1985

A high speed special purpose processing unit for logic simulation

Vineet Kumar
Iowa State University

Kumar, Vineet

A HIGH SPEED SPECIAL PURPOSE PROCESSING UNIT FOR LOGIC SIMULATION

Iowa State University PH.D. 1985

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark __✓__.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages __✓__
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received _____
16. Other _____

A high speed special purpose processing unit for logic simulation

by

Vineet Kumar

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering and Computer Engineering
Major: Computer Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1985

TABLE OF CONTENTS

NOMENCLATURE vii

INTRODUCTION 1

LITERATURE REVIEW 4

OVERVIEW 16

DESCRIPTION OF THE PROCESSING UNIT 18

DESCRIPTION OF UNITS IN THE SUBPROCESSING UNIT 35

Model Accessing and Updating Unit 35
Evaluation Unit 53
Communication Unit 54
Scheduling Unit 66
Diagnostics/WCS Controller and Host Computer Interface Unit . . 83

CONCLUSION 99

BIBLIOGRAPHY 105

ACKNOWLEDGEMENT 107

APPENDIX A: PROPOSED IMPLEMENTATION OF THE EVALUATION UNIT 108

Hardware 108
Normal Mode 108
Diagnostics and writable control store mode 143
Description of the Microinstruction 146
Firmware 152

APPENDIX B: DATA SHEET 171

Cascadable 16 Bit Error Detection and Correction Unit (AM 2960) 172
SSR Diagnostics/WCS Pipeline Register (AM 29818) 178
Clock Generator and Microcycle Length Controller (AM 2925) . . . 180
Microprogram Controller (AM 2910-1) 183
16 Bit Bipolar Microprocessor (AM 29116) 191
2048 x 8 Bit Bipolar PROM (AM 27S191A) 214
1024 x 4 Bit Static RAM (AM 2149) 216
256 x 4 Bit TTL Bipolar RAM (AM 93422A) 220
First-In First-Out Memory (TDC 1030) 224
Synchronous 8 Bit Up/Down Counter (74AS869) 228

LIST OF FIGURES

Figure 1. Integrated circuit complexity trends 1

Figure 2. Typical operations during a time frame 6

Figure 3. Ring pipeline architecture of a simulator 7

Figure 4. Yorktown Simulation Engine overview 8

Figure 5. Logic processor overview 10

Figure 6. Architecture of proposed multiprocessor simulator . . . 12

Figure 7. Slave unit configuration 13

Figure 8. Master configuration 14

Figure 9. Host computer - processing unit interface 19

Figure 10. Architecture of the processing unit 21

Figure 11. Architecture of the subprocessing unit 23

Figure 12. Flow of data for a source device with external fanouts . 25

Figure 13. Flow of data for a source device with internal fanouts only 27

Figure 14. Flow of data for a fanout device which has an external source 28

Figure 15. EDC device in check-only configuration 33

Figure 16. Numbering of inputs and outputs of a device 37

Figure 17. COMM's view of the processing unit 57

Figure 18. Arrangement of 2K words of COMM memory 61

Figure 19. Time wheel 67

Figure 20. Arrangement of SCHED memory 71

Figure 21. Information of a device stored on the time wheel 78

Figure 22. Partitioning of a network into subnetworks 85

Figure 23. Typical configuration of a unit 89

Figure 24. Connections for Diagnostics/WCS registers 91

Figure 25. Additional connections for writing into WCS 94

Figure 26. Connections to perform Diagnostics and write into WCS . 97

Figure 27. Block diagram of EVAL 109

Figure 28. FIFO 111

Figure 29. 256 x 16 bit external RAM 112

Figure 30. Control store 116

Figure 31. Design of EVAL in Normal mode 119

Figure 32. Timing diagram of $\overline{\text{RESET}}$ 122

Figure 33. Timing diagram for reading and writing into 29116s' RAM within one cycle 125

Figure 34. Timing diagram for using the CT output of 29116 126

Figure 35. Immediate instruction cycle timing 127

Figure 36. Timing diagram to write into the external RAM and load counters from 29116 128

Figure 37. Timing diagram to read from the external RAM by 29116 . 130

Figure 38. Timing diagram for reading from FIFO into 29116 132

Figure 39. Timing diagram for writing into FIFO from 29116 134

Figure 40. Timing diagram for using 29116 to generate the next microaddress 135

Figure 41. Timing diagram for writing into FIFO and reading from the external RAM 136

Figure 42. Timing diagram for reading from FIFO and writing into the external RAM 137

Figure 43. Timing diagram for latching data in the D-latch of 29116 139

Figure 44. Timing diagram for the JMAP instructin in 2910-1 141

Figure 45. Connections in EVAL to support Diagnotics/WCS 144

Figure 46. Arrangement of input and output signal values 157

Figure 47. Arrangement of output signal values packed in one memory word 162

LIST OF TABLES

Table 1. Digital devices grouped into categories 36

Table 2. Model of a simple device 38

Table 3. Model of a functional device with 3 signal values, single delay, and 32 inputs - 32 outputs 40

Table 4. Model of a functional device with 3 signal values, variable delay, and 32 inputs and 32 outputs 41

Table 5. Different selections from 'command of the device' field in model memory 43

Table 6. Fanout information of an external source device 62

Table 7. Function table for external RAM 115

Table 8. Constant delay in control path 140

Table 9. Length of clock outputs 142

Table 10. Status flags 147

Table 11. Microcycle length requirements for various fields in the microword 149

Table 12. Reset routine and subroutines 153

Table 13. Truth table for a 2 input NAND gate 156

Table 14. Microroutine for a 2 input NAND gate using a table look-up scheme 158

Table 15. Microroutine to convert a 2 or a 3 input NAND gate into a 4 input NAND gate 161

Table 16. Microroutine of a 4 input NAND gate using a table look-up scheme 164

Table 17. Microroutine for a 7 input NAND gate evaluated without tables 168

Table 18. NAND gate evaluation time for the three methods 170

NOMENCLATURE

ALU = Arithmetic Logic Unit

AMD = Advanced Micro Devices

CAD = Computer Aided Design

CDM = Captured Data Memory

CMOS = Complementary Metal Oxide Semiconductor

COMM = Communication Unit

CPU = Central Processing Unit

ECL = Emitter Coupled Logic

EDC = Error Detection and Correction

EVAL = Evaluation Unit

FDM = Fanout Data Memory

FIFO = First In First Out

HOST INTERFACE = Diagnostics/WCS Controller and Bus Interface Unit

IBM = International Business Machines

IC = Integrated Circuit

ID = Identification number

K = 1024

LIFO = Last In First Out

LSB = Least Significant Bit

LSH = Least Significant Half

LSSD = Level Sensitive Scan Design

Mega = 1.024×10^6

MODEL = Model Accessing and Updating Unit

MOS = Metal Oxide Semiconductor

MSB = Most Significant Bit

MSH = Most Significant Half

MSI = Medium Scale Integrated

ns = Nanoseconds

PROM = Programmable Read-Only Memory

PU = Processing Unit

RAM = Random Access Memory

ROM = Read Only Memory

RTL = Resistor Transistor Logic

SCHED = Scheduling Unit

Sub PU = Sub Processing Unit

TTL = Transistor Transistor Logic

us = Microseconds

VLSI = Very Large Scale Integrated

WCS = Writable Control Store

INTRODUCTION

Software-based simulators have long been used as an aid in the design of integrated circuits. The number of components in a chip has on an average been doubling every 1.5 to two years, as shown in Figure 1 (1). There are at present one million components in a chip, and it is anticipated that this number will reach ten million by 1990. The larger the complexity becomes, the more time-consuming design, error detection, and correction becomes. This leads to an increasing emphasis on design verification as early as possible in the development cycle to shorten the time and to cut the development cost. Simulation can provide warnings of possible design mistakes at a time when correcting them is relatively cheap.

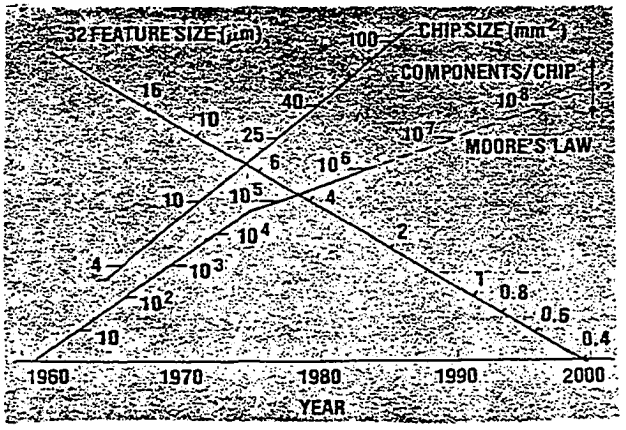


Figure 1. Integrated circuit complexity trends

The growing complexity of integrated circuits has resulted in a significant growth of computing requirements of CAD tools and has made their design dependent on the existence of adequate CAD tools. The architecture of general purpose computers supports only sequential algorithms; thus their speed is limited. Consider an example from IBM (2): An IBM 370/168 can perform about 30,000 gate evaluations/second; that is, 33 us per gate evaluation. It took 1800 hours of CPU time to verify the logic design of one-fourth of a medium-scale System 370 CPU. Four 370/168s were used for months (two shifts a day) to verify the logic of a high end System 370 CPU.

One obvious solution to the problem of increasing simulation time is to use faster and larger computers with the same programs. Unfortunately, the speed of large mainframes is not increasing as fast as the increase in simulation time caused by the increase in gates/chip. Hence, this solution may not work due to the large cost of mainframes and for the simple reason that the gate/chip size may increase beyond the capability of the machine. In fact, software-based simulation is being stretched to the limit by modern computer techniques. Another solution is to implement the existing algorithms in hardware. This approach does not offer any flexibility to changing technology and is unattractive because of large expense incurred due to low volume production. Reasonable solutions are to develop better algorithms which have a polynomial degree closer to one (i.e., as gates/chip size increases the simulation time increases only linearly) and to design specialized processors whose architecture fits the problem they are to solve.

Extensive use of logic simulation in the design cycle requires a massive amount of computing power and time to simulate a large digital logic network. The use of a special purpose simulator can significantly reduce the iteration time in the design cycle and can transform an impossible task into a feasible one. Also, the declining cost of hardware has made the use of hardware simulators very attractive. It can shorten the simulation time to somewhere in the order of 2-1000, as compared to current software based simulators. Such hardware accelerators are being considered and developed by various manufacturers to meet the challenge posed by VLSI circuits.

In this dissertation, the architecture of a specialized Processing Unit (PU) has been proposed. Methods have been developed to fully exploit the parallelism inherent in logic simulation. The PU communicates with the host computer via a high speed bus. The host computer loads the memories of the PU with simulation data and does the pre-processing and post-processing by handling the primary inputs and the output data. However, the simulation of the logic network is done entirely by the specialized PU.

LITERATURE REVIEW

The history of simulators has been discussed elsewhere (3, 4) in books and papers. In this chapter, some of the important papers on special purpose logic simulators will be discussed briefly, and their speed will be compared to the current software-based simulators.

A **network** to be simulated consists of a description of every device and connections between them. Each digital **device** has a **type** (ex., AND, OR, flip flop, or a functional module) which identifies its functions. It can have any number of inputs and outputs, and there should be a relationship between the inputs and the outputs. The logic signal values on an input or an output could be anywhere from two (0, 1) to eight (0, 1, unknown, high impedance, rising, falling, stable, and changing). **Primitive devices** are built into the simulator, whereas **nonprimitive devices** should be specified through the hardware description language. **Configuration** of a device consists of signal values at the inputs, outputs and any internal state variables. An **event** occurs due to a change in signal value on the signal line. A change in input causes the device to be **evaluated**, and its new outputs and state variables stabilize after some time equal to the **delay** associated with the device. **Scheduling** an event is marking it to occur at some future time. All concurrent events occur at the same **simulated time** or **time frame** and are processed during a **simulation cycle**, after which time is advanced to the next time frame. All scheduled events are maintained in the **event list**. The above method of simulation is referred to as an **event-driven simulation**. Any device

which uses about the same amount of memory, and can be evaluated in about the same time as a gate, is referred to as **simple.** Other devices are called **functional.**

Typical operations which occur during a simulation cycle are shown in Figure 2. Abramovici et al. (5) made use of the pipelined characteristic of logic simulation to design a simulator which employs distributed and parallel processing and resembles a ring pipeline as shown in Figure 3. All the processing units contain local memory and microprogrammable processors so that the specialized tasks are implemented directly in microcode. Communication between neighboring processing units is through FIFO message buffers. The Event List Manager orders the events in the Event List memory in increasing order and stores the concurrent and most recent events in a block. The Current Event Processing Unit retrieves events from this block and sends them to the Simple or Functional Configuration Processing Unit (depending on whether the device is simple or functional) and to the Model Accessing Unit. The Model Accessing Unit finds all the fanout devices and their type and sends this information to the Configuration Processing Unit, which updates the configuration of the source and fanout devices and sends it to the Evaluation Processing Unit. This unit evaluates the incoming devices and generates an event if its output has changed. The Scheduling Unit adds the delay of the device to the current simulation time and sends it to the Event List Manager.

One important feature of this machine is that the number of devices it can evaluate per second is independent of the total number of devices in the network. The authors claim that, dependent on the logic network,

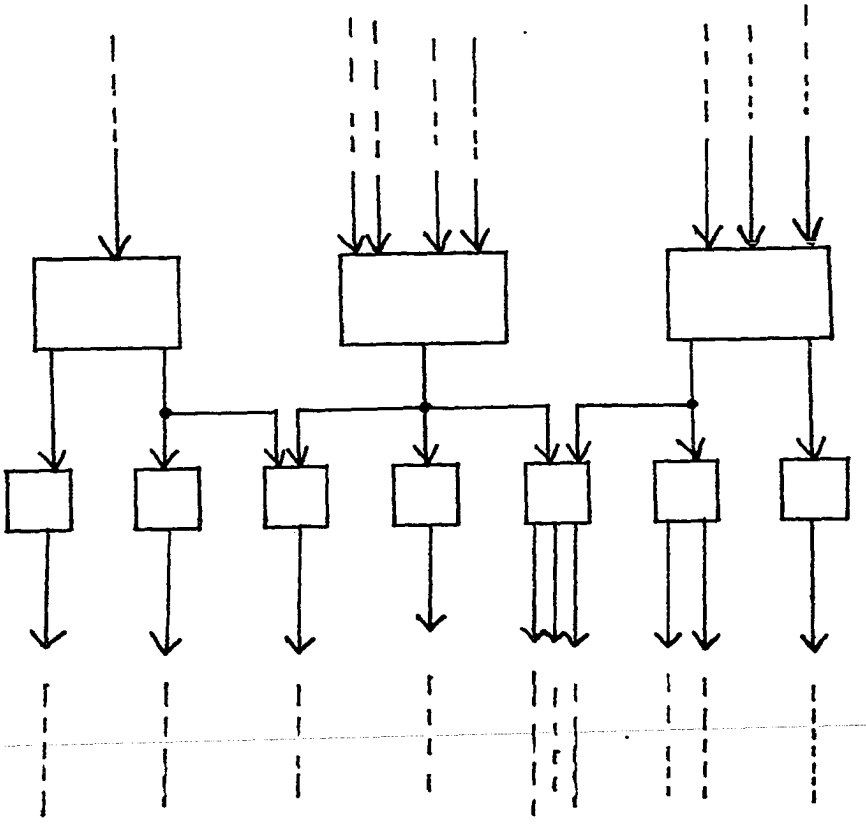
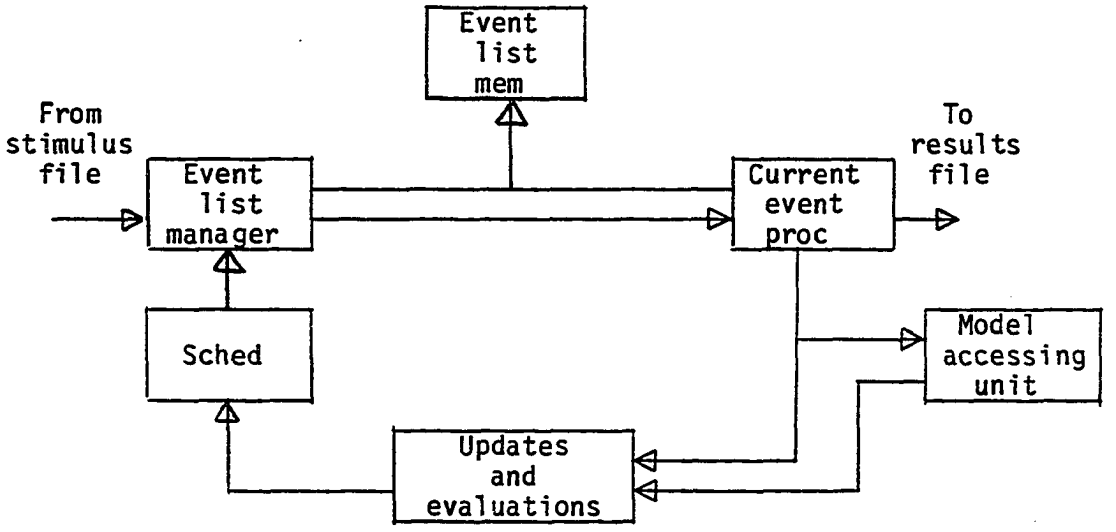
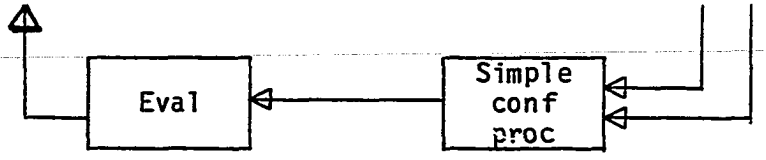


