The DIMOND: A Component for the Modular Construction of Switching Networks

PIERRE G. JANSEN AND JOEP L. W. KESSELS

Abstract—**The DIMOND is a building block for communication networks in which throughput is more important than transmission delay. Its main attraction stems from the fact that it allows the routing of messages through a network to be completely distributed over the individual building blocks.**

The DIMOND has two input and two output ports. A port provides for a path consisting of a number of data lines. A message arriving at an input port of the DIMOND is transferred to a designated output port where it is temporarily stored.

A functional description and an implementation of the DIMOND are given together with examples of its use in communication networks. Moreover, the construction of a FIFO buffer and a sorting network out of DIMOND's is described.

Index Terms—**Cellular logic, crossbar, distributed processing, FIFO buffer, sorting network, switching component, switching network.**

I. Introduction

ONE of the characteristic aspects of a computer architecture is the interconnection of communicating units such as processors, peripherals, and memories. In present-day computers the communication function is often implemented by means of a single switch (bus or crossbar). This implementation is in harmony with the other main functions (such as processing and storage) also being performed by single modules.

However, such an implementation of an essential function may suffer from one or more of the following shortcomings (for a detailed discussion see [1]).

1) *Vulnerability:* A failure in an essential module has a catastrophic impact on the system.

2) *Potential Performance Limitations:* A shared module can easily become the bottleneck of the system.

3) *Poor Adaptability:* In the light of specific requirements, the function often proves to have been under- or overdesigned.

These potential shortcomings argue strongly in favor of the distribution of all essential functions. Moreover, it is expected that the use of microprocessors will lead to systems with distributed processing. For such systems it is natural to distribute the communication function as well. In that context, this paper presents the DIMOND (Dual Interconnecting Modular Network Device), a switching component with two input and two output ports allowing the modular construction of communication networks. A message (containing routing information) arriving at a DIMOND is switched to a designated output port, where it is stored in a register.

Our aim in designing the DIMOND was to achieve a building block for communication networks which would be both universal and elementary. Furthermore, it should allow the routing of messages through a network to be completely distributed over the individual building blocks. The DIMOND is universal in that it comprises both the fork and the join function for messages streams. In other words, it is able to separate and merge message streams. The DIMOND is elementary in that it comprises the minimum number of ports with which these functions can be realized. Moreover, ad hoc control functions are avoided, i.e., the component comprises only those control functions which, in our view, are essential for a switching component to be generally applicable.

Manuscript received February 4, 1978; revised November 26, 1979.

The authors are with Philips Research Laboratories, Eindhoven, The Netherlands.

II. Functional Description

As stated, the DIMOND (Fig. 1) has two input (in_0 and in_1) and two output ports (out_0 and out_1). A port provides for a path consisting of a number of data lines. Each output port is connected to the output of a buffer register (reg_0 and reg_1). The input ports of the DIMOND are connected to the inputs of these buffer registers via a two-by-two crossbar. Whether or not a register holds a message is recorded in an associated status flip-flop, assuming the values *full* or *empty*.

The operation of the DIMOND is governed by signals on the following control lines.

1) *Input Control Lines:*

$creq_0$, $creq_1$: Copy request lines for in_0, in_1, respectively, indicating whether or not the corresponding input carries information (a message) to be stored in one of the buffer registers.

des_0, des_1: Destination lines for in_0, in_1, respectively, indicating the index (0, 1) of the register into which the message is to be stored.

prio: Priority line, indicating the index (0, 1) of the input to be served first in the event of contention, viz., when $creq_0 \wedge creq_1 \wedge (des_0 = des_1)$.

rel_0, rel_1: Release lines for reg_0, reg_1, respectively, indicating whether or not the status of the corresponding register is to be reset to *empty*.

2) *Output Control Lines:*

$cack_0$, $cack_1$: Copy acknowledge lines for in_0, in_1, respectively, indicating whether or not a message on the corre-

0018-9340/80/1000-0884$00.75

sponding input has been stored in the designated register. Messages are only stored when the destination register is empty.

