A FLEXIBLE CHANNEL EMULATOR FOR COMMUNICATIONS PROTOCOL AND ARCHITECTURE RESEARCH

by

Amanda M. Kao and Lester F. Ludwig

Memorandum No. UCB/ERL M85/83

11 November 1985

A FLEXIBLE CHANNEL EMULATOR FOR COMMUNICATIONS PROTOCOL AND ARCHITECTURE RESEARCH

by

Amanda M. Kao and Lester F. Ludwig

Memorandum No. UCB/ERL M85/83

11 November 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A FLEXIBLE CHANNEL EMULATOR FOR COMMUNICATIONS PROTOCOL AND ARCHITECTURE RESEARCH

by

Amanda M. Kao and Lester F. Ludwig

Memorandum No. UCB/ERL M85/83

11 November 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A FLEXIBLE CHANNEL EMULATOR FOR COMMUNICATIONS PROTOCOL AND ARCHITECTURE RESEARCH

Amanda M. Kao

AT&T Bell Laboratories

Lester F. Ludwig

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

ABSTRACT

This paper describes the design of the programmable 10Mbps physical level emulation system for the UC Berkeley Protocol Workroom facility. The Fast-TTL design presented supports flexible linking of 32 node interfaces and can operate from actual terminal equipment. A wide variety of bus, ring, point-to-point, radio (fixed or mobile), tree, and many other topologies are easily supported, and gateways between subnetworks may be freely introduced. The local topological configuration at each node connection port can be independently specified. Three independent signals (data, code violation status, and carrier) are delayed in ensemble between interconnected nodes, via programmable bidirectional delays. Each internode delay is independently adjustable over a wide range of distances in 10 meter increments and may be adjusted in real time. A number of fault conditions can also be introduced in real-time. The operating speed design can be advanced upward towards 100 Mbps via ECL, gate array, and space-division techniques, limited by the effects of gate propagation delays.

May 27, 1986

A FLEXIBLE CHANNEL EMULATOR FOR COMMUNICATIONS PROTOCOL AND ARCHITECTURE RESEARCH

Amanda M. Kao

AT&T Bell Laboratories

Lester F. Ludwig

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

1. INTRODUCTION

This document discusses the channel emulator system designed for use within the Protocol Workroom, an experimental communications protocol, architecture, and distributed system research facility under development at UC Berkeley [1]. Under programmable control of an external master computer, this hardware system implements functional equivalents to a network's entire physical layer and associated collision-detection mechanisms. The channel emulator system provides the synchronous/asynchronous 10Mbps interconnection fabric within the experimental network of this facility. It supports flexible physical level interconnection of the facility's 32 node interfaces which may be connected to the Protocol Workroom's emulators [2] or other terminal equipment. The work discussed in this report is the expansion and implementation of an original design by one of the authors [3]. The implementation of a preliminary prototype of this design served as the masters project of the other author [4].

This paper begins with a brief overview of the Protocol Workroom facility and the channel emulator concept. Section 3 provides a functional description of the channel emulator. Section 4 discusses the architectural organization of the channel emulator, including detailed explanations of hardware subsystems. Section 5 discusses an optional control panel. Section 6 describes possible extensions and the current implementation at UC Berkeley.

2. OVERVIEW OF THE PROTOCOL WORKROOM FACILITY

The goal of the Protocol Workroom is to provide a software-controlled hardware facility for emulation-oriented research in the fields of communication protocols and communication network architectures. The facility is to be capable of supporting the real-time hardware emulation of a wide variety of network architectures and protocols, so as to complement the theoretical development and

software simulation techniques currently used in these areas of research. A more detailed discussion of the overall facility may be found in [1].

Figure 1 illustrates the conceptual architecture of the overall Protocol Workroom facility. Functionally, it may be viewed as a number of intelligent nodes (each emulating an intelligent digital device) connected to two separate communication networks, the *experimental* and *backbone* networks. The experimental network emulates the network whose behavior is under study. Only data and control signals that would be carried over the modeled network are handled by the experimental network. All other data communications involved in the operation of the facility (configuration, control, monitoring, and analysis) are handled by the backbone network.

The experimental network consists of two principle subsystems. One of these subsystems is the physical level channel emulator that is the subject of this paper. The other subsystem is a group of 32 node emulators providing software-defined hardware implementations of link and higher level protocols of modeled digital devices. The node emulators utilize a number of special hardware subsystems controlled by stimulus/response state machines and a 68020 to accurately emulate a very wide variety of protocols in real time at rates up to 10Mbps [2]. The node emulator can be configured with multiple protocol engines, an elaborate switch fabric, and control provisions to realize very flexible multiport entities such as gateways and centralized integrated-services switches. Together, the node and channel emulators permit extremely flexible control of physical and logical aspects of the experimental network, facilitating the interactive study of an unprecedented wide range of network types in real or near-real time. The experimental network operates synchronously or asynchronously at data rates up to 10 Mbps. It can emulate different (higher) transmission rates through the use of time-distance scaling. Monitoring occurs both within the channel emulator (collision detection) and node emulators. All monitoring information, however, is collected by the node emulators; there is no direct monitoring communications between the master computer and the channel emulator. The channel emulator is controlled, however, directly by the master computer.

Figure 2 illustrates the actual architecture that will be used in realizing the facility. Each node emulator is realized with a Pacific Microsystems 68010 computer connecting to one or more of the custom protocol engines. The Pacific board within each node emulator is hosted by a cluster-concentrating Sun. One of these cluster-concentrating Suns also hosts an Omnibyte OB68K230 I/O board that is used to control the channel emulator. Each protocol engine has a single dedicated connection to an interface port on the channel emulator as shown in Figure 2. The multiple-port nodes that are created by connecting multiple protocol engines to the same Pacific board can be used for emulating multiple-port entities such as gateways and centralized switches.

3. FUNCTIONAL DESCRIPTION OF CHANNEL EMULATOR

The channel emulator illustrated at the bottom of Figure 2 is a software-controlled hardware system, implemented in digital logic, performing the following communications network functions in either synchronous or asynchronous modes:

1. Separate interfacing with each node port, accepting data, timing, and carrier signals for transmission from the port while simultaneously presenting received timing, data, carrier detection, and collision monitors to the port;
2. Code violation indications for out-of-band control signals;
3. Support of a wide range of classical, modern, and uninvestigated node connection topologies and gateway configurations, including multiport nodes;
4. Insertion of programmable propagation delays into the synchronous or asynchronous serial data streams between any pair of connected nodes;
5. Real-time insertion of a wide variety of local and global fault conditions;
6. Simultaneous support of a variety of collision detection schemes at each node site; and
7. Distribution of global timing reference for synchronous operation and for system-context time-stamps in monitoring records.

The channel emulator thus provides for the flexible logical emulation of a wide class of physical layer networks. Further, through the incorporation of very long delay times, it also permits geometric stretching for time-distance scaled modeling of high-bandwidth networks. The key concept in this system is that there is no analog encoding of transmitted data, no physical transceiver, and no physical medium involved in the interconnection of nodes. Instead, the entire physical layer and associated collision-detection functions are precisely emulated by a centralized hard-wired logic system. A sequence of command instructions is used to program the channel emulator's configuration and parameters. These instructions can be modified during an experiment in real-time. In the Protocol Workroom facility, these commands are provided by an I/O board hosted by one of the Suns. An important remark is that the channel emulator can function as a meaningful research and development tool outside its original application within the Protocol Workroom. In fact, almost any sort of terminal equipment can be used with the channel emulator to study that equipment's performance in different network topologies and under various operating conditions. Such applications are suggested by Figure 3.

4. ARCHITECTURE OF CHANNEL EMULATOR

The channel emulator transports four classes of 10 Mbps serial bit streams between the ports connecting to the external nodes. It also introduces operations on these four classes of signals so as to emulate the behavior of specified physical level systems. The four classes of signals are termed *data, violation, carrier, and timing*. These are routed and processed as a four-bit wide signal stream throughout the channel emulator. The timing signal provides the periodic timing reference defining the clock periods for the other three signals. The data signal carries the actual serialized data stream, while the violation signal provides an additional identical but independent path for a second data signal. This second signal can be used to signify the occurrence of code violations which are used in some protocols for out-of-band control signals. The carrier signal indicates whether the node it originates from is in a transmission state or not. As a result, this signal can be interpreted as indicating the validity of the other three signals. When the carrier is high, the corresponding data, violation, and timing signals are carrying active information. When the carrier is low, these signals are viewed as being in an idle state. Thus the transmit carrier signal means "transmit this information" while the received carrier means "this data has arrived". In addition to these functions, the carrier signal is critical to internal operation of the channel emulator. Carrier signals are used to arbitrate timing sources during asynchronous operation, determine collisions, and make directional indications for received signals in bus topologies.

An important feature of the channel emulator implementation is that the data and violation signals are low when the carrier is low. If this behavior cannot be guaranteed by the node equipment, additional logic circuitry such as that shown in Figure 4 can be added to force this condition. (Note that such a circuit introduces a few nanoseconds of delay due to gate propagation.) With this behavior assured, all possible collision points for data, violation, and carrier signals can be implemented as simple OR-gates. This concept will be illustrated later. Timing signals must be arbitrated among the candidate sources at the collision points, however, in order for the channel emulator to function properly after a collision occurs. The arbitration is controlled by the status of applicable carrier signals as will be explained later in this document.

A high-level view of the architecture used to implement the channel emulator is illustrated in Figure 5a. Figure 5b illustrates an expansion of the block diagram of Figure 5a showing the specific treatment of the four signal classes. Note that in Figure 5 the Tap blocks have been turned on their side and enlarged into large rectangular solids whose node interface port resides behind them. Beyond this level of detail the channel emulator consists of a control system, and submodules within the illustrated blocks. The submodules are termed *cells*. The overall architecture is now discussed in

more detail.

The entire channel emulator is controlled by a single control system consisting of a centralized controller and a control bus. The remainder of the channel emulator is organized with respect to the four categories of blocks as illustrated in Figure 5a. The Tap blocks provide the link between each of the node interface ports and the rest of the channel emulator. Each Tap block has one node interface port and two other types of ports to the rest of the channel emulator. These two other types of ports are termed *path 1* and *path 2*. All three ports carry both transmitted and received signals for data, violation, carrier, and timing. Path 1 provides the principal connection between the Tap blocks and the rest of the channel emulator. It is employed in every topology supported by the the channel emulator. Path 2 is employed only for a few topologies, such as the bidirectional bus, folded buses, radio, passive tree topologies, and in an optional failure-recovery mode for counter-rotating rings. The precise usage of this second path will become clear in later discussion.

The Delay-Input-Routing blocks, Delay blocks, and Delay-Output Routing blocks together form an interconnection fabric for the transport of signals among the Tap blocks. A mathematical view of this function in terms of a Boolean matrix operator is shown in Figure 6. The two Routing blocks connect the Tap block with any of the delays in any of the Delay blocks in the channel emulator. Broadcast transmission is naturally supported in the Delay-Input Routing block. The Delay-Output Router block provides special circuitry to support multiple-source broadcast reception.

The number of blocks required to support a given number of nodes varies with the types of topologies to be supported. An N-port channel emulator supporting up to M interconnections from each node requires M delay blocks with N Tap, N Delay-Input, and N Delay-Output blocks. For a K-node bus, ring, double ring, and full-connectivity radio network M must be at least 2(K-1), K, 2K, and K^2, respectively. The channel emulator design discussed in this paper sets N=32 and M=64, allowing the support of up to 32 node bus networks and ring networks as well as full-connectivity radio networks up to 8 nodes. The blocks are divided into cells, some of which are programmable, as indicated:

* Tap Block:
    - o Control bus cell;
    - o Topology/fault cell for data, violation, and carrier signals (programmable);
    - o Topology/fault cell for timing signals with clock arbitration (programmable);
    - o Collision detection cell (optional);
* Delay-Input Routing Block:

- o Control bus cell;
- o Path 1 routing cell (programmable);
- o Path 2 routing cell (programmable);

* Delay Block:
  - o Path 1 delay cell (programmable) with control bus interface;
  - o Path 2 delay cell (programmable) with control bus interface;

* Delay-Output Routing Block:
  - o Control bus cell;
  - o Path 1 data/violation/carrier "masked-OR" cells (programmable);
  - o Path 1 clock arbitrator cells (programmable);
  - o Path 2 data/violation/carrier "masked-OR" cells (programmable);
  - o Path 2 clock arbitrator cells (programmable).

These various blocks and the cells comprising them are described in the sections that follow. Also included is a discussion on the realization of subnetworks, gateways, and tree topologies.

4.1. Centralized Controller

This controller is responsible for establishment and real-time modulation of the channel emulator configuration and operational modes. It receives external commands from the facility's master computer or other external controlling device and translates these into sequences of state assignment commands which are put onto a controller bus. The controller bus is monitored by special cells within every other block in the system. In the Protocol Workroom the Centralized Controller is implemented with a user-wired Omnibyte OB68K230 Multibus I/O controller card. This card is fitted with custom output circuitry in its provided user-wired area, and is hosted by one of the cluster-hosting Suns. For applications outside the Protocol Workroom, any other arrangement capable of sequentially transmitting a sequence of control words onto a bus can be used. Note that since each subsystem within the channel emulator examines the address of each command sent over the control bus the actual sequence of commands can be listed in any order.

4.2. Control Bus

This unidirectional bus carries commands from the I/O controller block within the channel emulator to all blocks within the channel emulator. Each block in the channel emulator includes a Control Bus interface cell which compares the address on the control bus with its own unique address and, upon a match, loads a value into a latch within that cell. The information stored in this latch is used to configure that cell. The bus is formed from four sub-busses: a two-bit *block-type*

sub-bus, a log(N) bit *block-index sub-bus* specifying a particular block within the specified type, a sixteen-bit *value sub-bus* setting the state of the cell, and a one-bit *load-command sub-bus* carrying a load signal that loads the selected block with the value obtained from the value bus. Table 1 summarizes the organization and addressing for the N=32, M=64 case. Note the interpretation of the value sub-bus information varies from block-type to block-type.

To support the programming of a 32-node channel emulator, a total of 26 bits are needed. Five bits are used for the block-index. Four bits are used to select the proper block within each index using the demultiplexing scheme shown in Table 1. Sixteen bits are needed for the value bus to program cells within the blocks. One bit is needed for the load-command signal that is used to signify that new address and value sub-bus quantities are valid and can be latched. Together, a 26-bit bus is required for programming the system. This bus can be read by every block in the system. In the Protocol Workroom facility, the Omnibyte OB68K230 I/O Board is custom adapted to provide sequences of 25-bit words and an appropriately timed load command. In this manner the master computer can transmit a configuration file to the channel emulator for both pre-experiment set-up and real-time modification.

Each block contains a cell or other provision which monitors the controller bus for commands issued to its particular block from the channel emulator's centralized controller. Recall that the control bus is comprised of four sub-busses: a block-type sub-bus, a block-index sub-bus, a value sub-bus, and a load-command sub-bus. The block-type and block-index sub-busses are constantly monitored by the Control Bus cell for the unique address of that cell's block. When the appropriate address is observed, the load-command sub-bus loads the current state of the value sub-bus into an internal 16-bit latch. The word in this latch sets the states of the programmable cells within each block. Each block contains a magnitude comparator which is used to determine whether the block has been selected or not by comparing its block address with the block-type and block-index sub-busses. If the block determines that it has been selected, the signal on the load-command sub-bus is used to strobe the internal latch. Depending upon the specific cell, the word length actually used is between 6 and 16 bits in length. Note from the list in the beginning of Section 4 that some of the blocks contain cells that are not programmable.

4.3. Tap Blocks

These blocks provide the link between the nodes of the network and the interconnection fabric. These blocks take the signals received from the Delay Output Routing block and derive signals transmitted to the node interface and the Delay Input Routing block. The circuitry used to derive these signals is programmable and is controlled by information taken from the channel emulator's

control bus. This signal derivation circuitry supports the following functions:

* Routing and selection among inputs, outputs, and internal elements;
* Collision emulation and detection;
* Clock arbitration;
* Fault condition emulation;
* Global time reference distribution.

The routing and selection functions permit a wide range of tap arrangements and local physical level phenomenon to be directly implemented. Clock arbitration is used to select between candidate timing sources in asynchronous operation. Fault conditions can be introduced and modulated in real-time. The global time reference is used for monitoring and is also employed as the system time base for synchronous operation.

Each Tap block is comprised of a number of cells: a control bus interface cell, a three-bit wide topology/fault cell (for data, violation, and carrier signals), a topology/fault cell with clock arbitration functions (for timing signals), and an optional collision detection cell. Each cell is described in this section.