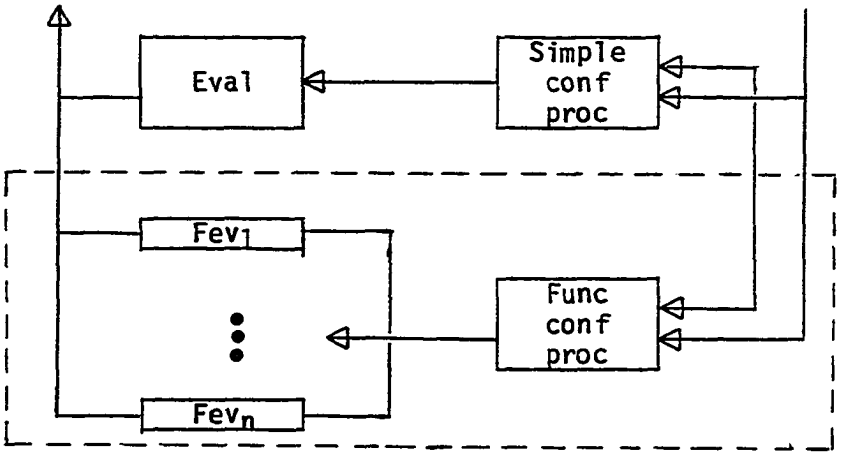
Figure 2. Typical operations during a time frame



a) Basic structure



b) Refinement for simple evaluations



c) Refinement for simple and functional evaluations

Figure 3. Ring pipeline architecture of a simulator

their machine can perform anywhere from 500,000 simple evaluations/second to one million simple evaluations/second. Typically, software simulators will perform less than 20,000 simple evaluations/second. A CDC Cyber 176 can perform 90,200 simple evaluations/second using vector processing (6). In any case, the specialized simulator runs about 10 to 60 times faster than the software based simulators.

A very different approach has been taken by IBM engineers in designing the Yorktown Simulation Engine (YSE) (2, 7, 8, 9). Instead of pipelining, the whole logical network is partitioned by the host computer into subnetworks, and each subnetwork is assigned to a logic processor, as shown in Figure 4. Because of the division in the network, signals from one subnetwork may be required by other subnetworks. This transfer of signals is done through a switch. A network should be partitioned in such a way so that interprocessor communication is minimized. Array

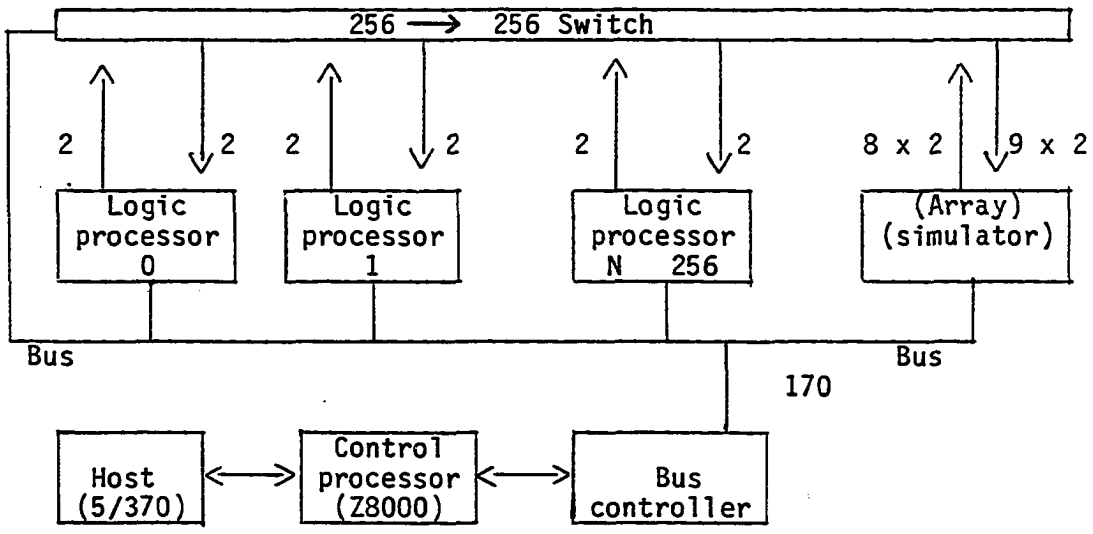


Figure 4. Yorktown Simulation Engine overview

processors are used to simulate RAMs and ROMs. A maximum of 256 processors can be connected, and communication between the host computer and the processor units is provided by an interface which consists of 170 twisted pairs of wires as a bus, a Bus Controller which is implemented with about 500 TTL, ECL, and MOS memory modules, and a standard Z8000 Control Processor. The Control Processor provides a general asynchronous interface capable of 10M bytes/second transfer rate. The overview of a logic processor is shown in Figure 5. The instruction memory stores the interconnections and function type information for a logical subnetwork of not more than 8K functions. The data memory holds the logic values for signals in the network, and the function unit evaluates logic functions. Each function (ex. AND, OR, etc.) has four inputs and one output, and each signal can have any one of the four logical values - 0, 1, undefined, tri-state high. Functions with more than four inputs can be computed by iteration. The logic processors are capable of three modes of simulation, unit delay, rank order, or a combination of the two, and are physically built from 600 TTL and 130 MOS memory modules. It should be noted here that the logic processor does not have the computational capability that one might expect a processor to have. Tables of functions are loaded into the memory modules of the functional unit, and all the processing is carried on by reading and writing from the various memory modules. By pipelining the instructions, a function can be evaluated in one instruction cycle of 80 ns.

In every instruction cycle, the switch transfers data from each of the 256 processors to any other processor. The switch consists of an

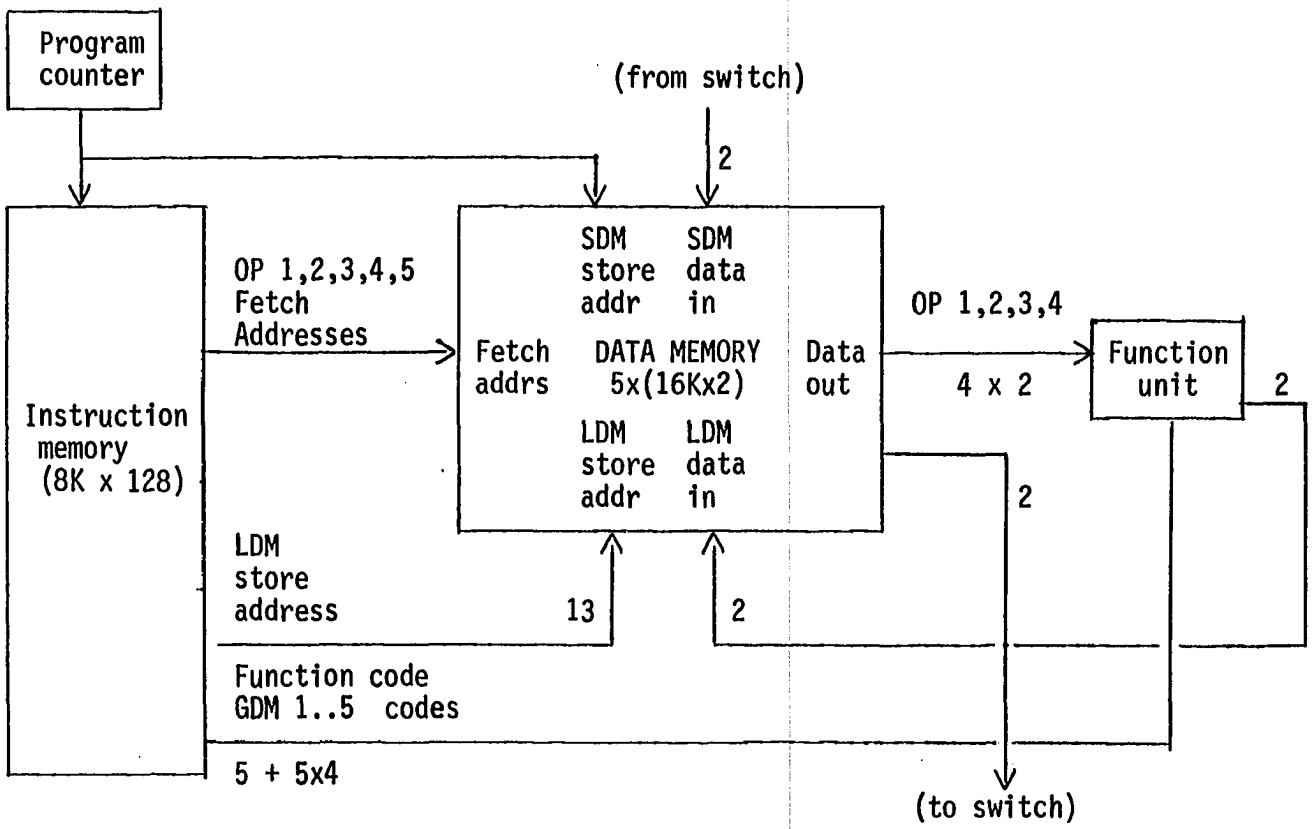


Figure 5. Logic processor overview

8K x (256 x 8) RAM which is loaded, before simulation, with all the information necessary for routing data during each instruction cycle with a maximum of 8K instruction cycles.

The YSE is capable of simulating a network of up to 2 million gates exceeding 3 billion gate evaluations per second. Apart from partitioning, the high speed has been achieved by designing a very specialized simulator for only gates with four inputs and one output, using unit delay and a four valued logic signal. In comparison, the simulator designed by Abramovici et al. supports logic simulation for both simple and functional devices, the architecture is transparent to logic values, and variable delays are used.

In the paper by Levendel et al. (10), microprocessors are used to design a logic simulator using distributed processing as shown in Figure 6. The performance of this simulator is better by over two orders of magnitude than the software based simulators. Unlike YSE, this system implements event-driven simulation with arbitrary delays for both simple and functional devices. For simple devices, the communication structure is cross-point based, and for functional devices it is a time-shared parallel bus. The authors claim that using both cross-point matrix and a parallel bus will prove cost effective. All communication between the host computer and the simulator is carried on through the master. During a simulation cycle, the slaves (simple or functional evaluators) transfer the scheduled events for the next time frame to other slaves and any primary output values and user-requested information to the master. Upon receiving a signal from the slaves that the current simulation cycle is