$infa_0$, $infa_1$: Information available lines for out_0, out_1, respectively, indicating whether or not information is available on the corresponding output.

When two DIMOND's are interconnected for communication, the data lines and the control lines that perform a hand-shaking procedure are interconnected directly (Fig. 2), i.e.,

$$in_1(B) = out_1(A)$$
$$creq_1(B) = infa_1(A)$$
$$rel_1(A) = cack_1(B).$$

In the sequel such a connection will be denoted as
$input_1(B) = output_1(A)$.

III. Communication Networks

One main application field for the DIMOND is the implementation of communication networks. The routing of a message through such a network can be performed sequentially by a number of DIMOND's.

A network interconnecting n senders with n receivers will be termed an n-by-n *node*. As stated in the Introduction, the DIMOND is the most elementary (two-by-two) node. Two basic structures, loop and tree, allowing the construction of an n-by-n node out of DIMOND's, are described below and their relative merits discussed.

In the examples given, the routing information takes the form of a destination address. More precisely, a message consists of two fields: an information field and an address field. Since four-by-four nodes are constructed, the address field contains two bits (ad_1, ad_0).

1) Loop Structure: In Fig. 3 the DIMOND's are connected in a loop structure. The remaining input and output of each DIMOND are connected to a sender and receiver, respectively.

Messages offered to a DIMOND, with a destination address equal to the receiver address of the DIMOND, are switched to the receiver. All other messages are sent on to the next DIMOND in the loop (sequential search). More precisely,

$$input_0(A) = output_0(D)$$
$$input_0(B) = output_0(A)$$
$$input_0(C) = output_0(B)$$
$$input_0(D) = output_0(C).$$

The signals on the destination lines depend on the address field of the message as follows:

$$des_0(A) = \overline{ad_1} \cdot \overline{ad_0} \text{ of } in_0(A); \quad des_1(A) = \overline{ad_1} \cdot \overline{ad_0} \text{ of } in_1(A)$$
$$des_0(B) = \overline{ad_1} \cdot ad_0 \text{ of } in_0(B); \quad des_1(B) = \overline{ad_1} \cdot ad_0 \text{ of } in_1(B)$$
$$des_0(C) = ad_1 \cdot \overline{ad_0} \text{ of } in_0(C); \quad des_1(C) = ad_1 \cdot \overline{ad_0} \text{ of } in_1(C)$$
$$des_0(D) = ad_1 \cdot ad_0 \text{ of } in_0(D); \quad des_1(D) = ad_1 \cdot ad_0 \text{ of } in_1(D)$$

Moreover, in order to minimize the probability of congestion, messages within the loop are given priority over incoming messages, i.e., for A, B, C, and D: $prio = 0$.

2) Tree Structure: In a tree structure, a path between any sender and any receiver consists of as many DIMOND's as there are bits in the address field. Each DIMOND switches a message depending on one particular bit in the destination address. In the example (Fig. 4), the DIMOND's A and B switch according to the most significant bit; C and D use the remaining bit (binary tree search). More precisely,

$$input_0(C) = output_0(A); \quad input_1(C) = output_0(B)$$
$$input_0(D) = output_1(A); \quad input_1(D) = output_1(B).$$

For A and B: $des_0 = ad_1(in_0)$ and $des_1 = ad_1(in_1)$.
For C and D: $des_0 = ad_0(in_0)$ and $des_1 = ad_0(in_1)$.

In order to avoid indefinite blocking of particular message streams, it is advisable to use alternating priority signals for A, B, C, and D.

In the table given below, a comparison is made between both structures with respect to the following aspects (it is assumed that n is a power of 2).

1) The number of DIMOND's needed to build a node.

2) The average number of DIMOND's a message has to pass through between sender and receiver.

3) Is the network prone to congestion? More precisely, can the throughput of the network decrease when the number of messages in the network increases? The throughput of the tree structure will reach a saturation point, after which additional messages in the tree will have no influence on the throughput. The loop structure, however, has a maximum throughput when the registers on the loop are alternately full and empty. Every additional message on the loop will decrease the throughput.