4.3.1. *Node Emulator Interface Cell*

The node emulator interface provides the connection between each node and the channel emulator. Figure 7 illustrates the lead assignments for each node emulator interface port. A common ground lead is used as a return for each of the logic signal leads identified below. Balanced-line transmission circuits can also be used. A transmit and receive lead are provided for each of the data and code violation signals. A transmit carrier lead is provided for the node to indicate its transmission status and a receive carrier lead is included for the incoming carrier signal. An additional lead is used specifically for the bidirectional bus topology to indicate directional-sense of the carrier. Directional-sense is important only in the case of a bidirectional bus configuration. This lead is ignored by the node emulator in all other cases. Two more leads are included for local timing; one for transmit timing, and another for receive timing. Two leads are provided for global timing signals which are transparently broadcast to all node emulators. One signal is the global 20MHz clock used for global time stamps and for synchronous operational modes. The other signal is a subharmonic signal from a synchronized K-bit master counter which is used to synchronize the K-bit global time counters in each node and to increment the counter for bits K+1 and above. Details of the global time stamp system are explained in [3]. Finally, two optional collision detection leads can be provided. One lead is based on comparing transmit and receive data (data discrepancy), while the other compares transmit and receive carrier signals. If the two optional collision detection leads are not

brought out in the interface, but rather realized in the node hardware, then a 25-pin connector can be used at the node interface to support balanced lines for the eleven remaining logical signals.

4.3.2. *Topology/Fault Cells*

These cells determine the handling of signals to be received and transmitted by a node to realize a number of different network topologies. Also supported in these cells are the real-time introduction of topological and transceiver faults. Circuitry is included to support the following configurations (in addition to many other possible configurations):

1. Bidirectional bus (BB);
2. Unidirectional ring (UR);
3. Folded bus [two versions, FBa, FBb)];
4. Counter-rotating ring using two ports (CRR);
5. Counter-rotating ring failure-recovery mode (CFR);
6. Unidirectional bus pair with physical level head-end (UBP);
7. Collision Star with head-end, dual-frequency radio (fixed and mobile) (S);
8. Full-connectivity radio (fixed and mobile)(R);
9. Gateways (G);
10. Multi-port centralized switches and other point-to-point connections (PP);
11 Passive trees (PT);
12. Trees of subnetworks (ST).

Most of these network and subnetwork topologies and their features are well known. A few comments are added below for clarification of terms.

Physical level head-ends are used in unidirectional bus pair topologies and dual-frequency radio networks where routing, reservation, and contention resolution is handled in a centralized fashion by a key node. In general they are nodes that perform centralized protocol controlling functions. They also are commonly used as gateways where they perform additional functions such as message routing. An illustration of a bus network employing a head-end is shown in Figure 8a. Note that collision stars with head-ends and dual-frequency radio networks have identical topologies; the equivalent topology is illustrated in Figure 8b. A full-connectivity radio topology is shown in Figure 8c.

Gateways are used in practical communications systems as a means to connect independent networks together. In the Protocol Workroom, these and centralized multiple-port switches are

implemented by multiport configurations of node emulators. Channel emulator support of these entities requires the use of more than one port per node site. For a gateway, each of these ports resides in a different sub-network emulated by the channel emulator. An example of a gateway connection linking two emulated networks is shown in Figure 8d. From a given subnetwork, a port connecting to a gateway node operates as any other node site within that subnetwork. Thus the connection of the relevant networks is accomplished transparent to the channel emulator. For a centralized switch, the channel emulator implements simple point-to-point connections between the multiple port node and the other nodes it connects to. This is shown in Figure 8e. Thus the channel emulator handles multi-port centralized switches as a point-to-point network.

It is possible to realize wide ranges of hierarchical networks by using gateways to interconnect subnetworks. Each subnetwork can be any kind of topology. An example of how to realize a given hierarchical topology with the channel emulator is illustrated in Figures 9a-c. Note that the node assignment in the realization is arbitrary. Figure 9a shows the tree configuration to be realized. In Figure 9b, this configuration is divided into subnets, and the nodes are numbered for easy reference. It can be seen from this diagram that gateways will occur at node location numbers 2, 5, and 7. In Figure 9c, the topology of each of the above subnets is defined and the gateway nodes are identified. This diagram shows an example consisting of a variety of subnet topology types: a star with head-end (A), a ring (B), a unidirectional bus pair (C), and a bidirectional bus (D). Note that the gateways are physically handled by the channel emulator as two separate nodes; it is the node emulator that treats gateways as if they were a single node. From this diagram, one can determine the necessary interconnections required of the channel emulator in order to realize the tree. Using this procedure, it is possible to construct a wide range of tree topologies and hierarchical networks by linking combinations of subnetworks in this manner.

Passive trees occur in the ISDN passive bus and passive CATV distribution systems. They are distinguished by having a tree topology of interconnected cables with non-neglectable lengths. Figure 10a illustrates a passive tree network, while Figure 10b illustrates an emulating circuit configuration.

The topology/fault cells also include provisions for emulating real-time faults within the system. These fault conditions are generated either by pseudo-random binary noise sources or programmed combinations of high and low logical values. The incorporation of these options allows for the introduction of a variety of fault conditions at any point in the channel emulator.

The global and local implementation levels of the topologies listed above are illustrated in Figures 11a-f. (Note Figures 11b and 11c do not use a head-end; see Figure 8a for a bus topology employing a head-end.) Each of these diagrams shows the global interconnection of Tap blocks on the

left and a logic circuit realization emulating the Tap block operations for each of the data, violation, and carrier signals on the right. The logic circuit realizations illustrated are identical for the data, violation, and carrier signals. Thus three copies of the illustrated circuit are needed within each Tap block. The design of the channel emulator's topology/fault cell for each of the data, code violation, and carrier signals is simply a generalization of these logical realizations with the inclusion of fault conditions. More complex circuitry is required for timing signals due to the need for clock arbitration. This will be discussed in Section 4.3.4.2.

4.3.2.1. *Topology/fault cell for data, violation, and carrier*

An illustration of the circuit used for the data, code violation, and carrier topology/fault cells is shown in Figure 12. The same circuit is repeated once for each of these three signals. The total topology/fault cell thus consists of these three copies, each sharing the same 9-bit configuration command. Each of the three copies consists of 8-to-1 data selectors and logical-OR gates. The method used in determining the possible selections for each of the multiplexers is summarized in Table II. This Table was constructed by systematically investigating each type of topology and determining the signal outputs of each of the multiplexers. These outputs are denoted in Figure 12 as Node RCV, Path 2 XMT, and Path 1 XMT. For example, by observing the local signal propagation for a bidirectional bus (Figure 11a), it follows that Path 2 XMT is the logical "OR" of Path 2 RCV and Node XMT (Path 2 RCV + Node XMT), Path 1 XMT is the logical "OR" of Path 1 RCV and Node XMT (Path 1 RCV + Node XMT), and Node RCV is the logical "OR" of Path 2 RCV, Path 1 RCV, and Node XMT (Path 1 RCV + Path 2 RCV + Node XMT). Similarly, for a folded bus (Figure 11b), the Path 2 XMT and Node RCV outputs of the Tap block are both simply Path 2 RCV while the Path 1 XMT output is the logical "OR" of Path 1 RCV and Node XMT (Path 1 RCV + Node XMT). In this way, the outputs for each of the supported topologies can be determined. The resulting options are listed in Table II. Note here that radio network transmit and receive signals require only routing of interface signals to the Delay-Input and Delay-Output Routing blocks since all necessary logical operations are provided by these blocks. Finally, note from Figure 10b that passive trees require path 2 to be used as a "feedback" provision; this is also supported. These selections and extra options which emulate fault conditions (forced logical "1", forced logical "0", and pseudo-random noise) were used to obtain the design shown in Figure 12. Note that gateways require no special treatment since each gateway port is completely characterized by the type of subnetwork it resides on.

4.3.2.2. *Topology/fault cell for timing*

The topology/fault cell for the timing signal is less straight-forward and is shown in Figure 13. It features clock arbitrator functions in order to provide the correct timing for network signal propagation and for the receiving nodes. Depending on the topology emulated, timing signals can be received from either the node interface or from external timing signals provided by the Delay-Output Routing block. The decision is determined by the chosen topology and in several cases also by the status of the carrier signal from the node associated with the Tap block. The local carrier signals are employed in clock arbitration functions used to resolve collisions. These are discussed below.

Clock arbitration functions are employed to realize a truly asynchronous interconnection environment. Three separate clock arbitrator functions are required: one for the receive timing at the node and one for each direction of data transmission. The derived timing they provide is used for three purposes:

1. Receive-data timing for the local node interface;
2. Clocking of subsequent delay cells in the network for carrier signal propagation;
3. Provision of timing reference for deriving received timing in subsequent blocks.

There are two cases to consider; these are distinguished by whether the given node is in a collision state or not. If there is no collision, the local transmit carrier signal can be used to determine whether the received timing is from the preceding block timing reference, or from the local node. If the node is not transmitting (i.e., no carrier signal is present), the clock arbitrator will choose the timing from the preceding block. However, if a carrier signal is present, the timing transmitted by the local node is selected. If the node is in collision state (as determined by the status of external and local carrier signals), data will be ignored by the local node and hence receive timing will not be needed for that particular node. A well-defined timing signal is needed, however, to propagate signals through subsequent delay cells. The clock arbitrators are designed to provide this signal. A precise emulation would have the clock arbitrator choose the timing associated with the node whose carrier signal arrived first so that the effective propagation velocity of the carrier "wavefront" is constant. If timing sources were suddenly switched, the resulting phase change in the timing would cause the delay cells to effectively creat an instantaneous change in propagation velocity. Note, however, that the detection of which of two asynchronous events first occurs can only be approximated by logic circuitry. Within some sufficiently small time interval any circuit will see the two events as being simultaneous with respect to its reaction time. A default will have to be chosen, resulting in a timing phase change should the default be the wrong choice. A high-speed RS flip-flop circuit could

be used to minimize this error. If a one bit-period maximum error is permissible, the flip-flop can be omitted by simply wiring in the default directly. This is the approach presented in this paper. The performance of this approximation is felt to be acceptable for foreseeable protocol studies and may be modified if needed without major rewiring. In particular, a major simplification results if the default is chosen to be the local timing source. In this case, the decision can be based on the local carrier and implemented precisely as it needs to be for non-collision states. As a result, the same circuit can be used for both cases under this minor approximation. The resulting functional schematic of such a clock arbitrator is shown in Figure 14.

The clock arbitration function described is combined with routing circuitry similar to that of Figure 12, resulting in the design illustrated in Figure 13. All IC's shown are 8:1 data selectors. ICs 1,2,3, and 4 perform clock arbitration functions through manipulation of one or more of their address lines by various carrier signals. ICs 1 and 2 derive the XMT timing signals for path 1 and path 2 respectively. Each has two address lines, A_1 and A_2, which are controlled by information stored in the value sub-bus latch. Each (A_1, A_2) ordered pair is interpreted as follows:

00 Arbitrate between Node XMT and Path RCV timing sources;

01 Node XMT only;

10 Path RCV only ("feedback");

11 Logic "0" ("no signal").

Address line A_0 is controlled by the Node XMT Carrier signal. IC 3 performs the clock arbitration function for the Node RCV timing based on the activity of the Node XMT carrier, Path 1 RCV carrier, and Path 2 RCV carrier signals. IC 4 is identical in function to IC 3, but with path 2 suppressed. This permits path 2 to be used for feedback purposes without interfering with "node"/"path 1" arbitrations. The outputs of ICs 3 and 4 are presented to IC 5 along with direct connections to Path 1 RCV, Path 2 RCV, and fault conditions. IC 5 selects one of these to obtain the Node RCV timing signal. Table III summarizes the usage and addressing of this topology/fault cell for the various example topologies. Note a sixth 8:1 selector could be included whose function is like that of IC 4 except with path 1 suppressed rather than path 2. This makes the usage of path 1 and path 2 completely interchangeable. This is attractive because the rest of the channel emulator is completely symmetric with respect to paths 1 and 2.

4.3.3. *Control bus cell*

This cell consists of a 16-bit latch and the wiring shown in Figure 15. The latch is strobed according to the addressing arrangement of Table I. The 16-bit word stored in the latch is used to program the entire block. Nine bits are used to program the topology/fault cells for data, code

violation, and carrier signals. Since the topologies and operations on these signals are identical, the same nine bit word is used for each of the three identical circuits. The remaining seven bits are used to program the topology/fault cell for the timing signal.

4.3.4. *Collision Detection Cell (optional)*

The optional collision detection cell provides the optional logical provisions to detect data discrepancies and carrier collisions. Although the Protocol Workroom node emulators will include the two gates required to derive collision detection signals from the data and carrier signals available at the interface, other stand-alone applications of the channel emulator may find it convenient to have collision detection signals provided at the interface. The circuit used to detect data discrepancies is the "EX-OR" of the Node XMT data and Node RCV data signals. When the output value is a logical 1, a data discrepancy has occurred. Data discrepancy detection is used in bidirectional bus topologies like Ethernet. An EX-OR circuit for the violation signal is also suggested. An overall data discrepancy signal can then be derived from "OR"-ing these two "EX-OR" results. A more useful collision detection mechanism often provided by physical level interfaces is the detection of multiple active carrier signals. To detect a carrier collision, the Node RCV and Node XMT carrier signals are logically "AND"-ed. If the "AND" output here is a logical 1, then a carrier collision has occurred.

4.4. Delay-Input Routing Block

The Delay-Input Routing blocks shown in Figure 5 are simply a collection of 2N 4-wire 2N:1 data selectors wired to operate as a unidirectional crossbar switch. For the 32 port channel emulator, each cell in this block consists of a 4-wire 64:1 data selector. This results in a collection of four sets of 64 64:1 data selectors wired as a stack of four 64x64 crossbar switches. This is equivalent to four copies of a collection of the arrangement shown in Figure 16. These four copies combined with the control cell form the entire Delay-Input Routing system shown in Figure 5a. (Note also that each of the four planes in Figure 5b are exactly the circuit shown in Figure 16.)

The state of each 64:1 multiplexer is controlled by a 6-bit command. The corresponding multiplexers in the four copies each get the same command. Thus the state of the entire Delay-Input Routing block is specified by a set of 64 6-bit words. These commands are carried over the value sub-bus as the first six least-significant bits and latched via the arrangement of Figure 15 in accordance with the addressing scheme outlined in Table I. Note that separate block-type addresses are used for path 1 and path 2. Even though these could have been combined to share a 12-bit subset of the value sub-bus, separate addressing was used to maintain programming consistency as illustrated in Table I.

4.5. Delay Blocks

These blocks are used to introduce programmable delays between nodes to simulate the propagation time introduced by a transmission medium. Each block consists of 2 identical delay cells, one for path 1 and one for path 2. Each delay cell propagates three binary signals in parallel with matched programmable delay. One of these signals represents the data stream, another represents the code violation status while the third signal represents the carrier. The timing signal is also handled by the delay cells as suggested by Figure 5b, but in a different manner as will be discussed. Each delay cell can have two stages, one of which is optional. First, a RAM-based delay is used to introduce the bulk of the delay in 1024 increments, each equivalent to 20 meters of cable at 10Mbps. Since current bipolar RAM technology prevents access cycles of less than 25ns, this circuit cannot operate faster than 10Mbps. Therefore, a optional 20MHz interpolator circuit implementing a half clock-period of delay can be included . This circuit can offset the bulk delay in increments equivalent to either 0 or 10 meters of cable, thus doubling the resolution of the delay cell.

The architecture of the RAM-based delay cell design is illustrated in Figure 17. In the design, a 1024x1 bit RAM supporting high-speed read/write switching is used to implement a bucket-brigade-like delay. A pair of counters with synchronous load capabilities are used to realize two rotating address pointers with an adjustable offset. A 2-to-1 multiplexer alternates between the outputs of these two counters, sequencing the read and write addresses that are issued to the 25ns 1K bit RAM. The RAM delay cell uses the actual timing signal accompanying the data, violation, and carrier signals to insure accurate sampling. The resulting system implements a programmable delay clocked with respect to the provided 10 MHz timing signal, adjustable in 100ns steps. As a result, the resolution (step size) of the RAM delay corresponds to that of one bit period. This time step is 100ns for the 10Mbps rate, corresponding to approximately 20 meters of cable. Logic gates required for topology switching are expected to introduce an approximately 50ns delay which forms the minimum internode delay, corresponding to 10 meters of cable. For the 10 MHz clock assumed, this gives a possible range of internode delays corresponding to separations of between 10 and 20,490 meters in 20 meter increments. The result of including the interpolator circuit into the delay scheme allows for a range of internode delays corresponding to separations of between 10 and 20,500 meters in 10 meter increments. For a 32-node network, this delay range per node can give an effective network length in excess of 640Km for use in time-distance scaling of high-bandwidth networks. When time-distance scaling is used to simulate higher transmission rates, the resolution is enhanced by the scale factor (e.g., increments are one meter for scaling to 100Mbps). The long delays are also directly useful in the study of geographically large networks. In addition to protocol studies, this capability

also permits study of synchronization problems in communications and power systems.

The interpolator circuit requires a timing signal of 20MHz. The 20 MHz clock must be synchronized with the 10Mbps timing signals from the transmitting node. The timing signals required for the operation of the RAM delay and interpolator must be provided by clock arbitrators as discussed previously. Thus, to include interpolators, each node must actually transmit a 20 MHz and 10 MHz timing signal and the existing timing circuitry must be copied to carry this faster timing signal in parallel with the original one. This adds considerably to the overall complexity of the channel emulator. (Note that noting can be gained by using divide-by-2 counters since a phase synchronization signal would then be required and this signal needs the same treatment required for a separate clock signal.) The interpolator circuit consists of a D flip-flop with bypass provisions used to implement a selectable 0 or 50ns delay increment and an EX-OR gate acting as a controllable invert/non-invert element. A diagram of the interpolator's incorporation within the delay system, including the source of the timing signals, is shown in Figure 18.