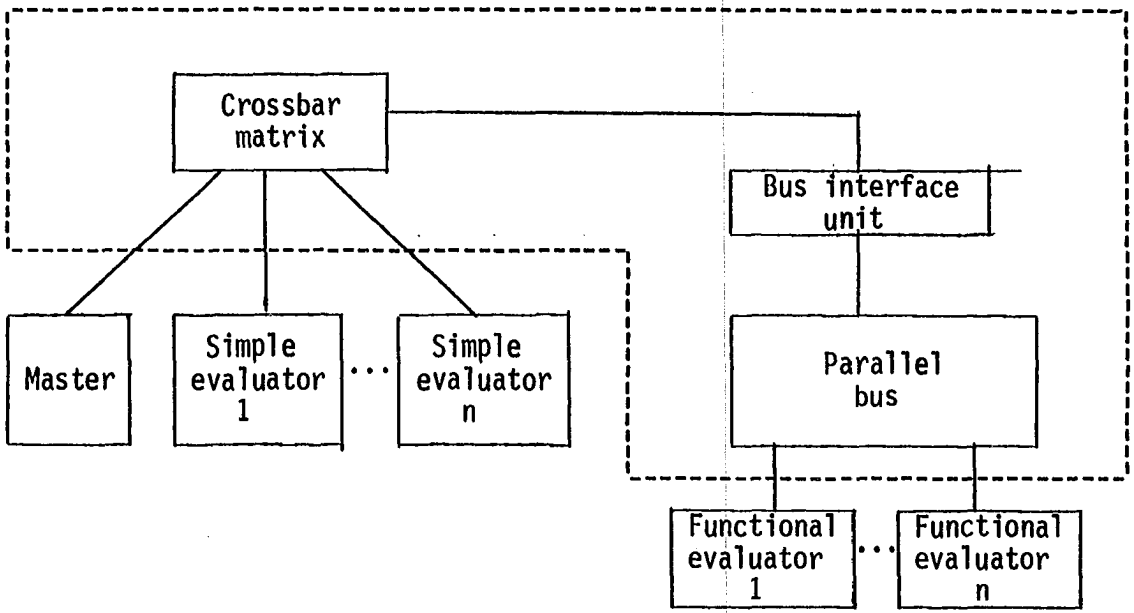


Figure 6. Architecture of proposed multiprocessor simulator

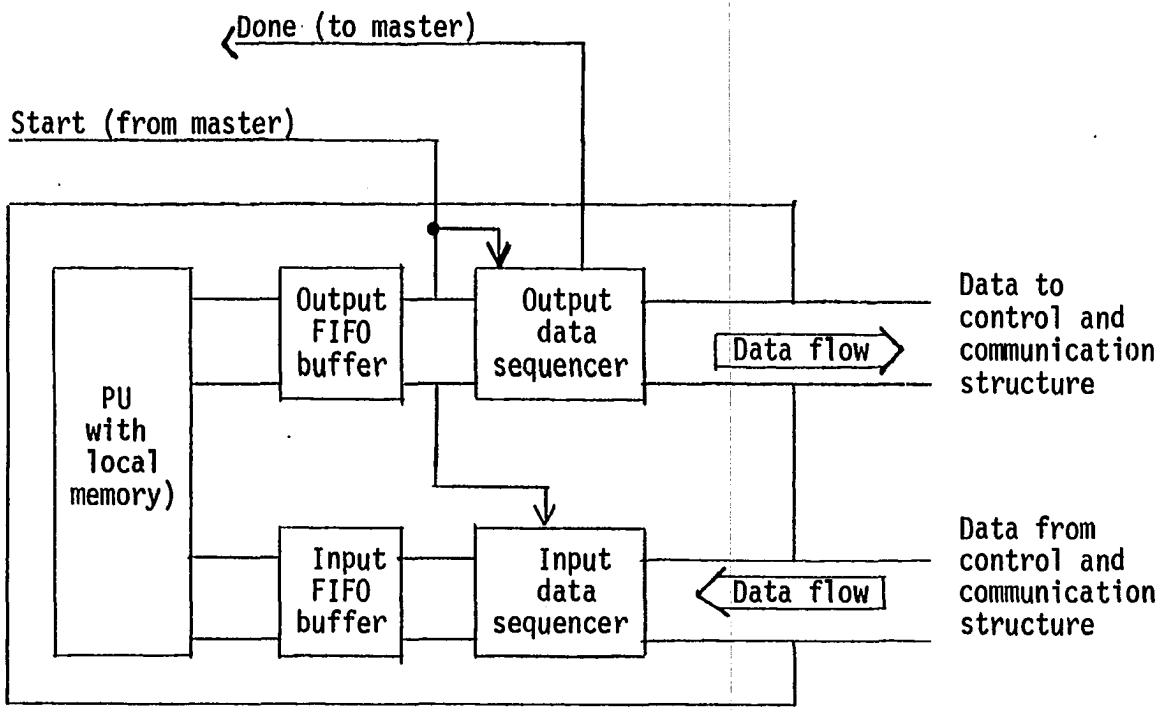


Figure 7. Slave unit configuration

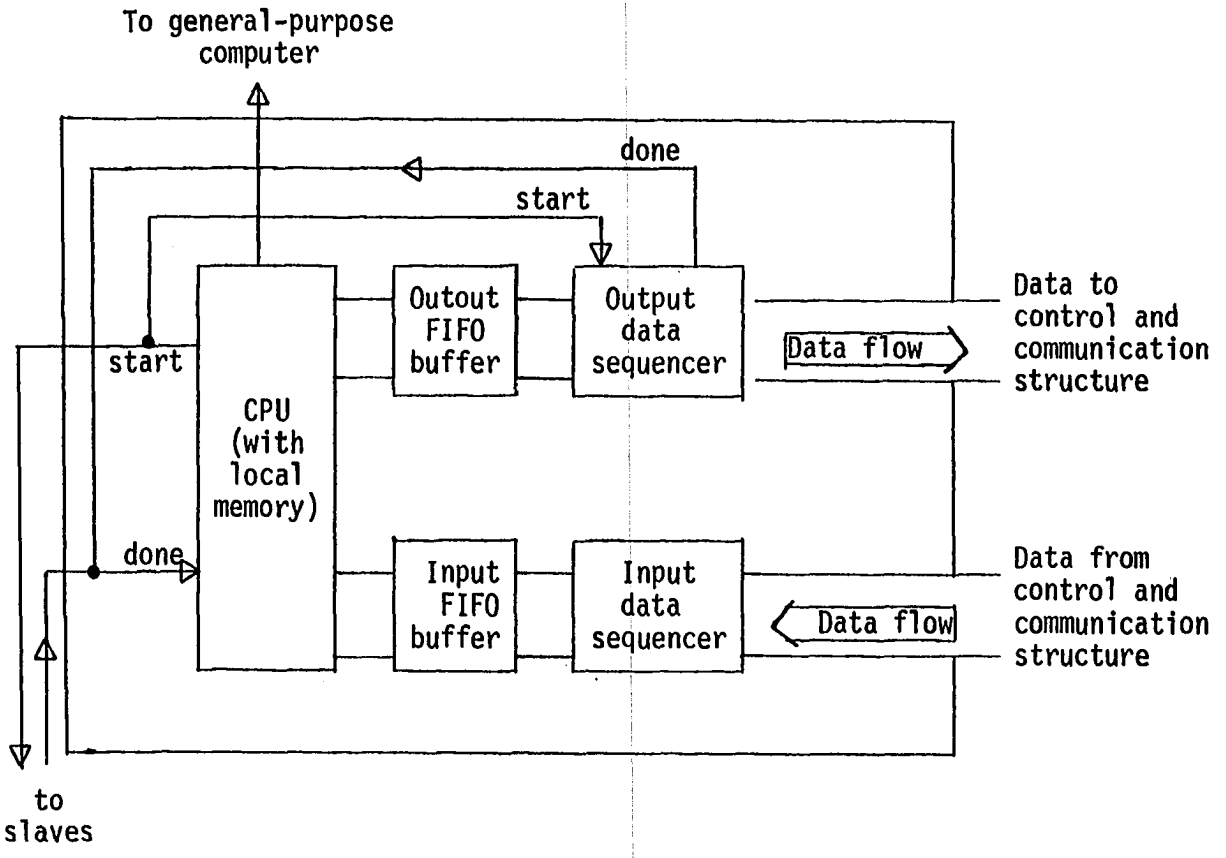


Figure 8. Master configuration

over, the master transfers any primary input values to the slaves for the next simulation time and then informs the slaves to start the next cycle.

The configuration of the slave unit and the master unit is shown in Figure 7 and Figure 8, respectively. The processor unit is a general purpose 16 bit microprocessor, and the input and output sequencers can be either specially designed or single chip microcomputers can be used. The slave units perform the device evaluation and scheduling. The simulation algorithms reside in the processing units' local memory. Both masters and slaves use Output Data Sequencer to send data and Input Data Sequencers to receive data via the communication structure. The processing unit and the data sequencers are isolated from each other by means of FIFO buffers.

Koike et al. (11) have designed a special purpose simulator for large scale computers, including CPU, main memory, cache memory, and microprogram memory. With 32 special processors running in parallel, it can execute up to 6 million IC evaluations per second (1.2 billion gate evaluations per second), can handle up to 15,000 random logic ICs (about 3 million gates), and 2000 memory ICs (up to 2 Mega bytes). Other hardware simulators have been designed by Sasaki et al. (12) and Barto et al. (13, 14).

OVERVIEW

In the following chapter, the proposed architecture of the Processing Unit (PU) is discussed. This architecture can be compared with the earlier work in this area that has been described in the previous chapter. Abramovici et al. (5) used distributed processing by assigning different tasks of the simulation algorithm to six dedicated units. Each unit has a microprogrammed processor and local memory, and devices are simulated in a pipelined fashion. Also, functional devices are evaluated in parallel by several processors in a unit. The idea of simulating devices in a pipelined fashion has been retained in the proposed architecture, but three dedicated units are used to simulate devices in a microprogrammed environment. Each unit takes the same amount of time to perform its tasks on a simple device, and longer time is required by each unit if functional devices are simulated. Both simple and functional devices are evaluated by a single processor. Denneau (7) used a different approach in which a logic network is partitioned into subnetworks, and each subnetwork is simulated in parallel by different but identical processors. Interprocessor communication is performed through a switch. Levendel et al. (10) proposed a cross-point based communication structure for simple devices and time-shared parallel bus for functional devices. Other methods for interprocessor communication have also been proposed (11, 12). In the proposed architecture, the concept of a "thick" capture bus has been introduced for interprocessor communication of data for both simple and functional devices. Also, the communication of data is done in parallel with the simulation of devices. A method has been developed

through which the user can program each device in the network for intermediate simulation results. These results are sent to the host computer, via a standard bus, in parallel with the simulation of devices. The use of a time wheel to schedule devices has been proposed (10). This idea has been extended in this dissertation in that memory for the time wheel is allocated dynamically. Since hardware costs are rapidly diminishing, this architecture utilizes inexpensive VLSI and LSI chips. Because the enormous amount of hardware required to implement the architecture implies the need for extensive diagnostics, a dedicated unit is used to perform the diagnostics function on the other units. The methods developed in this dissertation provide parallelism in hardware, high speed, and efficient utilization of system resources.

Following the description of the architecture, functions of the specific parts in the PU are described in detail to demonstrate the power of the proposed architecture. Devices are divided into categories to utilize the resources of the PU efficiently. The kind of information required and the information moved within the PU for each category of devices are described in detail. An estimate of the amount of memory required and the speed of the PU is made. Methods have been developed to handle the data in a natural manner at a high speed. Memory is allocated either statically or dynamically, depending on the situation at hand.

Finally, in Appendix A, the design of the Evaluation Unit is described. Timing diagrams are used frequently to explain the timings involved in the circuit. The unit is designed to operate within the range of commercial temperatures.

DESCRIPTION OF THE PROCESSING UNIT

The Processing Unit (PU) is capable of simulating 16.77 million simple devices at a maximum speed of 128 million simple evaluations per second. A network to be simulated can have any combination of simple and functional devices. A digital device consists of inputs, outputs, and delays between the inputs and the outputs. The outputs are a function of inputs only. This kind of a model excludes the possibility of internal state variables, though these could be included as an extension of the model. A simple device is defined as consisting of a maximum of four inputs and one output, three signal values (possibly 0, 1, and unknown), and ten internal fanouts with no external fanout or eight internal fanouts with ten external fanouts. Devices that do not fit this category are defined as functional devices. The model of a device consists of its configuration (input signal values, output signal values, and delays) and connections to fanout (both internal and external) devices.

As shown in Figure 9, the Host computer communicates with the PU via a high speed standard bus. During initialization, the Host programs the PU depending on the characteristics of the network of devices to be simulated. The key to high speed simulation lies in exploiting the characteristics of the network in order to utilize the resources of the PU most efficiently. For example, the PU should be programmed for the number of signal values to be used (3, 7, or 15 signal values). The host computer is also responsible for pre- and post-processing of the simulation data.

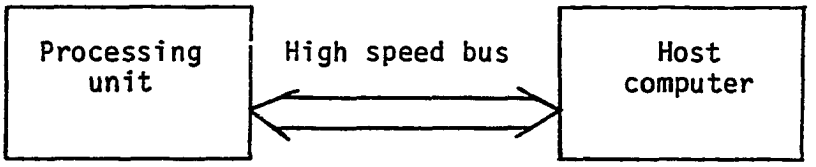


Figure 9. Host computer - processing unit interface

It loads the memories of the PU with simulation data. After the simulation is over, it receives the results of the simulation from the PU. The simulation is done entirely by the PU. The evaluation programs of all widely used devices are built into the PU. However, there are provisions for loading programs of new devices from the Host. The PU is also capable of performing diagnostics on its hardware upon receiving a request from the Host. The results of the diagnostics are sent to the host. This feature increases the reliability of the PU tremendously. Diagnostics should be performed either at regular intervals or when the PU is not being used.

A network of devices consists of primary inputs and primary outputs. The outputs of the network are defined for each set of primary inputs. If the inputs are changed, the outputs may also change. In fact, for each combination of unique inputs, there are unique outputs. The next outputs are a function of the previous outputs (or the state of the network) and the new inputs. If there are no feedbacks in the network, then the outputs are a function of inputs only.

In order to simulate a network, a set of simulation vectors (primary inputs), along with their predetermined results, is prepared. The

sequence in which these vectors is applied is very important. The network must first be brought into a known state before any effective simulation can be carried on.

The model of the network to be simulated is loaded into the PU. The Host sends the simulation vector to the PU. The PU simulates the network for the given vector and sends the simulation result (primary outputs) to the Host which then compares it with the predetermined result. This operation is carried on until an error is detected or the network has been simulated for all the sequence of vectors. If an error is detected, a more thorough check of the network can be carried on by specifying the problem devices. The state of these devices will then be sent to the Host for each time frame.

The amount of traffic on the bus will depend on the amount of simulation data that have been requested by the user. If only the simulation results are requested, then the bus will be used by the PU only after the simulation has completed.

In order to reduce the interaction between the Host computer and the PU, all the simulation vectors and predetermined results can be stored in the PU. The PU can then simulate the network for each vector and notify the Host only if the simulation result and the predetermined result disagree. This method, though faster, requires extra memory in the PU to store the simulation vectors and predetermined results. Either method can be used, and in this dissertation the first method discussed will be used.

As shown in Figure 10, the PU is divided into 256 sub PUs. The sub PUs are exact copies of each other. The number of sub PUs could vary depending on the needs of the installation. Each sub PU can simulate a maximum of 64K simple devices. The network to be simulated is partitioned into 256 subnetworks, and each subnetwork is assigned to a sub PU. The partitioning must be done along the length of the network, starting from primary inputs and ending at the primary outputs. All sub PUs simulate devices in their subnetwork for the same time frame. The external data for the current time frame are transferred between sub PUs via the capture bus. The current time frame is advanced to the next time frame after all the sub PUs are done simulating the active devices for the current time frame.

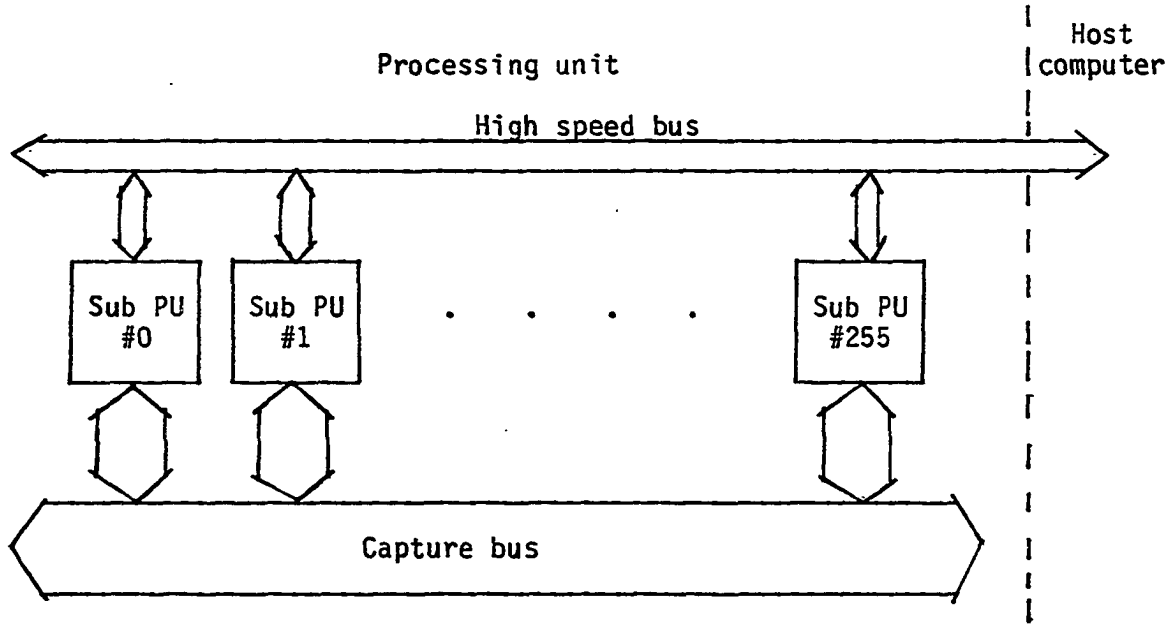


Figure 10. Architecture of the processing unit

The capture bus is a nonstandard, very high speed, "thick" bus consisting of 256 lines for data, 32 lines for check bits, and some lines to synchronize the transfer of data between the sub PUs. The 256 data lines are just by coincidence equal to the 256 sub PUs. The standard bus is used entirely for communication between the Host and the sub PUs. Communication between sub PUs is not done on this bus but on the capture bus. In order to send the data to the Host, the sub PU must first acquire the bus. The Host can send data to one or more sub PUs at a time.

Each sub PU is divided into five units, as shown in Figure 11. The dedicated units work concurrently on separate tasks of the algorithm and are designed with standard, commercially available chips in a microprogrammed environment. Each unit has its own memory and processors. Any sharing of information between units is done through the high speed FIFO buffers. The distributed architecture resembles that of a data flow machine where the unit is activated upon the arrival of data at its inputs. Since there is no direct communication between units, the throughput of the sub PU is greatly enhanced.

