) is in descending sequence.

If we append an ascending sequence after a descending sequence, we obviously get the desired bitonic sequence. Therefore, when applying the data to the bitonic merging network we simply append the horizontal output after the vertical output (Fig 4.7).
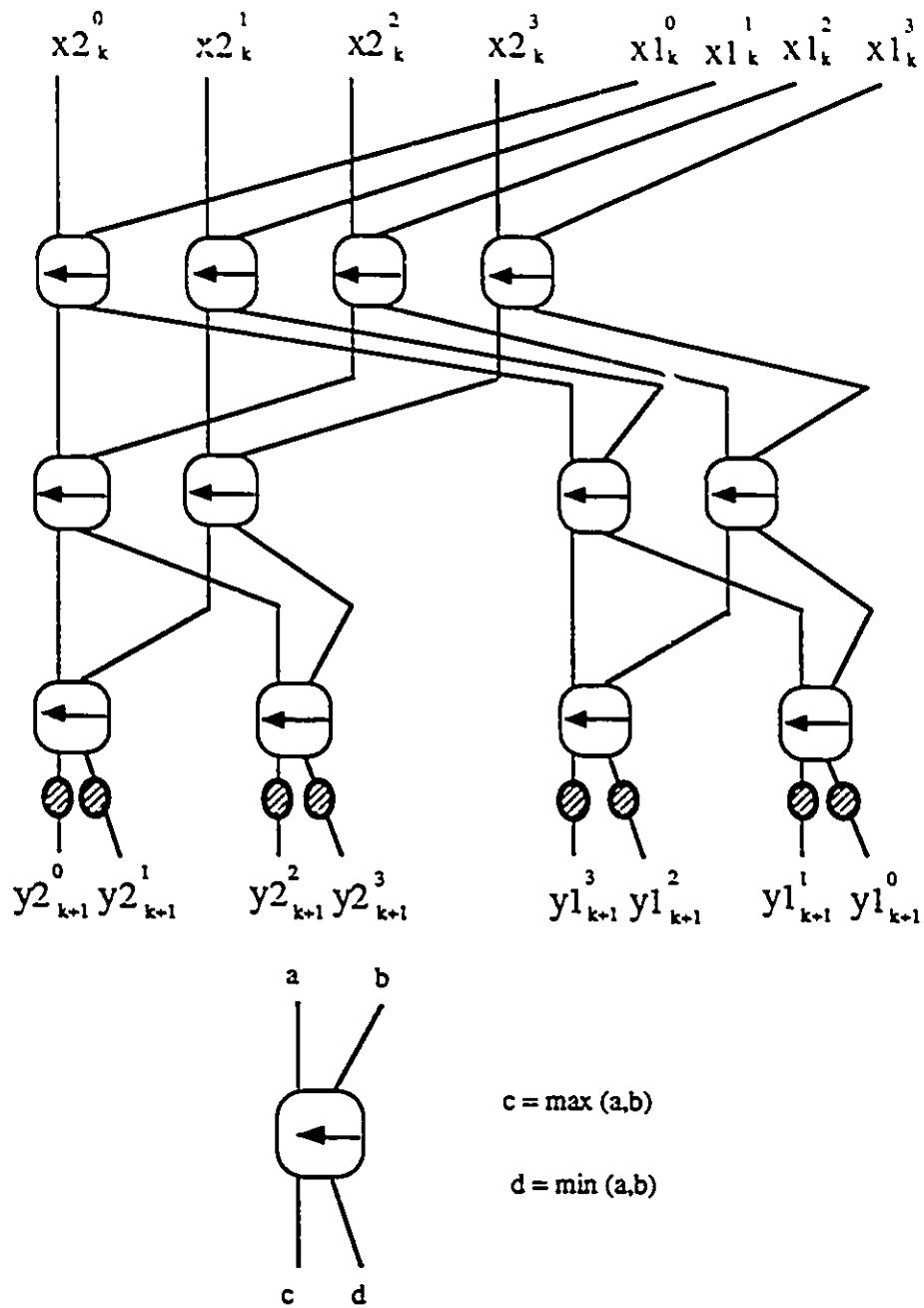
Figure 4.6 Bitonic merge sorting for r=4.
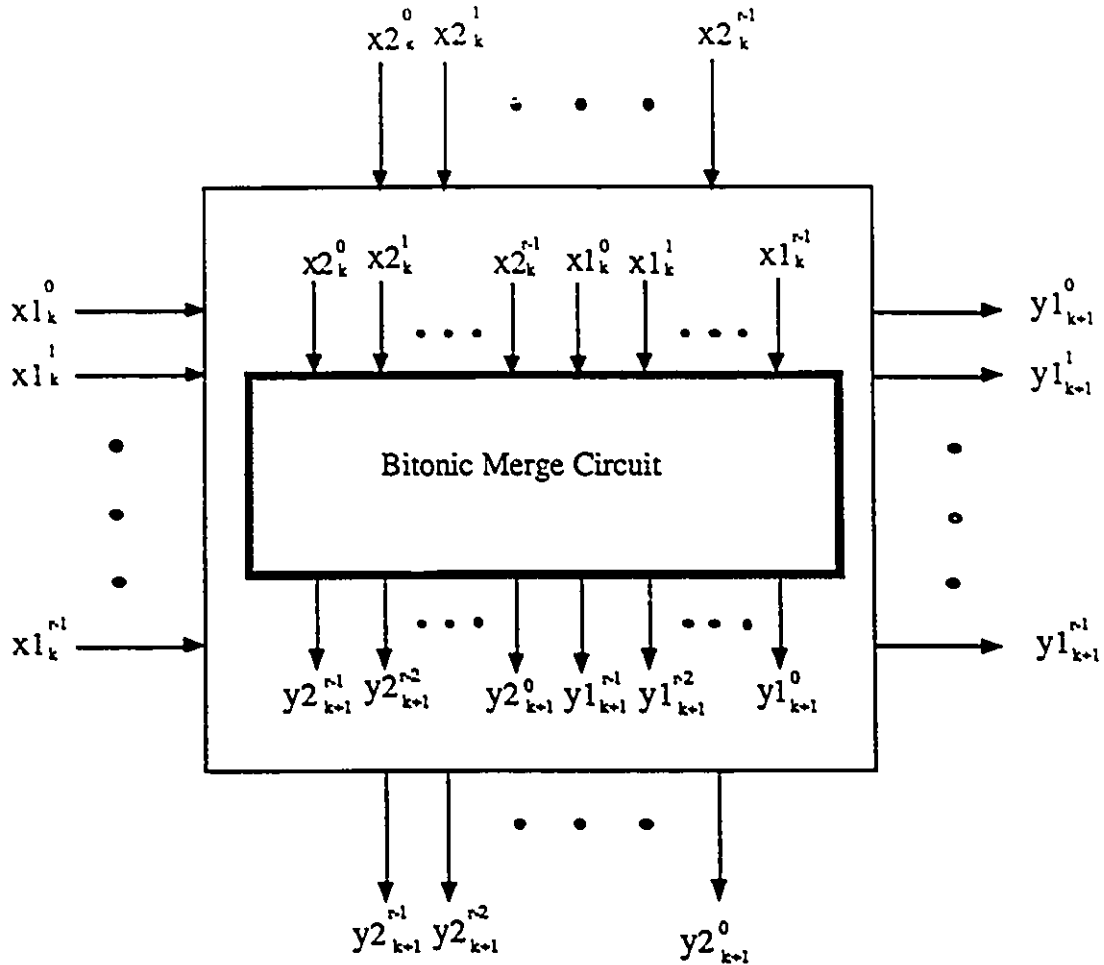
Figure 4.7 Relationship between the bitonic merge circuit (Fig 4.5) and cell of systolic array (Fig 4.2)

4.4.2 A CELL FOR ODD-EVEN MERGE SORTING

A network to merge two ordered lists $X1_k$ and $X2_k$ to give an ordered list as the output is shown in Fig 4.8. The algorithm used in this approach is called odd-even merging.

For simplicity, in our description, we assume that both $X1_k$ and $X2_k$ have r elements each so that the sorted output has 2r elements.
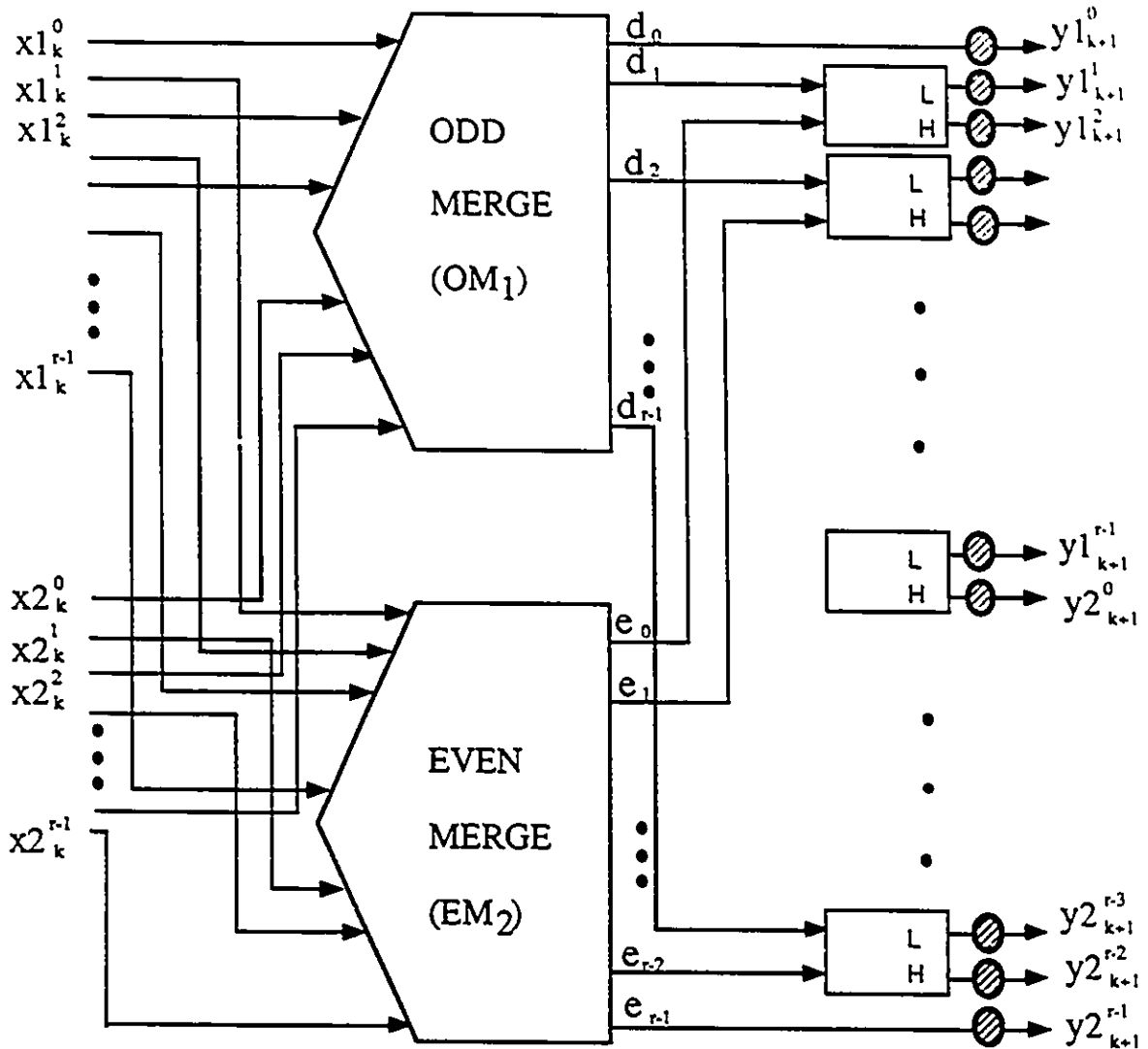
Figure 4.8 Odd-even merging network

The network is defined recursively in terms of two smaller odd-even merging networks OM_1 and EM_1 where the network OM_1 (EM_1) merges two lists each with $r/2$ elements. As shown in Fig 4.8, the algorithm works by dividing the inputs $X1_k$ ($X2_k$) into two ordered lists EA and OA(EB and OB) of equal size, so that the successive odd elements of A(B) are successive elements of OA(OB). Similarly, successive even elements of A(B) are successive elements of EA(EB). Clearly both EA and EB (OA and OB) are ordered. The

58

network $OM_1(EM_1)$ merges OA and OB(EA and EB) to produce two ordered lists D and E each with r elements. Let D(E) be the ordered list $[d_0, d_1, ... , d_{r-1}]$ $([e_0, e_1, ... , e_{r-1}])$ produced by $OM_1(EM_1)$. It has been shown [5] that the relationship between the elements of D, E, $Y1_k$ and $Y2_k$ is as follows:

- $y1^0_{k+1} = d0$

- $y1^{2i+1}_{k+1} = $ minimum of (d_{i+1}, e_i), for all i, $0 \leq i < r/2$

- $y1^{2i}_{k+1} = $ maximum of $(d_i \ e_{i-1})$, for all i, $1 \leq i \leq r/2$

- $y2^{2i}_{k+1} = $ maximum of $(d_{i+r/2+1}, e_{i+r/2})$, for all i, $0 \leq i < r/2$

- $y2^{2i-1}_{k+1} = $ minimum of $(d_{i+r/2+1}, e_{i+r/2})$, for all i, $1 \leq i \leq r/2$

- $y^{2r-1}_{k+1} = e2r$

It may be readily verified that the network shown in Fig 4.8 produces $Y1_k$ and $Y2_k$ from D and E as defined above. A cell for odd-even merging using two lists with 4 inputs each is shown in Fig 4.9.

4.5 PIPELINING WITHIN THE CELL

The cell for bitonic merging that we discussed in 4.4.2 has 2log r stages. We have tacitly assumed, so far, that each subcell consists of just a 2X2 switch and a comparator; however, these subcells do not include any latches for pipelining. This assumes that the clock period is long enough to ensure that a cell with 2 ordered lists of r elements as its input can always generate a ordered list of 2r elements as its output within a clock cycle.
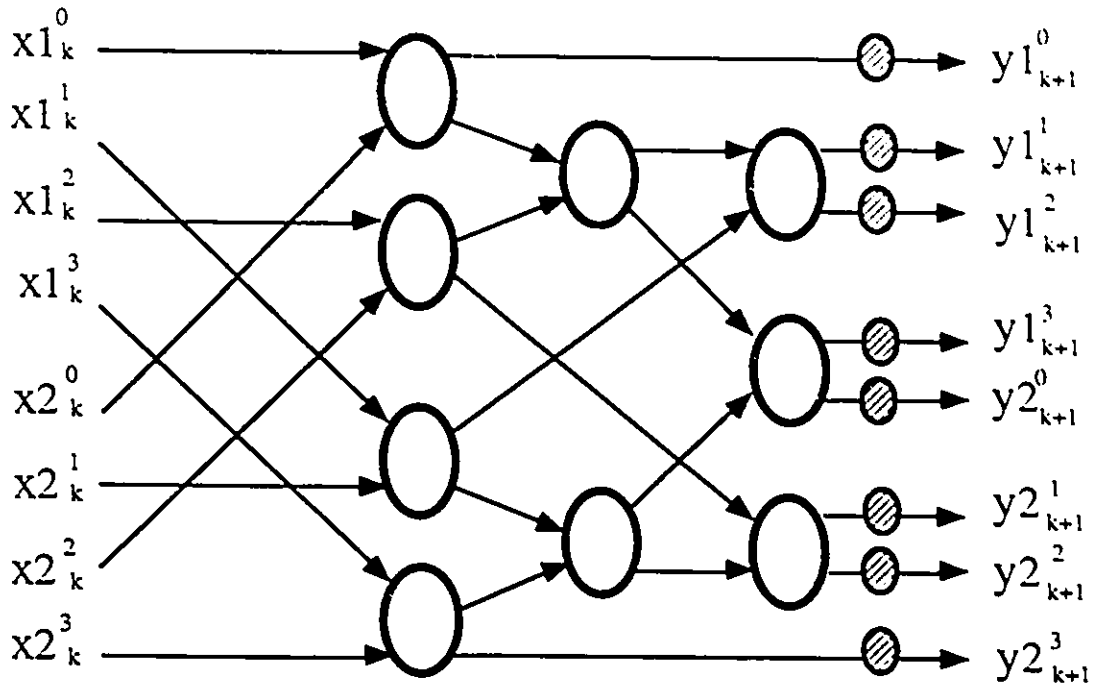
Figure 4.9 Eight inputs/outputs odd-even merge cell

We can always decrease the pipeline period by putting a pair of latches within each subcell. The discussions for routing networks presented in Chapter III (Section 3.4) are equally applicable and, for maximum throughput, the pipeline period must be longer than D_{max} where D_{max} is the maximum delay of propagation from one subcell to the next. As we have seen from the analysis above, this D_{max} is linearly proportional to r.

4.6. ANALYSIS OF SYSTOLIC ARRAYS FOR SORTING USING COMPLEX CELLS

The area for a single shuffle exchange network of r inputs is $O(A_{sw}r^2 / log^2 r)$ where A_{sw} is the size of a single switch. We need $O(\log r)$ stages in any complex cell for the merging network. Therefore we need an area A for a complex cell where $A = O(A_{sw}r^2 / \log r)$.