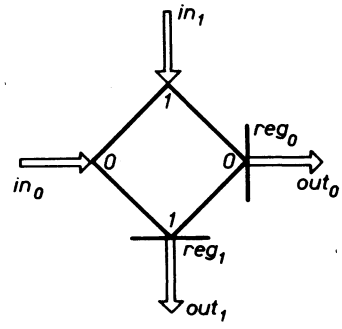


Fig. 1. Symbolic representation of the DIMOND.

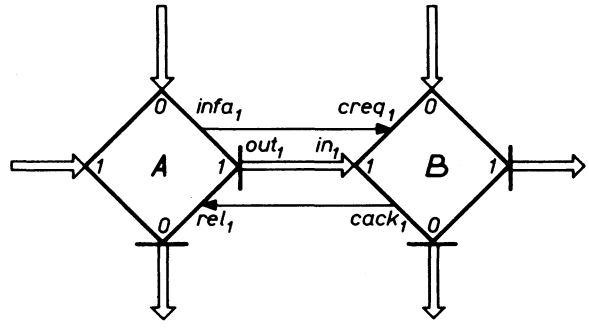


Fig. 2. Two DIMOND's interconnected for communication.

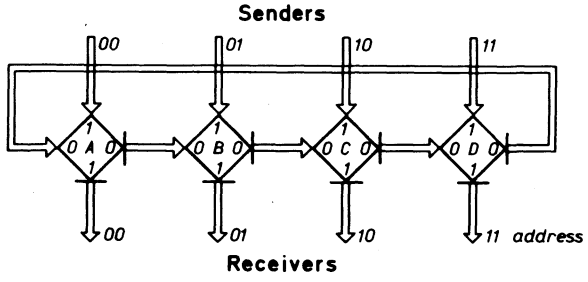


Fig. 3. Four DIMOND's in a loop structure.

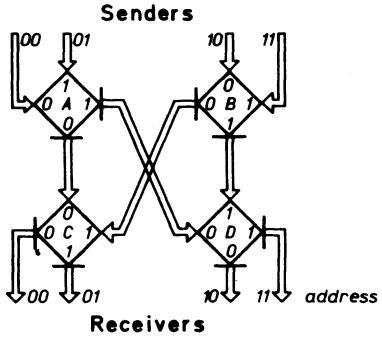


Fig. 4. Four DIMOND's in a tree structure.

In the extreme case—in which all registers on the loop contain a message—the throughput will even be zero (deadlock). Due to the fact that messages in the loop have priority over incoming messages, the deadlock situation can only arise when all senders put a message on an empty loop simultaneously. The extension of the loop with one additional DIMOND will obviously prevent such a situation from arising since one sender will never put a message on the loop.

4) The need for external circuitry to implement the routing function.

5) The number of bits needed for addresses if a receiver requires the address of the sender. In a loop structure the message then has to contain both the destination and the sender address. In the tree structure each DIMOND consumes a particular address bit not to be used again by any of its successors; hence, that bit location is available to insert one bit of the sender address. The structure can thus be made to let a message start with its destination address and to arrive with the address of its sender.

Aspect	Loop	Tree
1)	n	$(n \times \log_2 n)/2$
2)	$n/2$	$\log_2 n$
3)	*yes*	*no*
4)	*yes*	*no*
5)	$2 \times \log_2 n$	$\log_2 n$

A device connected to both an input and an output port of a node is called a *subscriber* of the node. It will be evident that n subscribers can be connected to an n-by-n node so as to allow bidirectional communication among all subscribers. Both implementations of the node allow simultaneous communication between a number of subscribers. In such applications,

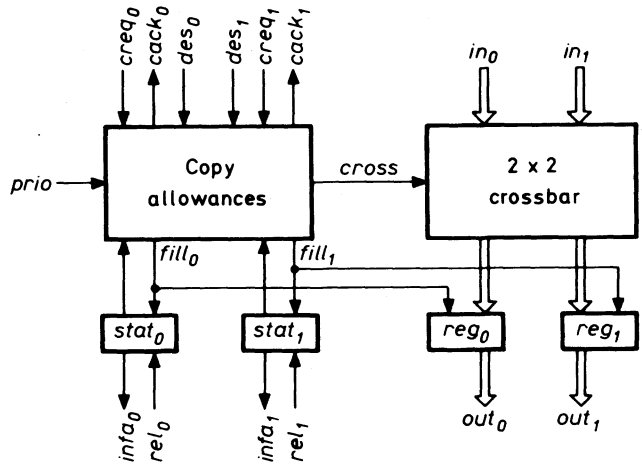


Fig. 5. Block diagram of an implementation of the DIMOND.

the loop structure has the advantage of allowing the allocation of one dedicated DIMOND to each subscriber.