Typical operations during a time frame consist of the following: 1) retrieve current source events, 2) update output signal values of source events, 3) determine fanout devices, 4) update input signal values of fanout devices (events are generated), 5) evaluate fanout events, and 6) schedule events.

The Scheduling Unit (SCHED) contains relevant information about all the source devices whose fanout devices will be evaluated in the future. A source device can either have only internal fanout devices or both

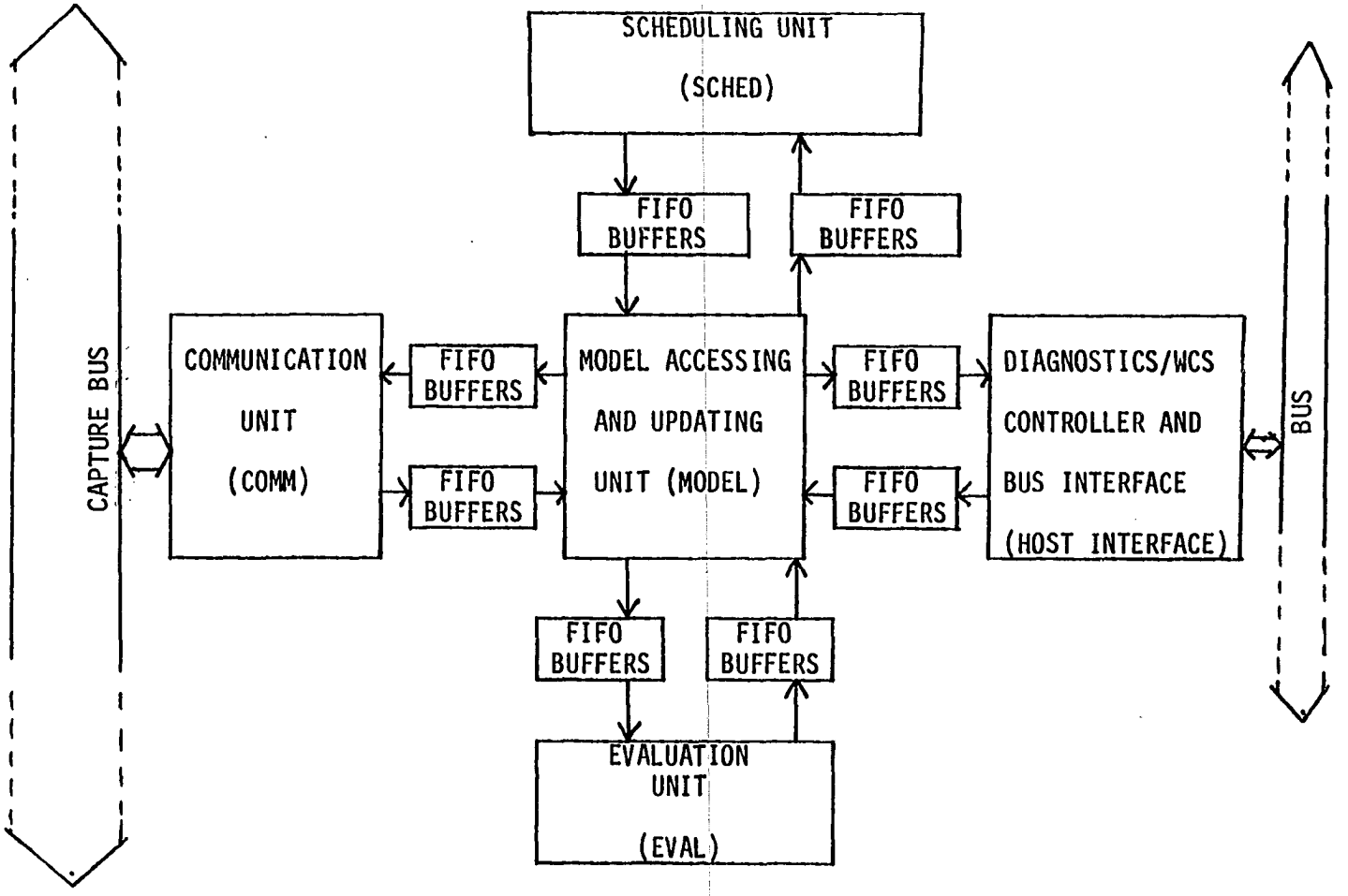


Figure 11. Architecture of the subprocessing unit

internal and external fanout devices or no fanout device. During a time frame, SCHED first sends to the Model Accessing and Updating Unit (MODEL) the information of active source devices with external fanouts followed by other active source devices. MODEL has the model (configuration and connections to fanout devices) of all the devices in the subnetwork. Upon receiving the information of a source device with external fanouts, MODEL first updates the output signal values. Any user requested information about the source device is sent to the HOST INTERFACE. Then the fanout devices are determined. The information of external fanout devices is send to the Communication Unit (COMM). The input signal values of the internal fanout devices are updated, and the information of each internal fanout device (event) is sent to the Evaluation Unit (EVAL). COMM stores the information it receives from MODEL. HOST INTERFACE sends to the Host Computer the information it receives from MODEL. EVAL evaluates the event, compares the new output with the previous output, and discards the event if the outputs are the same. If the outputs are not the same, the event is sent to MODEL. Upon receiving the event, MODEL retrieves relevant information from the model of the device and sends the information to SCHED which then schedules the event (source device). All source devices with external fanouts originate from SCHED and follow the same path as has been described above. Figure 12 shows the flow of data between units. When MODEL has received from SCHED all the source devices with external fanouts, it signals COMM to start communication of external fanout data via the capture bus. When COMMs of

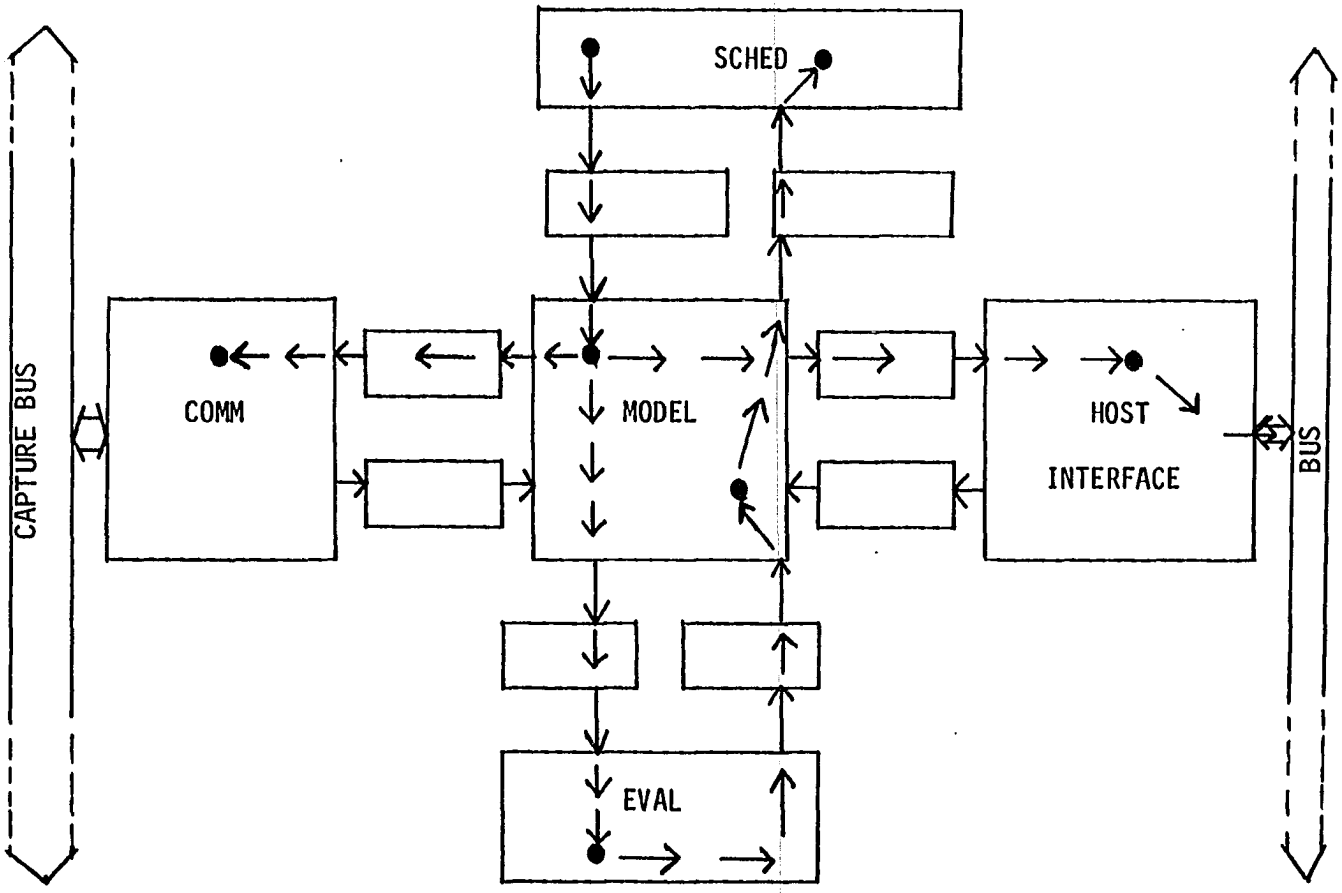


Figure 12. Flow of data for a source device with external fanouts

all the sub PUs are ready, each COMM sends the information of source devices with external fanouts, for the current time frame, on the capture bus and receives information of external source devices whose fanouts are in its subnetwork. From the information of external source devices, fanout devices are determined, and their information is send to MODEL.

While COMM is busy with the transfer of external fanout information, MODEL is receiving from SCHED information of the remaining source devices for the current time frame. These source devices follow the same path described for the source devices with external fanouts, except that there are no external fanouts and no information is sent to COMM. Figure 13 shows the flow of data between units for this case.

When all the source devices from SCHED have been simulated, MODEL starts receiving information of fanout devices from COMM. The input signal values of the device are updated, and the information of the device is sent to EVAL. EVAL evaluates the event and sends it to MODEL if the output changes. MODEL retrieves the pertinent information for the event and sends it to SCHED which then schedules the event. Figure 14 shows the flow of data between the units for this case.

When all the devices from COMM have been simulated, the time frame is advanced, and the simulation of devices for the next time frame begins.

Even though it may not be apparent from the architecture of the sub PU, the devices are simulated in a pipelined fashion. Each of the five units are specialized to perform certain operations. For example, in Figure 13, while SCHED is performing operations on event #5, MODEL is performing operations on event #4 and EVAL is evaluating event #3. Event