The area A_{sw} is $O(\log N_{max})$ where N_{max} is the biggest number that we wish to process. In this computation, it is reasonable to assume that the value of r and N_{max} are fixed. Therefore, in this case, the area A of a complex cell is $O(1)$. To implement an array with N inputs, we need an array with m = {(N/r)-1} cells at the base of the triangular array of Fig 4.2. Such a systolic array has $(m^2+m)/2$ cells; the area of the systolic array is, therefore, $O(N^2)$. We have seen, in Chapter 3, that the worst delay per stage is determined by the longest wire for communication (in other words the pipeline period is $O(r)$, where r is determined by user requirements and also by the fabrication technology). Since the pipeline period is determined by user requirements, it means that r, as mentioned above, is a constant. The pipeline period is $O(1)$. Thus the asymptotic area time metric of this array are identical to the S-array.

The above analysis, in a certain sense, is rather disappointing; we might as well use the S-array so far as the asymptotic bounds are concerned. However, we are ignoring the fact that a complex cell is smaller than an equivalent array of simple cells since, in reality, the actual area occupied by wires is smaller than the area occupied by the active area even though the asymptotic bounds are identical. We therefore use a different technique to look at the performances of the triangular array for merge sorting network with cells of varying complexities. A traditional metric for complexity of the sorting network is the delay or latency measured by the number of levels of switches and the number of comparators used [49]. Table 4.1 shows the total number of comparators and the latency periods required for different sizes of networks using complex cells based on odd even merging. We have assumed that there are latches between stages of subcells within a cell. It is interesting to note that the latency period of the network using four inputs cells is the same as that of a network using two input cells, since a parallel merger for four inputs requires two parallel steps. Thus a systolic array using 4 input cells is no faster than a systolic array using 2-input cells (just a comparator and latches). A complex cell to merge 2 pairs of numbers has

3 comparators. A complex cell to merge 2 sorted lists of 4(8) inputs requires 9(25) comparators. For convenience, we have also shown this information in diagrammatic form (Fig 4.10).

TABLE 4.1: Performance of sorting network.

N	inputs (R)	Total number of cells	Comparators	Latency
64	2	2016	2016	63
64	4	496	1488	62
64	8	120	1080	45
1024	2	523776	523776	1023
1024	4	130816	392448	1022
1024	8	32640	293760	768
1024	16	8128	203200	508

4.7 DESIGN OF A SYSTOLIC ARRAY FOR ROUTING

The intercell communication architecture for the systolic array for routing is very similar to that used in Fig 4.2. The only difference is that we do not require the preprocessing nodes when we design an array for routing. The complex cells may use any convenient scheme for intra cell communication so long as the scheme corresponds to that for a rearrangeable network. We have chosen a Benes network to illustrate the approach. We have ignored the additional control lines which are used to set individual switches in a subcell to either "straight " or "cross" mode [52]. In this particular approach, all the problems of

determining the settings of the switches, as mentioned in chapter 3, are applicable. The array is shown in Fig 4.11.
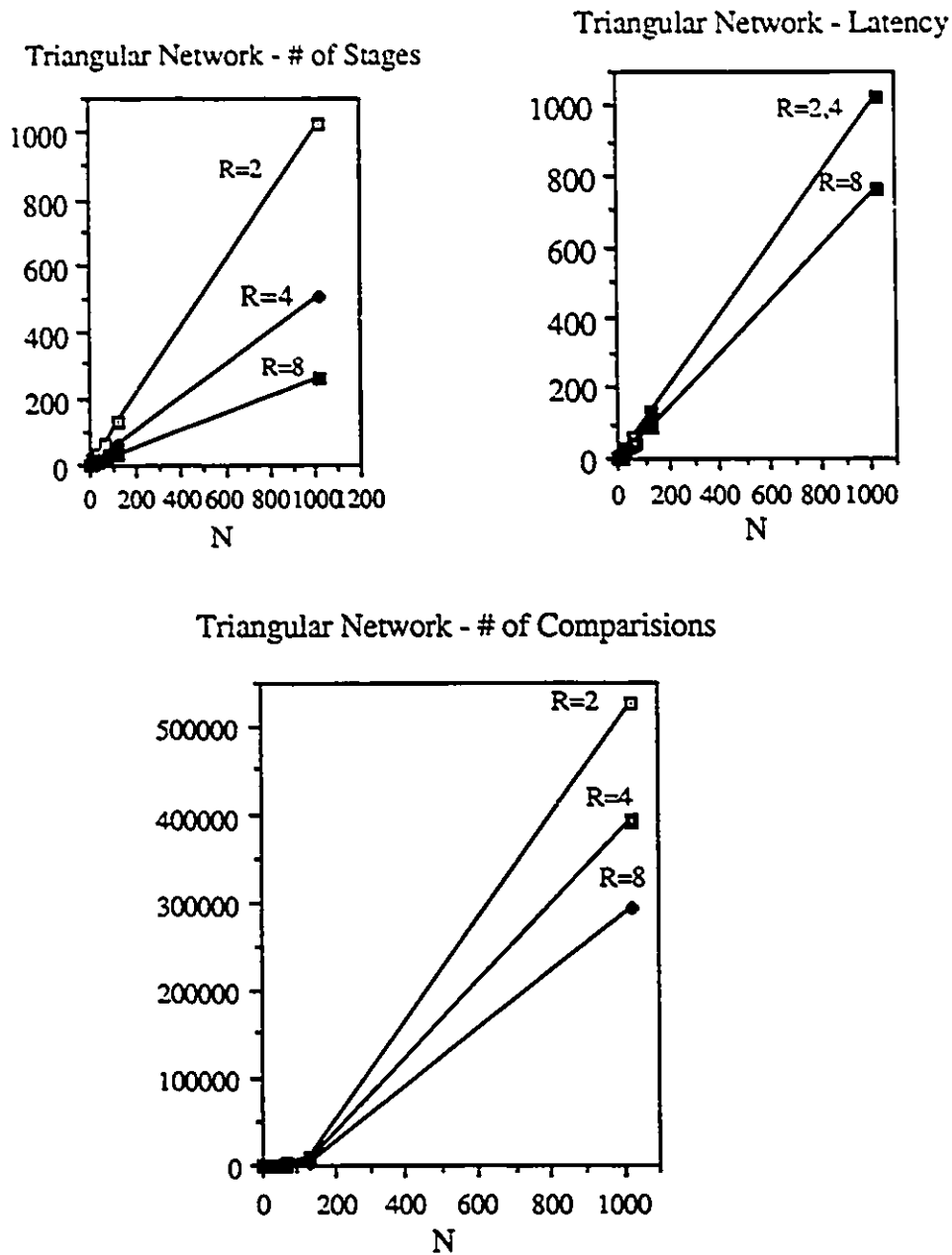
Figure 4.10 Shows the graphical comparisons for the triangular array network

4.8 DESIGN OF A NEW TWO DIMENSIONAL SYSTOLIC SORTING NETWORK

In this section we present the design of another 2-D systolic network for parallel sorting. The input (output) of each cell is connected to two neighboring cells. Before generalizing the method for implementing a complex cells we consider the simple situation where every cell, as in the S-array, consists of a 2 input comparator and buffers to pipeline the output.
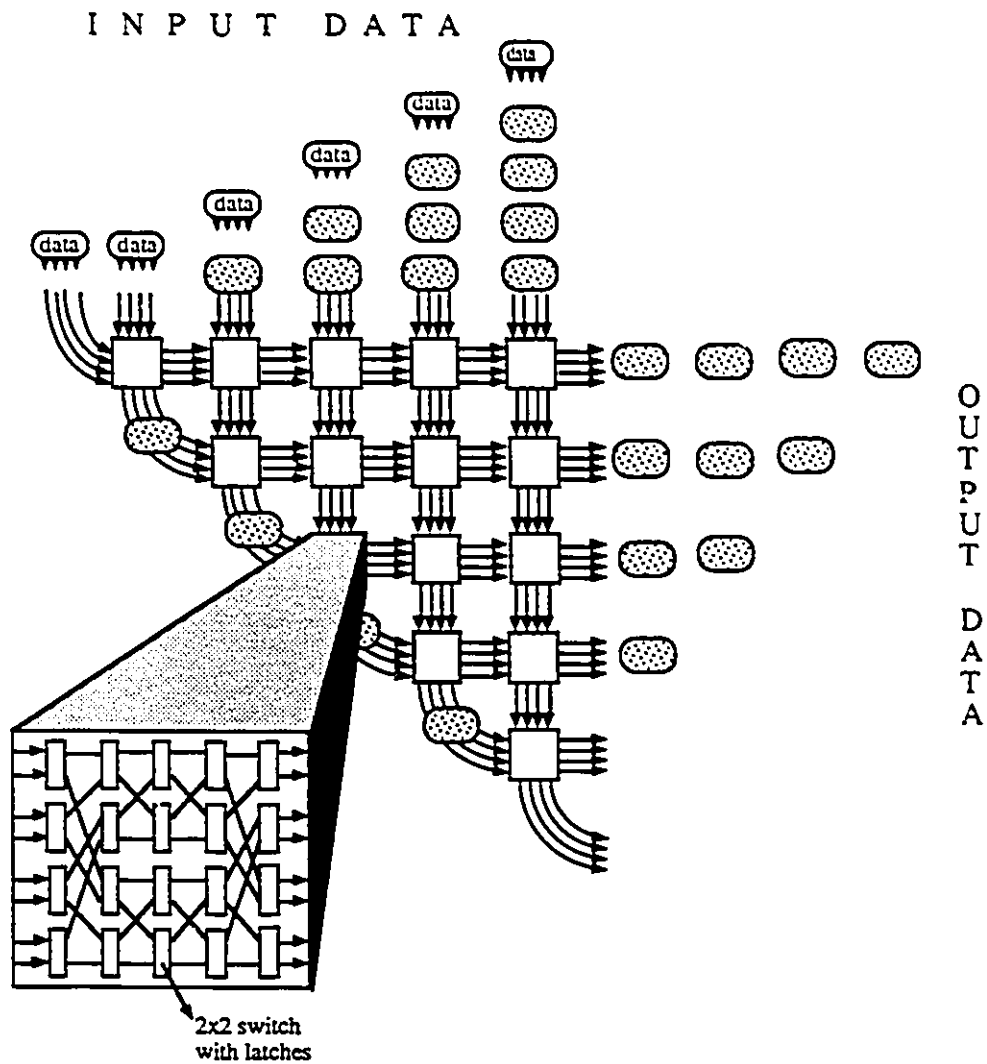
Figure 4.11 Routing network with cells of controllable complexity

The sorting network for 8 inputs is shown in Figure 4.12. The array, in general, has $N = 2n$ inputs, entering from the left. It utilizes only local connections but is different from the usual mesh pattern. It is convenient to visualize an array of this type as having two halves- the left and the right with somewhat different characteristics, as discussed below.
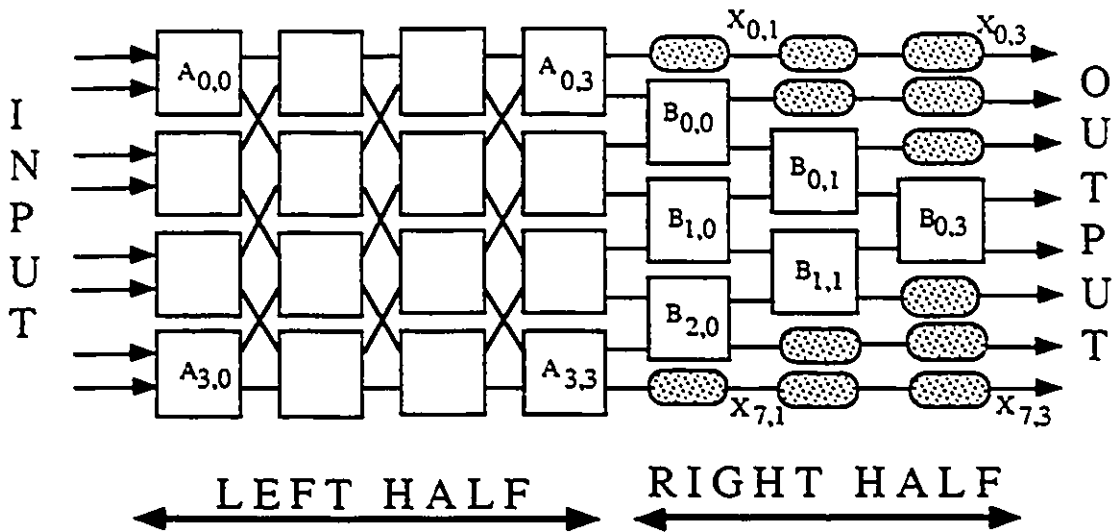
Figure 4.12 A novel parallel systolic sorting network for 8 inputs.

There are n columns of cells in the first half where each column has n cells. We will number the columns from 0 to n-1 and, within a column, we will number the cells from 0 to n-1. We will use $A_{i,j}$ to refer to the ith cell in jth column in the first half of the array. Cell $A_{i,j}$ has two inputs that come in from the left and generates two outputs from its right hand side. We will use $A1_{i,j}^{IN}$ and $A2_{i,j}^{IN}$($A1_{i,j}^{OUT}$ and $A2_{i,j}^{OUT}$) to refer to the two inputs to (output from) cell $A_{i,j}$. Cell $A_{i,j}$ compares $A1_{i,j}^{IN}$ and $A2_{i,j}^{IN}$. If $A1_{i,j}^{IN} \leq A2_{i,j}^{IN}$ ($A1_{i,j}^{IN} > A2_{i,j}^{IN}$) then $A1_{i,j}^{OUT} = A1_{i,j}^{IN}$ and $A2_{i,j}^{OUT} = A2_{i,j}^{IN}$ ($A1_{i,j}^{OUT} = A2_{i,j}^{IN}$ and $A2_{i,j}^{OUT} = A1_{i,j}^{IN}$). The outputs $A1_{i,j}^{OUT}$ and $A2_{i,j}^{OUT}$ are stored in two buffers so that they become input to the cells in the next column in the next clock cycle. The inputs to $A_{i,0}$ are supplied from outside, for all i, $0 \leq i \leq n-1$. The outputs from $A_{i,n-1}$ are input to the

65

array on the right half of the network, for all i, $0 \leq i \leq n-1$. The interconnections for cell $A_{0,j}$ and cell $A_{n-1,j}$ is different from the interconnections of other cells in column j, for all j, $0 \leq j < n$. The interconnections for cell $A_{i,j}$, $1 \leq i \leq n-1$ is given below :

- $i=0$: $A1_{0,j}^{OUT}(A2_{0,j}^{OUT})$ is connected $A1_{0,j+1}^{IN}(A1_{1,j+1}^{IN})$

- $i=n-1$: $A1_{n-1,j}^{OUT}(A2_{n-1,j}^{OUT})$ is connected $A1_{n-2,j+1}^{IN}(A2_{n-1,j+1}^{IN})$

- $i \neq 0$ and $i \neq n-1$: $A1_{i,j}^{OUT}(A2_{i,j}^{OUT})$ is connected $A2_{i-1,j+1}^{IN}(A1_{i+1,j+1}^{IN})$

There are n-1 columns of cells in the second half where the jth column has n-j-1 cells, for all j, $0 \leq j < n$. In addition, the second half also contains a number of delay elements represented by shaded ovals. The purpose of the delay element is to ensure the integrity of the wave front, since the top and bottom outputs from column j do not need to be rerouted any further. We will number the columns and rows as before. We are ignoring the delay elements and will use $B_{i,j}$ to refer to the ith cell in jth column in the second half to the array. Cell $B_{i,j}$ has two inputs that enter from the left and generates two outputs from its right hand side. Using a notation similar to that for the left half, $B1_{i,j}^{IN}$ and $B2_{i,j}^{IN}$ ($B1_{i,j}^{OUT}$ and $B2_{i,1}^{OUT}$) refer to the two inputs to (outputs from) cell $B_{i,j}$. Cell $B_{i,j}$ compares $B1_{i,j}^{IN}$ and $B2_{i,j}^{IN}$. If $B1_{i,j}^{IN} \leq B2_{i,j}^{IN}$ ($B1_{i,j}^{IN} > B2_{i,j}^{IN}$) then $B1_{i,j}^{OUT} = B1_{i,j}^{IN}$ and $B2_{i,j}^{OUT} = B2_{i,j}^{IN}$ ($B1_{i,j}^{OUT} = B2_{i,j}^{IN}$ and $B2_{i,j}^{OUT} = B1_{i,j}^{IN}$). The inputs to $B_{i,0}$ are supplied from the left half of the array. The interconnection is such that $B1_{i,j}^{OUT}(B2_{i,j}^{OUT})$ is connected $B2_{i+1,j}^{IN}(B1_{i+1,j+1}^{IN})$

Now that we described the interconnection of this network, we introduce another notation to refer to the output lines in the second half of the array; this notation will help us prove the functionality of the array. We note that the right half of the network has n-1 columns of cells and that there are always 2n output lines. We number the columns 1, ... , n-1 and number the output lines from top to bottom in the sequence 0, 1, ... , 2n-1. The output of

66

the left half will be termed column 0. We will use $X_{i,j}$ to refer to the ith column ($0 \le i < n$) and jth line ($0 \le j < 2n$). The relationship between $X_{i,j}$ and $B1_{i,j}{}^{IN}$ and $B2_{i,j}{}^{OUT}$ is straightforward. Figure 4.13 illustrates this array on a sort of 8 numbers.
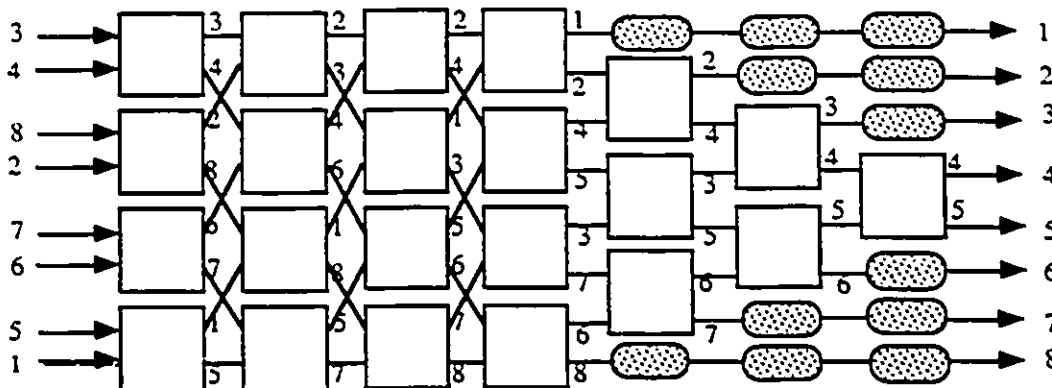
Figure 4.13 An illustration for eight inputs sorting.