For the orderly construction of composite networks, the following property is of interest; any two nodes, regardless of structure and size, can be interconnected by joining one output port of one node to one input port of the other, and vice versa. Each constituent node (with its subscribers) may then be considered as a single subscriber of the other. Thus, for example, hierarchical networks can be implemented.

IV. Implementation

Fig. 5 shows an implementation of the DIMOND which requires one central clock for all interconnected DIMOND's. The central clock has two phases.

In the first clock phase, it is determined which inputs have to be copied into which registers. The copy allowances so determined are stored in four flip-flops: c_{00}, c_{01}, c_{10}, and c_{11} (c_{01} is the allowance to copy in_0 into reg_1 etc.).

In the second clock phase, two actions are performed concurrently:

if required, inputs are copied in the output registers, and

the flipflops $stat_0$, $stat_1$ (status of reg_0, reg_1, respectively) are adapted.

More precisely, we have the following.

First Clockphase:

$$c_{00} := creq_0 \cdot \overline{des_0} \cdot \overline{stat_0} \cdot (\overline{creq_1} + \overline{des_1} + \overline{prio})$$
$$c_{01} := creq_0 \cdot \overline{des_0} \cdot \overline{stat_1} \cdot (\overline{creq_1} + \overline{des_1} + \overline{prio})$$
$$c_{10} := creq_1 \cdot \overline{des_1} \cdot \overline{stat_0} \cdot (\overline{creq_0} + \overline{des_0} + prio)$$
$$c_{11} := creq_1 \cdot \overline{des_1} \cdot \overline{stat_1} \cdot (\overline{creq_0} + \overline{des_0} + prio).$$

The following control signals are combinationally defined by the copy allowances.

copy acknowledge signals:

$$cack_0 = c_{00} + c_{01}$$
$$cack_1 = c_{10} + c_{11};$$

internal control signals controlling *the crossbar*

$$cross = c_{00} + c_{11}$$

and *the output registers*

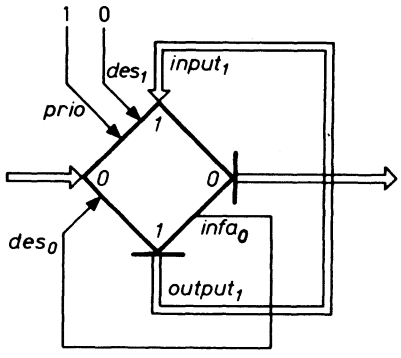


Fig. 6. FIFO buffer with two places.

$fill_0 = c_{00} + c_{10}$
$fill_1 = c_{01} + c_{11}$.

Second Clockphase: Inputs are copied into the output registers under the following conditions:

$fill_0 \rightarrow reg_0 := \overline{cross} \cdot in_0 + cross \cdot in_1$
$fill_1 \rightarrow reg_1 := cross \cdot in_0 + \overline{cross} \cdot in_1$.

The status flip-flops are set or reset under the following conditions:

$fill_0 \cdot \overline{rel_0} \rightarrow stat_0 := 1$
$rel_0 \rightarrow stat_0 := 0$
$fill_1 \cdot \overline{rel_1} \rightarrow stat_1 := 1$
$rel_1 \rightarrow stat_1 := 0$.

The information available lines are connected to the status flip-flops:

$infa_0 = stat_0$
$infa_1 = stat_1$.

If the DIMOND is implemented in terms of integrated circuits, it is advisable to design two separate circuits:

a control circuit comprising the copy allowances and the status flip-flops, and

a switching circuit comprising the crossbar and the output registers (see Fig. 5).