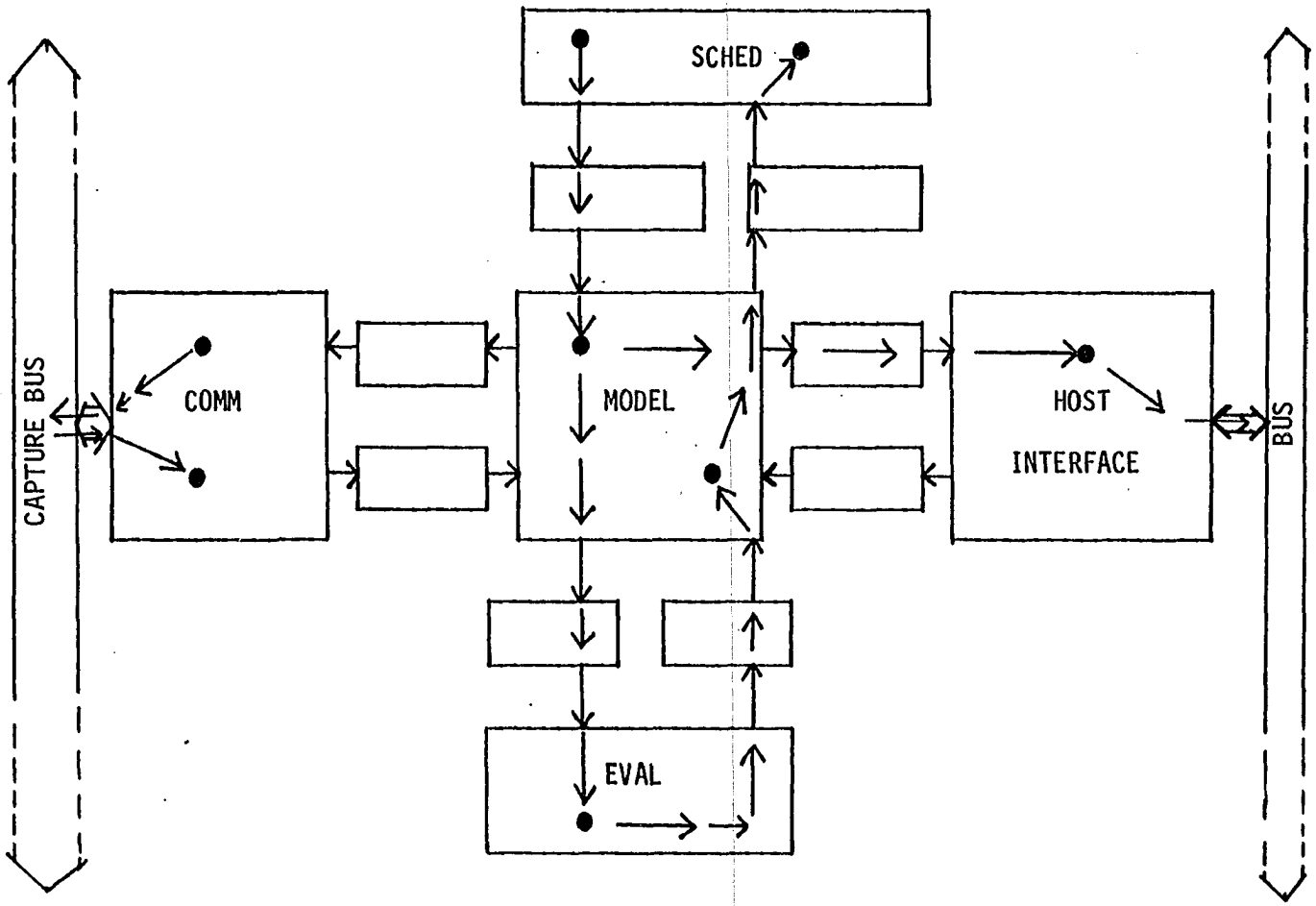


Figure 13. Flow of data for a source device with internal fanouts only

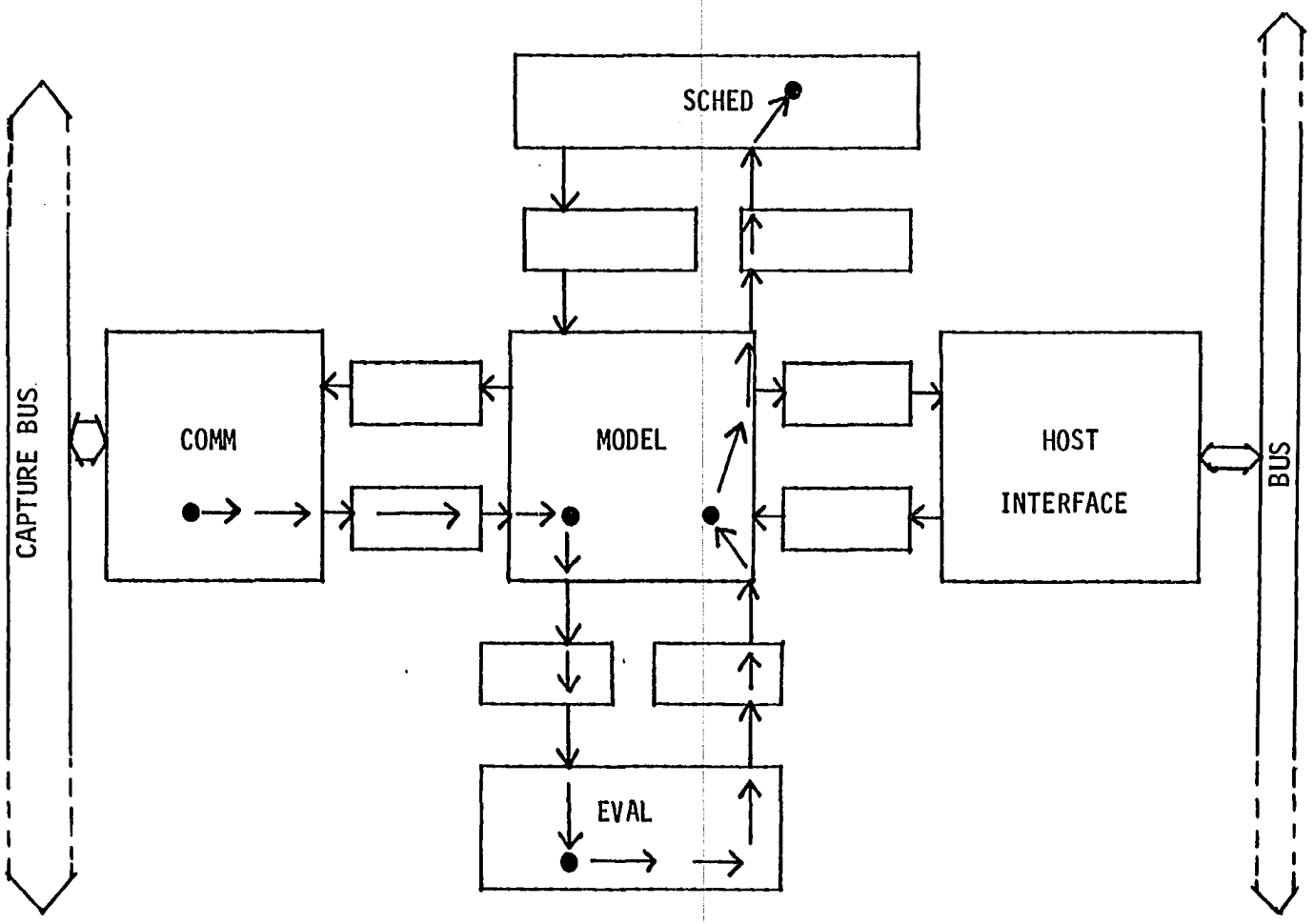


Figure 14. Flow of data for a fanout device which has an external source

#2 has already been evaluated and is waiting on the FIFO buffers for MODEL. Event #1 is waiting on the FIFO buffers to be scheduled by SCHED.

Due to the nature of pipelining, multiple input events are caused. Two or more source devices may activate the inputs of the same fanout device. This will cause several events, of the same fanout device, to be generated. Only the last event generated is correct, and all other events must be cancelled. Besides scheduling events and retrieving events of the current time frame, it is the responsibility of SCHED to detect multiple input events, cancel all previous events of the same fan-out device, and schedule the latest event.

The throughput of the sub PU will be degraded if multiple input events occur often. In the case of simple devices, the activation of multiple inputs in a time frame is rare. However, this may not be true for functional devices, all of whose inputs must first be updated before the device is evaluated by EVAL.

Each unit is specialized to perform certain operations on a device. EVAL will evaluate a simple device within 1.5 us to 3 us. On an average, it will take 2 us to evaluate a simple device. Other units must take the same time to perform all of their operations. Longer time is required to evaluate functional devices. All the units will take a proportionally longer time, as compared to simple devices, to perform their operations. In this way, none of the units will be idle while the simulation is going on. Also, the traffic on the FIFO buffers will not be blocked. The communication of external fanout data on the capture bus is done, by COMM, in parallel with the simulation of internal fanout devices by the rest of

the units. The communication on the capture bus must be done at a very high speed in order to ensure that the rest of the units will not be waiting for data from COMM. The amount of traffic in the HOST INTERFACE, during simulation, will depend on how much data have been requested by the user. MODEL is the central unit of the sub PU. Since this unit has the model of all the devices in the subnetwork, it is here from where the information of each device is sent to other units. This unit should have the maximum traffic because all other units send/receive data to/from this unit.

Since the units are highly specialized, their architecture is tailored to execute the required operations very efficiently at a high speed in a microprogrammed environment. Any future changes can be accommodated by simply changing the microcode, without any alteration in the hardware of the unit. General purpose microprocessors could be used instead of microprogrammable processors to reduce the overhead in microprogramming. It is expected, however, that the amount of microprogramming required would not be enormous. Besides, more than one microprocessor will be required to achieve the same computing power as a microprogrammable processor. In the proposed architecture, the benefits received by using microprogrammable chips outweigh the extra time required to microprogram. A microprogrammable microprocessor (AM 29116), a microprogram controller (AM 2910-1), and other microprogrammable chips should be used to design the architecture of each unit. Two TDC 1030s can be cascaded to design the 64 x 16 bit high speed FIFO buffers. (For data sheets of these chips, see Appendix B.)

The devices are simulated in a pipelined fashion, and a constant time is required by each unit to perform operations on a device. Therefore, as the number of devices in the sub PU increases, the simulation time increases only linearly. In order to estimate the maximum speed of the PU, assume that a simple device can be evaluated in 2 us. Ignoring the overhead in filling the pipeline at the beginning of each time frame, and assuming that all the 256 sub PUs are busy throughout the simulation, a maximum of 128 million simple devices can be evaluated in a second.

During simulation of the network, activity is centered around a certain portion of the network. A look-ahead mechanism could be used to transfer data of the active portion from the main to cache memory. This kind of arrangement could save memory costs at the expense of extra hardware for look-ahead and memory management.

The cost of semiconductor memory is decreasing typically at the rate of 20% or 30% per year (15). Nowadays, high density memory chips, 64K and 256K bits, are available with access times in the range of 55 ns to 300 ns at a relatively low cost (16, 17). In an architecture where high speed is the primary goal, it is cost effective to interface high density, high speed memories directly to the processor in a one level memory system. The purpose of having high speed microprogrammable processors is defeated if the memory cannot keep up with processor requests. Any disparity in the speed of the processor and the memory could be reduced by having memory systems with wide words (128 bits and 256 bits). Also, the access time could be overlapped with processor operations by reading the next word (or writing into memory) while the processor is operating on

the current word. This kind of processor-memory arrangement should be used to design the units of the sub PU.

Large memory systems requiring many memory chips may have inadequate reliability. Therefore, EDC is required to improve the reliability of large memory systems. Am 2960s (see Appendix B for data sheets) can be used to detect and correct errors which may occur in the memory or on the Capture Bus. Am 2960s can correct all single-bit errors and detect all double and some triple-bit errors. In fact, double-bit memory errors can be corrected if one of the two-bit errors is a hard error. A maximum of four 2960s can be cascaded along with MSI chips to handle 64 bits of data and 8 check bits. To obtain maximum performance, the EDC device should be used in the check-only configuration, as shown in Figure 15.

On a read from memory, data is read onto the data bus. At the same time, the data and the check bits are read into the EDC to check for errors. If one or more errors are detected, ERROR is active; if exactly two errors are detected, then DOUBLE ERROR is active; and for three error cases and some double error combinations, MULTERROR is active. Also, if only ERROR is active, then only one error has been detected. These error flags could be used to interrupt the processor when an error occurs. The syndrome bits may then be decoded by the processor. Since the EDC device is operating in the correct mode, the processor, upon interrupt, can read the corrected data from the output latch if only a single error has been detected and the error is not in the check bits. If error is in the

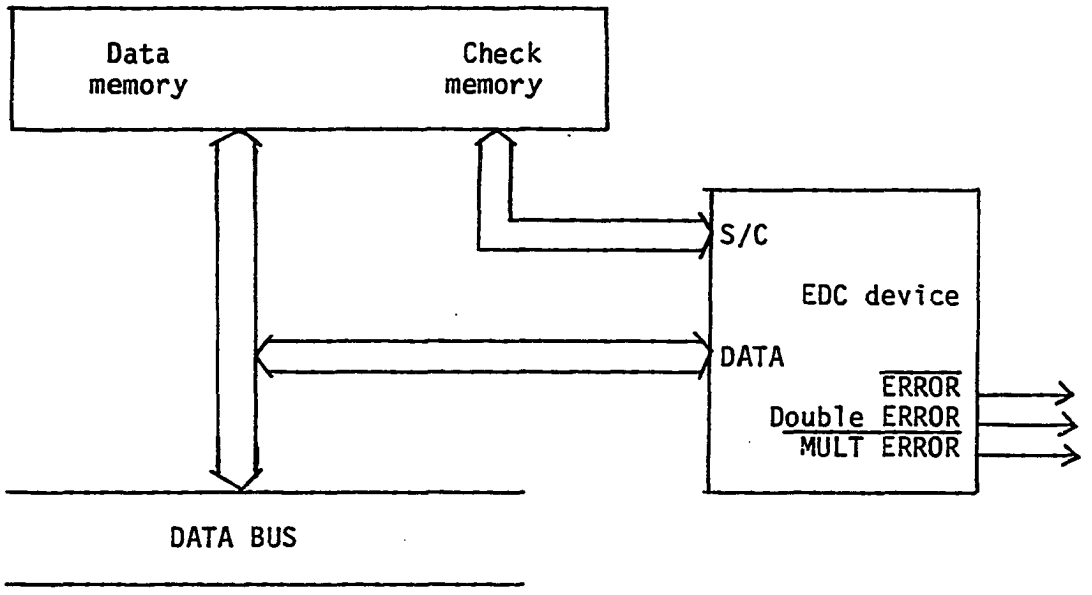


Figure 15. EDC device in check-only configuration

check bits, then the data word read from memory does not have errors but the check bits do. The error should be corrected. In this way, the EDC is made transparent to the processor, and the time to read from memory is equal to the access time of the memory. A slow down of the system occurs only when an error is detected, and this happens infrequently.

On a write to memory, the data word could be buffered by using the Am 2961/62 Data Bus Buffers (18) while the check bits are being generated. Thus, the time required to generate the check bits is made transparent to the processor.

Besides sending/receiving data to/from the Host computer and the MODEL, the HOST INTERFACE can also perform diagnostics on the hardware of

the other four units of the sub PU, and can also write new evaluation programs in the WCS of EVAL. The reliability of the PU is increased tremendously by providing diagnostics at the sub PU level, and EDC with the memory systems and the capture bus. Some form of EDC should also be provided for the standard bus.

In summary, parallelism in a logic network is exploited by partitioning the network into subnetworks and assigning each subnetwork to a sub PU. Within a sub PU, pipelining is used to simulate devices. The resources of the PU are used efficiently by programming the PU depending on the characteristics of the network. Diagnostics and EDC for memories and buses is provided to increase the reliability of the hardware. Finally, the hardware is designed in a microprogrammed environment to achieve high speed and flexibility.

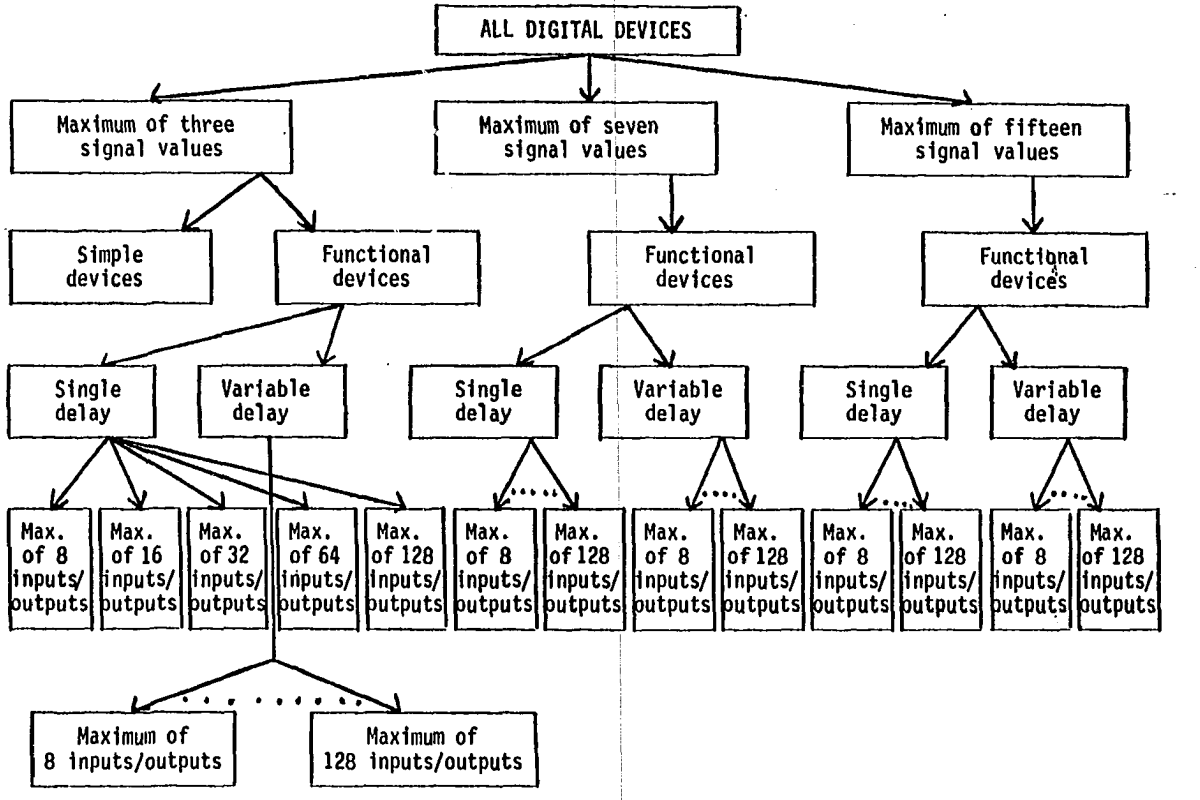
DESCRIPTION OF UNITS IN THE SUBPROCESSING UNIT

In this chapter, each of the five units in the subprocessing unit will be discussed. Most of the discussion will be centered around simple devices.

Model Accessing and Updating Unit

A digital device consists of inputs, outputs and delays. The delay is the time it takes for the signal to propagate from the inputs to the outputs, and it may or may not be a function of inputs. The output is always a function of inputs. Table 1 shows how digital devices can be grouped into various categories. Before simulation starts, the Host must program the PU for the maximum number of signal values that will be used. If more than three signal values are used, all devices are functional devices. However, if three signal values are used, then the devices are divided into simple and functional devices. Functional devices are subdivided into single delay and variable delay devices. For single delay devices, the propagation delay between inputs and outputs is the same, irrespective of which outputs are active. In the case of variable delay devices, the delay at each output is a function of either a combination of inputs and the signal values at those inputs or a function of previous output signal values. Each of the two categories in functional devices is further divided into six categories, depending on the number of inputs and outputs. The category with a maximum of eight inputs/outputs will consist of all functional devices which have less than eight inputs and

Table 1. Digital devices grouped into categories



eight outputs. A functional device with 31 inputs and 10 outputs will fit in the 32 inputs/outputs category. This division is mainly made to save space in memories of all the units.

According to Table 1, digital devices are divided into 31 categories with the simplest of all being the simple device and the most complex being the functional device with variable delays, 128 inputs, and 128 outputs. The division in this manner is not exhaustive and can be extended to include other cases not covered here. The inputs and outputs of a device are counted from left to right, as shown in Figure 16.