Theorem 4.2

The systolic array shown in Fig 4.12 is a sorting network.

Proof:

To prove that the network sorts properly, let us assume that, at time t_0, a list of distinct numbers $[a_0, a_1, \dots, a_{2n-2}, a_{2n-1}]$ is input to the network. We have to prove that, after an appropriate latency period, an output in the form of a list $[b_0, b_1, \dots, b_{2n-2}, b_{2n-1}]$ is generated from the right side of the network, where b_i appears in the list $[a_0, a_1, \dots, a_{2n-2}, a_{2n-1}]$ and $b_i < b_j$, for all j, $i < j \le 2n - 1$. We first apply a transformation so that b_i is transformed to i. This is clearly permissible since this does not change the magnitude relationship of any pair of elements from the list $[a_0, a_1, \dots, a_{2n-2}, a_{2n-1}]$. Thus we have to prove that the output from the right end is the ordered list $[0, 1, 2, \dots, 2n-1]$ if, for all i, $0 \le a_i \le 2n-1$ and $a_i \ne a_j$. In other words, $X_{i,n-1} = i$, for all i, $0 \le i < 2n$.

We notice that we only have one comparator (cell $B_{n-2, 1}$) in column $n-2$ so that only $X_{n-1,n-2}$ and $X_{n,n-2}$ may differ from $X_{n-1,n-1}$ and $X_{n,n-1}$. Since $X_{i,n-1} = i$, for all i, $0 \leq i < 2n$, the following relationships must hold :

- if $i \neq n-1$ and $i \neq n$, $X_{i,n-2} = i$

- otherwise one of $X_{n-1,n-2}$ and $X_{n,n-2}$ must be $n-1$ and the other must be n.

In other words, $X_{n-1,n-2}$ (and $X_{n,n-2}$) $\in$ { $n-1, n$ }. Using a similar argument, $X_{n-2,n-3}$ (and $X_{n-1,n-3}$) $\in$ { $n-2$, $n-1, n$ }; $X_{n+1,n-3}$ (and $X_{n,n-3}$) $\in$ {$n-1, n$, $n+1$ }. If we apply induction, it is easy to determine the restrictions on $X_{i,j}$ for all j, $0 \leq j \leq n-1$. We omit this analysis since this is straight forward. This analysis for $j = 0$ (i.e., the output of the left array) shows that :

- $X_{0,0} = 0$

- $X_{2n-1,0} = 2n-1$

- $X_{2i-1,0}, X_{2i,0} = \{k \mid i \leq k \leq n-1+i\}, 0 < i < n$

In the left half of the array, the interconnection is uniform except for the top and the bottom. Therefore, in order to analyze what has to appear at the input of the cells in the left half, we logically subdivide the cells in three categories :

- cells in the top row

- cells in the bottom row

- all other cells.

We will consider only the first and the last category of cells since the analysis for the bottom row of cells is similar to that for the top.

In view of the above relation involving $X_{i,j}$ it is easy to see that $A1_{i,n-1}^{OUT}$, $A2_{i,n-1}^{OUT}$ must satisfy the following relationships :

$$A1_{0,n-1}^{OUT} = 0$$

$$A2_{n-1,n-1}^{OUT} = 2n-1$$

$$A1_{i,n-1}^{OUT} = \{k \mid i \leq k \leq n-1+i\}$$

$$A2_{i-1,n-1}^{OUT} = \{k \mid i \leq k \leq n-1+i\}$$

One convenient way to look at this requirement is as follows :

-   The input i must appear as $A1_{k,n-1}^{OUT}$ or as $A2_{k-1,n-1}^{OUT}$ for some k, $1 \leq k \leq i$, for all i, $1 \leq i \leq n-1$

-   The input i must appear as $A1_{k,n-1}^{OUT}$ or as $A2_{k-1,n-1}^{OUT}$ for some k, $n-1 \geq k \geq i + 1 - n$, for all i, $2n-2 \leq i \leq n$.

We will prove that $A1_{0,n-1}^{OUT} = 0$ and that the input i must appear as $A1_{k,n-1}^{OUT}$ or as $A2_{k-1,n-1}^{OUT}$ for some k, $1 \leq k \leq i$, for all i, $1 \leq i \leq n-1$. The proof for the other parts are similar.

We notice that the output $A1_{i,0}^{OUT}$ is the minimum of $A1_{i,0}^{IN}$ and $A2_{i,0}^{IN}$, therefore 0 must be $A1_{i,0}^{IN}$ for some i. Thus 0 cannot be an input to cell $A_{n-1,1}$. Proceeding in this way, we note that, regardless of wherever 0 appears in a given column, in the next column, it moves up by one, unless it happens to be in row 0. In other words, 0 advances to row 0 and stays there. Thus $A1_{0,n-1}^{OUT} = 0$.

If i has to appear in $A1_{k,n-1}^{OUT}$ or in $A2_{k-1,n-1}^{OUT}$ for any k, $1 \leq k \leq i$, for all i, $1 \leq i \leq n-1$, then in the left half of the array, as we go from column n-1 to column n-2, we find that i has to appear as an input in cell $A_{j, n-2}$ for any j, $1 \leq j \leq i + 1$. Similarly, as we go to

69

successive columns on the left ,and we reach column k, i has to appear in cell $A_{j, n-k}$ for

any j, $1 \leq j \leq i + k-n+1$. Clearly when we reach column 0, we find that i has to appear in

cell $A_{j, 0}$ for any j, $0 \leq j \leq n- 1$. In other words, i may occur in any column.

◻

4.9 COMPLEXITY OF THE ALGORITHM

The total number of comparators used in this 2n input network has n^2 in the left half and $\frac{(n-1)(n-2)}{2}$ in the right half. So far as the area and time complexities are concerned, we note that we need a total of $\frac{(3n^2-n)}{2}$ comparators for N=2n inputs. Thus the area may be approximated by $A = \frac{(3N^2-2N)}{8}$. In this estimation, we have ignored the delay elements.

If we compare this scheme to the scheme used in the S-array, we find that the number of

comparators is less; however, the asymptotic bounds are still $O(N^2)$. Since we use only

local fixed length wires for communication, the pipeline period is $O(1)$.

4.10 A NEW NETWORK FOR SORTING WITH COMPLEX CELLS

The network shown in Fig 4.14 may be generalized so that each cell, instead of including

just a comparator and latches, is a cell for sort merging. As in the network described in Fig

4.2, each cell receives two sorted lists of r inputs. The cell sort merges the inputs and

generates a pair of ordered lists of r numbers where largest element in the first list is smaller

than the smallest element of the second list. Such a network for 16 inputs is shown in

Figure 4.14.

We now have to show that the network sorts properly. This proof is very similar to

theorem 4.2 and is omitted.

4.11 DESIGN OF AN FFT SYSTOLIC ARRAY

We will now present an example of the application of our architecture technique to a well known Digital Signal Processing (DSP) algorithm; the Discrete Fourier Transform (DFT). We show that we are able to generate well known efficient realizations using our LLGS technique.
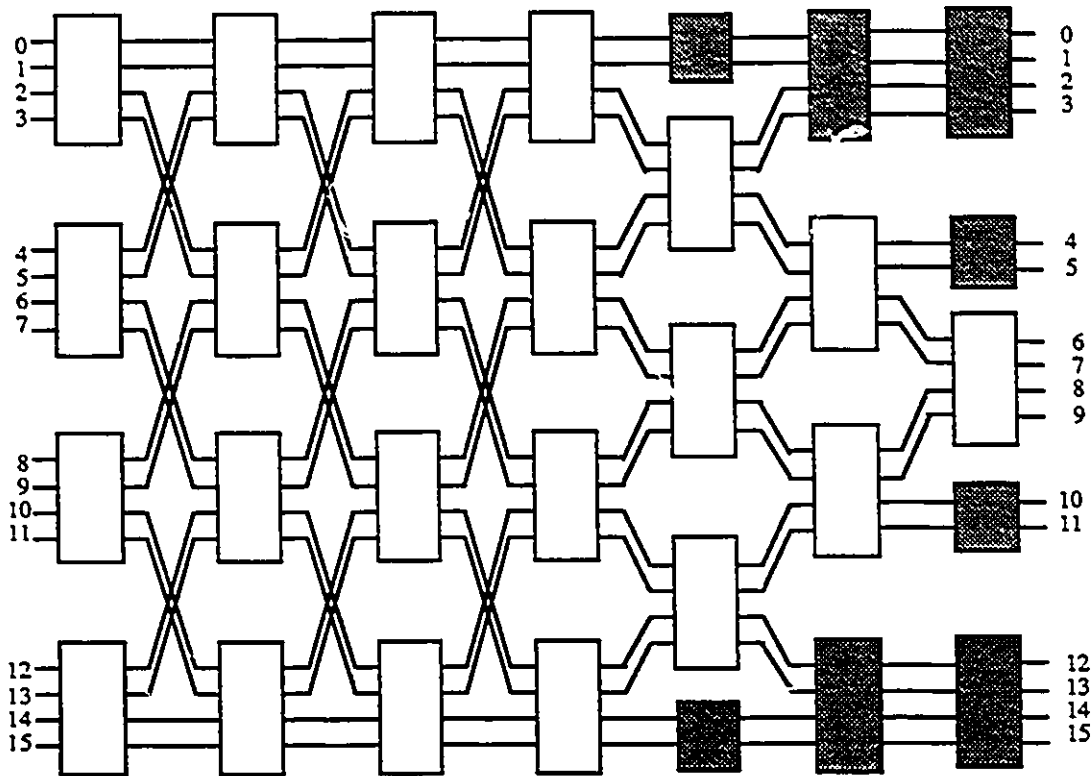
Figure 4.14 A Compact parallel systolic sorting network for 16 inputs

The DFT plays a significant role in the field of spectral analysis, and is a common tool for mapping between time and frequency domains in digital signal processing [54]. The DFT transforms an N point sampled time series into an equivalent N-point frequency series.

Let {x(n), n= 0,1,2, ... N-1} be a finite-length sequence. The DFT of x(n) is defined as

71

$$X(k) = \sum_{n=0}^{(N-1)} x(n)\, W_N^{nk} \tag{4.1}$$

for all k, $0 \le k < N$ and $W_N = e^{-j2\pi/N}$. The computation of the term within the summation sign (i.e., $x(n)\, W_N^{nk}$) requires one complex multiplication which is usually implemented with four real multiplications and two real additions. To compute $X(k)$, $0 \le k < N$ requires N^2 complex multiplications and $N(N-1)$ complex additions, i.e., $O(N^2)$ operations.

The DFT computation can be viewed as a matrix-vector multiplication as in eqn (4.2).

$$
\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ \cdots \\ X(N-1) \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & W & W^2 & \cdots & W^{N-1} \\
1 & W^2 & W^4 & \cdots & W^{2N-2} \\
1 & W^4 & W^8 & \cdots & W^{4N-4} \\
\cdots & \cdots & \cdots & & \cdots \\
1 & W^{N-1} & W^{2(N-2)} & \cdots & W^{(N-1)(N-1)}
\end{bmatrix}
\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ \cdots \\ x(N-1) \end{bmatrix}
\tag{4.2}
$$

This matrix-vector formation of the DFT can be computed using systolic arrays. The data flow diagram is shown in Figure 4.15.

Theorem 4.3:

If we are carrying out a N point DFT using systolic arrays where the largest number that we need to process is $\le N_{max}$ then

i. Area (A) of the systolic array network is $O(N^2 \log^2 N_{max})$

ii. Time (T) required for computation is $O(N \log^2 N_{max})$

iii. The pipelined period (P) of this network is $O(\log^2 N_{max})$.

Proof :

To perform this computation we need N systolic arrays where each systolic array has N cells requiring a total of N^2 cells. Each cell performs a complex multiply-add operation

which requires, in the worst case, multiplying(or adding) two complex numbers each of which is $\log N_{max}$ bits long.

$$W^{4(N-1)}$$

$$\bullet \quad W^{2(N-1)}$$

$$W^6 \quad \bullet \quad W^{N-1}$$

$$W^4 \qquad \bullet \quad 1$$

$$1 \quad W^2 \quad W^2 \quad \bullet$$

$$1 \quad W \quad 1$$

$$1 \qquad 1$$

$$1$$

$$x_3 \quad x_2 \quad x_1 \quad x_0 \longrightarrow$$

$$X_0 \qquad X_1 \qquad X_2 \qquad \qquad X_{N-1}$$

Figure 4.15 1-D Systolic array for DFT .

We assume that the pipeline period is long enough to complete one complex multiplication followed by one complex addition. It is, of course, possible to increase the throughput rate by using a completely pipelined multiplier; however, we ignore this possibility since our interest is in exploring the importance of N rather than N_{max}. Multiplication takes a longer time and is $O(\log^2 N_{max})$; the area is $O(\log^2 N_{max})$. Thus each cell has size $O(\log^2 N_{max})$ and the pipeline period is $O(\log^2 N_{max})$. The latency period is the time for the signal to

propagate through N cells, i.e., $O(N \log^2 N_{max})$ and the area for N^2 cells is $O(N^2 \log^2 N_{max})$.

$\square$

The number of operations in a Fourier transform may be reduced from $O(N^2)$ to $O(N \log N)$ by using the well known Fast Fourier Transform (FFT) [54]. We now describe, briefly, a well known algorithm which formulates $X(k)$ in terms of the sum of 2 DFT's each of which involves only $N/2$ points. This is called the decimation in time algorithm. We show that a simple variation of this algorithm allows us to design a number of linear systolic arrays of complex cells and we show that, at some point in time, such architectures may be attractive in implementing the DFT where N the number of points (also called samples) is large and we have stringent pipeline period requirements.

It is convenient to assume that $N = 2^m$ for some m. We may now restate 4.1 as follows:

$$X(k) = \sum_{n \ even} x(n) \, W_N^{nk} + \sum_{n \ odd} x(n) \, W_N^{nk} \qquad (4.3)$$

$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r) \, W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1) \, W_N^{(2r+1)k} \qquad (4.4)$$

$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r) \, W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) \, W_{N/2}^{rk} \qquad (4.5)$$

$$X(k) = G(k) + W_N^k \, H(k) \qquad (4.6)$$

The term G(k) (H(k)) is an N/2 point DFT of the even (odd) points of the original sample. By recognizing the periodicity of G(k) and H(k) we obtain the well known butterfly realization of the FFT (Fig 4.16).
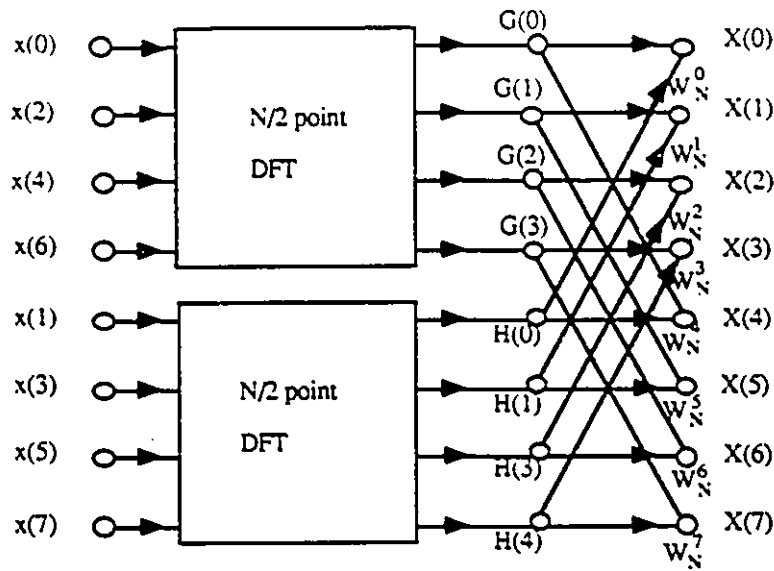
Figure 4.16 Generating FFT using a butterfly organization.