Such an implementation allows a modular composition of the data path, since one control circuit can control an arbitrary number of switching circuits.

V. A FIFO Buffer with Minimal Delay

In this section the implementation of a FIFO (first in, first out) buffer with the aid of DIMOND's is described. First, a FIFO buffer with two places will be constructed out of a single DIMOND (Fig. 6). Let the control and data lines of A be connected as follows:

$des_0 = infa_0$: if reg_0 is empty, an incoming message (offered at in_0) is directed to reg_0 ($des_0 = infa_0 = 0$); otherwise it is diverted to reg_1 ($des_0 = infa_0 = 1$).

$input_1 = output_1$: a message in reg_1 is offered at in_1.

$des_1 = 0$: a message offered at in_1 leaves the buffer via reg_0.

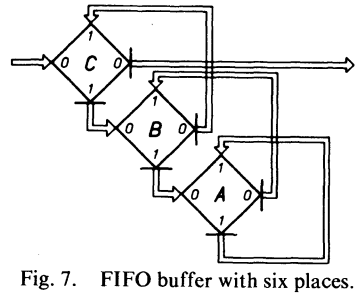


Fig. 7. FIFO buffer with six places.

$prio = 1$: a message in reg_1 has priority over a subsequent incoming message, hence the order of messages is preserved.

Extending this idea, a FIFO buffer with six places is shown in Fig. 7. The DIMOND's A, B, and C are interconnected as follows:

$input_1(A) = output_1(A)$
$input_1(B) = output_0(A); input_0(A) = output_1(B)$
$input_1(C) = output_0(B); input_0(B) = output_1(C)$,

while for all DIMOND's

$des_0 = infa_0$;
$des_1 = 0$;
$prio = 1$.

By induction from the previous case, it can be proved that this configuration indeed behaves as a FIFO buffer with six places. The type of FIFO buffer described has two attractive properties.

1) By adding one DIMOND, the capacity of the buffer is modularly augmented by two places.

2) The buffer has minimal delay in the sense that the delay of an empty buffer equals the delay of *one* DIMOND. More generally, the number of registers a message has to pass through depends only on the number of messages already waiting in the queue and *not* on the buffer capacity.

VI. Processing Networks

In communications networks, messages are transmitted from one DIMOND to another without modification. The control signals are then used as described in the previous sections. However, it is also possible to use DIMOND's in processing networks. As a case in point, we present a processing network which accepts messages and delivers them in sorted sequence according to some ordering relation (Fig. 8).

Assume an unordered sequence of data items to be offered at $in_1(A)$. Let each DIMOND perform the following action: if reg_0 is empty then a data item offered at in_1 is stored in reg_0, where it is kept until a smaller data item is offered. In that case, the item in reg_0 is sent to reg_1 and the new data item is stored in reg_0. All other data items offered at in_1 are transferred to reg_1. Hence, each DIMOND retains in reg_0 the smallest data item offered to it. By implication, then,

$$reg_0(A) \leq reg_0(B) \leq reg_0(C) \leq \cdots.$$

In order to achieve these actions the control lines of all DIMOND's are connected as follows:

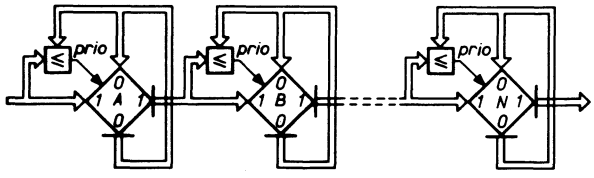


Fig. 8. A sorting network with DIMOND's.

$des_1 = infa_0$: if reg_0 is empty, a data item offered at in_1 is stored in $reg_0 (des_1 = infa_0 = 0)$.

$input_0 = output_0$: a data item in reg_0 is offered at in_0.

$des_0 = creq_1$: as long as no subsequent data item is offered at in_1, the data item in reg_0 is directed to the register it occupies ($des_0 = creq_1 = 0$); hence, it will stay in reg_0.