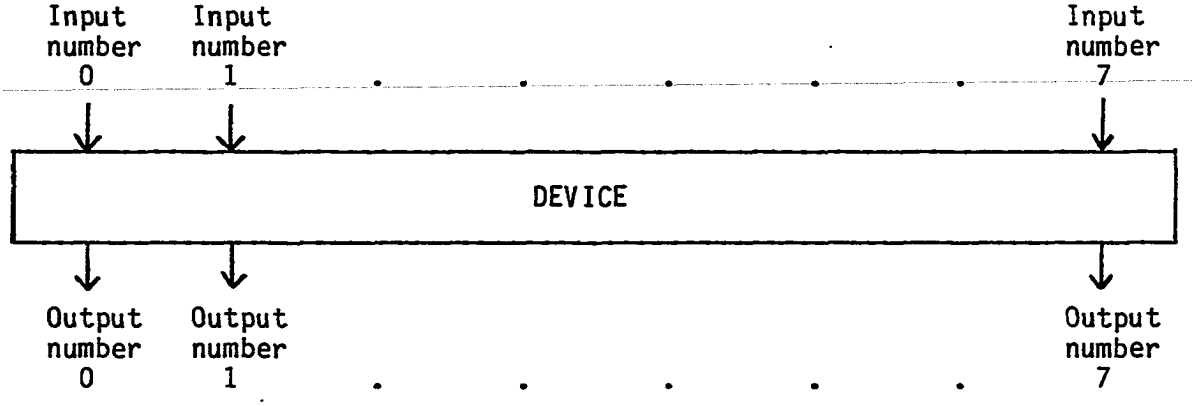


Figure 16. Numbering of inputs and outputs of a device

The MODEL memory is a 64K x 144 bits memory, with 128 bits of data and 16 check bits in a word. It contains the configuration and interconnections of all devices in the subnetwork. A simple device occupies one word of MODEL memory. The address of the location which the simple device occupies is also its identification number (ID). Since there are

Table 2. Model of a simple device

Command for the device	= 4 bits
Type of device	= 12 bits
Delay	= 8 bits
Signal values of 4 inputs and 1 output	= 10 bits
Presence of an external fanout	= 1 bit
Address of the external fanout	= 13 bits
Number of internal fanouts	= 4 bits
Identification number of the first internal fanout	= 16 bits
ID of the second internal fanout	= 5 bits
.	
.	
.	
ID of the last internal fanout	= 5 bits
Input number of the first internal fanout	= 2 bits
.	
.	
.	
.	
.	
Input number of the last internal fanout	= 2 bits

64K words, models of (64K-1) simple devices can be stored. The last location is used for other purposes and will be explained later. The configuration and interconnection data for simple devices are shown in Table 2. For a device to qualify as a simple device, it must have no more than three signal values, a maximum of four inputs and one output, a maximum of ten internal fanouts with no external fanout, or eight internal fanouts with a maximum of 10 external fanouts and must occupy only one word of MODEL memory. The simple device category will include almost all TTL gates plus other gates and devices; any other device could be modeled as a functional device with no restrictions whatsoever.

Both Table 3 and Table 4 show the model of a three signal value functional device which has no more than 32 inputs and 32 outputs. Table 3 includes a single delay device, whereas Table 4 includes a variable delay device. As is apparent from these tables, the model of a functional device will usually occupy more than one word of the MODEL memory. The identification number of every device is the address of the first word in the memory. The models of all devices occupy consecutive locations in the MODEL memory.

Each of the fields in the MODEL memory that constitute the configuration and interconnection of the device will be described as follows:

1. Command for the device: Through this field, the user can exercise full control over each device of the network. These four bits can be used to select one of the 16 different combinations. An example is shown in Table 5. This field is useful in locating faults in the network. Initially, simulation can be carried on so that only the primary

Table 3. Model of a functional device with 3 signal values, single delay, and 32 inputs - 32 outputs

Command for the device	= 4 bits
Type of device	= 12 bits
Delay	= 8 bits
Number of inputs	= 5 bits
Number of outputs	= 5 bits
Signal value for the first input	= 2 bits
.	
.	
.	
.	
Signal value of the last input	= 2 bits
Signal value of the first output	= 2 bits
.	
.	
.	
Signal value of the last output	= 2 bits

For each output, the following information should be provided:

Presence of external fanout	= 1 bit
Address of the external fanout	= 13 bits
Number of internal fanouts	= 5 bits
Identification number of the first internal fanout	= 16 bits
ID of the second internal fanout	= 8 bits
.	
.	
.	
ID of the last internal fanout	= 8 bits
Input number of the first internal fanout	= 5 bits
.	
.	
.	
Input number of the last internal fanout	= 5 bits

Table 4. Model of a functional device with 3 signal values, variable delay, and 32 inputs and 32 outputs

Command for the device	= 4 bits
Type of device	= 12 bits
Number of inputs	= 5 bits
Number of outputs	= 5 bits
Signal value of the first input	= 2 bits
.	
.	
.	
Signal value of the last input	= 2 bits
Signal value of the first output	= 2 bits
.	
.	
.	
Signal value of the last output	= 2 bits

For each output, the following information should be provided:

Presence of external fanout	= 1 bit
Address of the external fanout	= 13 bits
Number of internal fanouts	= 5 bits
Identification number of the first internal fanout	= 16 bits
ID of the second internal fanout	= 8 bits
.	
.	
.	
ID of the last internal fanout	= 8 bits
Input number of the first internal fanout	= 5 bits
.	
.	
.	
Input number of the last internal fanout	= 5 bits
Number of delays	= ? bits
Delay as a function of either inputs or outputs	= 1 bit

If the delay is a function of outputs, then the following information is provided for each delay:

Delay	= 8 bits
Previous output signal values	= ?

Table 4. Continued

If the delay is a function of inputs, the following information is provided for each delay:

Delay	= 8 bits
Input numbers	= ?
Signal values	= ?

Table 5. Different selections from 'command of the device' field in model memory

0000	====	send output to Host whenever it changes
0001	====	send output to Host when simulation is over
0010	====	STOP simulation if any output has a logic value of 1
0011	====	data should not be sent to the Host
.		
.		
.		
.		

outputs are received by the Host for some combination of primary inputs. If the output differs from the expected output, then trouble shooting can be done through a more comprehensive simulation, by programming the 'command for the device' field in such a manner so that the problem devices are monitored, whenever active, during every time frame.

2. Type of device: This 12 bit field is shared by MODEL, SCHED, and EVAL. In the case of EVAL, the 12 bit field is used as a microaddress to directly map to the microroutine of a device in the 4K control store. Before the simulation starts, the Host programs the units for the maximum signal values to be used. During simulation, there are only ten or eleven different devices to be identified (see Table 1) by the MODEL processor. These devices can be identified from a portion of this field. Through this field, the MODEL processor jumps to the microroutine of the device stored in its control store and accesses the data from the MODEL of the device.

3. Delay: This field of eight bits will cover delays in the range of 0 to 256 units of time. Typical delays of ECL gates is 2 ns, and MOS gates is 100 ns. Between these two extremes, the delays of other logic families lie.

In the case of variable delay functional devices, more than one field is required to model delay. Each output can have more than one delay, and every delay can be a function of either inputs and the signal values of the output. In the case of simple and single delay functional devices, only one 8 bit field is required.

4. Inputs: This field consists of the signal values at the inputs of the device. A simple device has four inputs. A functional device has an extra field which contains the number of inputs that the device has.

5. Outputs: This field consists of the signal values at the outputs of the device. Simple devices have only one output, whereas functional devices can have any number of outputs. An extra field, for functional devices, contains the number of outputs that the device has.

6. Fanouts: For the two extremes, an RTL gate can drive, in the worst case, five gates, whereas a CMOS gate can drive more than 50 gates. Between these two extremes, an ECL gate can drive up to 25 gates, a TTL can drive up to 10 gates, and a MOS can drive up to 20 gates.

A simple device has one output which can drive a maximum of either ten internal fanouts and no external fanout or eight internal fanouts and ten external fanouts. A 1 bit field specifies whether the source device has any external fanouts. If it does not have external fanouts, then the 13 bit external fanout field is absent; otherwise this field gives the address of the fanout in the COMM. A 4 bit field specifies the number of internal fanouts that the source device has. The 16 bit field contains the identification number of the first internal fanout device. The identification numbers of the other internal fanouts must be within 32 IDs of the first fanout, and are specified by 5 bit fields. Here, we are excluding such rare possibilities where a simple device has some functional devices as fanouts which occupy so many locations of MODEL memory that it is impossible to assign IDs in the range of 32 for all fanouts. The ID of the internal fanouts could be specified by 16 bits, but this will

waste memory. Another field contains the input number of the internal fanout device to which the output of the source device is connected. One such field of 2 bits is required for each simple internal fanout.

In the case of functional devices, each output consists of all the fanout fields that have been mentioned for a simple device. A maximum of 32 external fanouts and 32 internal fanouts can be specified for each output. Also, the first internal fanout of a functional source device could be assigned any ID, but the other fanouts must have IDs within 256 IDs of the first fanout.

SCHED will send information to MODEL of those source devices which are active during the current time frame. First, the source devices which have external fanouts are sent to MODEL. The information consists of the device ID, followed by the output signal value in the case of simple and single delay functional devices. For variable delay functional devices, the device ID, number of active outputs, output numbers, and the signal value of each active output is sent to MODEL. For each source device, the MODEL processor executes the following six steps:

1. Using the device ID, access the model of the source device.
2. From the model, retrieve a portion of the 'type of device' field which is used as a macroinstruction to jump to the appropriate microroutine.
3. Retrieve the command for device field and send appropriate data to the Host Interface.
4. Update the outputs of the source device.

5. For every active output, send the information of external fanout to COMM.
6. For every active output, determine the internal fanout devices for each internal fanout
   a. Update the input
   b. Send information to EVAL.

In the beginning of each time frame, SCHED first sends information of source devices which have external fanouts followed by a control word of sixteen 1's which marks the end of source devices with external fanouts. In fact, there is only one control word which consists of sixteen 1's. The devices are assigned all possible 16 bit combinations for IDs, except the combination of sixteen 1's which is reserved for the control word. When MODEL encounters this control word, it sends it to COMM. Next, the SCHED send information of source devices which have only internal fanout devices followed by another control word which marks the end of all source devices in SCHED for the current time frame. For every source device with internal fanout devices, the MODEL processor executes the same six steps, mentioned above, except step 5. When the second control word is received from SCHED, MODEL starts receiving information of active devices from COMM which are external fanout devices from other sub-processing units. The information of each device consists of device ID, input number, and input signal value. For each device, the MODEl processor executes the following three steps:

1. Using the device ID, access the model of the source device.

2. From the model, retrieve a portion of the 'type of device' field and jump to the appropriate microroutine.
3. a. Update the input.
   b. Send information to EVAL. When a control word is encountered from COMM, which marks an end of all devices for the current time frame, MODEL sends this control word to EVAL and the Host Interface.

Care should be taken in microprogramming the MODEL processor because only one of the three different algorithms (one for source device with external fanouts, another for source device with internal fanouts, and the last one for the device from COMM) mentioned above should be executed, and a portion of the 'type of device' field can be used to jump to only one microroutine.

In the six-step algorithm, the operations involved at steps 4, 5 and 6 depend on whether the device is simple, single delay functional delay, or a variable delay functional device. Updating the output of a simple device merely consists of replacing the old signal value with the new. In the case of a single delay functional device, the new signal values and the old signal values are first compared to find the active outputs. These outputs are then updated, and the information on which outputs are active is also used in steps 5 and 6. In the case of variable delay functional devices, the information on which outputs are active is sent by SCHED. Because the delay at an output is variable, the signal value may have been changed in the previous time frames. Therefore, the new signal value and the old signal value at each active output are first

compared. If they are the same, this output is discarded and no event will be generated at this output. If the two signal values are different, the output is updated, and steps 5 and 6 will be executed for this output.

For simple devices, step 5 is executed by retrieving the field which contains the address of the external fanout, from the model of the device, and sending this information along with the new output signal value to COMM. In the case of functional devices, the fanout address and signal value is sent for every active output.

For simple devices, step 6 is executed by retrieving, from the model of the source device, the field which contains the number of internal fanouts. Through the ID of the first internal fanout, the model of the fanout is accessed. The input number provided by another field is used to update the input of the fanout with the output signal value of the source device, and the information which consists of fanout device ID, device type, input and output signal values, is sent to EVAL. For the other fanouts, the ID is found by adding their 5 bit partial ID field to the 16 bit ID of the first fanout, and the same operations are performed to send information to EVAL. In the case of functional devices, for each active output of the source, the same operations are performed as for simple devices, in order to send information of fanouts to EVAL.

Since the information in the model of both source and fanout device is updated, this new information must be written back into the MODEL memory before information of another source device is read from SCHED.

MODEL sends information of the device to EVAL which evaluates the device and sends to MODEL the device ID and information on whether outputs have changed. If outputs have changed, the extra information sent consists of either the new output signal values if the device is simple or a single delay functional device, or the number of active outputs, output numbers, and the signal values of active outputs if the device is a variable delay functional device. If the outputs of the device have not changed, then MODEL retrieves a portion of the 'device type' field from the model of the device and sends this information along with the information received from EVAL to SCHED. This information is important for SCHED to detect multiple input events. If the outputs have changed, then the following six steps are executed:

1. Using the device ID, access the model of the device.
2. From the model, retrieve a portion of the 'type of device' field, and jump to the appropriate microroutine.
3. Only for single delay functional devices, retrieve the field which contains the number of outputs.
4. Retrieve the 1 bit field which contains the information on whether the device has external fanouts or not.
5. Retrieve the field which contains the delay.
6. Send the information extracted in steps 3, 4 an 5 plus the information from EVAL to SCHED.

In the case of single delay functional devices and simple devices, one 8 bit delay field contains the delay. However, in the case of variable delay functional devices, each output may have variable delays which

depend on either the inputs and the signal values on these inputs, or the previous signal value of the output. The active output numbers and the new signal values provided by EVAL are used in conjunction with the information in the model of the device to find the delay. All active outputs which have the same delay constitute one event. Outputs with separate delays constitute separate events. In this manner, several events could be produced. For simple devices, the information sent to SCHED consists of device ID, presence of external fanouts, delay, and the output signal value. For single delay devices, the information consists of device ID, presence of external fanouts, delay, number of outputs, and the output signal values. For variable delay devices, the information consists of the device ID and the total number of active outputs. For each set of active outputs that have the same delay, the information includes the number of outputs, output numbers, presence of external fanout, delay, and the signal values of the outputs. When the MODEL receives a control word from EVAL, this marks the end of all active devices that have been evaluated during the current time frame, it sends it to SCHED. All the operations involved in the above algorithm consist of reading information from the model of the device. Since the model is not updated, there is no need to write back into the MODEL memory.

All the information that the MODEL sends or receives to or from the FIFOs of different units consists of information of one device at a time. The information of a device consists of the device ID followed by the rest of the information of the device. When the information of all the devices has been sent/received, the last word is the control word instead

of the device ID of another device. This control word marks the end of information for the current time frame.

The MODEL memory contains the model of (64K-1) simple devices. Whenever a device ID is read from FIFOs, the model of the device is accessed, and a portion of the 'type of device' field is used to jump to the microroutine of the device. When the control word is accessed from the FIFOs, location 64K is accessed from the MODEL memory. The portion of the 'type of device' field contains a unique macroinstruction which transfers control to a unique microroutine. This routine keeps track of the control words received from each unit during a time frame.

If an uncorrectable error occurs within the SCHED, COMM, or the EVAL, an extra control word is sent to MODEL. This extra control word is detected by this microroutine. The MODEL processor stops the simulation by not reading or writing into any of the FIFO buffers of these three units and conveys the information to the Host Interface which in turn sends it to the Host. If an uncorrectable error occurs in the MODEL, then this information is sent directly to the Host Interface by sending an extra control word. An uncorrectable error in the Host Interface could be sent directly to the Host. The diagnostic controller could be used to detect the errors as long as the error does not occur in the diagnostic controller.