In Chapter 3, we looked at the butterfly organization. We recall that, even though a butterfly has Nlog N processors, if we wish to fabricate a butterfly using VLSI/ULSI technologies on a single die, we need area $O(N^2 A_{but})$ and pipeline period $O(N P_{but})$ where $A_{but}(P_{but})$ is the area (pipeline period) of a processor in the butterfly. A processor in the butterfly requires area $A_{but} = O(log^2 N_{max})$ and $P_{but} = O(log^2 N_{max})$.

As N increases, the fastest possible pipeline period degrades and, at some point in time, may fall below the requirements of the user. In this situation, we can use an approach similar to that used for perimeter sorting. In a nutshell, if we have to design an array for an N point FFT, we determine the size of the biggest butterfly that can operate at the speed specified by the user.

Let this butterfly have 2^m inputs. The complex cell that we use is simply a butterfly of this size. A linear systolic array of such complex cells generates 2^m transforms. To generate all the N transforms we use $N/2^m$ linear systolic arrays of complex cells.

We now describe the details of this scheme.

Let $N_1 = 2^m$, $N = N_1.N_2$. Let $n = N_1r + s$ where $0 \leq r < N_2$ and $0 \leq s < N_1$. We may rewrite equation 4.1 as follows :

$$X(k) = \sum_{s=0}^{(N_1-1)} \sum_{r=0}^{(N_2-1)} x(N_1r + s) \; W_N^{(N_1r + s)k} \qquad (4.7)$$

$$X(k) = \sum_{s=0}^{(N_1-1)} W_N^{sk} \sum_{r=0}^{(N_2-1)} x(N_1r + s) \; W_{N_2}^{rk} \qquad (4.8)$$

$$X(k) = \sum_{s=0}^{(N_1-1)} W_N^{sk} \; E_s(k) \qquad (4.9)$$

where $E_s(k) = \sum_{r=0}^{(N_2-1)} x(N_1r + s) \; W_{N_2}^{rk}$

A systolic array for a 16 point Fourier Transform is shown in Fig 4.17. We have shown only one linear array in this Figure. In order to determine all 16 transforms X(0), ... X(15) we need a total of four such systolic arrays.

This is not a new approach to computing the DFT. A network similar to that shown in Fig 4.17 is proposed in [74]. What we show is the fact that our generic architecture for design in a ULSI environment, when applied to the DFT, gives us a network which has been already implemented.

Of course, the basis for our structure is the inherent limitations on pipeline delay in a butterfly. For butterfly networks with 4 inputs the delay per cell is not likely to be the limiting criterion.
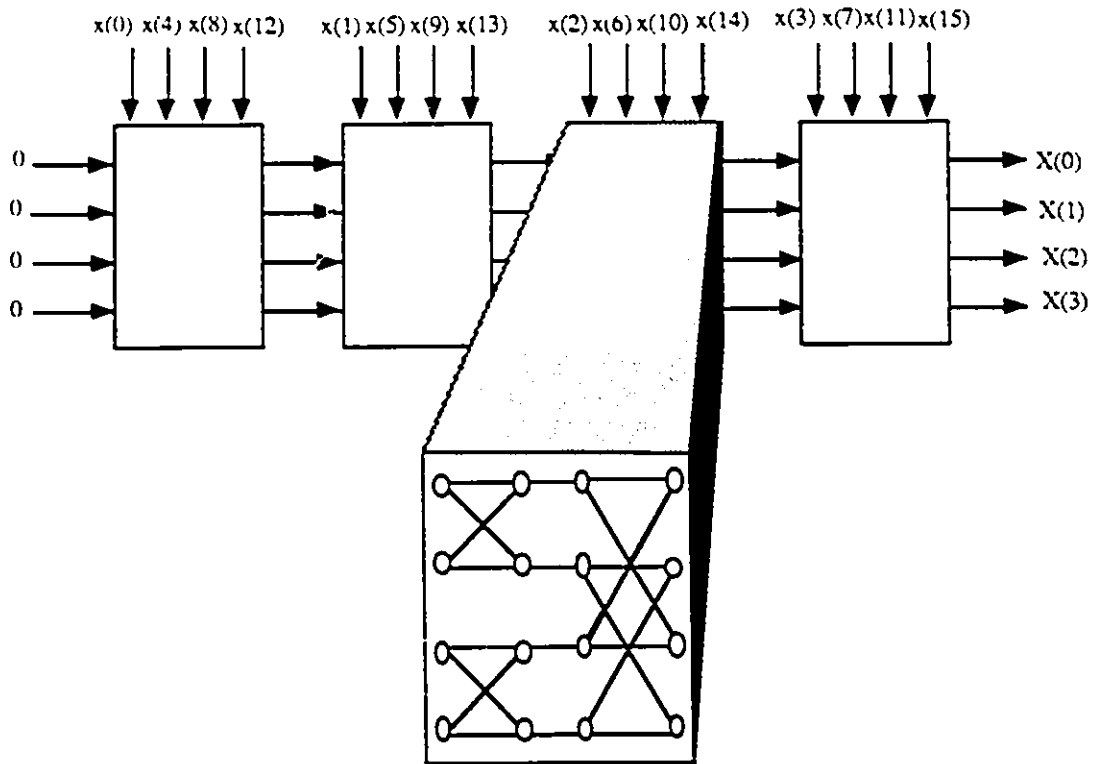
Fig 4.17   A linear array of complex cells for FFT.

4.12 SUMMARY

In this chapter we have proposed a novel architecture for supercomputing. Systolic arrays inherently have fast throughput rates due to the fact that the cells are connected only to selected local neighboring cells. There are, however, many applications which require substantial movement of data; typically, systolic architectures are not appropriate for such applications. These applications may be conveniently solved using a complex

77

interconnection structure between processors (e.g., shuffle exchange or hypercube). Such structures are expensive in terms of VLSI layout areas and have inherent limitations due to propagation delays in communication over relatively long distances. As a compromise position, in this chapter we have developed the concept of locally long but globally short (LLGS) connections. We have shown that a systolic array of relatively complex cells may solve many important problems involving significant data movement; the internal structure of these large cells may use a high wire layout. A very interesting property of this approach is the fact that the designs are flexible, so that, depending on the technology we are using and the pipeline requirements of the user, the cell size will vary from application to application.

We have examined the area of perimeter sorting in detail, using a mesh type triangular array. The same triangular array architecture is also applicable to interconnection networks. We have also looked at a different architecture for perimeter sorting. This architecture has a very interesting intercell connection and offers some savings over triangular arrays. We have also examined Fourier Transforms and have replicated existing architectures that show that it is possible to obtain throughput rates faster than potentially achievable using a hypercube alone. With the ongoing advances in VLSI fabrication technologies, it is likely that the technology limitations on long wires will be the bottleneck in future. Our complex cell technique gives us a generic approach to solve this problem.

CHAPTER
5

FABRICATION OF
VLSI
INTERCONNECTION
ARRAYS

5.1 INTRODUCTION

In this chapter, we examine experimental results in fabricating VLSI arrays using a target CMOS 1.2μm, double metal, double polysilicon, N-well process technology. Our objective is to look at the design of fully pipelined interconnection networks and find out whether systolic arrays with complex cells are potentially useful.

We first design a cell which consists of a 2×2 switch with two latches and determine its area and timing characteristics. Using this as a building block, we design a systolic array to implement an 8 input router, and use extrapolated measurements to predict the areas of any arbitrary sized array.

We follow this with the design of a butterfly network; the layout is produced using two separate approaches:

1) The first approach is to use a place and route package, where we show that the layout area increases quadratically with the number of inputs. This approach does not, unfortunately, allow us to develop an extrapolation model due to the fact that the layout varies substantially from a butterfly of one size to a butterfly of another.

2) The second approach uses a custom layout following a uniform style so that, based on measurements of this network, we develop a model for butterfly networks of any size. The model uses the area of the butterfly and the length of the longest wire; using a simple transmission line model within the HSPICE analog circuit simulator, we estimate the delay of a wire (both in metal1 and metal2 layers) for different lengths. By measuring the length of the longest wire we can now determine the expected delays in arbitrary sized butterfly networks. We show that the pipeline period of a butterfly, as expected, degrades as the number of inputs to the butterfly increases. This means that, after a certain point, our complex cell approach becomes viable. We examine different user requirements for speed and see how these affect the complexity of our cells.

5.2 DESIGNING A LATCHED SWITCH

The block diagram of a 2x2 latched switch is shown in Figure 5.1a. The circuit diagram for the latched switch is shown in Figure 5.1b. The switch is built using complementary transistors (transmission gates) and the output of the switch is latched using a dynamic true single phase clock structure [27]. The cell, consisting of both the switch and two latches, measures $114\mu \times 50\mu$ and is shown in Figures 5.2a, 5.2b &5.2c. The results of HSPICE simulations to measure the transient response of the cell are shown in Figure 5.3; they indicate that the cell is potentially capable of operating at 250 MHz.

We also need to design butterfly networks where the pipelining occurs at a higher level (latching at the outputs of complete butterfly networks). The layout of a simple switch, that can be used in such networks is shown in 5.2b; SPICE simulations indicate a delay of 0.6ns. The cell measures $40\mu \times 50\mu$. It is important to emphasize that in both the circuits

80

shown in Figures 5.2a and 5.2b, we have ignored the control signal which determines the setting of the switch.
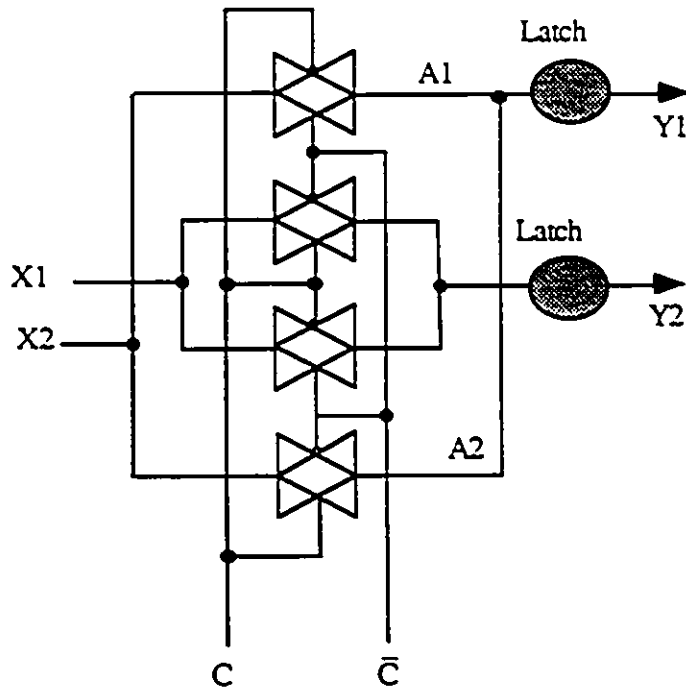

Figure 5.1 a The block diagram of a latched switch using transmission gate.

5.3 ESTIMATION OF WIRE DELAYS

In this section we discuss a model for estimating the wire delay. For simplicity we assume that the wire is fabricated over field oxide alone; the edge components of parasitic capacitance (fringing field components and wire-to-wire capacitance), which are difficult to model, have been ignored, and only the area components are considered. We feel that this simple model will give sufficient indications about the delays and, in any case, predict delays which are shorter than the actual delay. In our approach, the justification for

complex cells becomes stronger if the delay is greater. If the estimation of delay is on the low side, we are simply being rather conservative.
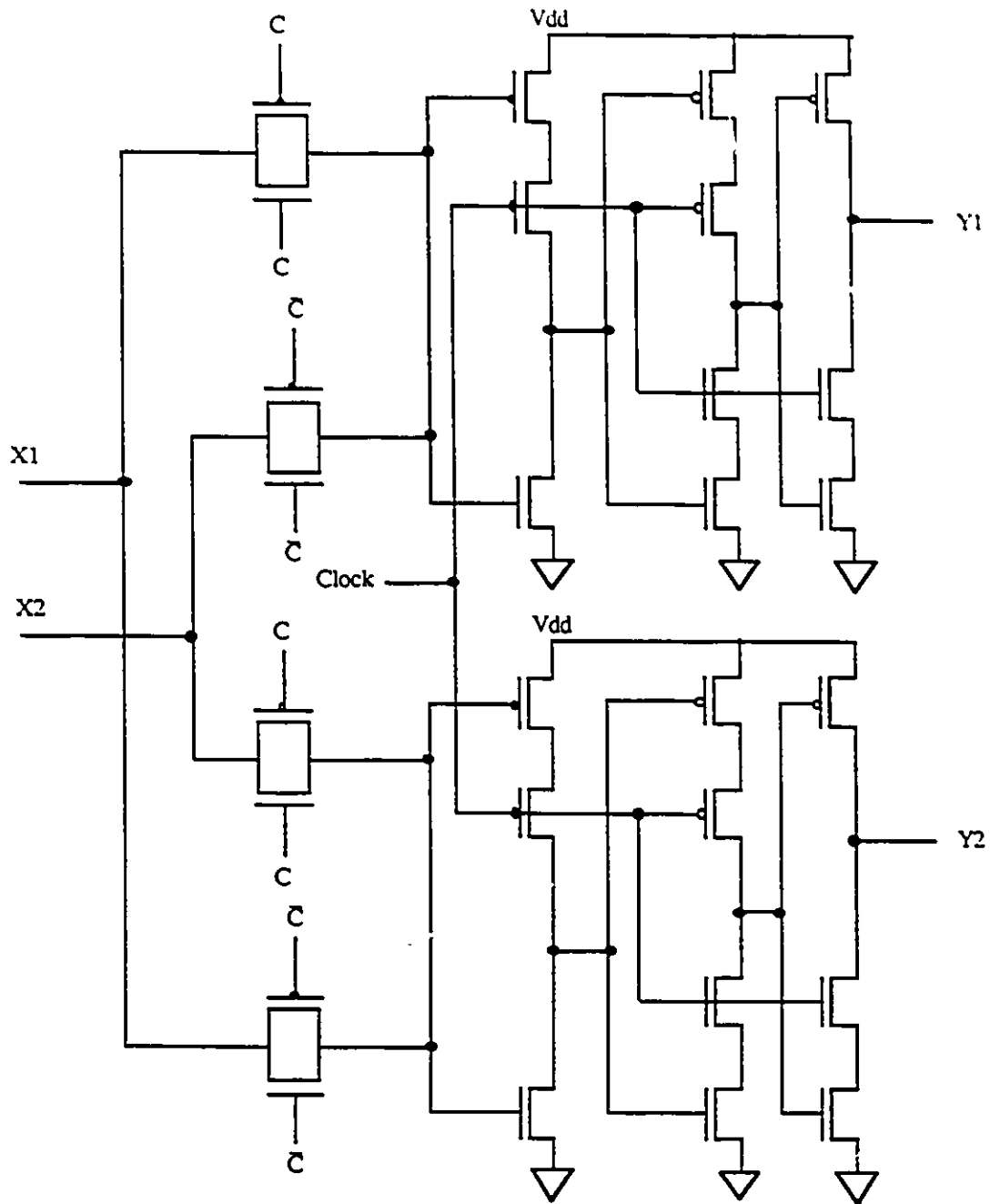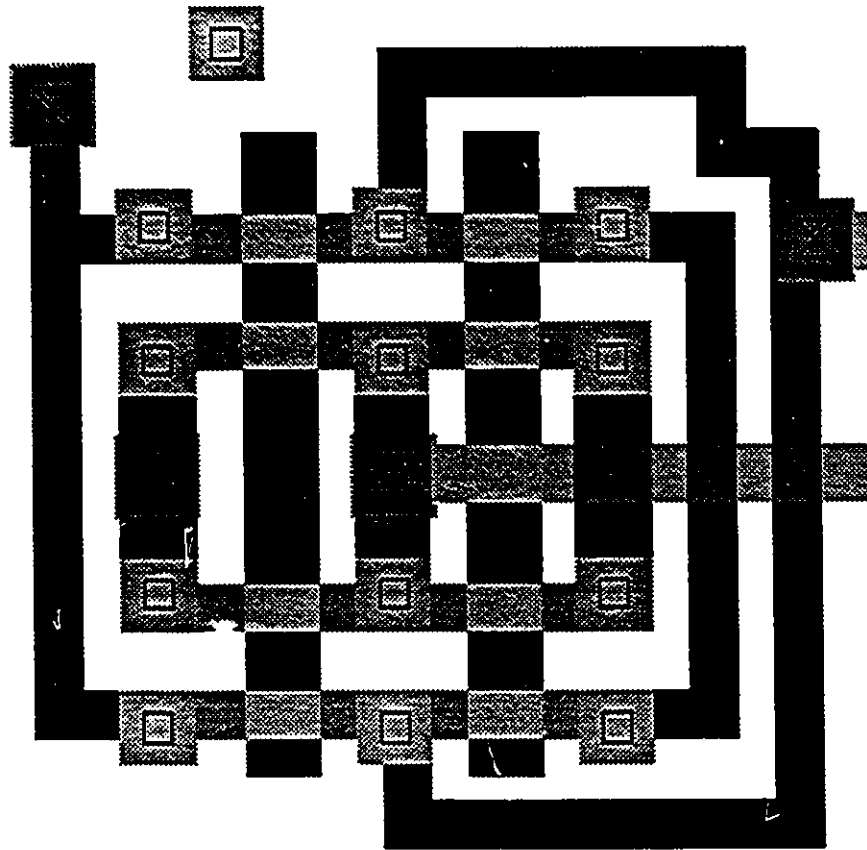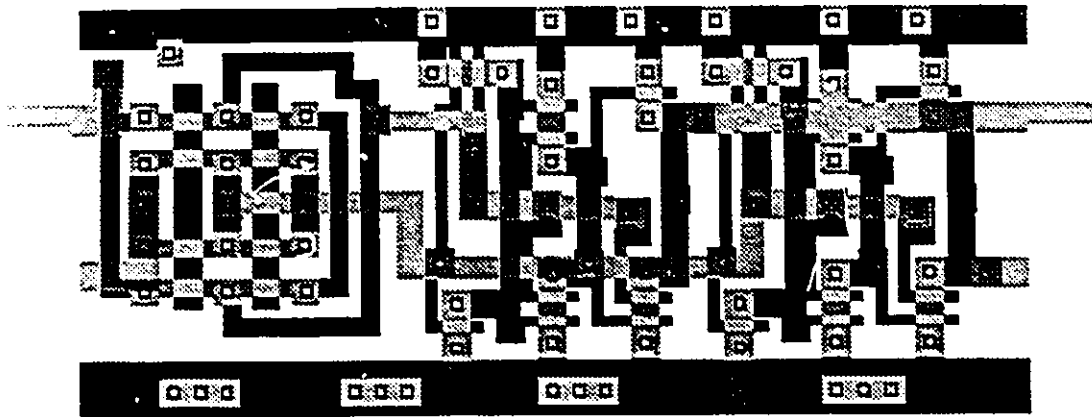
Figure 5.1b The circuit diagram of the latched switch as shown in Figure 5.1a.