It is noted that a special provision must be included for mobile radio topologies. When the separation distance between a pair of nodes changes, there is a change in the number of delay stages needed. It is reasonable to assume the mobile nodes will be moving sufficiently slowly that all such changes will be limited to at most one step per transmission. To prevent the introduction of extraneous bits or the dropping of bits in the propagation paths, changes that would occur during transmission are deferred to after that transmission is completed. Increments and decrements to the pointer separation may be realized at the programming level by interrupting the clock to the appropriate counter when the appropriate carrier signal is low. Increment and decrement commands may be generated asynchronously at any time. These commands must be latched until an appropriate version of the carrier falls low. For an increment command, the write counter clock is interrupted for one clock period when the incoming carrier signal falls low. For a decrement command, the read counter clock is interrupted for one clock period when the delayed carrier signal falls low.

4.6. Delay-Output Routing Block

Star, radio, and point-to-point topologies are not neighbor-oriented (as are bus and ring topologies). An enhanced interconnection system is provided in the Delay-Output Routing block to easily realize these topologies. This 64x4-signal system is capable of supporting radio networks of up to 8 nodes and independent collision-protocol stars terminating on any of the 32 nodes.

The design of the collision star with four nodes is shown in Figure 19a. It is realized by logically "OR"-ing all node transmit-data lines and broadcasting the result to all node receive-data lines. This logical operation is reproduced two more times, once each for the code violation and carrier

signals. The number of nodes actually functional within the system is determined through the logical "AND"-ing of the inputs from each node with an enable signal. In this way, one can set the number of active nodes in a star topology to anywhere between 1 and 32.

The design of a full-connectivity radio network using four nodes is shown in Figure 19b. For radio topologies with N nodes, the superposition of N primitive stars is required. These primitive stars are similar to the collision-protocol star circuit except that each node requires its own collection of N-1 delayed signals from the other N-1 nodes and its own private logical "OR"-ing of these received signals. Since there are N(N-1)/2 possible bidirectional paths between N given nodes, there is a total requirement of N(N-1) delay cells to support full duplex N-node radio topology. Since the 32-node facility contains 64 delay cells, provisions are made to support only up to 8-node radio networks (which will use 56 delay cell switch non-zero delay values). As above, the number of functional nodes used in the system is determined by the logical "AND"-ing of the inputs from each node with an enable signal.

A flexible switching arrangement can be used to implement bus and ring interconnections as simply point-to-point links between Tap blocks. Point-to-point connections are simply realized using data selector functions. Note this facilitates the emulation of systems involving several independent subnetworks.

To be able to realize these three classes of topologies in a flexible manner each of the data, violation, carrier, and timing signals are separately handled by independent copies of the configuration shown in Figure 20a. Note that each block can selectively access the path 1 and path 2 inputs to each Tap block. Each of the cells represents an identical functional element that uses an AND-gate "mask" to block the introduction of signals from paths not relevant to the topology local to the node associated with that cell. For the data, violation, and carrier signals, each cell represents a "Masked-OR" cell shown in Figure 20b. This cell uses an AND-gate mask to block signal paths not relevant to the local topology of the associated node and logically "OR"s the remaining signals. Note that if the mask blocks all but one path, the cell behaves as a simple data selector. The cell is used in this mode for all topologies other than radio, collision stars, and passive trees. In these topologies several possible signals can collide at the same point. The OR gate implements the collision mechanism, just as was done in the Tap blocks. For the timing signals, all the cells are the "Masked-Clock Arbitrator" cell shown in Figure 20c. This system is similar in function to the clock arbitrators within the Tap blocks. They differ from those in the Tap blocks in that they do not handle local timing signals and up to 64 clock signals can be involved rather than at most 2. A similar arbitration approximation is also made. An AND-gate mask is used to pass only relevant carrier signals to a priority encoder.

The priority encoder selects the most significant address of the carrier signals that are high and passes this address to the data selector that chooses the timing source. The approximation thus is the use of a hard-wired priority to arbitrate clocks rather than use of the first arriving high carrier. The approximation is especially attractive because of the remarkable circuit simplification and the surprisingly short (< 5nsec) propagation delay for existing priority encoder chips. Note that if the mask blocks all but one path, the cell behaves as a simple data selector. The resulting approximation is the same (a maximum of one bit period error) and high speed flip-flop circuits can be used to improve the approximation if desired.

Note that each block has cells for both path 1 and path 2 data, violation, carrier, and timing signals. All four of these cells associated with the same block and path are configured by the same 64-bit mask. As a result, the circuitry can be reduced by one-fourth by sharing the AND-gate mask used for the carrier cells with the clock arbitrator cells if layout permits.

5. OPTIONAL DISPLAY PANEL

A display panel can be added to the channel emulator to permit visual monitoring of the behavior of the channel emulator. Such visual monitoring is most useful when the network is run in slow motion so that the propagation of carrier wavefronts can be visually followed. The panel can also be extremely useful in system debugging and in testing the integrity of configurations. For full-speed operation, pulse-width expanding circuits can be added to capture the occurrence of rare events, such as collisions on a lightly loaded network. The panel can also be made to include stand-alone demonstration features so that the operational principals of various physical level architectures can be illustrated without the need for node equipment. This can be done by simply including a very simplified node simulator circuit generating effective signals for transmission as controlled by a single panel switch.

A possible display layout for one node on such a panel is shown in Figure 21. In this approach each node display is divided into three distinct parts: a group of LEDs that monitor different conditions concerning the channel emulator, a collection of seven-segment displays that indicate the configuration of each block, and a section of controls that provides control of a simplified node simulation for use in stand-alone operation. A set of LEDs is used to visually follow the propagation of signals from node to node, as well as signal routing within the node. This would be used primarily when the network is run slowly (at 1-10 Hertz, for example). A pulse-stretching switch can be included with these LEDs to aid capture of instantaneous events, such as collisions and carrier activity, at high timing speeds. Numerical displays can be included in the panel to show the status of the delay cells and topology switching cells. For the topology/fault switches for data, code

violation, and carrier signals, a display may be associated with each of the data selectors contained within the cell. The displayed numbers will be associated with the condition selected on each multiplexer. These displays provide orientation during demonstration and also serve as a diagnostic tool to determine whether each topology/fault cell has been properly programmed.

In order to operate as a stand-alone unit for internal debugging or demonstration, it is necessary to independently generate the signals that would ordinarily be supplied by the node emulator external to the channel emulator. Therefore, provisions are made for the simulation of a simple internal node emulator. This node emulator would provide the data, code violation, carrier, and timing signals. A switch that chooses between internal stand-alone operation or external operation in conjunction with the node emulator can be included.

6. EXTENSIONS AND UC BERKELEY STATUS

Values of N and M other than N=32 and M=64 can be implemented almost directly from the material presented in this paper. The major consideration is simply the modification of the addressing organization shown in Table 1. Designs with large values of N and M require care due to the need to minimize propagation delay. This need reflects the effective transparency required by the switching and routing functions in the channel emulator as each 100 nsec of delay translates into 20 meters of cable.

An important remark is that the design can be refined for high-speed operation. As many new node equipments will have protocol operation speeds in excess of 10 Mbps, the real-time support of higher bit-rate emulations of channels is attractive. ECL and GaAs gate array technology can be used to increase the speed of operation and reduce propagation delay in the combinatorial logic. Space-division techniques can be used to increase the speed of the delay cells and other circuitry. A 100Mbps design has been outlined using these techniques for an interested research institution. Two final enhancement remarks are given. The first is that the delay cell interpolator circuitry can be expanded with additional D-flip flops to increase cable-length resolution. In addition, it is also very straightforward to add external processors to enhance real-time operation, such as simulating multipath and fading channel conditions in radio links.

Currently, prototype hardware has been assembled for the support of four nodes. This system is currently being tested and debugged. The VLSI group within EECS at UC Berkeley is studying implementation of a chip set realizing the blocks comprising the facility. A number of industrial research and development laboratories are actively reproducing adapted versions of the system with UC Berkeley guidance.

7. ACKNOWLEDGEMENTS

Special thanks are extended to Professor M. Graham for the inspiring suggestion of using logic circuiting to emulate Ethernet systems and to Professors P. Varaiya and J. Walrand for their suggestions, encouragement, support, and editorial review. Thanks are also extended to I. Ketvirtis, R. Hill, M. Handler, and O. Awom for their work in the construction of the prototype. Finally, we thank Bell Communications Research, the National Science Foundation (Grant ECS-8118213/85-06337), JSEP (Grant F49620-79-0178), MICRO, AT&T Bell Laboratories, National Semiconductor, and an anonymous donor for their generous grants and donations.

8. REFERENCES

[1] A. Fawaz, D. Giralt, and L. Ludwig, "The Protocol Workroom: An Experimental Protocol and Distributed System Research Facility For UC Berkeley", ERL M85-82, 1985, UC Berkeley EECS Department.

[2] A. Fawaz, L. Ludwig, M. Peck, "Node Emulator Architecture for the UC Berkeley Protocol Workroom Facility", ERL M85-84, September 1985, UC Berkeley EECS Department.

[3] L. Ludwig, "Channel Emulator for the Protocol Workroom", Protocol Workroom Document No. 85-1, version 1, June 1985, UC Berkeley EECS Department.

[4] A. Kao, "Channel Emulator for the Protocol Workroom", Master's Thesis, August 1985, UC Berkeley EECS Department.

TABLE I ADDRESS ORGANIZATION

NAME	(4 BITS) BLOCK TYPE AND PATH SELECT	(5 BITS) BLOCK INDEX	USE OF VALVE BUS
TAP BLOCK	"0"	0-31	9 BITS FOR D/V/C 7 BITS FOR TIMING
DELAY INPUT BLOCK	PATH 1 = "1" PATH 2 = "2"	0-31	6 BITS FOR SELECTION
DELAY BLOCK	GROUP 1 = "3" GROUP 2 = "4"	0-31	(LSB) 1 BIT FOR INTERPOLATOR 10 BITS FOR DELAY VALUE
DELAY OUTPUT BLOCK	PATH 1 "5"-"8" PATH 2 "9"-"12"	0-31	16 BITS FOR 1/4 OF MASK

TABLE II TOPOLOGY/FAULT CELL FOR DATA, VIOLATION, AND CARRIER

TOPOLOGY	NODE INTERFACE RCV	PATH 1 XMT	PATH 2 XMT	9-BIT ADDRESS
BB	(P1 RCV)+(P2 RCV)+(N XMT)	(P1 RCV) + (N XMT)	(P2 RCV) + (N XMT)	101 011 101
UR/CRR	(P1 RCV)	(N XMT)	(GND)	100 110 111
FBa/FBb	(P2 RCV)	(P1 RCV) + (N XMT)	(P2 RCV)	111 011 100
EAST CFR	(P1 RCV)	(GND)	(N XMT)	100 111 110
WEST CFR	(P2 RCV)	(N XMT)	(GND)	111 110 111
UBP NODE	(P2 RCV)	(P1 RCV) + (N XMT)	(P2 RCV)	111 011 100
UBP HEAD END	(P1 RCV)	(GND)	(N XMT)	100 111 110
S NODE	(P2 RCV)	(N XMT)	(GND)	111 110 111
S HEAD END	(P1 RCV)	(GND)	(N XMT)	100 111 110
R	(P1 RCV)	(N XMT)	(GND)	100 110 111
PP	(P1 RCV)	(N XMT)	(GND)	100 110 111
PT	(P1 RCV)	(N XMT)	(P2 RCV)	100 110 100

TABLE III TOPOLOGY/FAULT CELL FOR TIMING

TOPOLOGY	NODE INTERFACE RCV. TIMING		PATH 1 XMT TIMING		PATH 2 XMT TIMING		7 BIT ADDRESS
	NODE CARRIER HIGH	NODE CARRIER LOW	NODE CARRIER HIGH	NODE CARRIER LOW	NODE CARRIER HIGH	NODE CARRIER LOW	
BB	(N XMT)	(ARB RCV)	(N XMT)	(P1 RCV)	(N XMT)	(P2 RCV)	011 00 00
UR/CRR	(P1 RCV)	(P1 RCV)	(N XMT)	(N XMT)	(GND)	(GND)	001 01 11
FB	(P2 RCV)	(P2 RCV)	(N XMT)	(P1 RCV)	(P2 RCV)	(P2 RCV)	010 00 10
EAST CFR	(P1 RCV)	(P1 RCV)	(GND)	(GND)	(N XMT)	(N XMT)	001 11 01
WEST CFR	(P2 RCV)	(P2 RCV)	(N XMT)	(N XMT)	(GND)	(GND)	010 01 11
UBP NODE	(P2 RCV)	(P2 RCV)	(N XMT)	(P1 RCV)	(P2 RCV)	(P2 RCV)	010 00 10
UBP HEAD END	(P1 RCV)	(P1 RCV)	(GND)	(GND)	(N XMT)	(N XMT)	001 11 01
S NODE	(P2 RCV)	(P2 RCV)	(N XMT)	(N XMT)	(GND)	(GND)	010 01 11
S HEAD END	(P1 RCV)	(P1 RCV)	(GND)	(GND)	(N XMT)	(N XMT)	001 11 01
R	(P1 RCV)	(P1 RCV)	(N XMT)	(N XMT)	(GND)	(GND)	001 01 11
PP	(P1 RCV)	(P1 RCV)	(N XMT)	(N XMT)	(GND)	(GND)	001 01 11
PT	(P1 RCV)	(P1 RCV)	(N XMT)	(N XMT)	(P2 RCV)	(P2 RCV)	001 01 10