It should be noted that if the last location in the MODEL memory is not reserved to include the macroinstruction of the control word, then the MODEL processor would have to distinguish between the device ID and the control word when information is received from the FIFOs.

An EDC device is provided in the check-only configuration (as discussed in chapter 3). The device consists of eight Am 2960s (two groups of four AM 2960s) to perform EDC on a word of 128 data bits and 16 check bits.

Evaluation Unit

MODEL sends the device ID, device type, input and output signal values of each device to EVAL. The 12 bits 'device type' field is used as a macroinstruction (and microaddress) to jump to the beginning of the microroutine for the device. EVAL evaluates the device, compares the new output signal values with the old, and sends to MODEL the device ID and information on whether the outputs have changed. Extra information, for a device, is sent if the outputs have changed.

In the case of single delay functional devices, all the output signal values need not be compared. It is enough to find a change in only one signal value in order to determine whether the outputs have changed. For simple and single delay functional devices, the extra information sent consists of the output signal values of the device. In the case of a variable delay functional device, all the new and old output signal values must be compared and a list of the outputs that have changed is made. The extra information sent to MODEL consists of the number of active outputs, output numbers, and the signal values of the active outputs.

When the information of all the active devices has been sent to MODEL, a control word follows. This marks the end of all devices to be evaluated during the current time frame. EVAL receives the control word from MODEL and sends it back to MODEL. It then constantly monitors the FIFO for information from the next time frame. It should be noted that the device IDs and the control word are not required by EVAL but are required by MODEL.

Communication Unit

COMM memory consists of a Fanout Data Memory (FDM), Captured Data Memory (CDM), and the rest of the memory contains information which is used to capture words from the capture bus and identify the fanout devices to which the captured data belongs. This information must be furnished by the Host computer before the simulation begins. A word in the 2K x 288 bit COMM memory consists of 256 data bits and 32 check bits. FDM consists of 32 x 288 bits, and CDM consists of 36 x 288 bits.

COMM receives information of a source device from MODEL in the form of an output signal value and a 13 bit address of FDM. The address specifies the FDM word and the first bit in the word from where the bits of the output signal value should be stored. For example, if a 2 bit signal value of 01 is to be stored in word 10 and bit 21, then the five bits of 01010 of the address specify word 10, and the eight bits 00010101 specify bit 21. The first bit of the signal value is stored in bit 21

and the second bit in bit 22 of word 10. Recall that before the simulation starts, the Host computer programs all the units in the sub PU with the number of signal values that will be used during the simulation. Either a maximum of three, seven, or fifteen different signal values could be used. The control store of COMM processor contains three micro-routines, one for each category of signal values, and only one micro-routine is used during the entire simulation of the network. When the control word is received from MODEL, it marks the end of all external fanout data for the current time frame. COMM is now ready for communication of fanout data on the capture bus. There is, however, one problem. Looking at the data in the FDM, there is no way of differentiating the new signal values which have been updated during the current time frame and the old signal values. This problem is solved by storing ones in the whole FDM before reading information, from the MODEL, of the current time frame. This is the reason why 2 bits for a signal value correspond to only three different combinations. The fourth combination of 11 is used by COMM processor to differentiate between the old and the updated signal values. The same is true if three or four bits are used for a signal value. Also, during each time frame only a small fraction of the FDM is updated. In fact, some words may not even be updated. For this reason, bit (0) of each word is used to determine whether the word has been updated.

A logic 1 in bit (0) means that the word is not updated. The maximum number of simple devices, with external fanouts, that can be accommodated in the FDM is 4080. COMM can easily distinguish between the control word

and the data word sent by MODEL because the data word will never consist of sixteen ones which distinguishes it from the control word.

Figure 17 shows how a typical COMM(X) may view the processing unit. It receives information from MODEL(X), stores all the output signal values of active source devices in the FDM, sends the data in the FDM onto the capture bus, receives the output signal values of external source devices and stores them in the CDM, maps the data in the CDM to the fanout devices, and sends the information of each active fanout device to MODEL(X). By the nature of the partitioning of network into subnetworks, almost all of the external fanouts of source devices in COMM(X) will belong to its neighbors -- COMM(X-1) and COMM(X+1). In most of the cases, the first 16 words of FDM will contain the output signal values of the source devices which have external fanouts in COMM(X-1), and the next 16 words will be from source devices which have external fanouts in COMM(X+1).

When all the COMMs are ready, the communication starts with COMM(0) word(0). COMM(0) sends its 32 words in the FDM onto the capture bus. The receiving COMMs capture the required words and store them in their CDM. When COMM(0) is done, COMM(1) sends data starting with word(0) and ending with word(31) of its FDM. Again, the receiving COMMs capture the required words. The communication, for the current time frame, is over when COMM(255) is done sending the 32 words in its FDM.

It has been pointed out that in most of the cases the source devices of a COMM have external fanouts in the neighbor COMMs. However, there

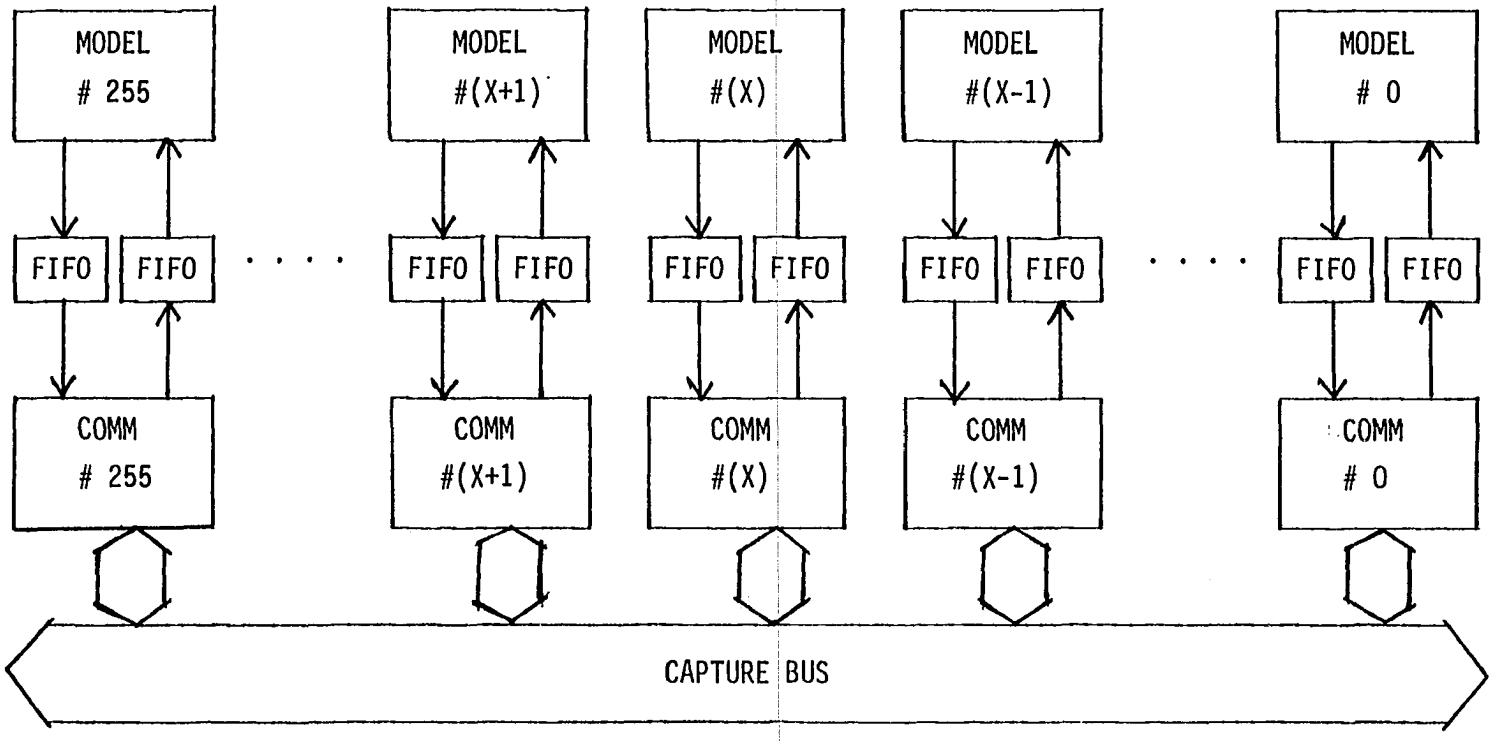


Figure 17. COMM's view of the processing unit

may be instances where some source devices of COMM(X) have external fanouts in COMMs other than COMM(X-1) and COMM(X+1). Another case, which is highly improbable, would be of a source device in COMM(X) that has external fanouts in several different COMMs. Regardless of the situation, the main idea is to have the output signal values of all the source devices that have external fanouts in some COMM(Y) assigned to the same word in FDM. In this way, when COMM(Y) captures a word on the bus, it will receive the signal values for all of its fanout devices. If there are more signal values than a word in FDM can hold, then, of course, consecutive words are assigned.

COMM(X) will capture only those words that it requires. The first word captured will be stored in location (0) of the CDM, and the following words captured will be written in consecutive locations. In some of the captured words, only a part of the word contains pertinent data, and the rest of the data belong to other COMMs. Since we have included the possibility that external fanouts of COMM(X) may be in COMMs other than the neighbor COMMs, the number of words captured may be greater than 32. A maximum of 36 words has been provided for the CDM. In the worst case, besides the two neighbor COMMs, data from four different COMMs could be captured.

For each word to be captured, an address of 13 bits is required. Eight bits specify one of the 256 COMMs, and five bits specify one of the 32 words in the FDM to be captured. The addresses of all the 36 captured words could be packed into 2 words of COMM memory. Some savings in memory could be made by taking advantage of the fact that consecutive words

in the neighbor COMMs (X-1) and (X+1) would be captured by COMM(X). Hence, the address of the first word and the number of words to be captured will suffice in the case of neighbor COMMs. But due to some unpredictability with which the external fanout devices could be found, this approach will not be taken, and one address of 13 bits will be provided for each word to be captured.

After the communication is over and no errors have been detected, the next step is to map the captured data to the fanout devices and send the information to MODEL. If bit (0) of the captured word is at logic 1, then the word does not have any new data. However, if bit (0) is at logic 0, then the bits of the signal values, in that word, which do not have ones, specify active signal values. Also, either the whole or just a portion of the captured word contains valid data, and the rest of the data belong to other COMMs. This portion of valid data will occupy consecutive bits in the word.

There are several ways of mapping the input signal values in the CDM to the fanout devices. One method suggested here is as follows:

1. Find a word in FDM which has valid data.
2. Find the portion of the word that has valid data.
3. From the valid data, find the active signal values.
4. For each active signal value, there may be one or more fanout devices. Find the address where the information for fanout devices is stored.

5. From the information of fanout devices, send to MODEL the device ID, the input number, and the input signal value for each fanout device.
6. Repeat (1) through (5) until the information of all active input signal values, in CDM, has been sent to MODEL.

An estimate of the total memory required by COMM is 2K words, as shown in Figure 18. For each input signal value in the CDM, a 19 bits address specifies the bit in a location of COMM memory, from where the information of the fanout devices starts. The information stored is shown in Table 6. A simple device with external fanouts can have a maximum of ten fanouts. On an average, it will have four external fanouts and one internal fanout. Therefore, fanout information for each simple device can be stored in 44 bits.

There are some data required for mapping which are not shown in Figure 18. For example, some data are required to map the signal value in the CDM to the 19-bit address which points to where the fanout information is stored. All the 19-bit addresses for each of the 4080 signal values are packed together in 303 words of memory. One way of mapping is by storing the address of the first location from where the 19-bit addresses start. If the number of the active signal value in CDM is known, then this number could be used along with the starting address to retrieve the corresponding 19 bit address.

One can also find ways of reducing the total memory capacity, but this can only be done at the expense of processor time. During the design of hardware, a trade-off between memory and processor time will

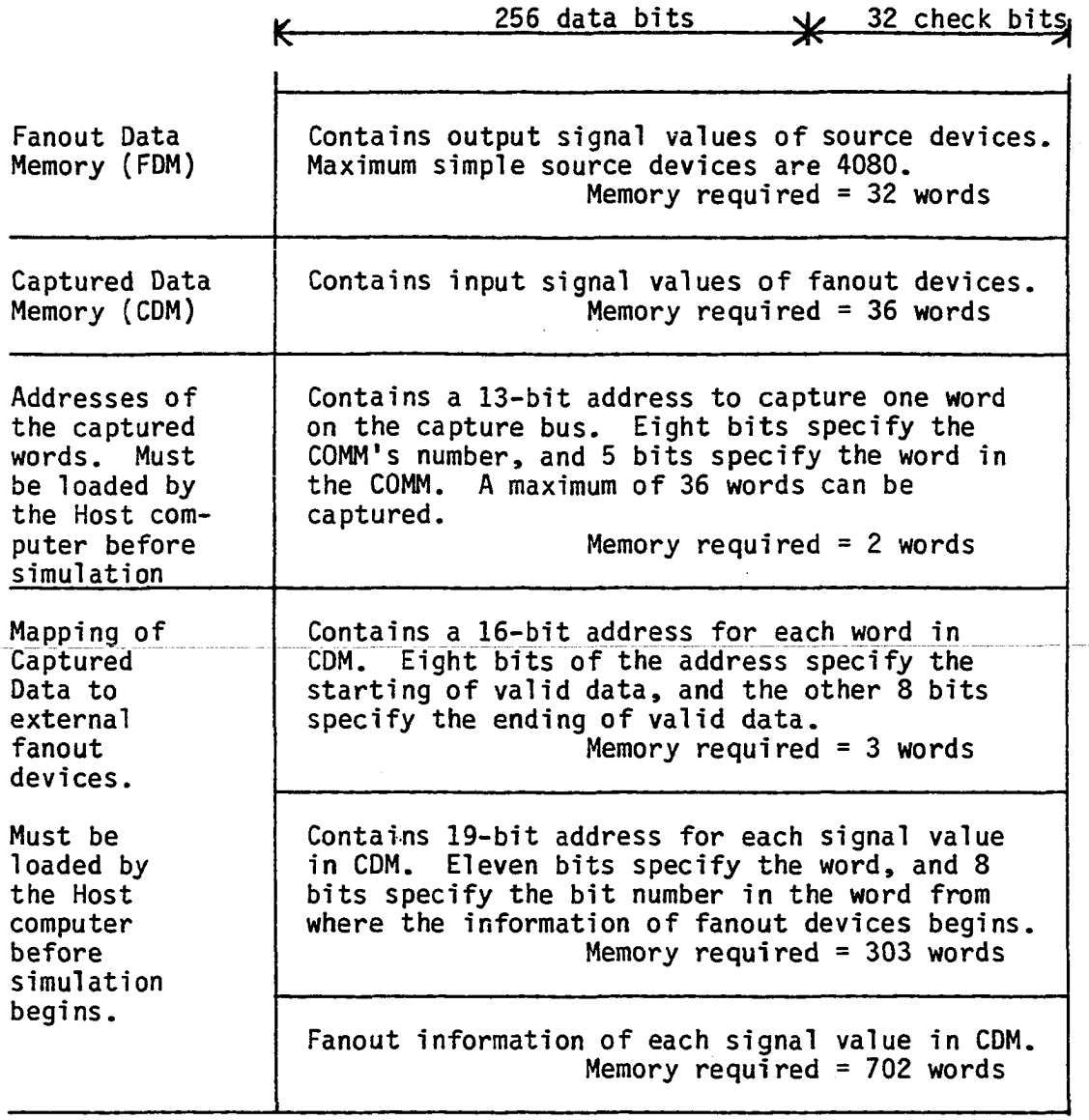


Figure 18. Arrangement of 2K words of COMM memory

Table 6. Fanout information of an external source device

Simple or functional device	= 1 bit

The following information is provided for simple devices:

Number of fanout devices	= 4 bits
ID of the first fanout device	= 16 bits
ID of the second fanout device	= 5 bits
.	
.	
.	
.	
ID of the last fanout device	= 5 bits
Input number of the first fanout device	= 2 bits
.	
.	
.	
Input number of the last fanout device	= 2 bits

The following information is provided for functional devices. Note that one output signal value could come from one output of the functional device.