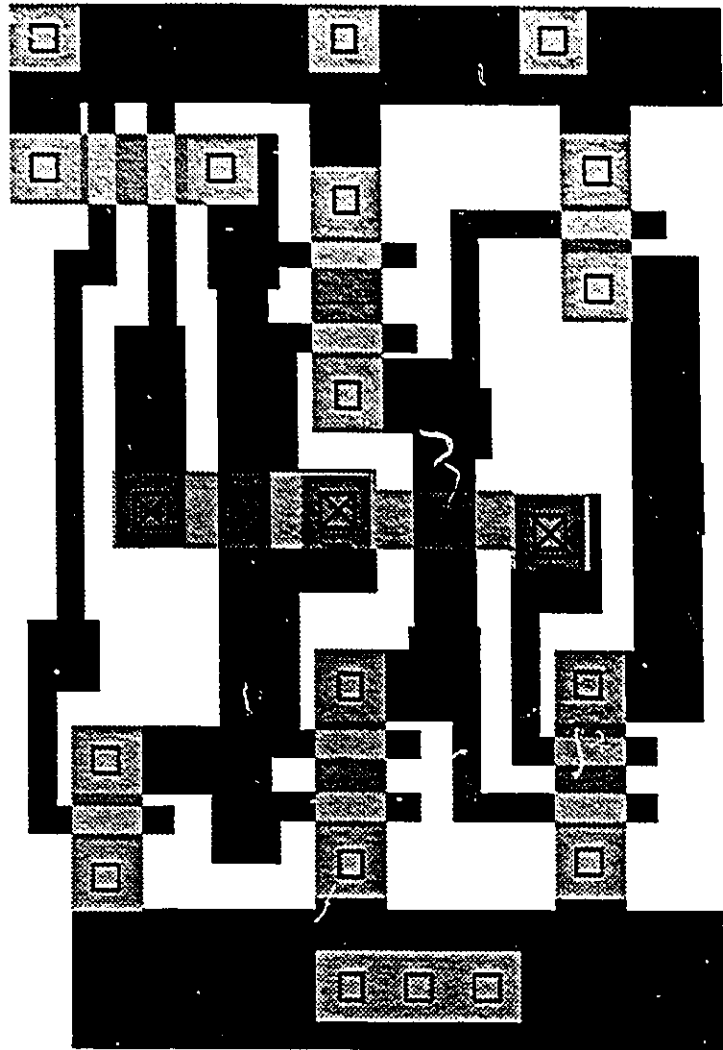Figures 5.2a & 5.2b Layout of the latched switch and switch shown in Figure 5.1b.

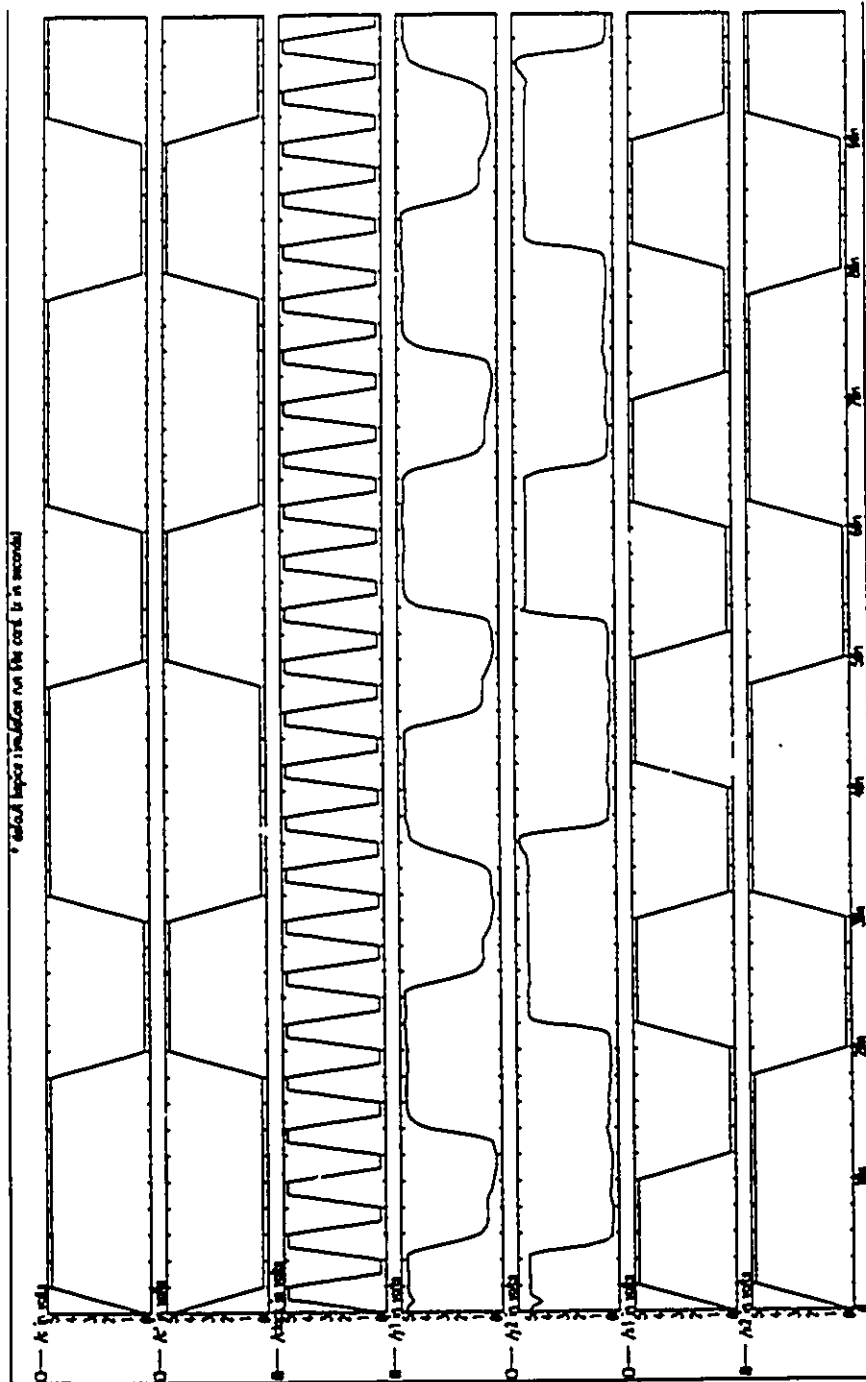Figure 5.2c Layout of the latches used in Figure 5.1a.

Figure 5.3: HSPICE simulation results for the layout shown in Figure 5.2a.

Our initial model is a transmission line model shown in Figure 5.4a, which consists of a driver transistor T_1, a transmission line and gate load T_2. The transmission line is initially charged to 5 volts, and the driver transistor T_1 is driven by a step input at time $t = 0$. The time interval during which the line voltage drops from 5 to .5 volts (90% drop) is used as the signal propagation delay along the wire. The propagation delay is confirmed using lumped resistance model (Figure 5.4b) as well as distributed resistance model (Figure 5.4c), which is included in HSPICE simulator.
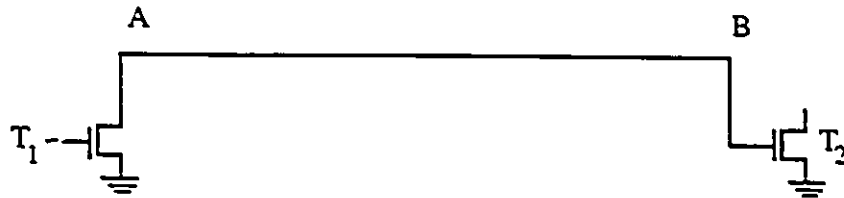

Figure 5.4 a Transmission line model for interconnect AB of gates T1 and T2.
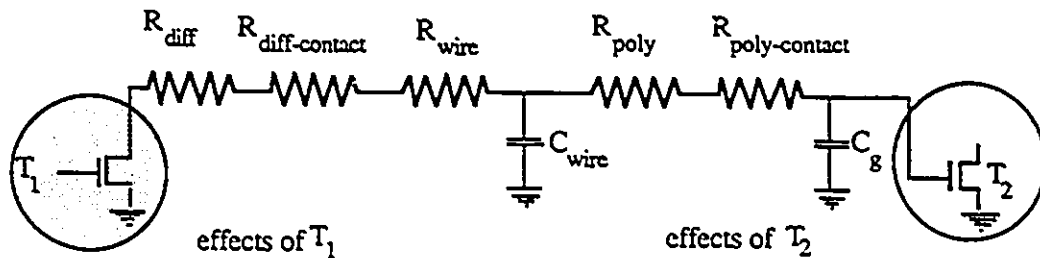

Figure 5.4 b. Transmission line lumped model for the interconnect shown in Figure 5.4a.
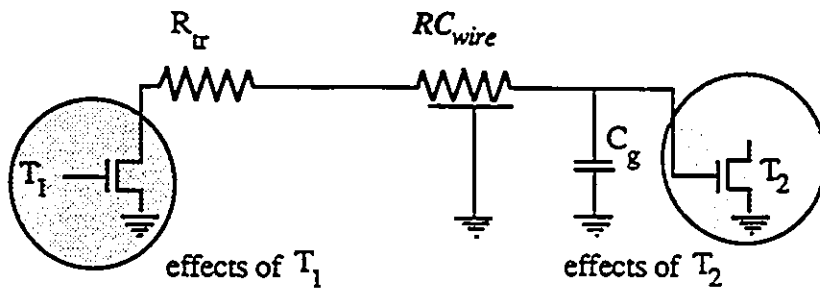

Figure 5.4 c. Transmission line of distributed model for the interconnect shown in Figure 5.4a using HSPICE wire model.

Table 5.1a: Simulated value for 3μ technology.

Length of Wire	Capacitance (pF)		Resistance (Ohms)		Delay(ns)	
(mm)	Metal 1	Metal2	Metal 1	Metal 2	Metal 1	Metal2
10	.58	.384	250	104	11	7
30	1.74	1.152	750	312.5	35	19
50	2.9	1.92	1250	520	60	32
100	5.8	3.84	2500	1040	125	65

Table 5.1b: Simulated value for 1.2μ technology.

Length of Wire	Capacitance (pF)		Resistance (Ohms)		Delay(ns)	
(mm)	Metal 1	Metal2	Metal 1	Metal 2	Metal 1	Metal2
10	.81	.42	116.67	100	13	7
30	2.43	1.26	350	300	40	22
50	4.05	2.1	583.33	500	75	39
100	8.1	4.2	1166.7	1000	160	80

This model uses representative contact resistances both at the drain of T_1 and the gate of T_2. Where R_{tr} is effective on-resistance of driver T_1, R_{wire} and C_{wire} are interconnection wire resistance and capacitance, and C_g is the input capacitance of the receiving gate T_2. We have ignored the VIA resistance since it is small $(.2\Omega)$ compared to the resistance of the wire itself. The HSPICE simulation results are shown in Table 5.1a and 5.1b. Table 5.1a shows that the delay is indeed linear and also that the delay is quite significant as the length

of wire is of the order of millimetres. As a consequence, the butterfly performance degrades significantly as the size increases.

5.4 DESIGNING A SYSTOLIC ARRAY OF SWITCHES

We show the layout of a 4×4 array of latched switches in Figure 5.5, each switch connects only to its neighbors which forms a simple 2 dimensional array of switches, as we discussed in 5.2. This array measures $460\mu \times 180\mu$. We use this value to estimate the sizes (length, width and area) of arrays of any arbitrary size (Tables 5.2). In Figure 5.5, we have packed cells without consideration for stage to stage interconnections. To realize the K-array (Chapter 3) we require an individual cell size of $130\mu \times 60\mu$, and the interconnection wire width requires an additional 10μ per cell.

Table 5.2: Relations between size and area of systolic array.

n	length mm	width mm	area mm2
4	.46	.18	.08
8	.91	.36	.32
16	1.82	.72	1.30
32	3.64	1.43	5.21
64	7.27	2.87	20.84
128	14.54	5.73	83.38
256	29.08	11.47	333.54
512	58.16	22.94	1334.14
1024	116.33	45.88	5336.46

The latency periods of the array are dependent on array size; the pipeline period, of course, is constant (independent of the size of the array). In the case of a systolic array, the interconnections are short wires and the communication delays due to the wires are negligible. Simulations show that this array is capable of operating at 250 MHz.
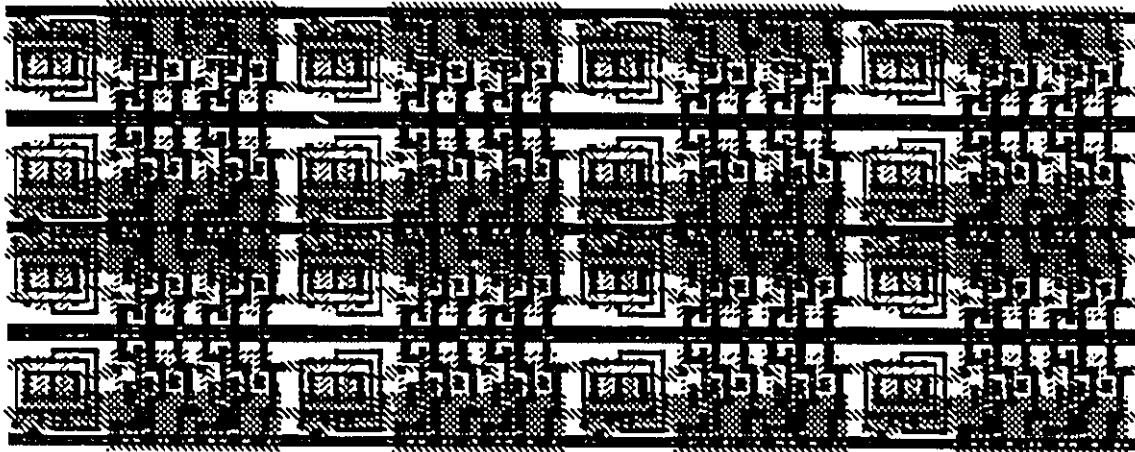
Figure 5.5: Layout for 4x4 cells of the two dimensional systolic array network.