$prio = (in_0 \leqslant in_1)$: if a data item is offered at in_1 while reg_0 is full, both data items are directed to reg_1 ($des_0 = creq_1 = 1$ and $des_1 = infa_0 = 1$). The larger data item has priority and will actually be stored in reg_1; the other data item will stay (or be stored) in reg_0.

The sorting network can be emptied by setting both destination lines to 1 for all DIMOND's so as to deliver an ordered stream of data items at the output of the last DIMOND.

A set of data items can also be sorted by a network consisting of a FIFO buffer and a one-DIMOND sorting module as described above. The buffer holds an unordered sequence of data items, and each time the contents of the buffer pass through the sorting module, the smallest item is selected and removed. Items not selected are restored to the buffer. Such a solution differs from the multi-DIMOND solution in that it requires less hardware but yields a smaller throughput.

Many other processing networks may be implemented with the aid of DIMOND's, such as an L.R.U. stack (least recently used stack) and an associative memory.

VII. Concluding Remarks

The DIMOND is an attractive building block for communication networks in which throughput is more important than transmission delay. According to the taxonomy in [1], DIMOND networks can be classified as IDD networks (indirect, decentralized routing, dedicated data paths). The main attraction of the DIMOND stems from the fact that it allows the routing of messages through a network to be completely distributed over the individual building blocks. This ability distinguishes the DIMOND from any related proposal of which the authors are aware.

In [3] a component is proposed consisting of a two-by-two line switch and a binary storage element controlling the switch. The networks described are capable of permuting a set of input lines onto a set of output lines. Messages are not stored in the network and part of the control for such a network has to be performed by a central module.

In [2], [4], and [5] three basic building blocks are described which allow the modular construction of networks composed of a hierarchy of interconnected loops. Around each loop, data blocks circulate which cannot be delayed. The data blocks may be full or empty. Control is not completely distributed over the subscriber modules, since each loop requires a module performing supervisory functions.

The implementation given in Section IV requires one central clock for all elements in a network. This implies that the distances between elements in one network should be small. This restriction on its application could be removed by designing the DIMOND so as to operate asynchronously with respect to its environment.

The amount of circuitry required for the implementation of the DIMOND is small for a large-scale integration (LSI) chip, leaving room for the following additions.

1) The data registers (reg_0 and reg_1) may be replaced by minimal delay FIFO buffers (Section V). This modification does not affect the minimum transit time of the module; it only increases the buffer capacity. Such an extension would alleviate the problem of congestion and eliminate the problem of deadlock in a loop structure.

2) By multiplexing the data lines it is possible to increase the number of bits in a message by a factor of two without affecting the transit time of the DIMOND. This can be achieved due to the fact that a message leaves a hole (empty register) after its transmission to a successor. Consequently, the data lines are not used by a subsequent message for at least one clock cycle. In the idle cycle the data lines can thus be used to transmit the second half of the message. In that case, a message will consist of a *truck* (first half) and a *trailer* (second half), where the *truck* contains the routing information.

Acknowledgment

At the Philips Research Laboratories an integrated circuit has been implemented that performs the control functions of the DIMOND. With this control circuit and commercially available multiplexers, a network connecting eight computers has been built. The authors would like to express their appreciation to H. Vrielink for his valuable suggestions with respect to the implementation of the DIMOND. Furthermore, acknowledgments are due to I.S. Herschberg.

References

[1] G. A. Anderson and E. D. Jensen, "Computer interconnection structures: Taxonomy, characteristics, and examples," *Comput. Surveys*, vol. 7, pp. 197-213, Dec. 1975.

[2] C. H. Coker, "An experimental interconnection of computers through a loop transmission system," *Bell Syst. Tech. J.*, vol. 51, pp. 1167-1175, July-Aug. 1972.

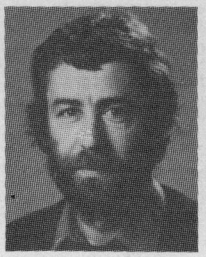
[3] W. H. Kautz, K. N. Levitt, and A. Waksman, "Cellular interconnection arrays," *IEEE Trans. Comput.*, vol. C-17, pp. 443-451, May 1968.