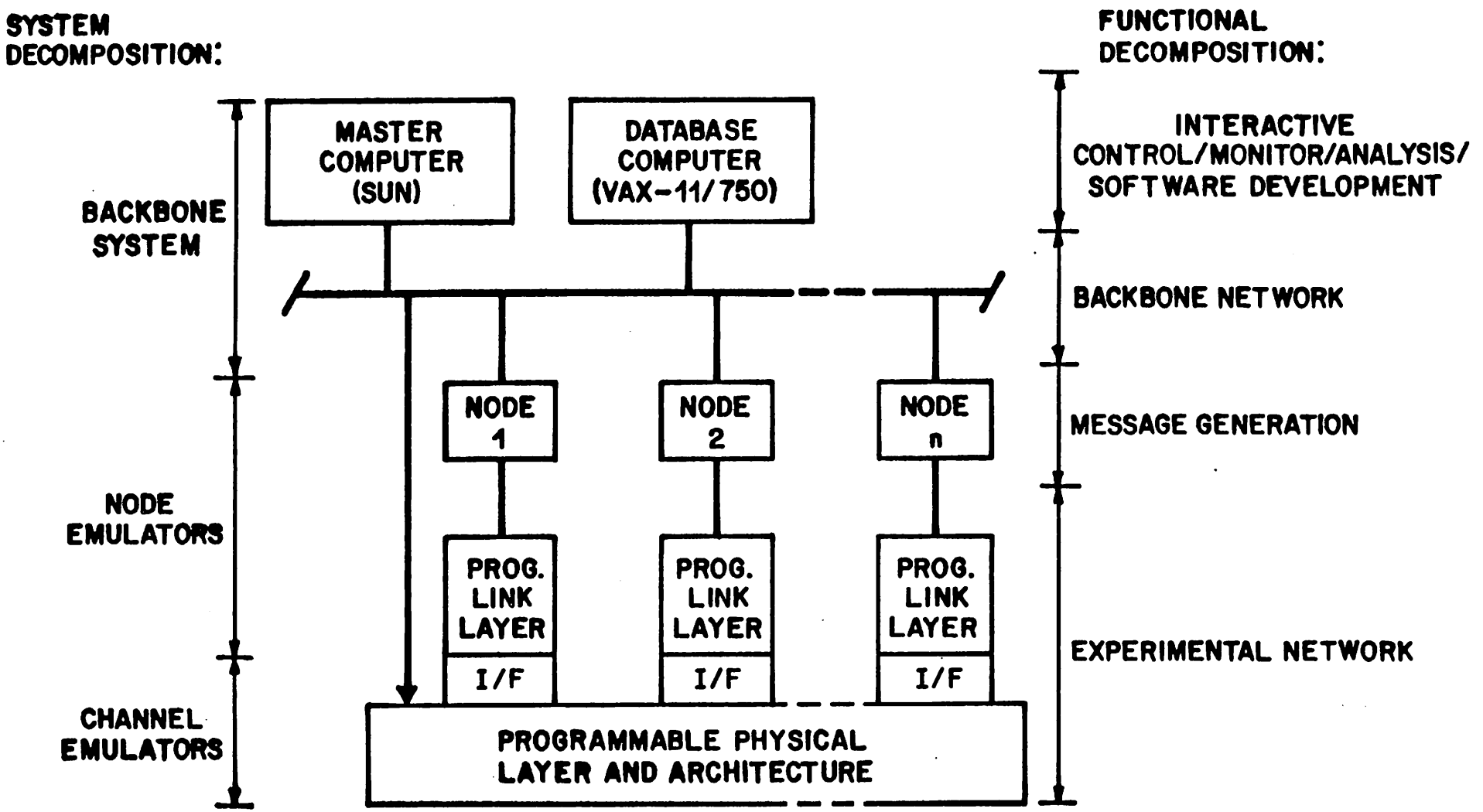


FIGURE 1 CONCEPTUAL ARCHITECTURE OF THE FACILITY

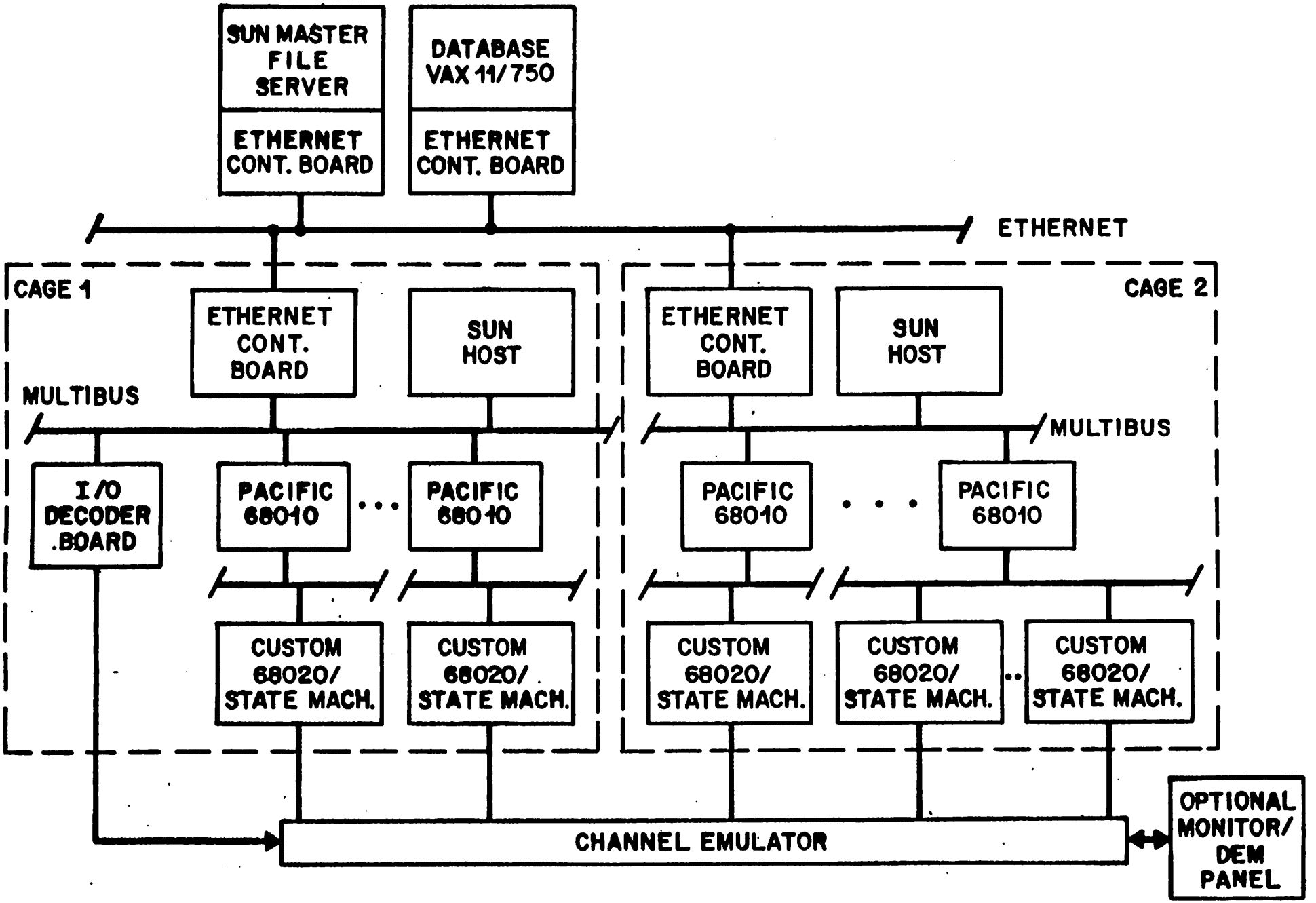


FIGURE 2 ACTUAL ARCHITECTURE REALIZING THE FACILITY

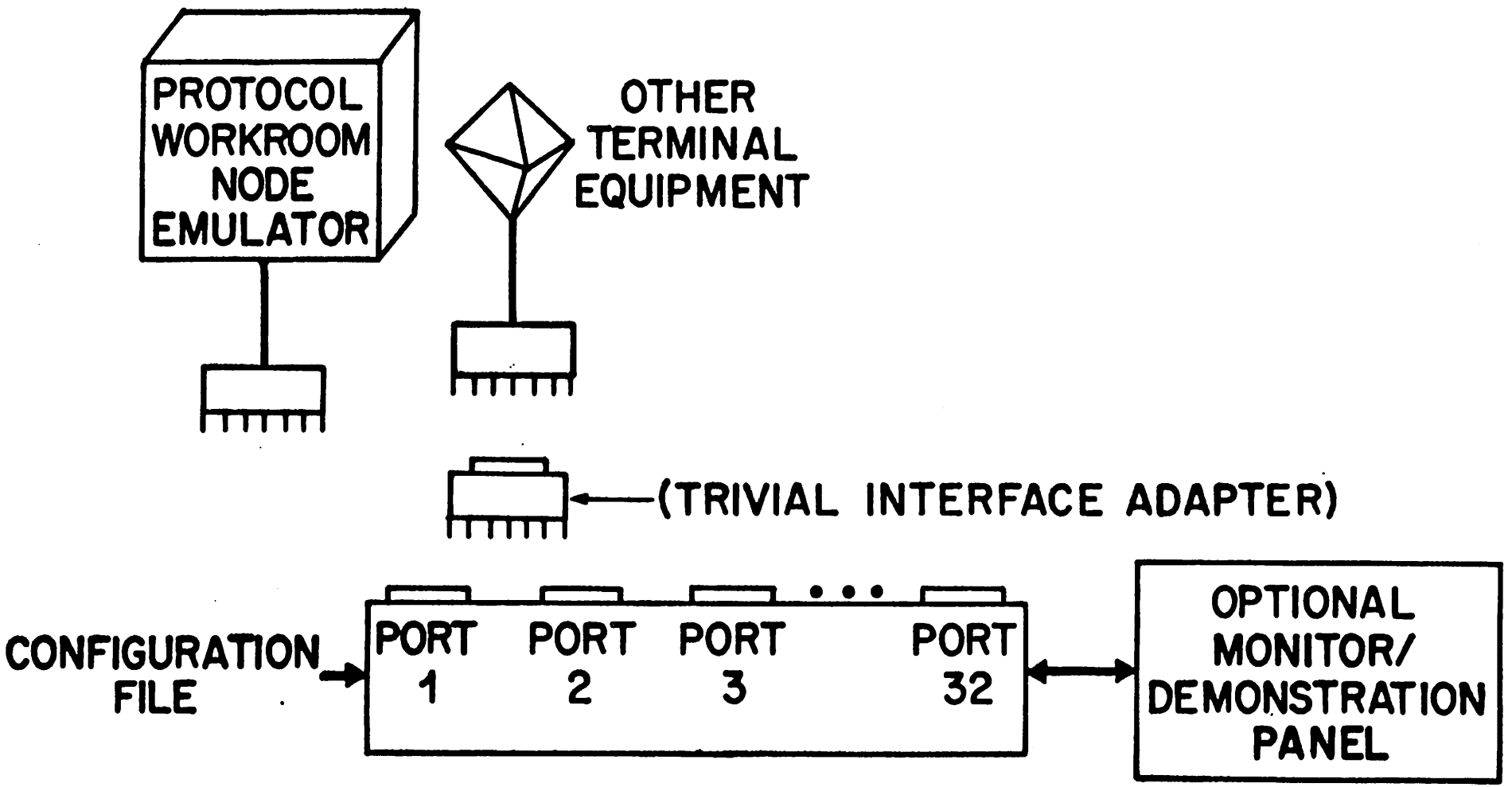


FIGURE 3 CHANNEL EMULATOR AS AN AUTONOMOUS R&D TOOL

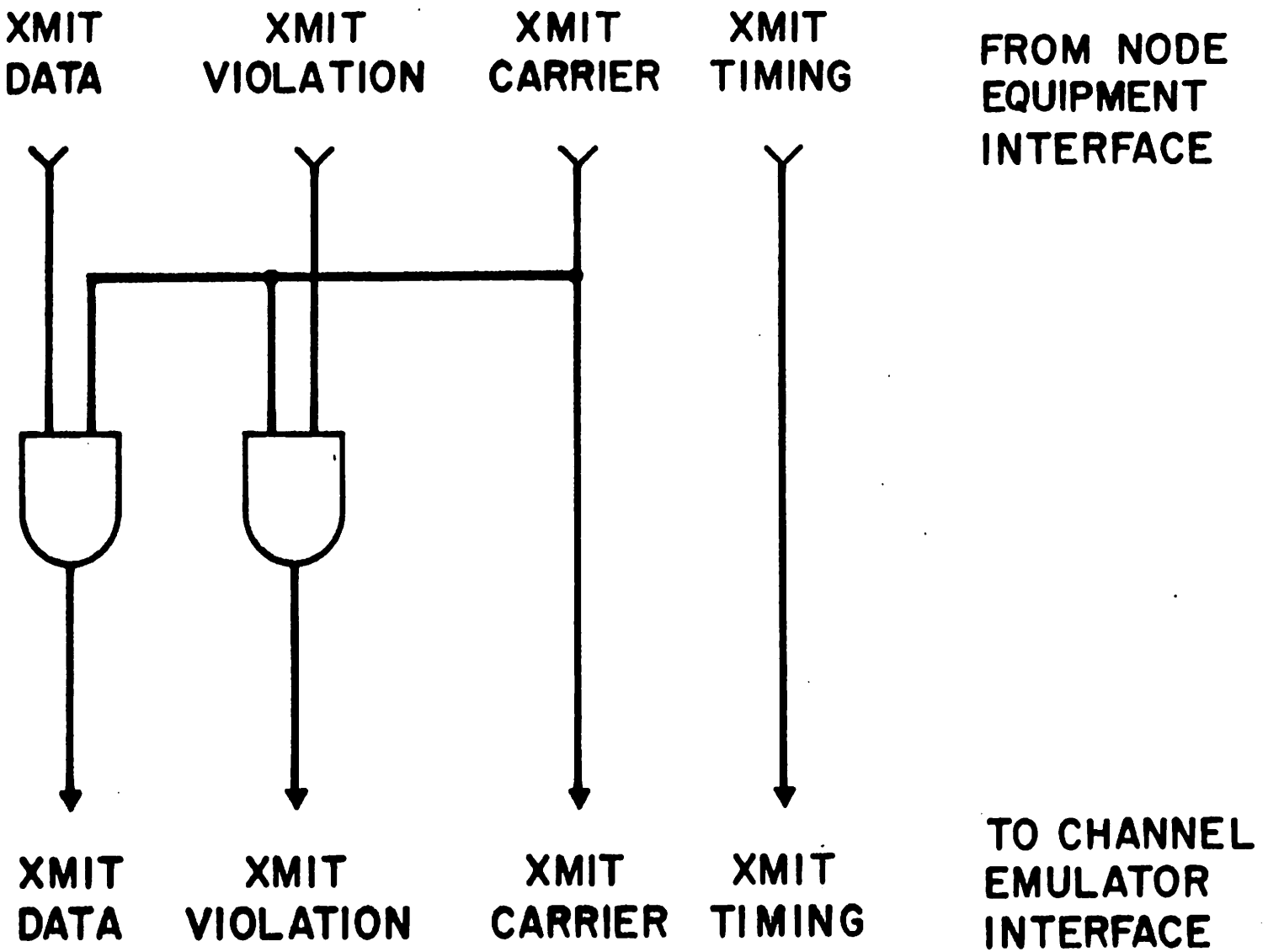


FIGURE 4 OPTIONAL LOGIC CIRCUIT FORCING THE NECESSARY IDLE CONDITION IN THE CASE WHERE IT CANNOT BE PROVIDED BY NODE EQUIPMENT AT THE INTERFACE