5.5 DESIGN OF A BUTTERFLY USING AN AUTOMATIC PLACE AND ROUTE PACKAGE

We first generate a butterfly network using the ELECTRIC package running on a SUN SPARC 2 work station. We use the interactive silicon compiler (QUISC) developed at Queen's University as an interface to the ELECTRIC package. This compiler tool uses a VHDL interface to generate an internal data base netlist for the QUISC Place and Route algorithm. The compiler uses a standard cell library (user defined standard cells are referred

to as macro cells). In this case, the macro cell is the latched switch described in section 5.3 The input to ELECTRIC is in the form of a VHDL structural description of the butterfly network (Figure 5.6 a). A standard cell library is required that contains the basic level cells that the functional VHDL description uses. The silicon compiler operates within the Electric environment, and within the particular technology file appropriate for the target foundry. In this case, the technology file corresponds to the 1.2μ double metal, double polysilicon CMOS technology at the Northern Telecom VLSI fabrication facilities. The VHDL compiler takes this input and generates a netlist. For efficient layouts, the user has to choose the number of rows of cells in the final network. We naturally choose four (eight) rows for a four (eight) input butterfly. Figure 5.6b (5.6c) shows the layouts generated by the package for 4 (8) input butterfly networks. The four (eight) input network measures .62mm × .58mm (1.53mm × .76mm).

```
package registers is
          type bit_vector is array( 1 to 16) of bit;
end registers;
use registers.all;
entity bfly8 is
          port(ins: in bit_vector;outs: out bit_vector);
end bfly8;
architecture structure of bfly8 is
          component switch2 port(a,b: in bit;c,d: out bit);
          end component;
          signal s1,s2,s3: bit_vector;
begin
          u1:  switch2 port map(ins(1),ins(2),s1(1),s1(2));
          u2:  switch2 port map(ins(3),ins(4),s1(3),s1(4));
          u3:  switch2 port map(ins(5),ins(6),s1(5),s1(6));
          u4:  switch2 port map(ins(7),ins(8),s1(7),s1(8));
          u5:  switch2 port map(ins(9),ins(10),s1(9),s1(10));
          u6:  switch2 port map(ins(11),ins(12),s1(11),s1(12));
          u7:  switch2 port map(ins(13),ins(14),s1(13),s1(14));
          u8:  switch2 port map(ins(15),ins(16),s1(15),s1(16));
          u9 : switch2 port map(s1(1),s1(4),s2(1),s2(2));
          u10: switch2 port map(s1(3),s1(2),s2(3),s2(4));
          u11: switch2 port map(s1(5),s1(8),s2(5),s2(6));
          u12: switch2 port map(s1(7),s1(6),s2(7),s2(8));
          u13: switch2 port map(s1(9),s1(12),s2(9),s2(10));
          u14: switch2 port map(s1(11),s1(10),s2(11),s2(12));
          u15: switch2 port map(s1(13),s1(16),s2(13),s2(14));
          u16: switch2 port map(s1(15),s1(14),s2(15),s2(16));
          u17: switch2 port map(s2(1),s2(6),s3(1),s3(2));
          u18: switch2 port map(s2(3),s2(8),s3(3),s3(4));
          u19: switch2 port map(s2(5),s2(2),s3(5),s3(6));
          u20: switch2 port map(s2(7),s2(4),s3(7),s3(8));
          u21: switch2 port map(s2(9),s2(14),s3(9),s3(10));
          u22: switch2 port map(s2(11),s2(16),s3(11),s3(12));
          u23: switch2 port map(s2(13),s2(10),s3(13),s3(14));
          u24: switch2 port map(s2(15),s2(12),s3(15),s3(16));
          u25: switch2 port map(s3(1),s3(10),outs(1),outs(2));
          u26: switch2 port map(s3(3),s3(12),outs(3),outs(4));
          u27: switch2 port map(s3(5),s3(14),outs(5),outs(6));
          u28: switch2 port map(s3(7),s3(16),outs(7),outs(8));
          u29: switch2 port map(s3(9),s3(2),outs(9),outs(10));
          u30: switch2 port map(s3(11),s3(4),outs(11),outs(12));
          u31: switch2 port map(s3(13),s3(6),outs(13),outs(14));
          u32: switch2 port map(s3(15),s3(8),outs(15),outs(16));
end structure;
```

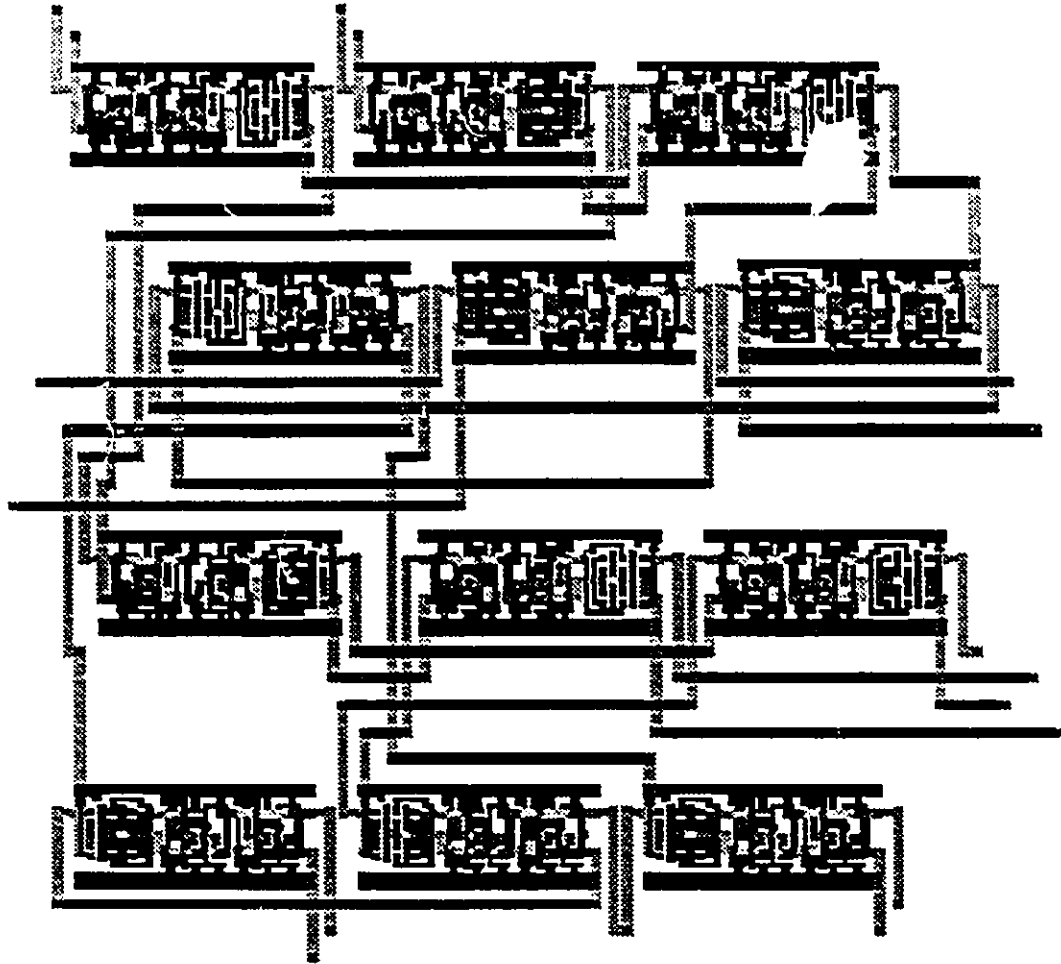Figure 5. 6 a VHDL code for the automatic layout.

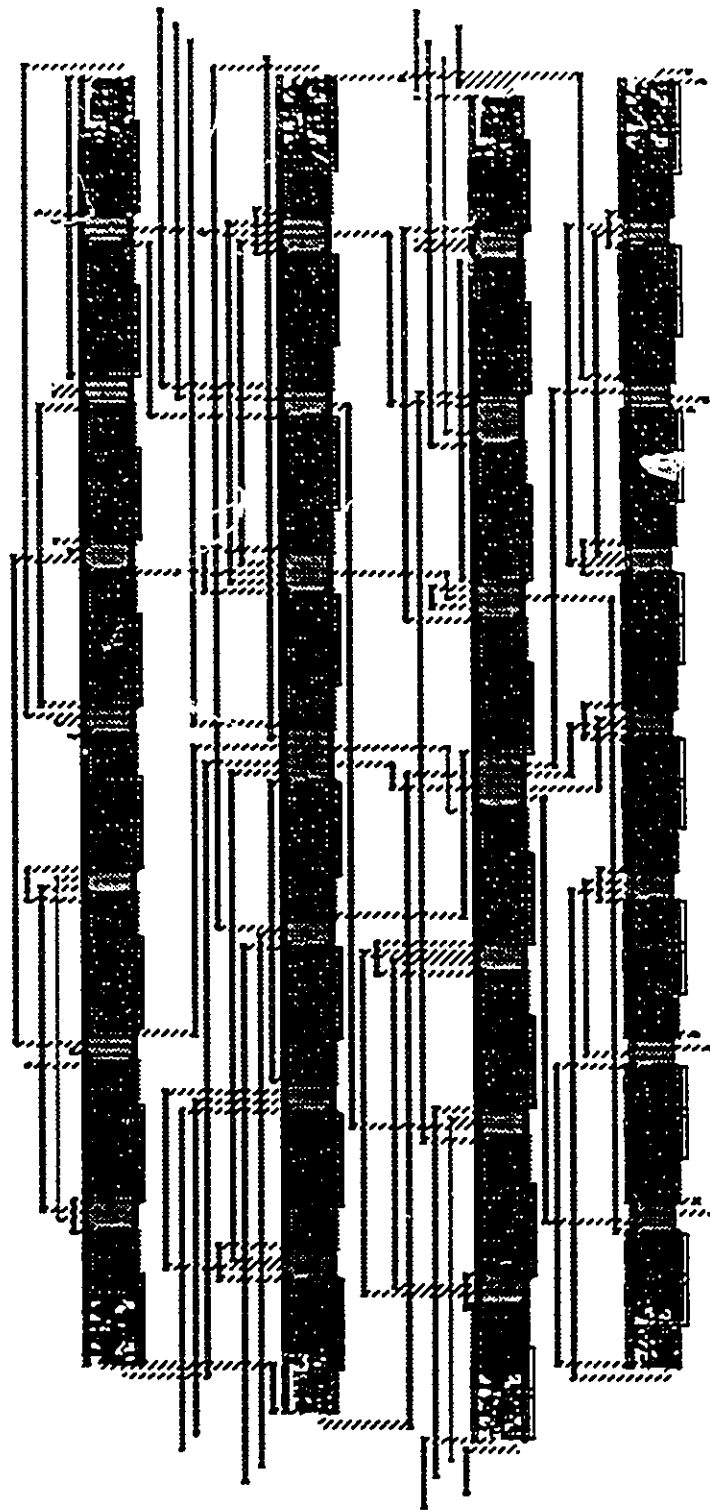Figure 5.6b Layout generated by QUISC for an eight input butterfly network.

Figure 5.6c Layout generated by QUISC for a sixteen input butterfly network.

5.6 DESIGN OF A BUTTERFLY USING CUSTOM LAYOUT

We now describe the network that we will use for the purpose of comparison with mesh type systolic layouts. We use the CADENCE software environment to obtain the cell layouts. An eight input butterfly is shown in Figure 5.7.

To implement this butterfly, we use the cell described in Figure 5.2b as the basic building block. We note that this cell does not have any latch. To use a butterfly as a cell in a systolic array, we have to use the cell shown in Figure 5.2a for the last stage of the butterfly. In other words, a butterfly with 2^m inputs has m+1 stages. The first m stages use the cell shown in Figure 5.2 b. The last stage uses the cells shown in Figure 5.2 a. We see that the j th stage of such a butterfly consists of a column of 2^m switches followed by interconnections from the output of the j th stage to the input of the j+1 th stage, for all j, $0 \leq j < m$. As expected, the area for this interconnection increases with j so that the area for interconnection is least (most) between the output of column 0 (m-1) and input of column 1 (m).

5.7 A MODEL FOR LAYOUTS OF BUTTERFLY NETWORKS OF ARBITRARY SIZE

In order to investigate whether the notion of complex cells is potentially useful, we need to look at relatively large butterfly networks. The basic scheme for the layout of a butterfly is shown in Figure 5.8. This figure also gives the lengths of wires which are critical in developing our model. We measure these lengths from the layout shown in Figure 5.7.
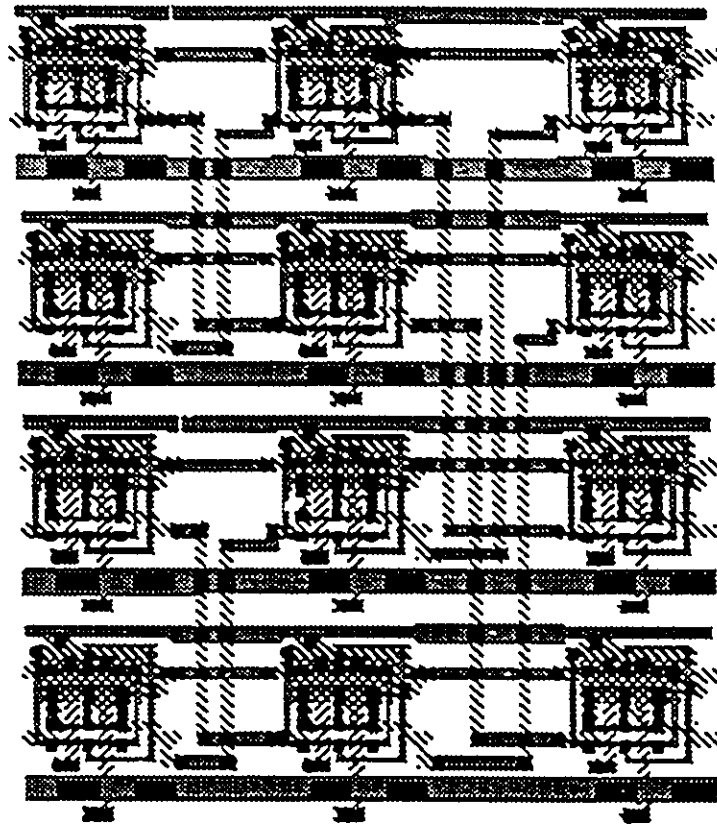
Figure 5.7 Layout of the eight input butterfly network.
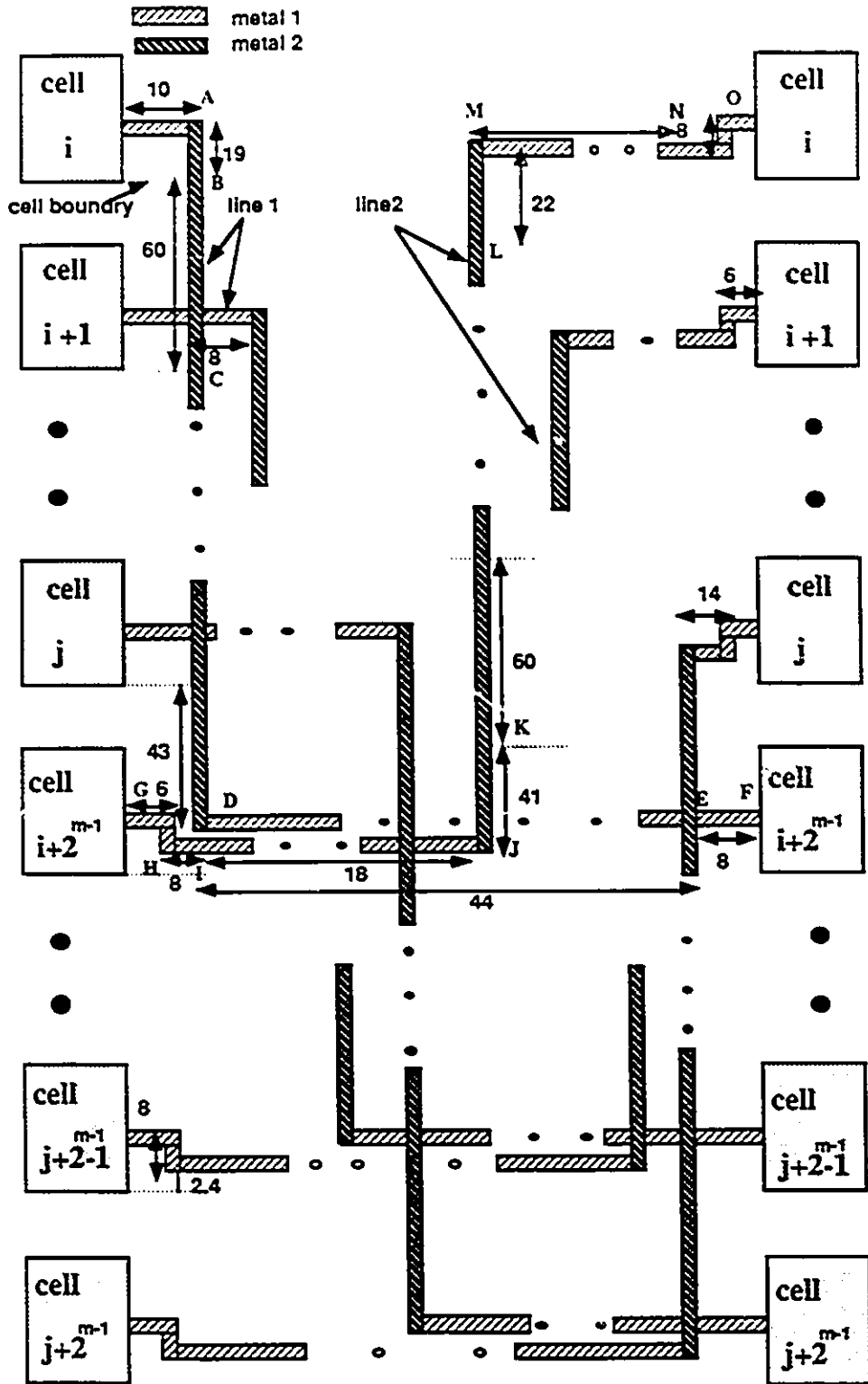
Figure 5.8 Model used to determine the wire delay in the butterfly network.
Table 5.3 a: Details of interconnect model for line1.