[4] W. J. Kropfl, "An experimental data block switching system," *Bell Syst. Tech. J.*, vol. 51, pp. 1147-1165, July-Aug. 1972.

[5] J. R. Pierce, "Network for block switching of data," *Bell Syst. Tech. J.*, vol. 51, pp. 1133-1145, July-Aug. 1972.

Pierre G. Jansen was born in Blerick, The Netherlands, in 1943. He received the M.Sc. degree in electrical engineering with honors from the Technical University at Eindhoven, The Netherlands, in 1970.

In 1970 he joined the Philips Research Laboratories and started to work in the fault diagnostic field. Since 1976 he has been engaged in research on multiprocessor systems. His current field of interest is the architecture of these systems, logic design, and multivalued logic.



Joep L. W. Kessels was born in Heerlen, The Netherlands, in 1942. He received the M.Sc. degree with honors from the Technological University of Eindhoven, The Netherlands, in 1967.

In 1969 he joined the Philips Research Laboratories where he first worked in the field of programming languages. Since 1976 he has been engaged in research on distributed processing. In this area his main interests are architecture, operating systems, and programming languages.

On the Design of Three-Valued Asynchronous Modules

ANTHONY S. WOJCIK, MEMBER, IEEE, AND KWANG-YA FANG, MEMBER, IEEE

***Abstract*—The use of three-valued signals in the design of asynchronous speed-independent modules is considered. The three-valued Post algebra is used as the mathematical basis for multivalued logic design. Asynchronous operation in a three-valued system is defined. Basis data/control signal detectors and control primitives are developed. It is shown that the design of the nonbinary modules only requires the use of standard binary design techniques and the detectors and logic primitives that are developed. A general monitoring and control structure that allows the interconnection of both combinational and sequential modules is described.**

***Index Terms*—Asynchronous logic, modular design, multiple-valued logic, Post algebra, speed-independent circuit.**

I. INTRODUCTION

THE analysis and synthesis of asynchronous networks is a topic that has attracted the attention of a number of researchers [1]–[4]. In particular, the study of asynchronous circuits has led to the development of several schemes for designing and controlling networks of interconnected asynchronous modules. One crucial factor underlying the different approaches is the nature of the delays that are assumed to occur in the network.

Two of the approaches to network design that are particularly useful with respect to current technology are the "speed-independent circuit" [1], [2] and the "propagation-limited network" [3] models. A basic assumption of the speed-independent model is that network delay is confined to the logic gates and that intergate (line) delay is zero. The propagation-limited approach assumes that delay between modules is significant while delay within a module is negligible. Taken together, the two views provide a reasonably accurate model of an asynchronous network composed of interconnected logic modules in which a module might be as small as a single LSI chip.

In such a network, one recognizes that there is only limited access available to each module. In addition to data paths between modules, it is necessary to have control paths between them to govern the movement of data and to monitor the operation of the modules themselves. Inclusion of control requires the use of access paths to a module, thus reducing the available data paths. Hence, although the potential data processing capability of a module is large, it will be restricted if the number of data paths is limited.

One remedy to this problem of maximizing the data processing potential of a module with a fixed number of access paths is to increase the capacity of the data paths by permitting more than two signal levels to be transmitted on each path. In this case, the use of multivalued logic circuits in the design of the modules is required. However, merely allowing multivalued signals to be used neither ensures that the data processing capability is increased nor guarantees that the control structure will not require more access paths. Further, if the logic design of a nonbinary network is considerably more difficult than that of a binary network, the effort to incorporate the multivalued signals may outweigh the benefit resulting from their use in a system.

The purpose of this paper is to briefly describe one possible approach to the design of networks composed of asynchronous

Manuscript received August 15, 1979; revised May 2, 1980. This work was supported in part by the National Science Foundation under Grants MCS74-01409A01 and MCS79-01689.

A. S. Wojcik is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.

K. Y. Fang was with the Atlantic Richfield Company, Harvey, IL. He is now with the Western Electric Company, Lisle, IL 60532.

0018-9340/80/1000-0889$00.75