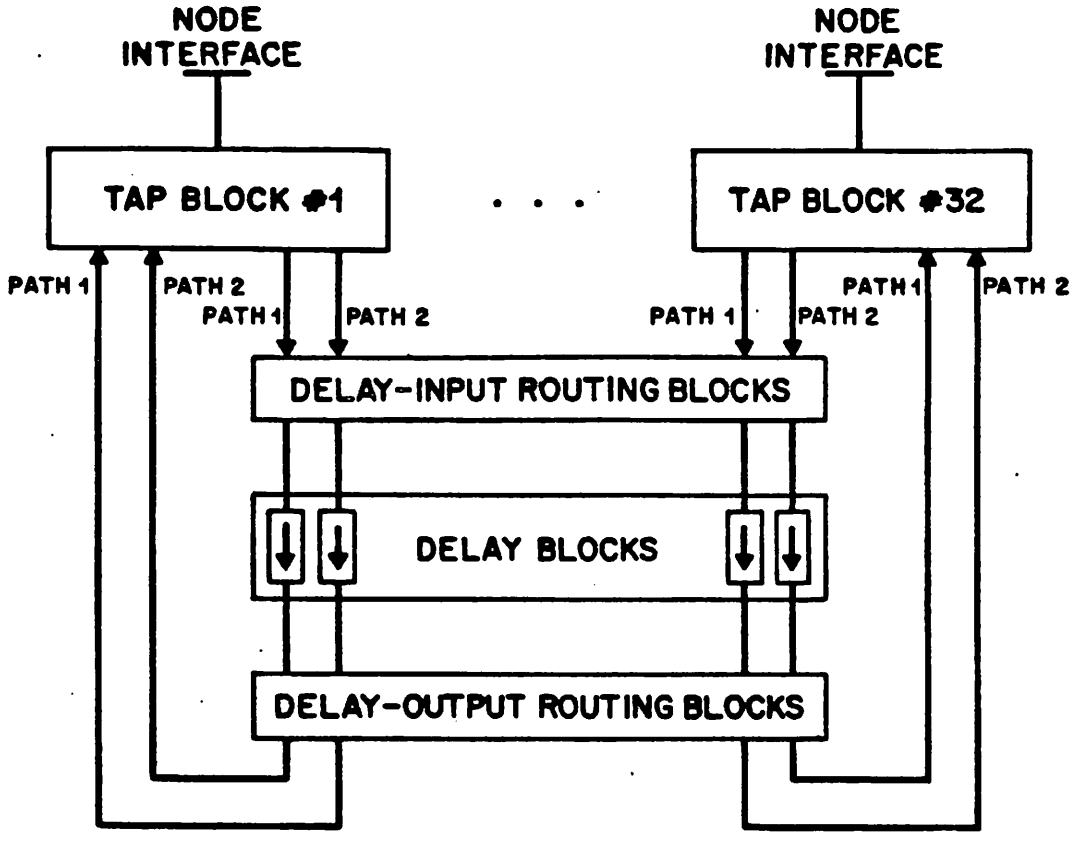


FIGURE 5a CHANNEL EMULATOR ARCHITECTURE

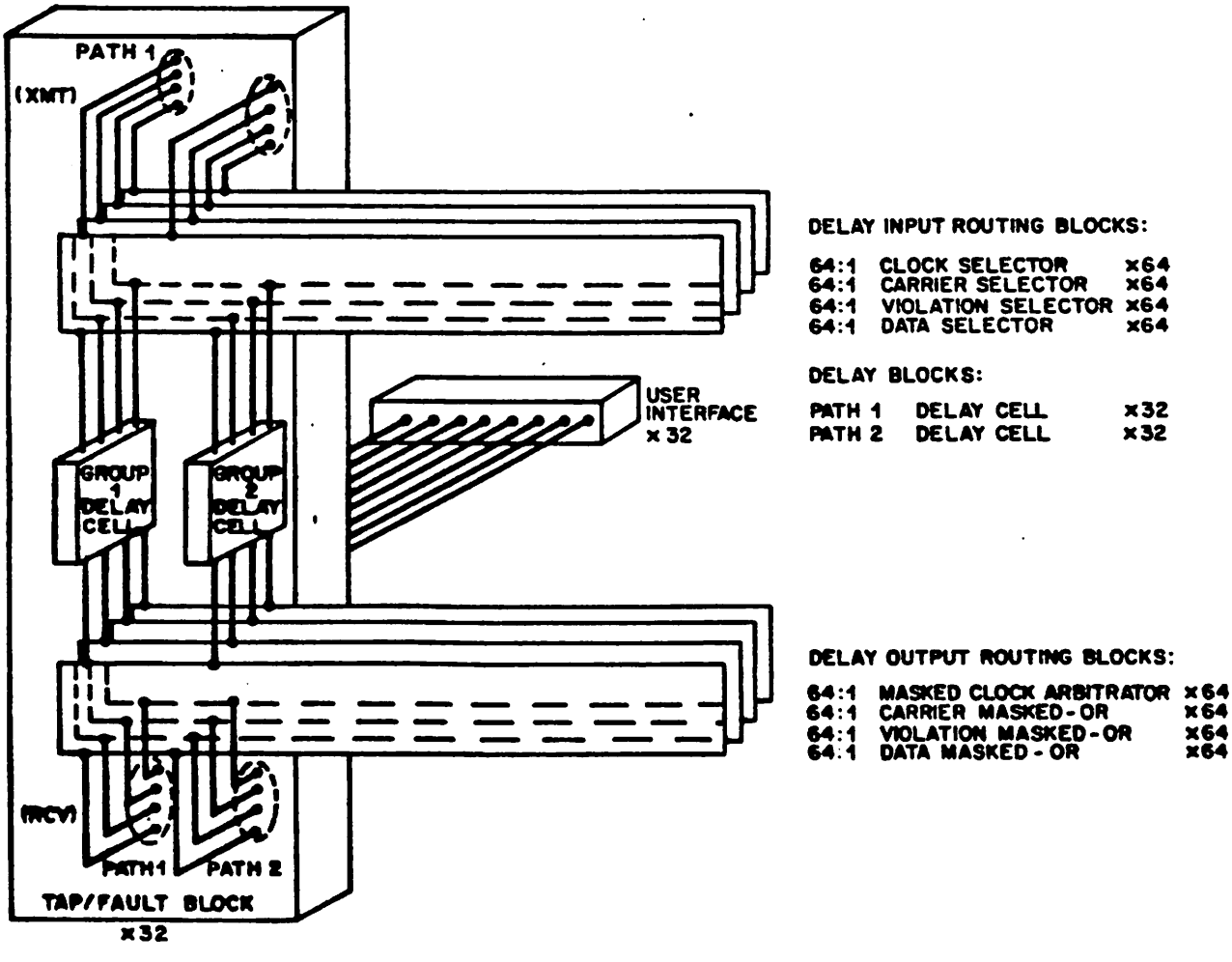
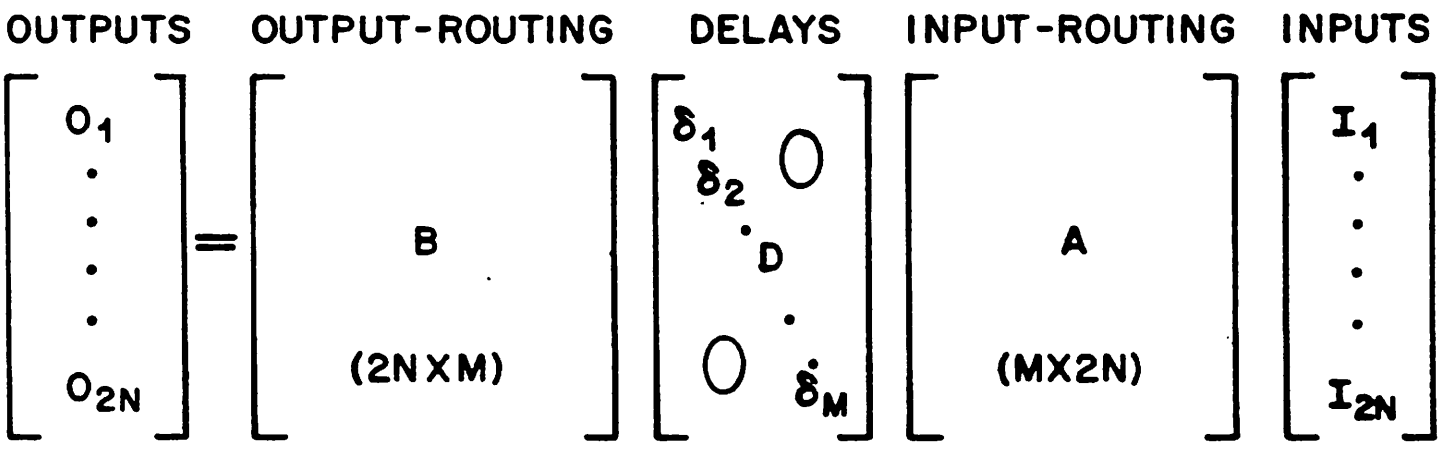


FIGURE 5b CHANNEL EMULATOR ARCHITECTURE SHOWING DETAILS OF SIGNAL PATHS & BLOCK



ELEMENTS OF A&B MATRICES ARE 1 OR 0. A HAS AT MOST ONE "1" PER ROW

ELEMENTS OF D DIAGONAL MATRIX ARE DELAY OPERATORS $\delta_K \in \{Z^0, Z^{-1}, Z^{-2} \ldots Z^{-D_{MAX}}\}$

ELEMENTS OF INPUT & OUTPUT VECTORS ARE BINARY SIGNALS FROM & TO TAP BLOCKS

SCALAR MULTIPLICATION DENOTES LOGICAL "AND", NOTATING A ROUTING FUNCTION

SCALAR ADDITION DENOTES LOGICAL "OR", NOTATING A COLLISION FUNCTION

MATRICES A & B CAN BE VIEWED AS UNIDIRECTIONAL CROSS-BAR SWITCH CONFIGURATIONS

FIGURE 6 OPERATOR VIEW OF CHANNEL EMULATOR INTERCONNECTIONS

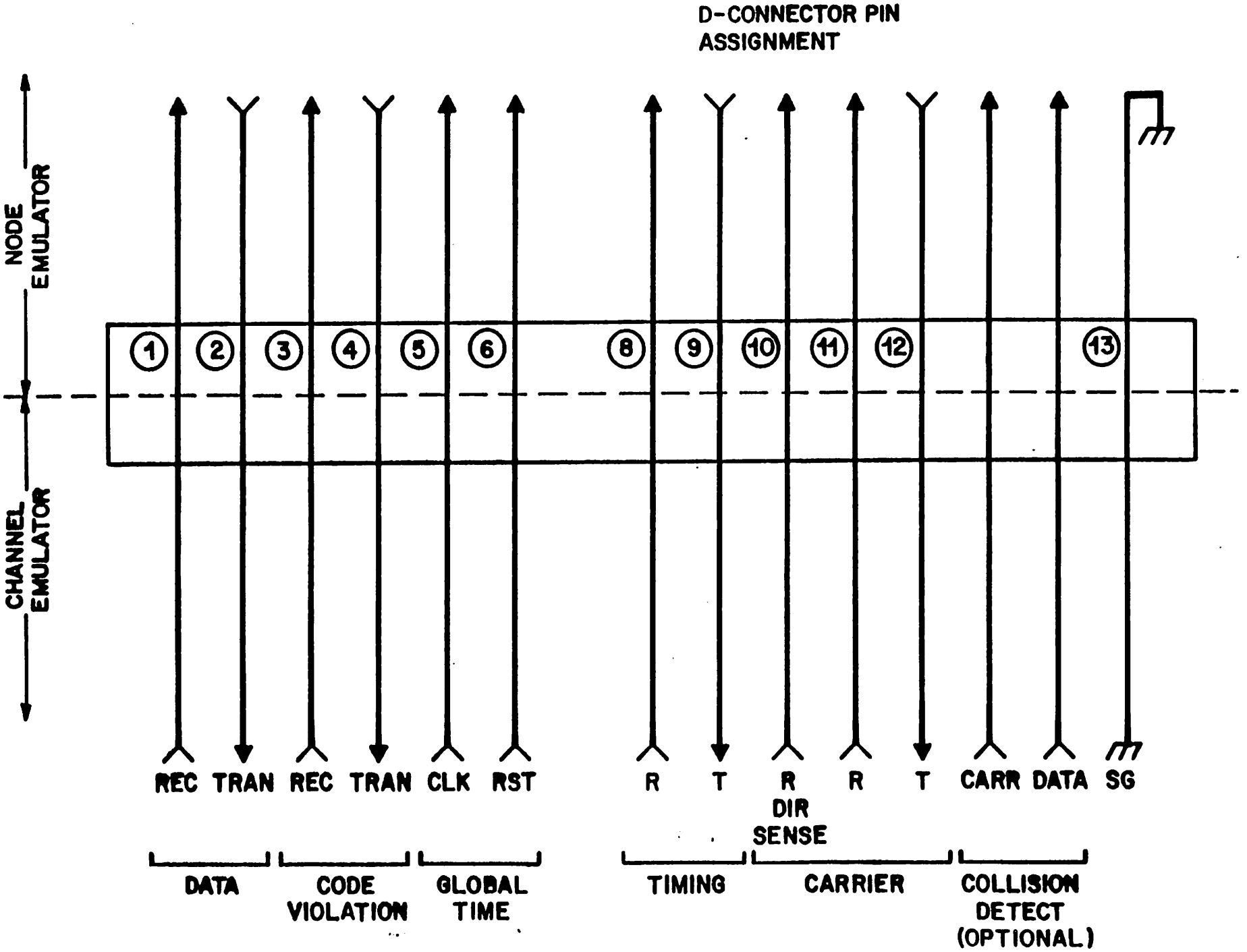
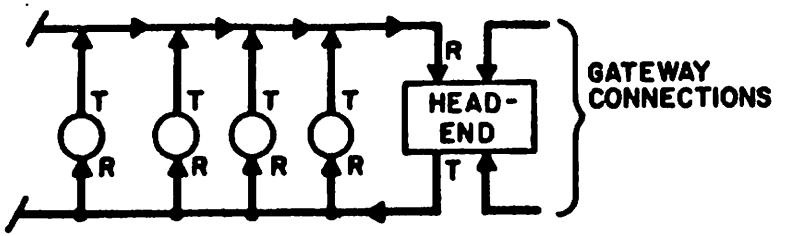
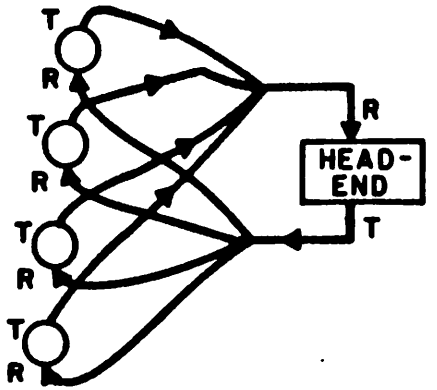


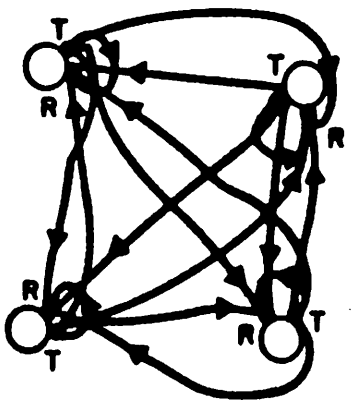
FIGURE 7 NODE EMULATOR/CHANNEL EMULATOR INTERFACE



a. NETWORK EMPLOYING A GATEWAY



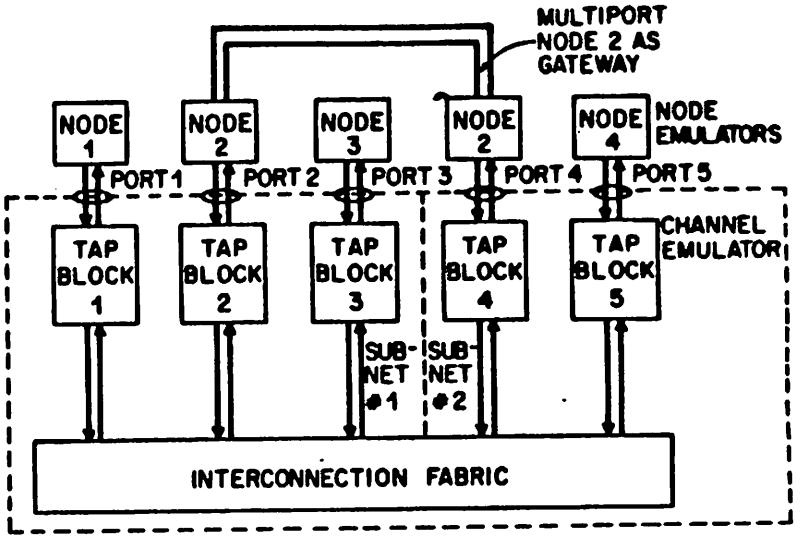
b. COLLISION STAR WITH HEAD END OR DUAL-FREQUENCY RADIO NETWORK



c. FULL CONNECTIVITY RADIO NETWORK

FIGURE 8

d. GATEWAY AND SUBNETWORKS



e. CENTRALIZED SWITCH WITH ONLY POINT-TO-POINT CONNECTIONS

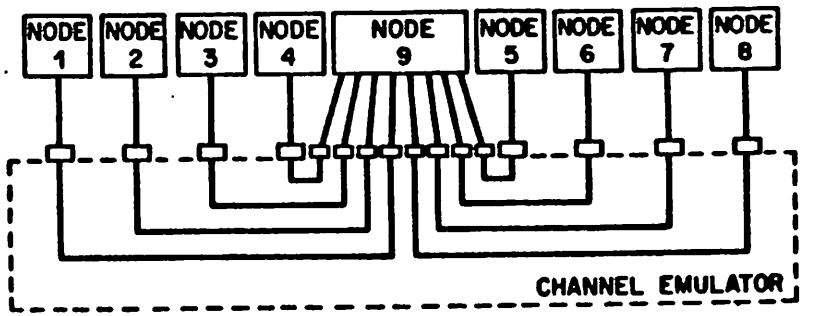


FIGURE 8 CONTINUED

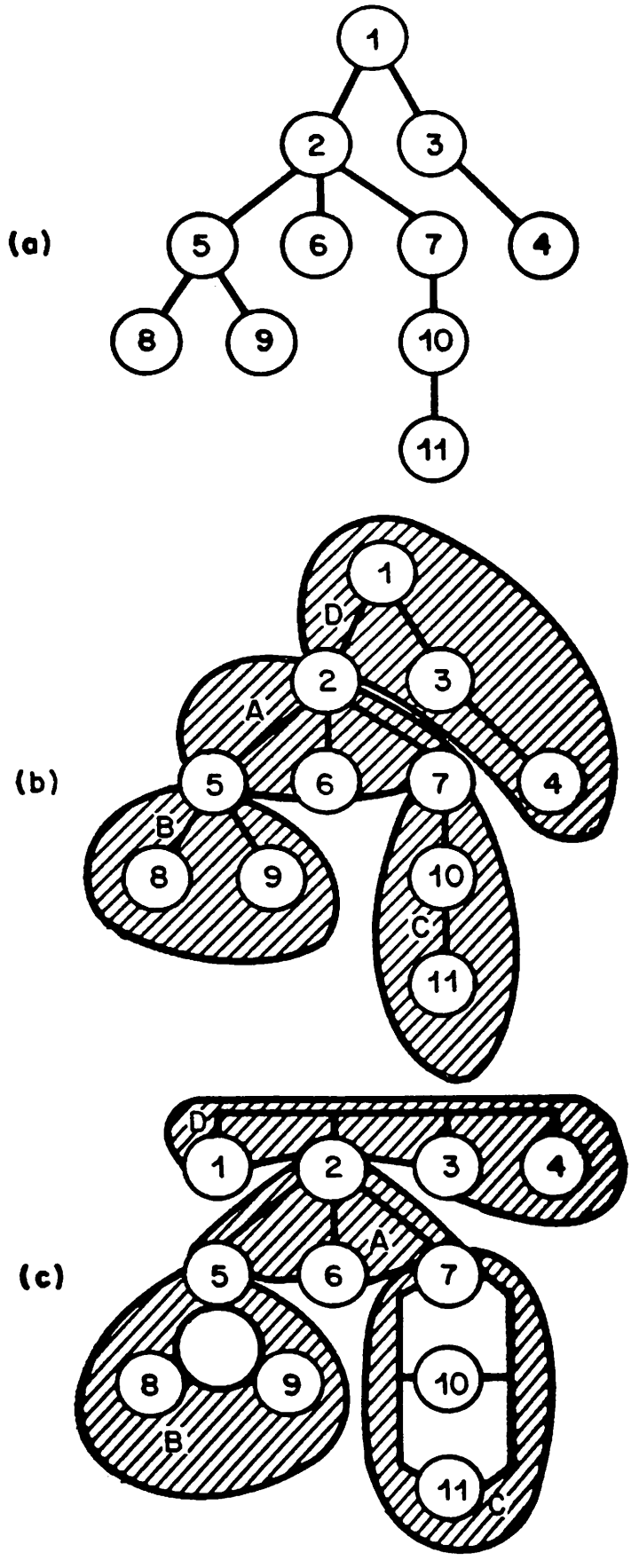
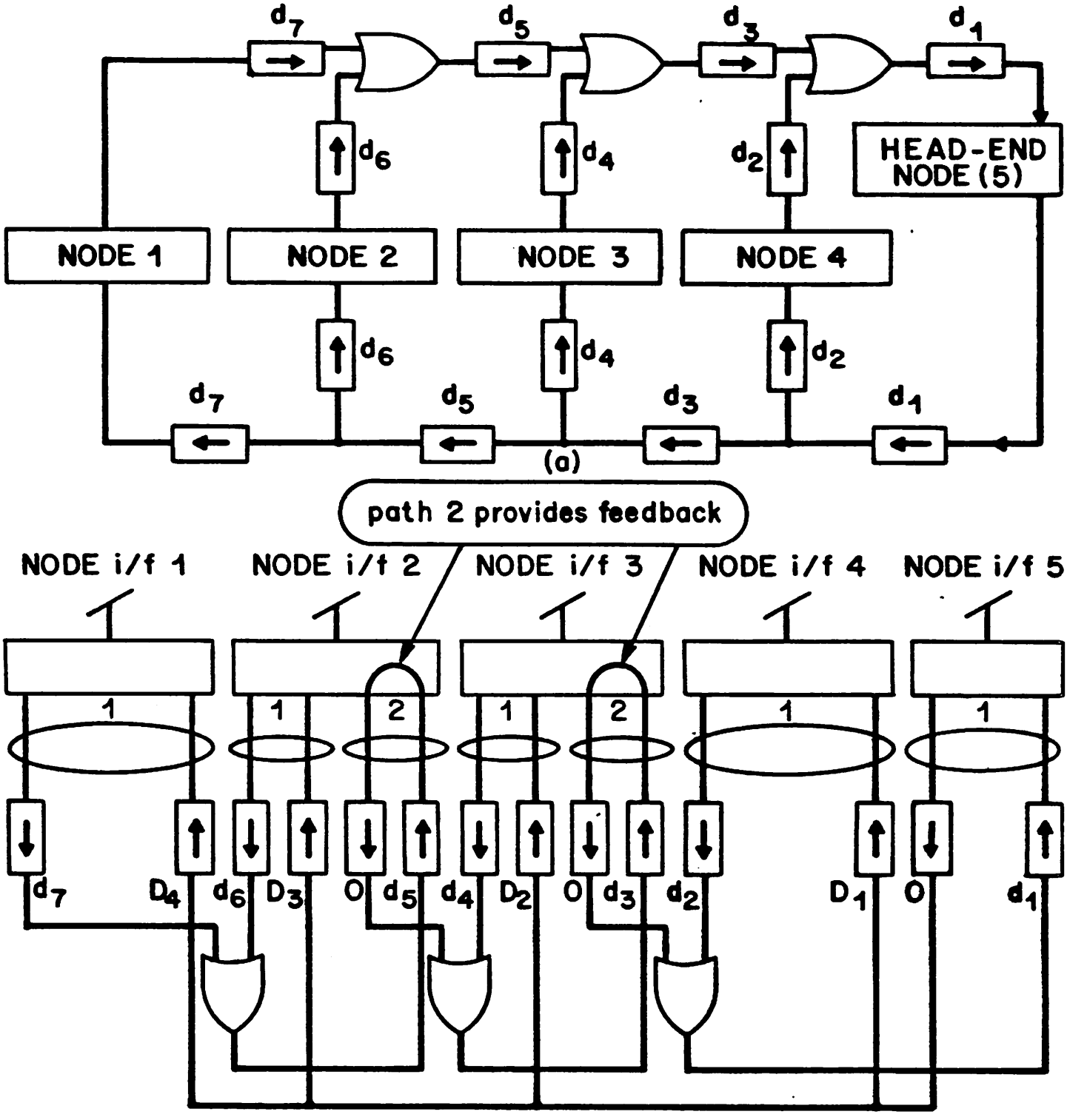