Line1 from cell$_i$ to cell$_{i+2^{j-1}}$, for cell i, $0 \leq i \leq 2^{j-1}$			
SEGMENT NUMBER	DESCRIPTION	LENGTH	LAYER
1	horizontal line from cell i to A	10	metal 1
2	vertical line from A to B (the end of the cell i)	19	metal 2
3	vertical line continued from B to C (the end of cell (i+2^{j-1}-1) ==> a total of 2^{j-1}-1 cells have to be crossed	60	metal 2
4	vertical line continued from C to D	43	metal 2
5	horizontal line from D to E ==> a total of 2^j-1 vertical lines here to be crossed	8 per vertical line	metal 1
6	horizontal line from E to F	8	metal 1

Table 5.3 b: Details of interconnect model for line2.

Line2 from cell i to cell i+2^{j-1}, for cell i, $0 \leq i \leq 2^{j-1}$			
SEGMENT NUMBER	DESCRIPTION	LENGTH	LAYER
1	horizontal line from cell i to G	6	metal 1
2	vertical line continued from G to H	10	metal 1
3	horizontal line continued from H to I	8	metal 1
4	horizontal line continued from I to J (2^{j-1} vertical lines in the worst case)	8 per vertical line	metal 1
5	vertical line fro J to K	41	metal 2
6	vertical line continued from K to L (2^{j-1}-1 cells here to be crossed.	60	metal 2
7	vertical line continued from L to M	22	metal 2
8	horizontal line from M to N (2^{j-1} vertical lines in the worst case)	8 per vertical line	metal 1
9	horizontal & vertical line from N to O	24 (8+8+8)	metal 1

To determine the propagation delay from the j th stage to the j+1th stage, $0 \leq j \leq m-1$, we have to consider two wires, Line1 and Line2. Each of the two wires consists of several logical segments as discussed below. Line1 is the wire connecting the output of cell (i) to input of cell $(i+2^{m-1})$ for all i, $0 \leq i < 2^{m-1}$. Line 2 is the wire connecting the output of cell $(i+2^{m-1})$ with the input of cell I(i) for all i, $0 \leq i < 2^{m-1}$.

The delays on the two lines are as follows:

$$\delta_{line1} = \left[\left\{ 10 + 8 \left(2^{j} - 1 \right) + 8 \right\} \right] \delta_{metal1} + \left[\left\{ 19 + 60(2^{j-1}-1) + 43 \right\} \right] \delta_{metal2}$$

$$\delta_{line2} = \left[\left\{ 6 + 10 + 8 + 8 \left(2^{j} - 1 \right) + 16 + 8 \right\} \right] \delta_{metal1} + \left[\left\{ 41 + 60(2^{j-1}-1) + 22 \right\} \right] \delta_{metal2}$$

Where δ_{metal1} (δ_{metal2}) is the propagation delay over a length of 1μ in metal1 (metal2).

To estimate the area of the array, we notice that the jth stage of a butterfly consists of 2^m switches and the interconnections from the output of the jth stage to the input of $(j+1)^{th}$ stage. Thus the length of the jth stage is 60.2^m and width of a cell is 40μ. The width of interconnection is the length of the wire line1 in metal1.

Thus the area for stage j is $60.2^m \{16 + 12(2^{j}-1)\}$, for all j, $0 \leq j < m$. To determine the area of the complete chip, we have to add the area of the latched switches in the last stage to the area of stage j, for all j, $0 \leq j < m$.

$$A_{layout} = \sum_{j=0}^{(m-1)} \left[60.2^m \left\{ 23 + 8(2^{j}-1) \right\} \right] + 60.2^m \ 113$$

$$= 60.2^m \left[\sum_{j=0}^{(m-1)} \left\{ 23 + 8(2^{j}-1) + 113 \right\} \right].$$

Table 5.4. Shows the delay and area of the butterfly for various values of inputs. Figure 5.9 shows how the areas and delays of butterfly networks compare with those for systolic networks of the type shown in Figure 5.5.

Figure 5.9 a is useful for the asymptotic values. Figure 5.9 b gives values for networks of practical size in terms of current technology. We immediately see that so far as the area is concerned, a butterfly requires only about 1/10 the area of a systolic array; since both areas increase quadratically with the number of inputs, the situation will not improve when we consider networks with more inputs. The pipelined period for butterfly networks, however, degrades linearly, while the pipeline period of the systolic network is independent of the number of inputs.

Table 5.4: Empirical relationships for the butterfly network.

Inputs	line1 (up) ns	line2 (down) ns	wire-area length mm	wire-area width mm	network width mm	network area mm^2	delay ns
4	0.17	0.08	.12	.07	.19	.02	1.37
8	0.29	0.21	.24	.1	.29	.07	2.25
16	0.52	0.45	.48	.17	.46	.22	3.38
32	1.00	0.92	.96	.3	.75	.72	4.98
64	1.95	1.88	1.92	.55	1.31	2.51	7.53
128	3.86	3.79	3.84	1.06	2.37	9.10	11.99
256	7.68	7.59	7.68	2.09	4.46	34.23	20.26
512	15.3	15.23	15.36	4.14	8.6	132.0	36.17
1024	30.56	30.48	30.72	8.23	16.83	516.88	67.33
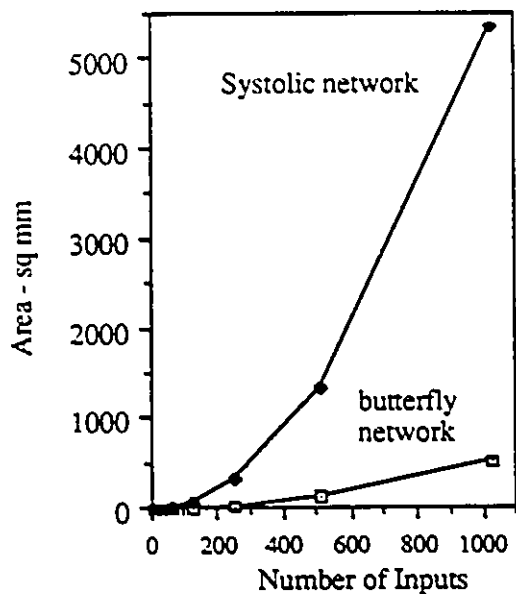
Figure 5.9a. Area of the butterfly network and systolic arrays for a large number of inputs.
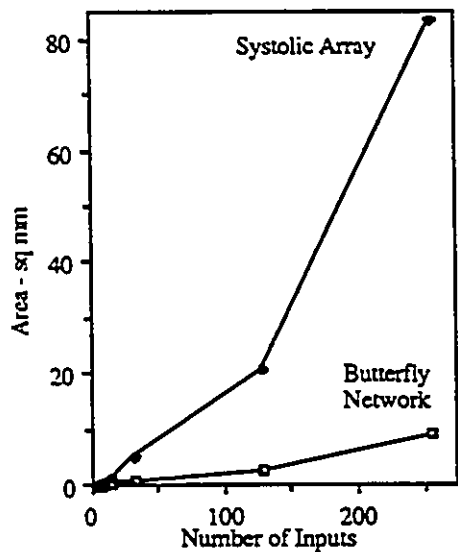
Figure 5.9b. Area of the butterfly network and systolic arrays for small number of inputs.

For our discussion, we assume that the latches in Figure 5.2a operates reliably up to 250 MHz clock rate. We also assume that if the delay of a network is δ, the maximum clock period canno: exceeded 2/δ. This assumption is merely a safe limit for the purpose of our discussions, and corresponds to the situation of dynamic pipelines of master/slave latches where we evaluate and pre charge adjacent cells within one phase of the clock.

We now consider various scenarios for designing the interconnection network.

Scenario 1: We wish to design the fastest interconnection network possible with this technology and cell library. In this case the speed of the latch is the determining factor. The maximum permissible delay is 2ns per cell. Thus the butterfly network we choose as a cell has 4 inputs, as determined from Table 5.4. The size of an interconnection network with 256 inputs, in terms of the number of cells, is (128X128-128)/2 i.e., 8128 cells. The total area required is 162.56 mm².
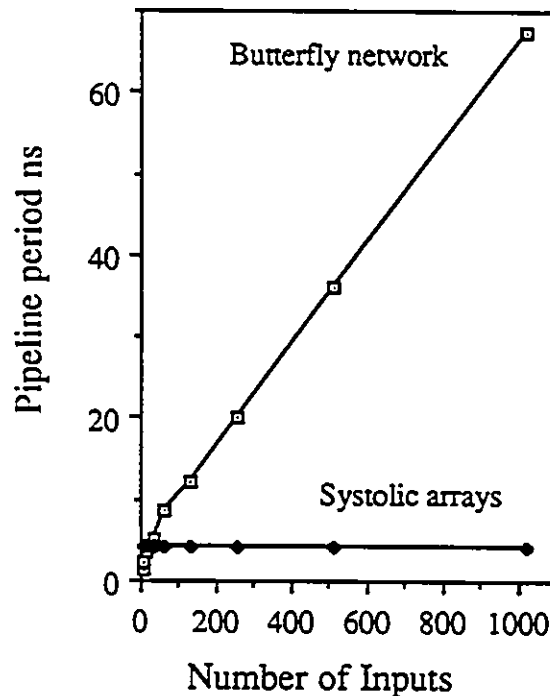
Figure 5.10 The latency of the pipelined systems of systolic array and butterfly network.

Scenario 2: We need a network which operates at 50 MHz. Here the maximum permissible delay per cell is 10 ns. Thus we may go for a butterfly of size 2^6 = 64 inputs. A network with 256 inputs will require 28 cells where each cell is a butterfly with 64 inputs. The area of the network is 70 mm^2.

Scenario 3: We need a network which will operate at 10 MHz. Here the maximum permissible delay is 50 ns. A single butterfly of 256 inputs is adequate for this. The area of this network is 34.2 mm^2.

We observe that we traditionally have two choices to solve this problem - a systolic array of switches or a butterfly. An interconnection network (K-array) of size 256 inputs, using cells with two inputs, requires (256X256-256)/2= 65408 cells, each cell of size 130μ x 60μ, with a total area of 510 mm^2. An interconnection network of the same size of 256 inputs in a butterfly configuration requires an area or 34.2 mm^2 silicon. A systolic array satisfies the speed requirements for all three scenarios, but is more expensive in terms of silicon area than the solutions that we have proposed. A butterfly is appropriate only in the last scenario. Figure 5.11 shows that there is an optimum size of the complex cell with respect to the operating frequency.

5.8 SUMMARY

In this chapter we have tested some of our theoretical results of previous chapters using a target 1.2μ CMOS technology.
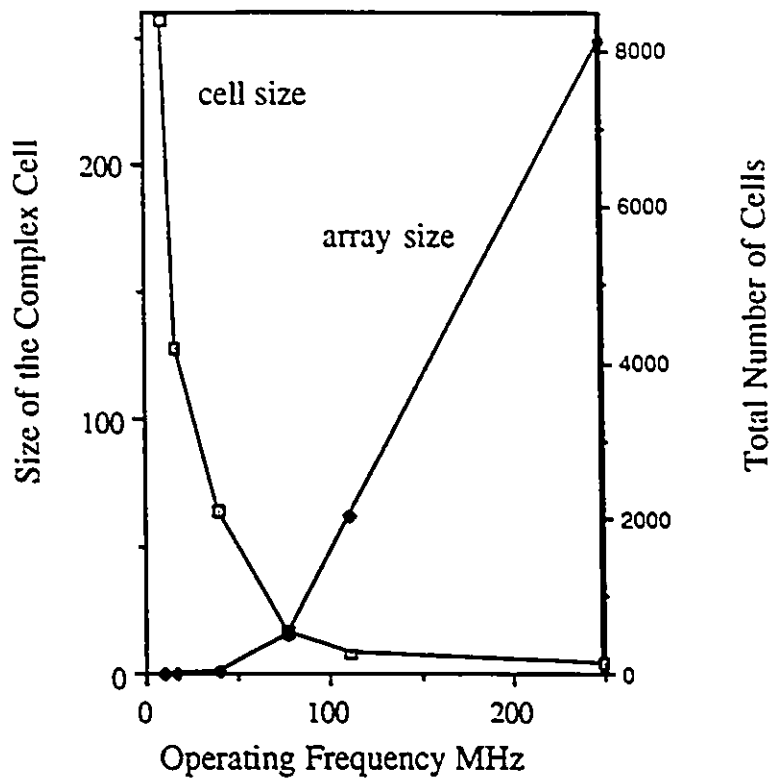
Figure 5.11 Relations between the speed of the cell and complex cell size along with array size.

We have initially designed a latched switch in the technology using mask extracted simulations to provide a measure of its performance. We have used the HSPICE transmission line model to empirically measure wire delays for arbitrary lengths; tables of values are presented. A butterfly network has been generated, using the regularity afforded by a silicon compiler running under the Electric™ CAD package. We have used detailed results from this network to provide a variety of design scenarios from which we can deduce that for time critical requirements the traditional butterfly network is not suitable. This verifies the results we had predicted from the theoretical considerations of the preceding chapters.

CHAPTER

6

CONCLUSIONS

6.1 CONCLUSIONS

The primary aim and objective of this dissertation has been the investigation of a suitable architecture and future directions for research in ULSI technology.

The main contributions of this dissertation lie in the following areas:

1. A model for ULSI computation has been recommended by reviewing and analyzing the present existing VLSI models.

2. Systolic architectures have been proposed in a ULSI environment for problems involving substantial movement of data, such as data routing and sorting applications, in addition to arithmetic intensive applications.

3. Considering the practical limitations and advantages of systolic organizatic 'ong with high wire organizations, a new, hybrid communication architecture utilizing a *locally long, globally short* connection methodology has been proposed. Our communication methodology trades the advantages of both of these two design methodologies against their disadvantages.

4. Using 1.2 μ CMOS technology, it has been verified that, even at the present level of integration, it is possible to generate both area and time efficient architectures with our new communication methodology.

6.2 DISCUSSION

In this thesis, we have investigated new architectures for problems which involve a substantial movement of data from processor to processor. Data routing in a multiprocessor, sorting and FFT architectures fall in this category. One way to implement networks to solve such problems is to use fairly complex interconnection schemes (e.g., shuffle exchange, butterfly, etc.). Such *high wire* organizations allow us to design networks with relatively few processing elements. Another, older, approach is to use a nearest neighbor connection scheme. The latter approach requires many processing elements and is not considered to be economic in terms of fabrication cost; however, such networks may be easily implemented in VLSI. This thesis has combined the two approaches into a pragmatic design technique.

In order to predict the performance and compare circuits with different characteristics, we have argued for a particular VLSI model (Chazelle's model where the wire delay is taken to be linearly proportional to its length) as being appropriate for current technology. We have shown that as we move from VLSI to ULSI, even from a conservative viewpoint, this model is quite applicable.

When implementing high wire networks in VLSI we face two difficulties:

   a. Such networks require substantial layout area for interconnection between cells.

   b. The longest wire in such networks grows linearly with time.

105

In chapter 3, we have analyzed a number of interconnection networks and sorting networks using Chazelle's model. We have analyzed and compared the asymptotic bounds and the composite performance metric of the following high wire organizations, namely the butterfly, shuffle exchange, recirculating shuffle exchange, cube connected cycle, Benes and Clos networks with the systolicized Kautz's network. We also analyzed and compared the Batcher's sorting network with Shreen's network.

We have shown that the area of these networks are comparable, but not the pipeline period and latency time. As a result, performance metrics which involve area, pipeline period and/or latency period (e.g., AT, AT^2, AP, ATP) are actually superior for systolic organizations when compared with high wire organizations. This means that, if the silicon area for fabricating circuits is allowed to be arbitrarily large, purely systolic networks will outperform high wire organizations when the networks have a large number of inputs.

Considering the practical limitation of the size of a die or a wafer, which determines the number of inputs of the network, the input size of the network further determines area, time and pipeline period. We have compared both limitations and advantages of the systolic and high wire organizations. Systolic and high wire organizations require $O(n^2)$ area, but systolic arrays requires n^2 processors, compared with $O(n\log n)$ for high wire organizations. Systolic arrays provide a constant communication overhead compared with a dominant high wire organizations. We have merged the communication advantages of systolic arrays along with an optimal number of processors of high wire organizations, into a novel communication methodology, referred as *Locally Long, Globally Short connection methodology*. This results in a new and important approach which gives users a whole class of networks to solve a given problem. This class of architectures can be further viewed as a systolic organization using smaller networks of sub cells whose area and wire delay can be controllable, and hence we refer to this as *systolic arrays with Cells of controllable complexity* .

106

We have shown that the communication architecture can be viewed as a two level hierarchy -- the intercell communication and the intracell communication. A complex cell of r-inputs is proposed as a building block with the intercell communication following a nearest neighbor (systolic) pattern, where it provides a constant pipeline period.

Using our approach the complexity of intracell interconnection will vary from application to application, depending on user requirements; thus there is a whole class of networks for solving a given problem. We have explored this basic concept in a number of application areas including sorting, interconnection networks and FFT computation.

We have proposed new sorting networks using triangular and square type arrays with our LLGS methodology. Bitonic and odd-even merge sorting are used as a basic communication structure, i.e., the local long communication architectures. These cells are connected with their nearest neighbor and/or selected nearest neighbors.

We have shown that a rearrangeable triangular interconnection network is possible which uses local Benes network and global systolic organizations.

In applying this generic architecture to FFT computation we have shown that, based on our approach, a linear systolic array of butterfly networks is an attractive solution. This solution for the FFT is not new but we have derived this structure from a different perspective - that of controlling the length of the longest wire depending on user requirements. We have also generalized our approach for FFT computation.