FIGURE 9 HIERARCHICAL OR ACTIVE TREE NETWORKS



$$D_1 = d_1 + d_2$$

$$D_2 = d_1 + d_3 + d_4$$

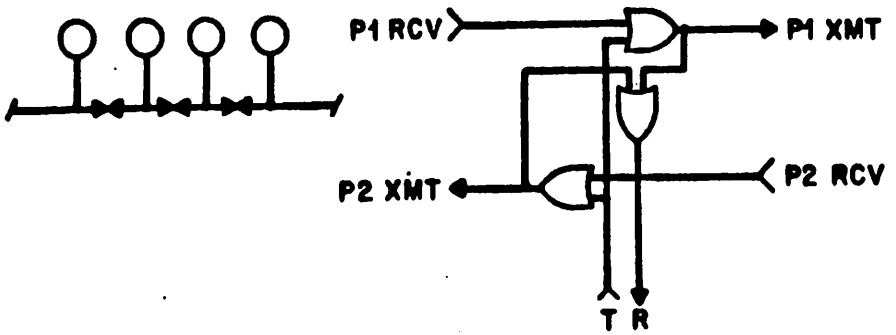
$$D_3 = d_1 + d_3 + d_5 + d_6$$

$$D_4 = d_1 + d_3 + d_5 + d_7$$

(b)

FIGURE 10 PASSIVE TREES

a. BIDIRECTIONAL BUS (BB)



b. FOLDED BUS (FBa)

c. FOLDED BUS (FBb)

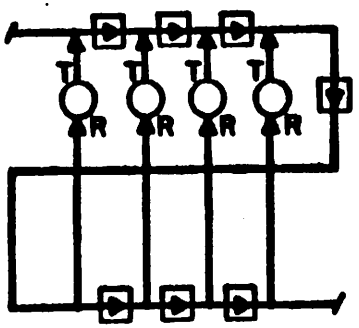
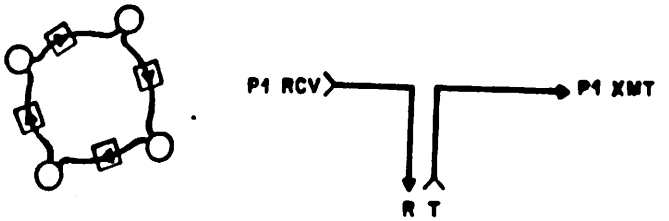
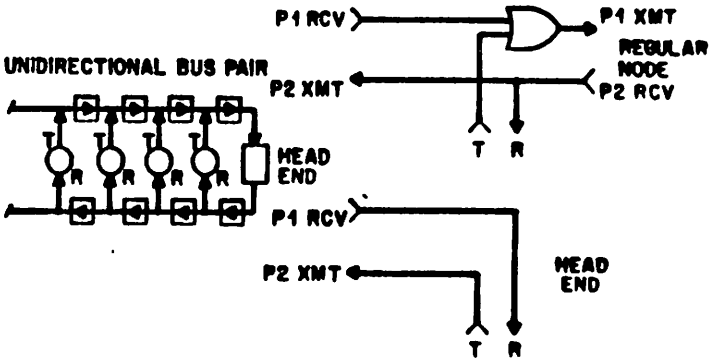


FIGURE 11 a–f GLOBAL AND LOCAL TOPOLOGY CONFIGURATIONS; LOCAL CONFIGURATIONS ARE FOR DATA, VIOLATION, OR CARRIER SIGNALS

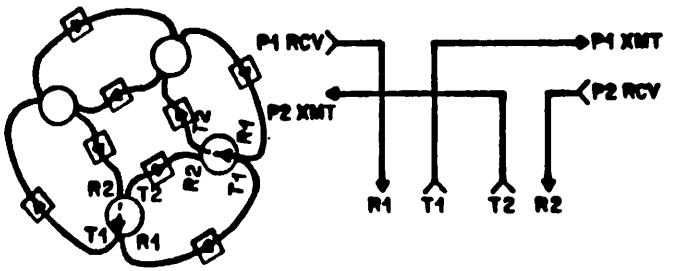
d. UNIDIRECTIONAL RING



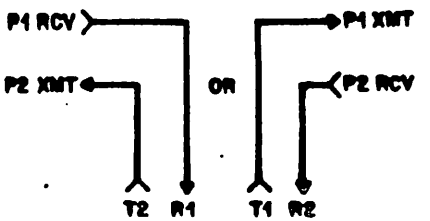
e. UNIDIRECTIONAL BUS PAIR



f. COUNTER-ROTATING RING USING 2 PORTS



DEAD NODE:



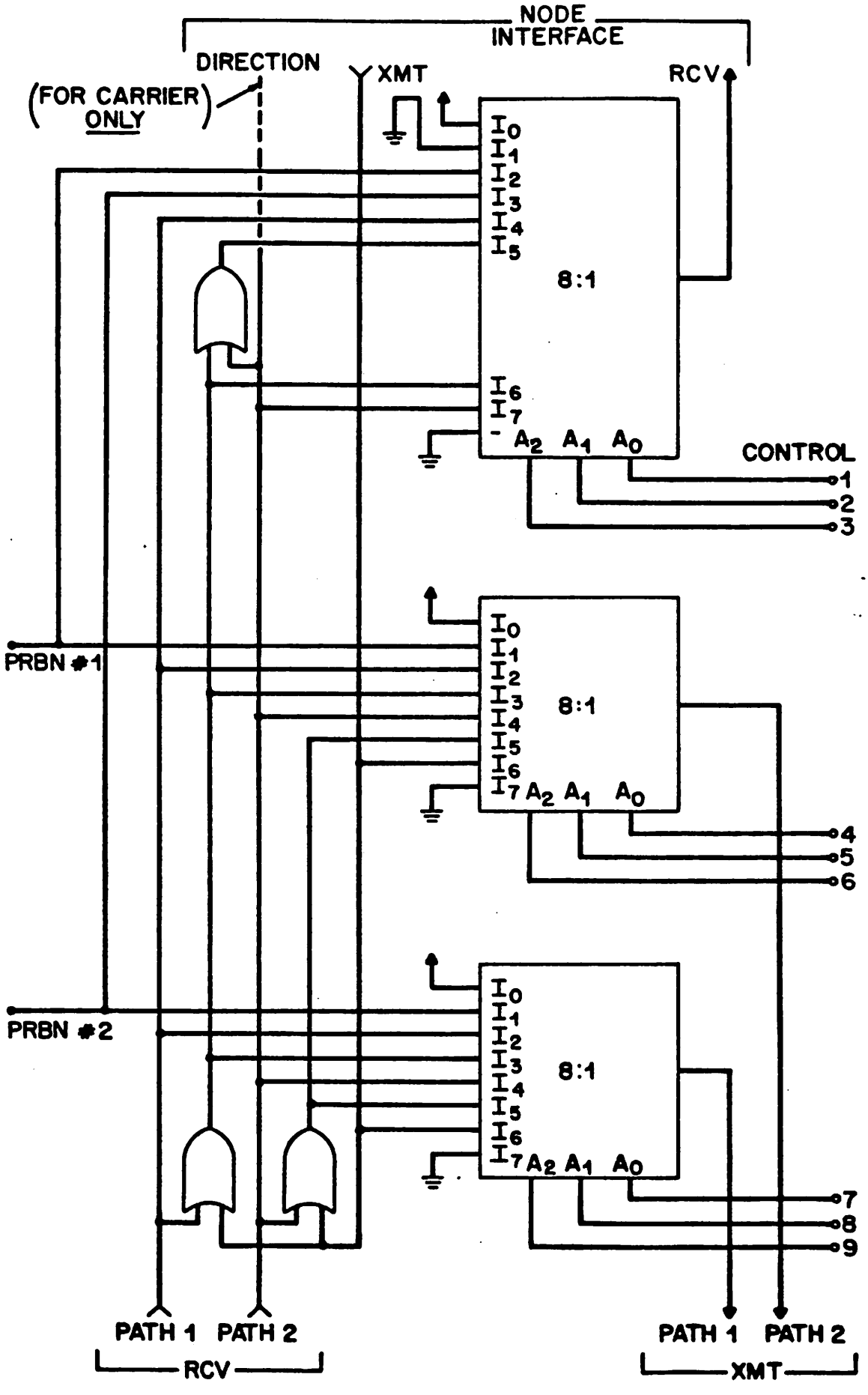


FIGURE 12 TAP/FAULT CELL FOR DATA, VIOLATION, AND CARRIER

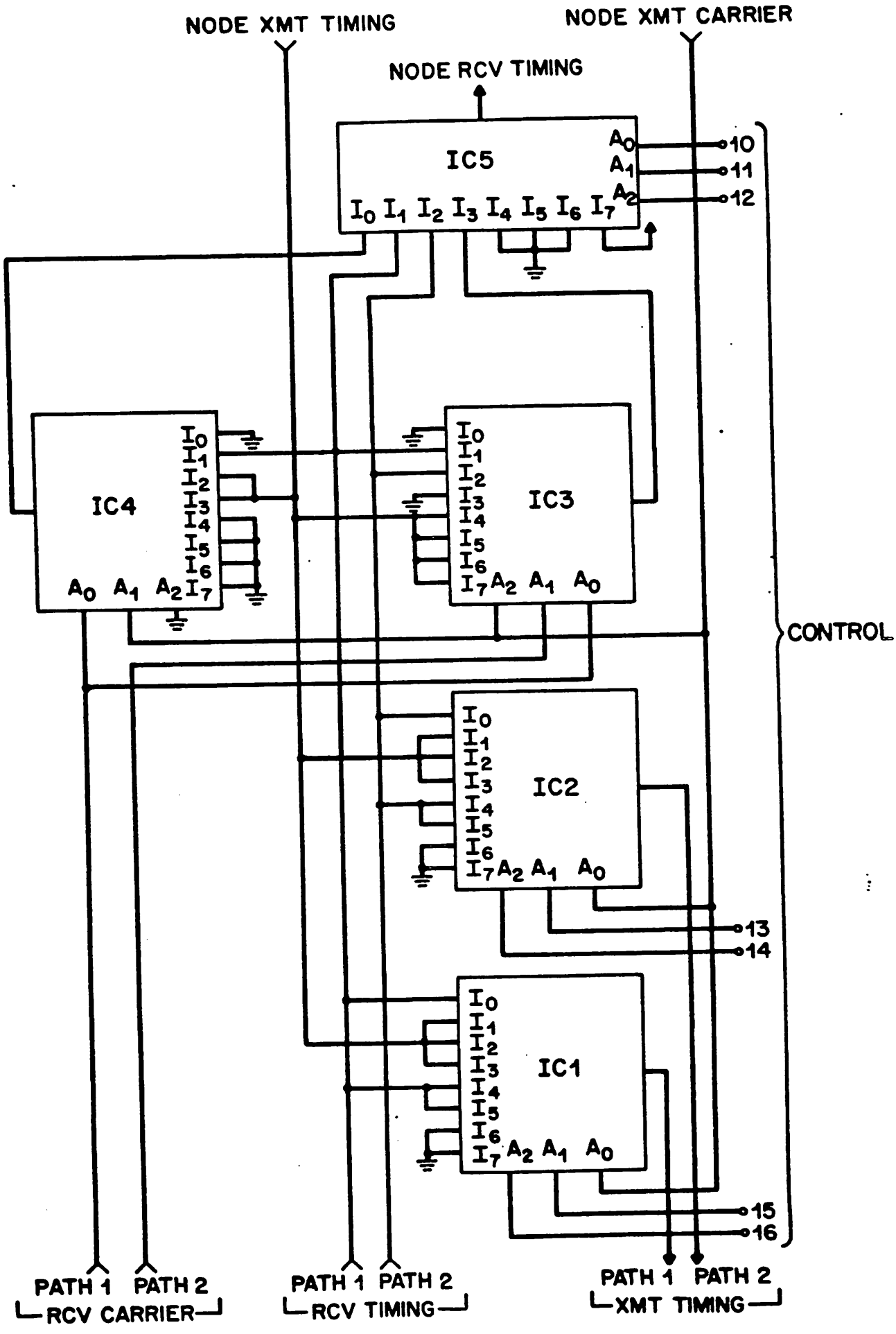


FIGURE 13 TOPOLOGY/FAULT CELL FOR CLOCK

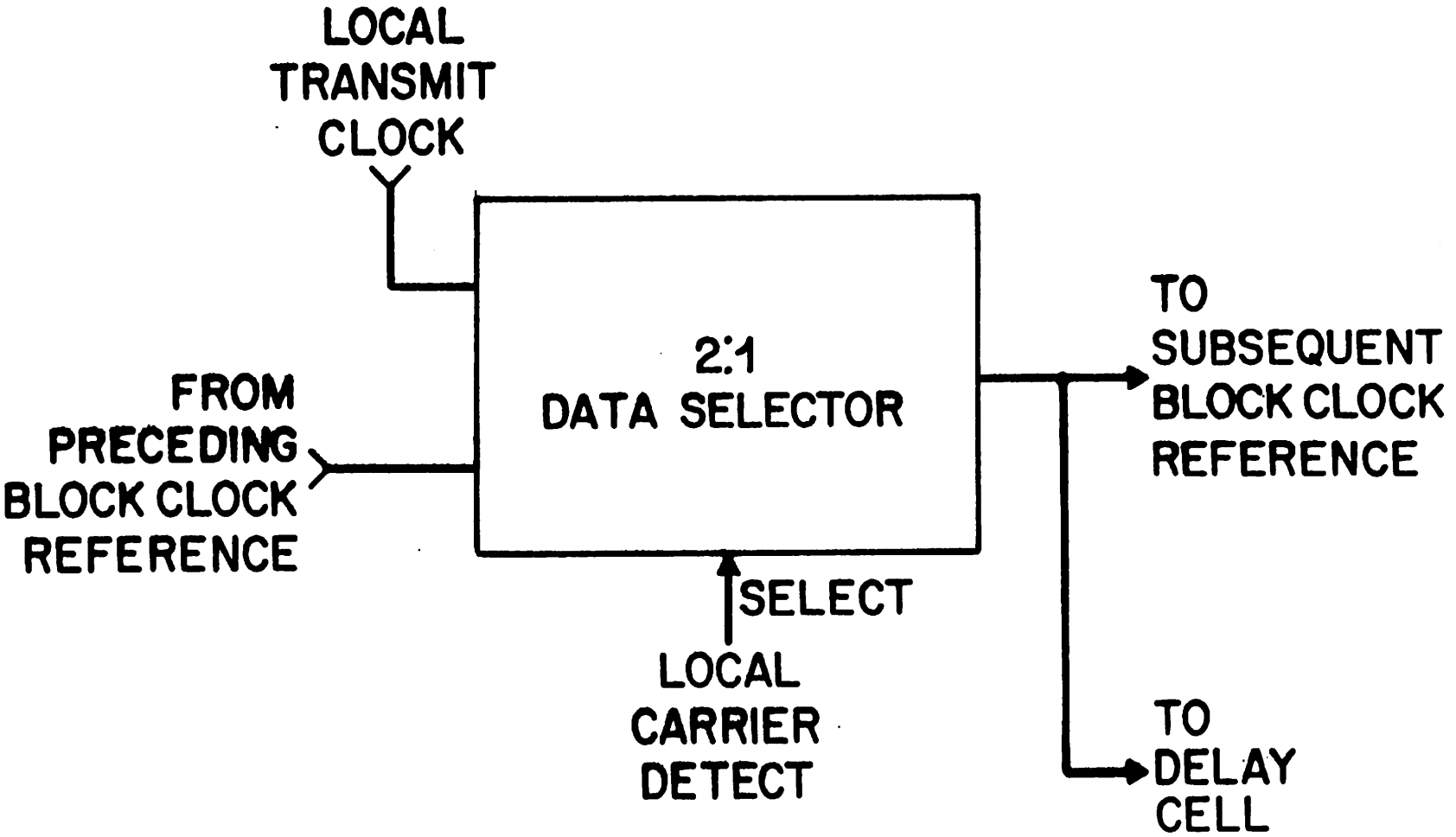


FIGURE 14 CLOCK ARBITRATOR CIRCUIT

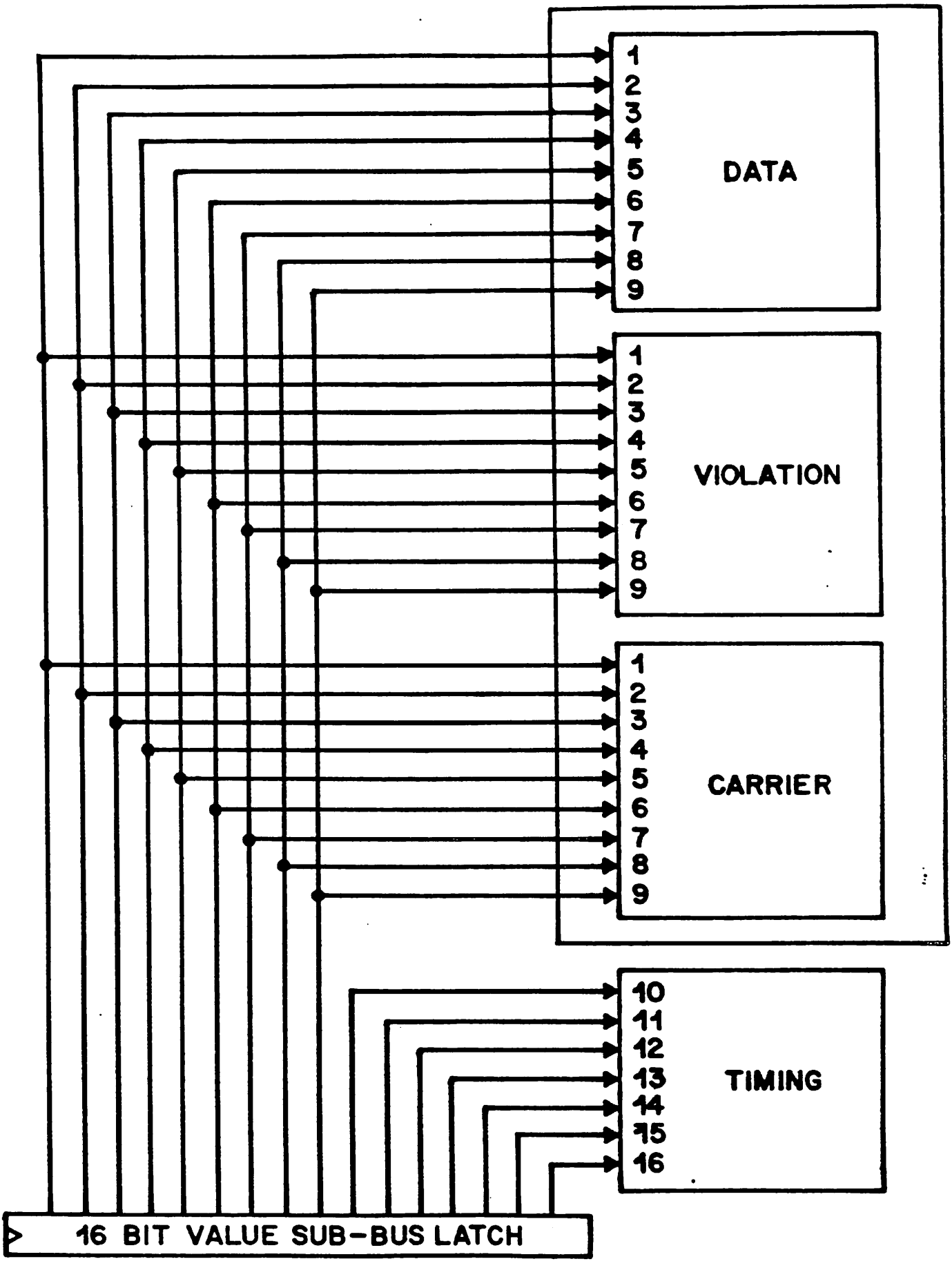


FIGURE 15 CONTROL BUS CELL WIRING TO TOPOLOGY/ FAULT CELLS WITHIN THE TAP BLOCK

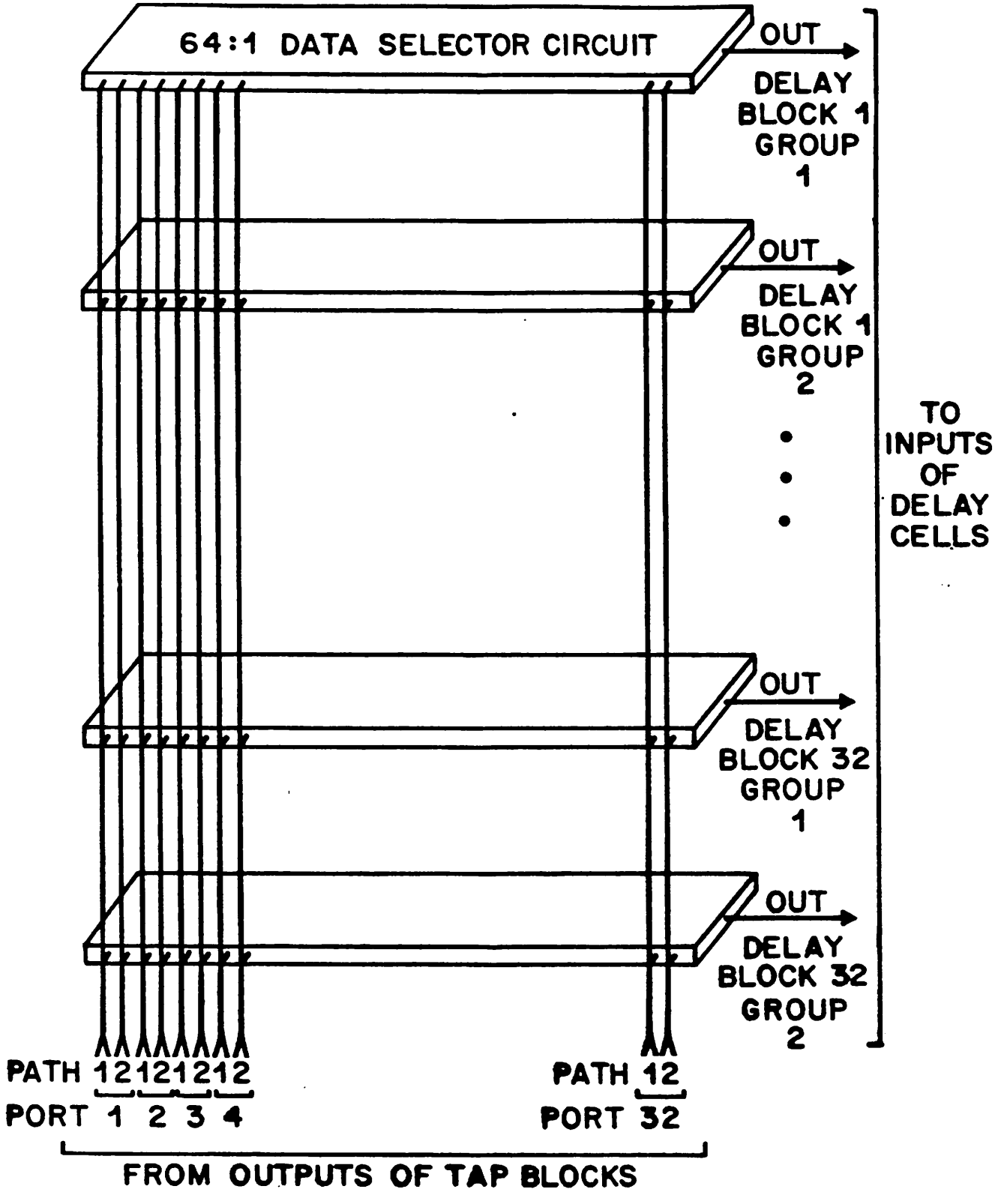


FIGURE 16 DELAY-INPUT ROUTING BLOCK. THIS IS REPEATED ONCE EACH FOR DATA, VIOLATION, CARRIER, AND TIMING SIGNALS

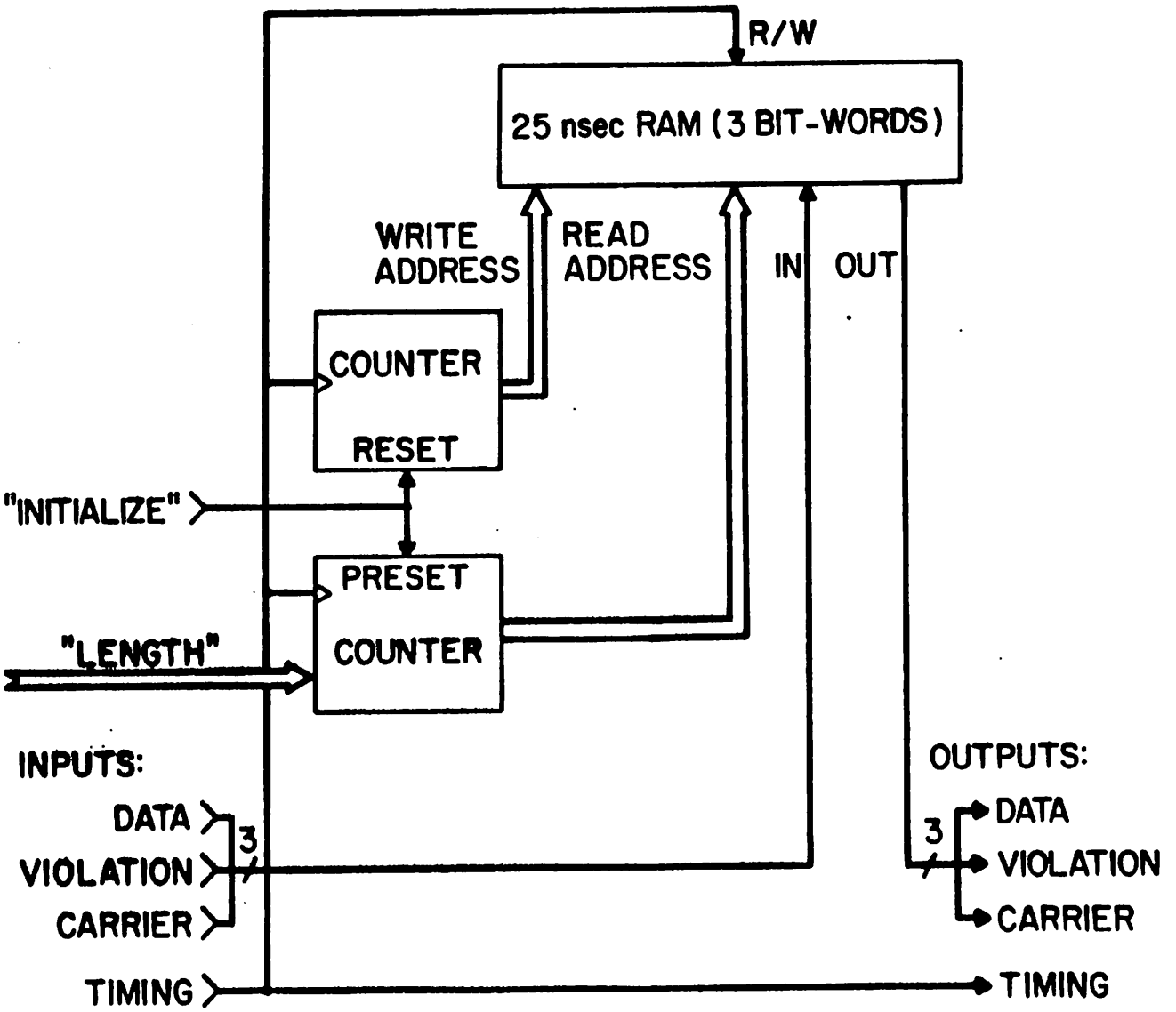


FIGURE 17 ARCHITECTURE OF RAM-BASED DELAY CALL

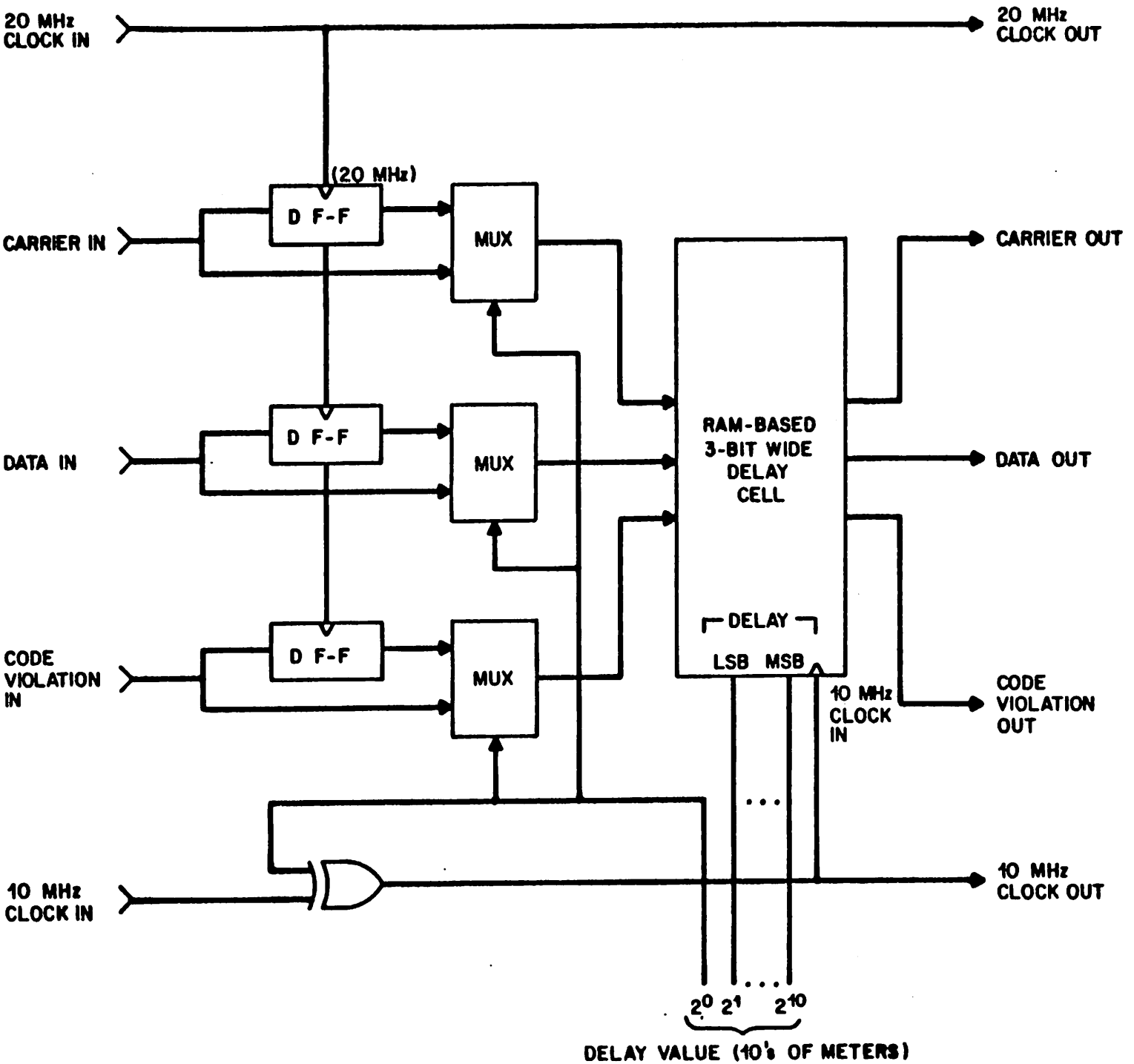
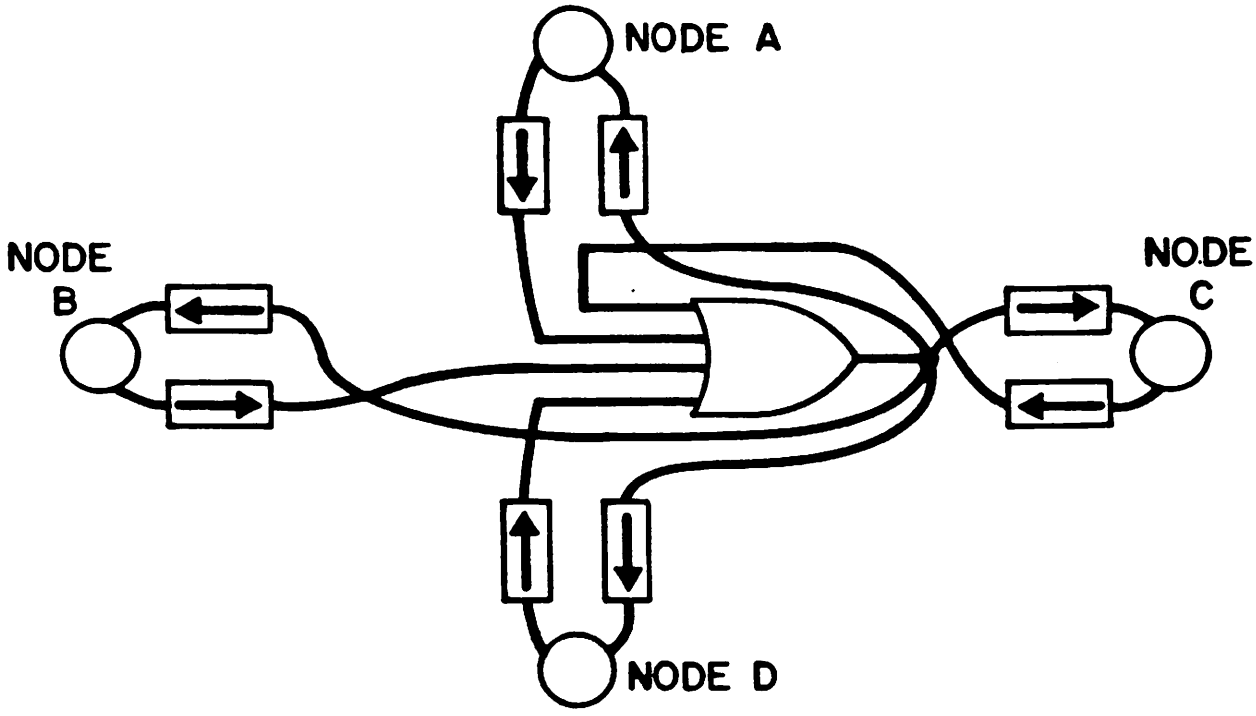