In each of the three application areas, we have shown that the problem may be handled by a systolic array of cells where the size and hence the complexity of cells is determined by the user requirements.

In chapter 5, we have looked at actual VLSI implementations in a case study of a butterfly network and a systolic network. We have shown that the area of the systolic network is far

greater than that of the butterfly and, as expected, both areas increase quadratically with increasing number of inputs to the network. We have modeled a standard technique for laying out a butterfly network and have used this to estimate the area, the latency period and the pipeline period of a butterfly of any arbitrary size. This technique has allowed us to predict the area, pipeline period and latency period of a systolic array where each cell is a butterfly network. It is interesting to note that, even at the current state of the art for routing applications, our approach of using a systolic array using cells of controllable complexity gives superior performance over existing solutions. We have further demonstrated how to choose different cell sizes according to the speed requirement. As a result, the interconnection area dominates the array. We expect that for applications such as sorting and FFT, we will see the same advantages when VLSI/ULSI technology allows a higher circuit density.

6.3 SUGGESTIONS FOR FUTURE WORK

In practice, metal layers do not scale in proportion to the scaling factor of channel length. For example, in the Northern Telecom 3μ CMOS technology, the metal1 and metal2 widths are both $3\mu m$, whereas in 1.2μ technology the metal1 and metal2 widths are $2\mu m$ and $2.4\mu m$, respectively. Even under such a relatively small change in technology, the assumption of ideal scaling fails. New scaling models must be developed that will predict such variations for much wider ranges of technology.

We expect, that with increasing fabrication density, more soft faults will occur in ULSI systems on silicon. Fault tolerance aspects of controllable complexity architectures are important to identify; this will be a fruitful area of future study.

Testability aspects of locally long, globally short connection methodology or cells of controllable complexity architectures are also an important and fruitful area for future study.

Wafer Scale Integration (WSI) provides very large silicon real estate and we project that controllable complexity architectures will be powerful structures in this fabrication medium. Such work will be further enhanced by expanding the types of bit level implementation within these architectures.

Multi Chip Modules (MCM) is another important area ,which effectively provides very large silicon area for processing, where the concept of locally long, globally short connections is significant.

Finally, it would be interesting to expand our architectural structures to a wide class of high speed arithmetic computation problems.

REFERENCES

1. Abelson, H., "Information Transfer and Area-Time Tradeoffs for VLSI Multiplication", *Communications of ACM.*, Vol. 23, No.1, pp 20-23, 1980.

2. Baccarani, G., M. R. Wordeman and R. H. Dennard, "Generalized Scaling Theory and its Applications to a 1/4 micrometer MOSFET Design", *IEEE Trans. on Electron Devices*, Vol. ED 4, No 4, pp 452-462, 1984.

3. Bakoglu, H. B., "Circuits, Interconnections, and Packaging for VLSI", Addison-Wesley Publishing Company, 1991.

4. Bakoglu, H. B. and J. D. Meindl, "Optimal Interconnection Circuits for VLSI," *IEEE Trans. on Electron Devices.* Vol. ED-32, pp 903-909, 1985.

5. Batcher, K. E., "Sorting Networks and Their Applications," AFIPS Spring Joint Computer Conference, Vol.32, pp 307-314, 1968.

6. Baudet, G. "On the Area Requirement by VLSI Circuits", VLSI Systems and Computations, H.T. Kung ed. Computer Science Press, 1981.

7. Bilardi, G., M. Pracchi and F. P. Preparata. "A Critique of Network Speed in VLSI Models of Computation", *IEEE Journal of Solid-State Circuits*, Vol. SC-17, No. 4, pp 696-702, 1982.

8. Brent, R. P. and H.T. Kung, "The Area-Time Complexity of Binary Multiplication", Technical Report, Dept. of Computer Science, Carnegie Mellon University, TR-CS-79-05, 1979.

9. Bromley, K., S.Y. Kung and E. Swartzlander, "Systolic Arrays", International Conference on Systolic Arrays, 1988.

10. Brown, D. and A. Scott, "Design rules and Process Parameters for the Northern Telecom CMOS4S Process", Technical Report from CMC, 1990.

11. Chazelle, B. and L. Monier, "A Model of Computation for VLSI with Related Complexity Results", *Journal of ACM.*, Vol.32, pp 573-588, 1985.

12. Chazelle, B. and L. Monier, "A Model of Computation for VLSI with Related Complexity Results," STOC., Milwaukee,.pp 318-325, 1981.

13. Clos, C., "A Study of Nonblocking Switching Networks", *The Bell System Tech. Journal.* Vol. 32, pp 406-424, 1953.

14. Cole, R. and A. Siegel, "Optimal VLSI Circuit for Sorting", *Journal of ACM.* Vol.35, pp.777-805, 1988.

15. Dennard, R. H., F.H. Gaenesslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions", *IEEE Journal of Solid_State Circuits*, Vol. SC-9, pp 256-268, 1974.

16. Fair, R. B., "Challenges to Manufacturing Submicron, Ultra-Large Scale Integrated Circuits", *IEEE Proceedings*, Vol. 78, pp 1687-1705, 1990.

17. Feng, T., "A Survey of Interconnection Network", *IEEE Computer*, No. 12, pp12-27, 1981.

18. Fortes, J. A. B., and B. W. Wah, "Systolic Arrays", Special issue, *IEEE Computer.* No. 7, 1987.

19. Franklin, M. A., "VLSI Performance Comparison of Banyan and Crossbar Communication Networks", *IEEE Trans. on Computer*, Vol. C 30, pp 283-291, 1981.

20. Fukuma, M., "Limitations on Mos ULSIs", Symp. VLSI Technology. 1988.

21. Goke, L. R. and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessors systems", First Annual Symposium on Computer Architecture, pp 21-28, 1973.

22. Goodman, F., Leonberger, S.Y. Kung, and Athate, "Optical Interconnections for VLSI Systems", *IEEE Proceedings*, pp 850-856, 1984.

23. Gradner, D. S., J. D. Meindl, and K. C. Saraswat, "Interconnection and Electron Scaling Theory", *IEEE Trans. on Electron Devices*, Vol. ED-34, pp 633-643, 1987.

24. Hoeneisen, B. and C.A. Mead, "Fundamental Limitations in Microelectronics-I. Mos Technology", Solid-State Electronics, Vol.15, pp 819-829, 1972.

25. Hoey, D. and C. E., Leiserson, "A layout for the Shuffle Exchange Network", IEEE International Conference on Parallel Processing, 1980.

26. Hwang, K. and F. A. Briggs, "Computer Architecture and Parallel Processing", McGraw-Hill Book Company, 1984.

27. Ji-Ren, Y., I. Karlsson and C. Svensson, "A True Single-Phase Clock Dynamic CMOS Circuit Technique", *IEEE Journal of Solid State Circuits* Vol. SC-22, pp 899-901, 1987.

28. Kautz, H. W., K. N. Levitt, and A. Waksman, "Cellular Interconnection Arrays", *IEEE Transactions on Computers*, Vol.C- 17, No. 5, pp 443-451, 1968.

29. Kedem, G. and A. Zorat, "Replication of inputs may save computational Resources in VLSI," Symposium on Foundation of Computer Science, 1981.

30. Kleitman, D., F.T. Leighton, M. Lepley and G. L. Miller, "New Layouts for the Shuffle-Exchange Graph", STOC., pp 278-292, 1981.

31. Knuth, D. E., "Sorting and Searching", Art of Computer Programming, Vol 3, Addison-Wesley Publishing Company, 1973.

32. Krishnan, M. S., and J. P. Hayes, "A Normalized-Area Measure for VLSI Layouts" *IEEE Trans. on Computer-Aided Design*, Vol 7, pp 411-419, 1988.

33. Kung, H. T., "Let's Design Algorithms for VLSI Systems", Caltech Conference on VLSI, pp 65-90, 1979.

34. Kung, H. T., "Why Systolic Architectures ?", *IEEE Computer*, 1982

35. Kung, S. Y., "VLSI Array Processors", Prentice Hall, 1988.

36. Kung, S. Y., E. Swartzlander, J.A.B. Fortes and K. W. Przytuta, "Application Specific Array Processors", International Conference on Application Specific Array, 1990.

37. Kurdahi, F. J., and A. C. Parker, "Techniques for Area Estimation of VLSI Layouts." *IEEE Trans. on CAD.*, Vol.8, pp 81-92, 1989.

38. Lawrie, D. H.,"Access and Alignment of Data in an Array Processor." *IEEE Trans. on Compute,* Vol. C-24, pp 1145-1155, 1975.

39. Leighton, T. and C. E. Leiserson, "Wafer-Scale Integration of Systolic Arrays." *IEEE Transactions on Computers*, Vol. C-34, pp 448-461, 1985.

40. Leiserson, C. E., "Area-Efficient VLSI Computation", Ph.D. Dissertation, Dept. Computer Science, Carnegie Mellon University, Pittsburgh, PA., 1981.

41. Leiserson, C.E. and H.T. Kung, "Algorithms for VLSI Processor Arrays", Introduction to VLSI Systems, Mead and Conway, Addison-Wesley, 1980.

42. Lighton, F. T.," Complexity Issues in VLSI Optimal Layouts for Shuffle-Exchange Graph and Other Networks", MIT Press, 1983.

43. Lipton, J. R. and R. Sedgewick, "Lower Bounds for VLSI," STOC., pp 300-307, 1981.

44. Mead, C. A. and. L. A. Conway, "Introduction to VLSI Systems", Addison Wesley, 1980.

45. Mead, C. A. and M. Rem, "Cost and Performance of VLSI Computing Structures." *IEEE Trans. on Electron Devices*, Vol. ED-26, No. 4, pp 533-540, 1979.

46. Meindl, J. D., "Opportunities for Gigascale Integration (GSI)", Symposium on VLSI Technology, pp 1-2, 1988.

47. Meindl, J. D., "Ultra-Large Scale Integration", *IEEE Trans. on Electron Devices,* Vol. ED-31, No. 11, pp 1555-1561, 1984.

48. Moore, W., McCabe and R. Urqhart, "Systolic Arrays", The First International Workshop on Systolic Arrays,1986.

49. Muller, D. E. and F. P. Preparata, "Bounds to Complexities of Networks for Sorting and for Switching", *Journal of ACM.* Vol.22, No. 2, pp 195-201, 1975.

50. Nassimi, D., and S. Sahni, "Parallel Algorithms to set up the Benes Permutation Network", *IEEE Trans. on Computer*, Vol.C 31, pp 1148-1154, 1982.

51. Ng, K., K. and W. T. Lynch, "The Impact of Intrinsic Series Resistance on MOSFET Scaling", *IEEE Trans. on Electron Devices*, ED-34, No. 3, pp 503-511, 1987.

52. Operman, D. C., and N.T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks Part I: Control Algorithm", *The Bell System Technical Journal*, Vol.50, pp 1579-1600, 1971.

53. Operman, D. C., and N.T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks Part II: Enumeration Studies and Fault Diagnosis", Vol. 50, pp 1601-1618, 1971.

54. Oppenheim, A. V. and R. W. Schafer, "Digital Signal Processing", Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

55. Panneerselvam, G., "Three Dimensional Systolic Cubic Architecture for Simultaneous Triple Matrix Multiplication", First International Conference on Supercomputing Systems, pp 499-506, 1985.

56. Patel, J. H., "Performance of Processors-Memory Interconnections for Multiprocessors", *IEEE Transactions on Computers*, Vol. C-30, pp 771-780, 1981.

57. Pease, M. C., "The Indirect Binary n-cube Microprocessor Array", *IEEE Trans. on Computer*, Vol C-26, pp 458-473, 1977.

58. Pippenger, N.,"On Crossbar Switching Networks", *IEEE Trans. on Communications*, Vol. COM-23, pp 646-659, 1975.

59. Preparata, F. P., and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation", *Communications of ACM.*, Vol 24, pp 300-309, 1981.

60. Reddy, S. M., " Complete Test Sets for Logic Functions", *IEEE Trans. on Computer*, Vol. 22, pp 1016-1020, 1977.

61. Reisman, A., "Device, Circuit, and Technology Scaling to Micron and Submicron Dimensions", *Proceedings of the IEEE.*, Vol. 71, pp 551-565, 1983.

62. Sai-Halasz, A. G., M. R. Wordeman, D. P. Kern, E. Ganin, S. Rishton, D.S. Zicherman, H.Schmid, M.R. Polcari, H.Y. Ng, P.J. Restle, T.H.P. Chang and R.H. Dennard, "Design and Experimental Technology for 0.1-μm Gate-Length Low-Temperature Operation FET's", *IEEE Electron Device Letters*, Vol. EDL-8, No. 10, pp 463-466, 1987.

63. Sai-Halasz, A. G., M. R. Wordeman, D. P. Kern, E. Ganin, S. Rishton, E. Ganin, H.Y. Ng, T.H.P. Chang and R.H. Dennard, "Inverter Performance of Deep-Submicrometer MOSFET's", *IEEE Electron Device Letters*, Vol. EDL- 9, No. 12, pp 633-635, 1988.

64. Saraswat, K. C., and F. Mohammdi, "Effects of Scaling of Interconnections on the Time Delay of VLSI Circuits", *IEEE Trans. on Electron Devices*, Vol.ED 29, pp 645-650, 1982.

65. Sasaki, H., "Directions and Strategies to Achieve ULSI", Symposium on VLSI Technology, pp 3-6, 1988.

66. Savage, J., "On the Complexity of VLSI Computations", VLSI Systems and Computations, ed. H.T. Kung, et.al., Computer Science Press, 1981.

67. Savage, J. E., "Area- Time Tradeoff for Matrix Multiplication and related Problems in VLSI Models", *Journal of ACM.*, Vol 22, pp 230-242, 1981.

68. Seitz, C. L., "System Timing", Introduction to VLSI Systems by Mead and Conway, ed. *Addison Wesley,* 1890.

69. Sheeran, M., "Designing Regular Array Architectures using Higher Order Functions", ACM FPCA. LNCS 201: pp 222- 237, 1985.

70. Siegel, H. J., "Interconnection Networks for Large-Scale Parallel Processing" Lexington Books, D.C. Health and Company, Lexington, 1985.

71. Solomon, P. M., "A Comparison of Semiconductor Devices for High-Speed Logic", *Proceedings of the IEEE.* Vol. 70, No. 5, pp 489-509, 1982.

72. Stone, H. S., "Parallel Processing with the Perfect Shuffle", *IEEE Trans. on Computer*, Vol. C-20, pp 153-161, 1971.

73. Sutherland, I. E., and D. Oestreicher, "How Big Should a Printed Circuit Board Be?", *IEEE Trans. on Computers*, pp 537-542, 1973.

74. Swartzlander, E. E., "Systolic FFT Processors", The First International Workshop on Systolic Arrays, pp 133-140, 1986.

75. Swartzlander, E. E., "Systolic Signal Processing Systems", Marcel Dekker, Inc. New York, 1987.

76. Swartzlander, E. E., "Systolic Arrays for Wafer Scale Integration", International Conference on Systolic Arrays, pp 179-184, 1989.

77. Taheri, M.,"VLSI Fault-Tolerant Systolic Architectures", Ph.D., Dissertation, Dept. Elect. Engg., University of Windsor, 1988.

78. Thompson, C. D., "Area-Time Complexity for VLSI", Caltech Conference on VLSI, pp 495-508, 1979.

79. Thompson, C. D., "A Complexity Theory for VLSI", Ph.D. Dissertation, Dept. Computer Science, Carnegie-Mellon University, Pittsburgh, P.A., 1980.

80. Thompson, C. D., "The VLSI Complexity of Sorting", *IEEE Trans. on Computer*, Vol. C-32, pp 1171-1184, 1983.

81. Thurber, J. K., "Interconnection Networks--A Survey and Assessment". National Computer Conference, pp 909-919, 1974.

82. Ullman, J. D., "Computational Aspects of VLSI", Computer Science Press, 1984.

83. Vuillemin, J., "Combinatorial Limit to the Computing Power of VLSI Circuits", *IEEE Trans. on Computers*. Vol. C-32, No. 12, pp 294-300, 1983.

84. Wong, J. W., and H.T. Kung," I/O Complexity: The Red-Blue Pebble Game", Annual Symp. Theory of Computation, pp 326-333, 1981.

85. Wu, C.-l., and T.-y. Feng, "On a Calss of Multistage Interconnection Networks", *IEEE Trans. on Computer*, pp 694-702, 1980.

86. Yao, C. A., "The Entropic Limitations on VLSI Computations", STOC, pp 308-311, 1981.

87. Zhou, D., F.P.Preparata, S. M. Kang, "Interconnection Delay in Very High-Speed VLSI", Vol 38, 779-790, 1991.

VITA AUCTORIS

Gopal Panneerselvam was born in Mayiladuthurai, Tamil Nadu, India in 1952. He graduated from University of Madras in 1973 with a B.Sc., in Physics. He received an M.Sc. degree from the same University in 1978. He received an M. Tech. degree from the Indian Institute of Science, Bangalore in 1980. In 1987, he received an M.Sc. degree in Electrical Engineering from University of Manitoba, Winnipeg. He is currently a candidate for the Ph. D. degree in Electrical Engineering at the University of Windsor.