FIGURE 18 DELAY CELL CIRCUIT WITH INTERPOLATER

a. STAR TOPOLOGY



b. FOUR NODE RADIO TOPOLOGY, DECENTRALIZED

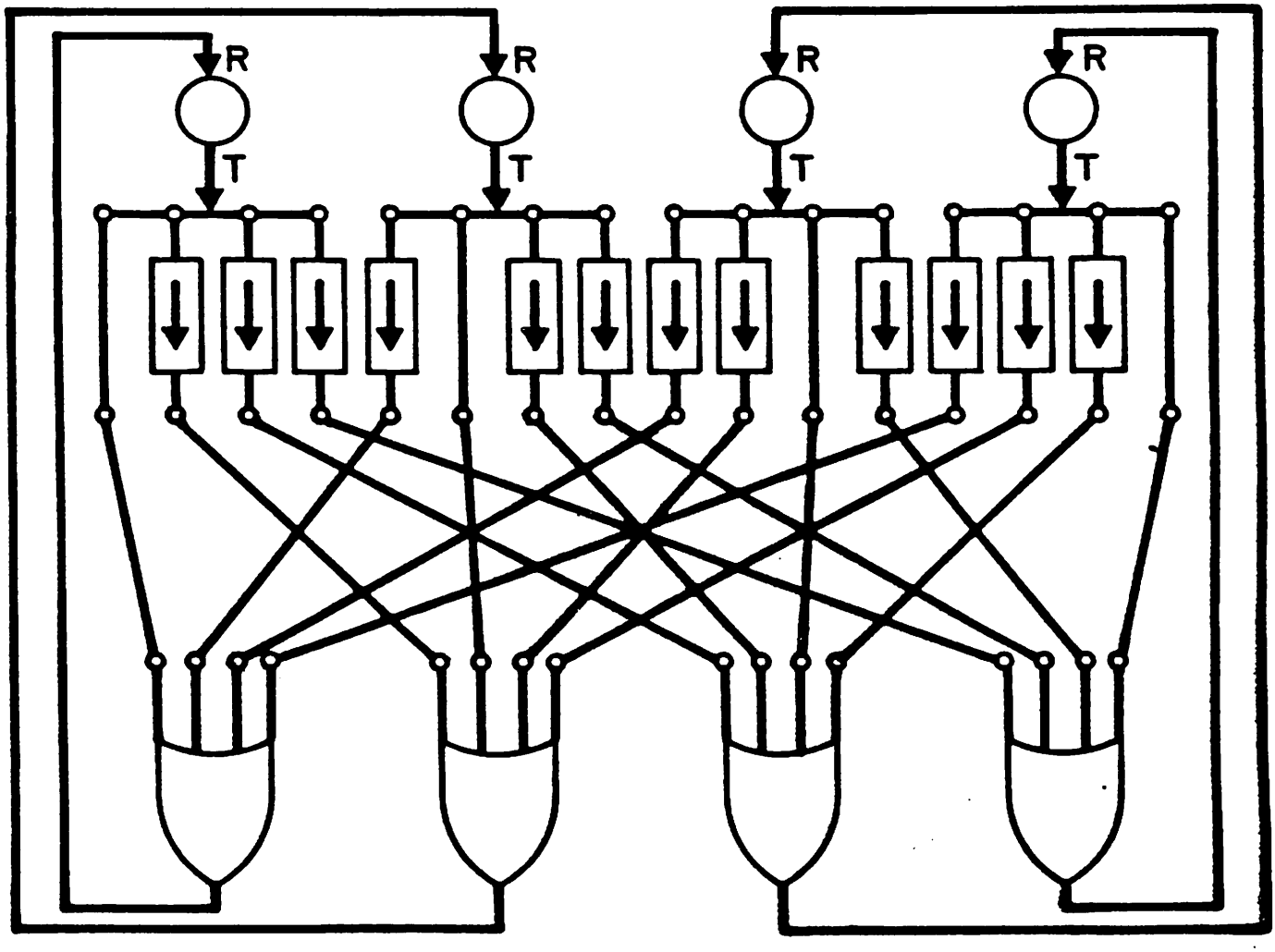


FIGURE 19

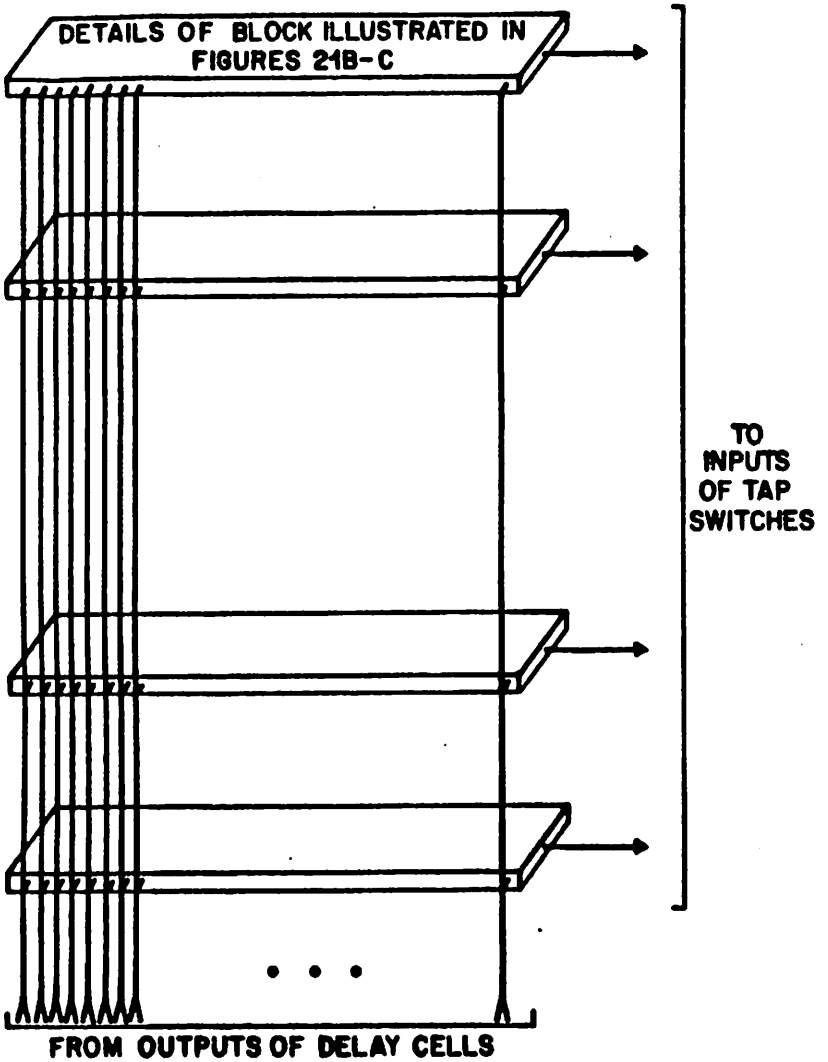
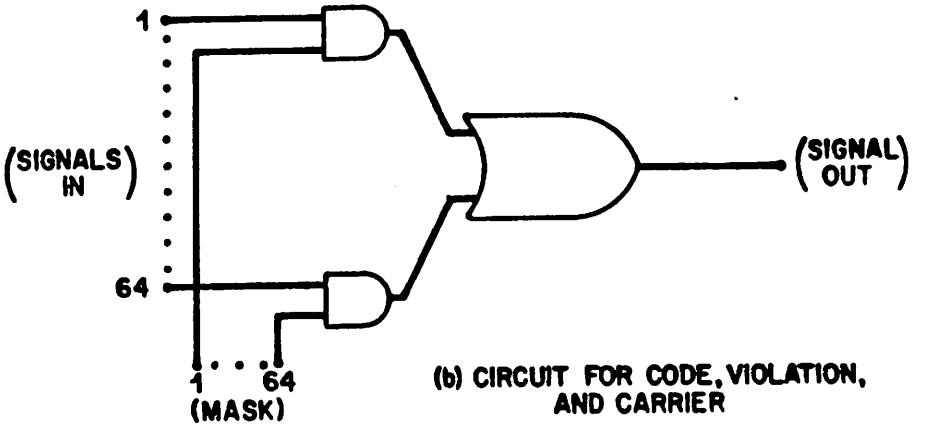
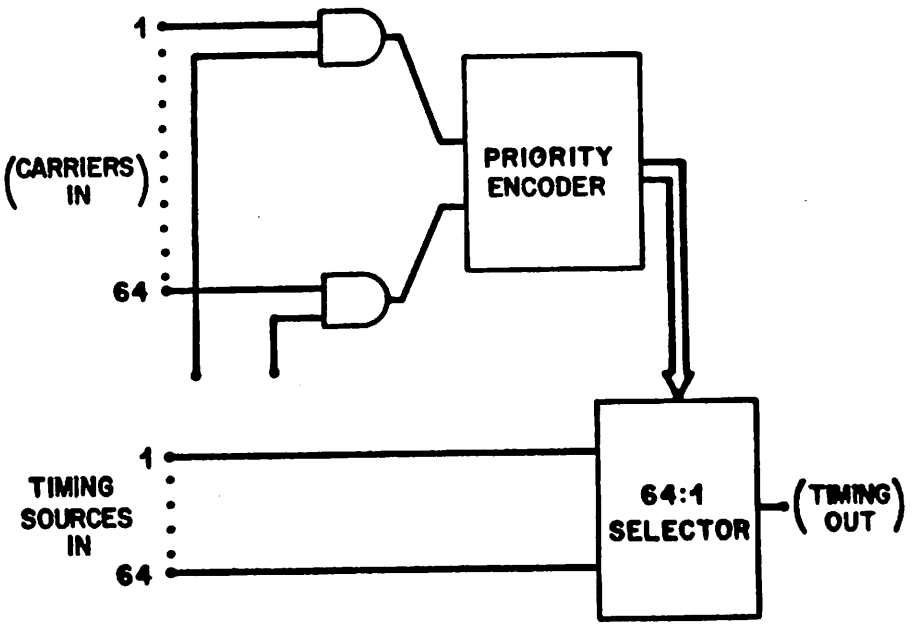


FIGURE 20a DELAY-OUTPUT ROUTING BLOCK



(b) CIRCUIT FOR CODE, VIOLATION, AND CARRIER



(c) CIRCUIT FOR TIMING

FIGURE 20 CONTINUED

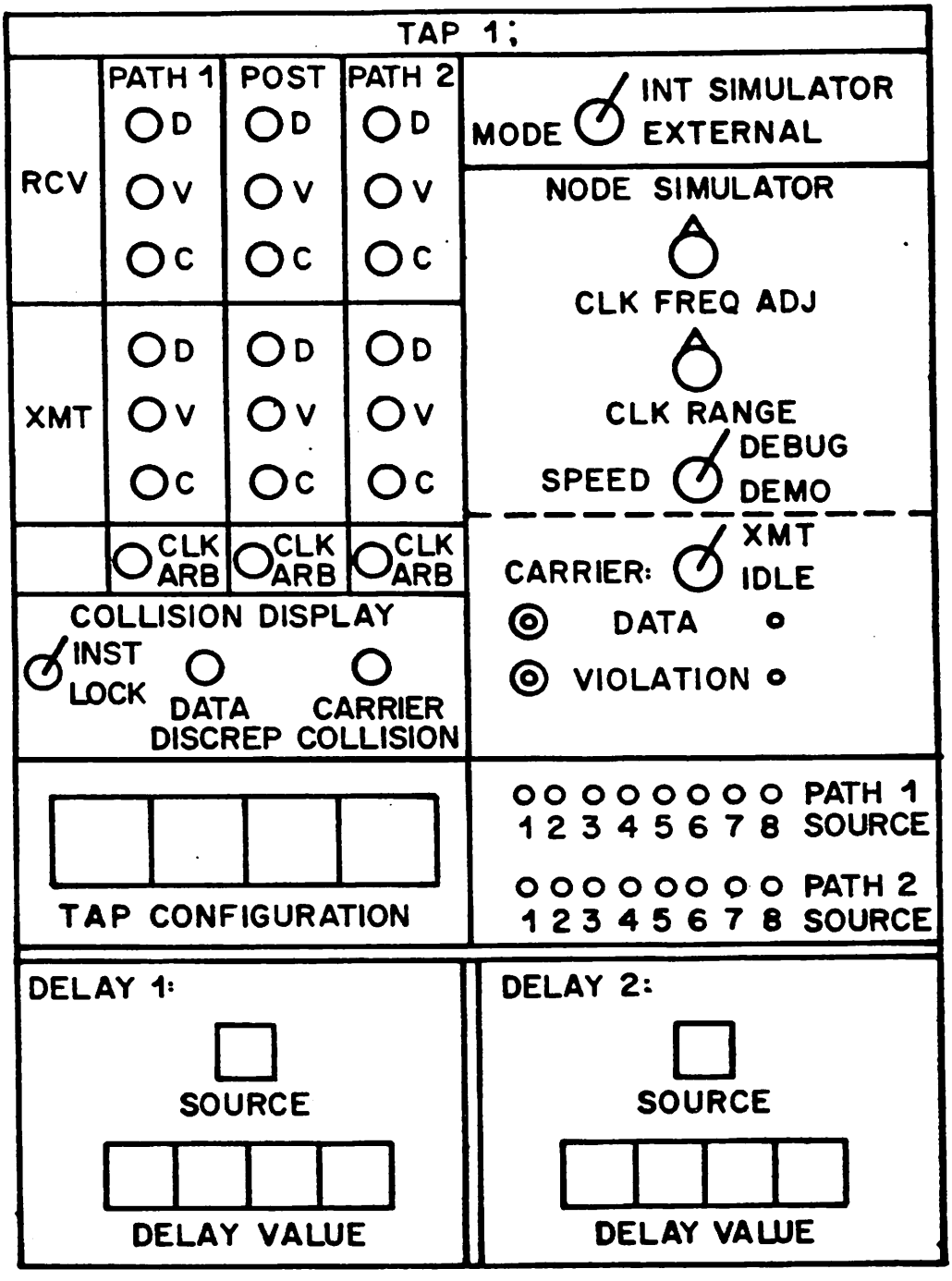


FIGURE 21 SAMPLE OPTIONAL CONTROL PANEL SEGMENT. FOR THE M=2N CHANNEL EMULATOR, THIS SEGMENT IS REPEATED N TIMES