Number of fanout devices	= 5 bits
ID of the first fanout device	= 16 bits
ID of the second fanout device	= 8 bits
.	
.	
.	
ID of the last fanout device	= 8 bits
Input number of the first fanout device	= 3 to 8 bits
.	
.	
.	
Input number of the last fanout device	= 3 to 8 bits

have to be made. In the discussion so far, memory has been allocated statically. Besides the 32 words of FDM, the rest of the memory could be dynamically allocated. This, however, will require much more sophisticated firmware.

MODEL will be sending or receiving information of a device at an average rate of 2 us. This time is sufficient for COMM to perform all the required operations. In between the receiving and sending of information to MODEL, COMM has to transfer external fanout data via the capture bus. If 2% activity is assumed, and using the ratio of devices with external fanouts to total devices as 1/16, approximately 2450 us are available for the communication of data. This corresponds to about 299 ns to send or receive a word on the capture bus. In parallel with the communication of data, 1225 internal fanout devices are evaluated and scheduled in each sub PU. During time frames when there are fewer than 1225 devices to be evaluated, the three units, SCHED, MODFL, and EVAL, will be waiting for COMM to send information. This ends up in the bottleneck of the system unless the communication is done at a higher speed. The communication time can be cut into half with the method that has been described.

Some of the factors that determine the speed of the bus include data skewing and the propagation delay of the signals, which is a function of the length of the bus and the time it takes for reflections on the bus to subside. Other factors include the time required to access the memory in order to send a word onto the bus, and the time required to access the address, capture the word, and store it in the memory. Previously it was

mentioned that the addresses, to capture the words on the bus, are stored in COMM memory. In order to capture words at a high speed, these addresses must be stored in high speed registers during communication of data on the bus. The bus has 256 data lines, 32 check lines and some lines for synchronization of the COMMs, during communication of data.

EDC capability is provided for the 2K x 288 bits memory and the capture bus. It consists of four groups of four cascaded AM 2960s. In all, sixteen AM 2960 chips are used to provide EDC for a word of 256 bits of data and 32 check bits. When the words of the FDM are read and sent onto the capture bus, the three error flags of the EDC device are monitored in the correct mode. The address of words that have errors is recorded. When the words are captured and written in the CDM memory, the EDC device is not in the generate but in the correct mode. The captured word has both data and check bits, and while it is written in the CDM, the EDC device checks it for errors. If an error occurs, not only the address of the word but also the syndrome bits are recorded.

Following the high speed transmission of data on the capture bus, each COMM checks whether errors have been detected in sending or receiving of data. If errors have not occurred, the COMMs will continue with their normal operations. However, if errors are detected, then each COMM will first consider those words which belong to the FDM. If the error cannot be corrected, then the COMM halts the simulation. If the error is correctable, then the word is read from the FDM, corrected, and retransmitted on the capture bus. The receiving COMMs capture and store the retransmitted word in the CDM, record its address if errors are not

detected, and both the address and the syndrome bits if errors are detected. When the transmission of error words from the FDM of all the COMMs are over, the next step for each COMM is to check if errors were detected on the captured words. Since corrected words of the fanout memory have been transmitted, the only other possibility is that the errors occurred on the capture bus. The syndrome bits of each error word could be decoded to determine the number of errors detected, and for single bit error which of the data or check bit is in error. If an error word cannot be corrected, the COMM, which is the source of the word, could be requested to send the word again. If uncorrectable errors occur again, then the simulation must be stopped and the cause of errors identified.

It is clear that high speed communication on the bus is vital to system performance. With the method discussed, the data on the bus will be transmitted as fast with EDC as without. An overhead is incurred only if errors are detected, and this will happen very seldom.

In summary, during each time frame the following operations occur in order:

1. Write ones in all the 32 words of the FDM
2. Receive information from MODEL and store the output signal values in the FDM.
3. When a control word is received from MODEL, communication of fanout data can start when all the COMMs are ready.
4. Start communication when all the COMMs are ready. The communication starts with COMM (0) word (0) and ends with COMM (255) word

(31). During communication, each COMM sends the output signal values of its source devices which have external fanouts and captures the input signal values of its fanout devices which have external sources.

5. All errors detected during the communication are corrected. If errors are uncorrectable, then the simulation is stopped and the source of the errors identified.
6. The active input signal values in the CDM are mapped to the fanout devices, and information of fanout devices is sent to MODEL.

Scheduling Unit

SCHED is responsible for scheduling events on the time wheel, retrieving events of the current time frame, and for cancelling previous events of the multiple input events.

The maximum allowable range of the simulation delay is 256. If a network consists of a slow microprocessor with a cycle time of 1 us, gates with delays in the range of 5 ns to 40 ns, and other devices covering a whole range of delays, then the real delay of the network may cover a range greater than the maximum allowable range. In this case, the range of the simulation delay must be increased. However, if the real delays, covering from 5 ns to 1 us, are divisible by 5, then the range of the simulation delay could be reduced to 200, well within the maximum allowable range. The delays of most of the digital devices in the network could be modelled to fall within a range of 256.

All the events are scheduled on the time wheel. A lot of work has been done in this area (19, 20). Each element of the time wheel represents a unit of simulation time. As shown in Figure 19, there are N elements. All events scheduled for the same time are placed in the same element of the time wheel. The time wheel has a range of N units of time. An event is scheduled by adding its delay to the current time (K) modulus N, i.e., (K + delay) mod N, and scheduling it in the first available space of the block in the resulting element. Since the current element (K) is not available until all the events have been removed, therefore all future events must have a delay equal to or less than (N-1) to be scheduled. Also, the block in which the event is scheduled must have an empty space.

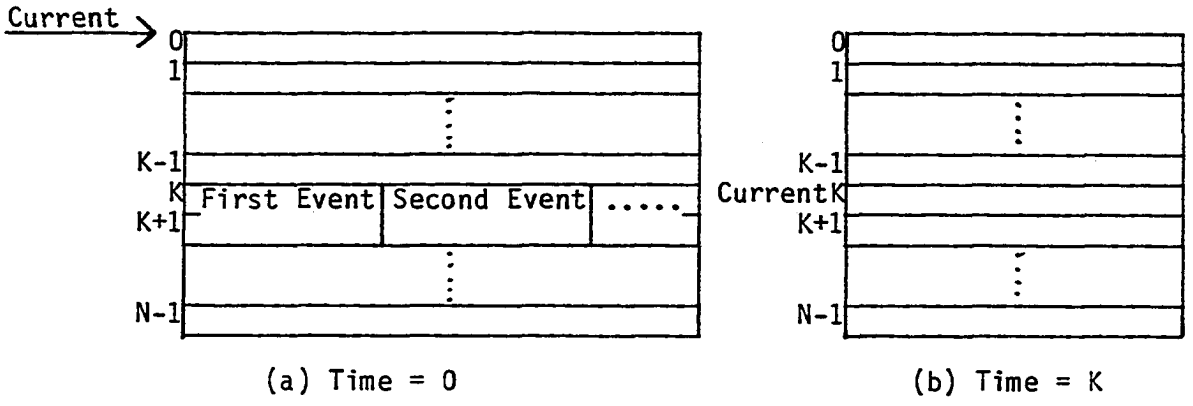


Figure 19. Time wheel

There are a whole lot of ways of assigning memory to the time wheel. One method is to assign the maximum memory ever required. Since the maximum allowable range of delays is 256, therefore, there are 257 elements. On an average, the activity in a network is 2%, but in this method the maximum activity must be known. Assume a maximum of 'X' active devices during a time frame. Since the time wheel contains only source devices, about X/2 source devices will be scheduled in one element. Here, it has been assumed that a source device has a fanout of 2. This is contrary to the previous assumption of an average of 5 fanout devices, but a safe number is 2. For each simple device, 4 bits of device type, 16 bits of ID, and 2 bits for the output signal value are stored. Therefore, the memory required by an element of the time wheel is (22xX/2) bits, and the total memory capacity is (257x22xX/2). If 4% is assumed to be the maximum activity in the network, then each element will consist of about 29K bits, and the total memory capacity is about 7.5M bits.

On an average, 2% activity is expected in a network. Therefore, most of the time at least half of the space in every element will not be used. This method is very wasteful of memory space. Since the memory requirements of each element are dynamically changing, a better method is to allocate memory upon demand. Also, the number of elements that should be assigned to the time wheel depend on the delay characteristics of the network. For example, a network may have devices with delays spread out over the range of 256 time units. In such a case, optimum memory could

be assigned to 257 elements of the time wheel. If more memory is required by an element, blocks from free space can be allocated. In another example, a network may have devices with delays in a very small range, starting from 1 unit. The range of the time wheel is one unit more than the range of delays in the network. Free space is used when more space is required by an element. In yet another example, the devices of the network may have delays in a small range or sets of ranges. For example, 90% of the devices may have delays within the ranges 20-40, 60-90 units of time whereas the other 10% have delays spread out, say at 5, 120, 200 units of time. Two approaches could be used in this example. In the first approach, the range of the time wheel corresponds to the maximum delay which is 200 from the example. In the second approach, the range of the time wheel is equal to a range within which most of the devices lie, which is 90 from the example. Events with delays greater than 90 are placed in the remote range blocks and their time of occurrence noted. The remote range blocks are periodically checked, and all events whose time of occurrence falls within the range of the time wheel are scheduled on the time wheel. When compared, the two approaches have their merits and demerits. The former approach is straightforward and will be used.

In order to determine the amount of memory required by the free space, a thorough research on the activity of networks needs to be done. However, as a rough estimate, a 2% activity per time frame with a fanout of 2 will give 656 source devices. A simple device requires 22 bits on the time wheel. If 2% activity is centered around the 16 elements and

the rest of the elements contain the same number of devices as the 16 elements do, then the amount of memory required is about 470K bits.

Since the range of the time wheel depends on the delay characteristics of the subnetwork, there are two methods of implementing the time wheel. In the first method, the range of the time wheel is equal to one more than the range of delay in the subnetwork. Microroutines of all widely used ranges are stored in the control store of the microprogrammed SCHED processor, and the same memory for scheduling events is interpreted in different ways, depending on which microroutine is used. Only one microroutine is used during the simulation of the network, and the Host computer chooses the appropriate microroutine for each subprocessing unit during initialization. In the second method, the range of the time wheel is fixed at 257 which is the maximum possible range of delays in a network. This method does not utilize the memory as efficiently as the first method, but it requires less microprogramming and relieves the Host computer from having to do extra work during initialization. Hence, the second method will be used here.

The SCHED memory, as shown in Figure 20, is divided into seven parts. The first part consists of the time wheel with 'N' elements giving the maximum simulation delay of 'N-1'. The second part consists of free space which is divided into blocks, and the availability of the blocks can be checked by interrogating the flags in the third part of the memory. One flag is assigned to each block in free space. If an element of the time wheel requires more memory than has been allocated, then the flags are checked, the first available flag is marked "occupied", and its

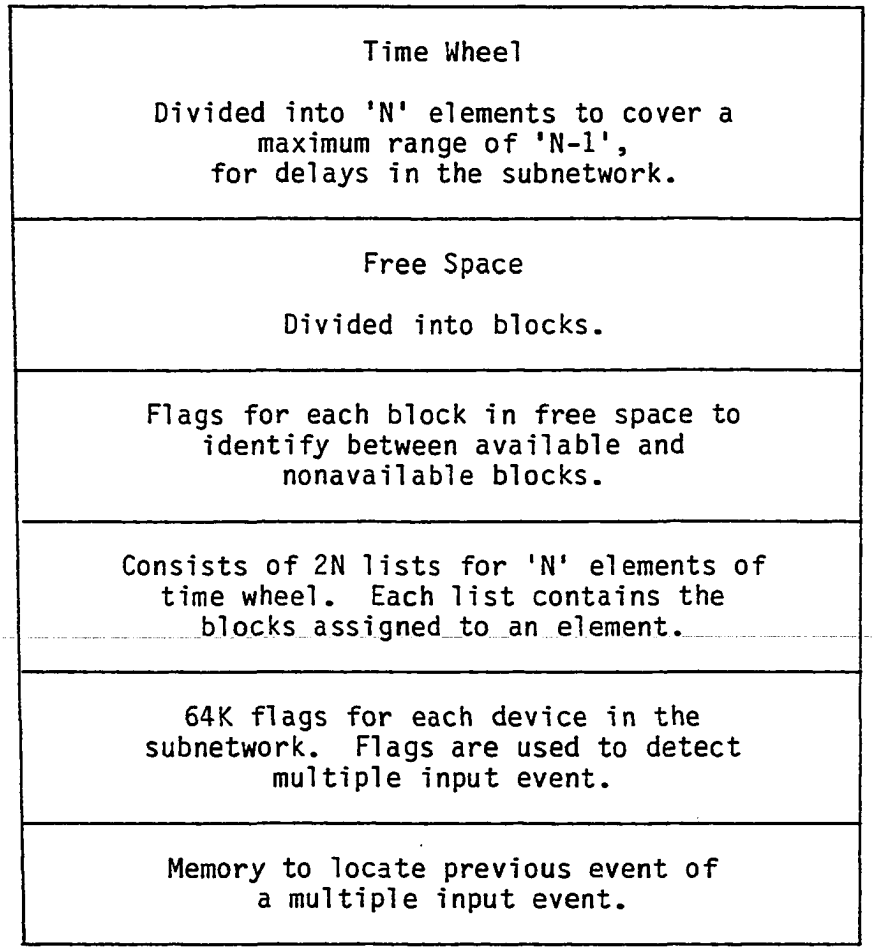


Figure 20. Arrangement of SCHED memory

corresponding block is assigned to the element. The physical location of the block can either be calculated from the knowledge of the number of the flag and the address in memory from where the free space begins, or a fourth part of the memory, which contains the addresses of all the blocks in free space, could be used.

For each element of the time wheel, the fifth part of the memory contains a list of all the blocks that have been assigned to it. There are two types of blocks. One type holds the information of those source devices that have only internal fanouts, and the other type holds the information of those devices that have external fanouts. For each element, two lists are provided, one for each type of blocks. Each list is divided into four fields. The first field contains the block number of the current block that is being used. The second field points to the first available bit in the block. Information of an event to be scheduled can be stored starting from this bit. The third field contains the block numbers of all the blocks that have been completely filled with information. The fourth field contains the number of blocks in the third field. As the current block is completely filled and more space is required by the element, the block number from the first field is moved to the third field, and the fourth field is incremented by one. The block number of the newly acquired block is recorded in the first field, and the second field is filled with zeros because the current block has not yet been filled with information. When the element becomes the current time frame, all the information that it contains is sent to MODEL. The

blocks that have been allocated to the element are released to the free space by marking their corresponding flags as "available".

The fifth and the sixth parts of the SCHED memory will be discussed later.

SCHED receives the information of the device to be scheduled from MODEL. The information consists of the device ID, device type, and information on whether the outputs have changed. If the output has changed, the extra information for a simple device includes the presence of external fanout, delay, and the output signal value. For a single delay functional device, in addition to the information received for a simple device, the number of outputs and one or more output signal values are also included. For a variable delay functional device, the extra information includes a number specifying the total number of active outputs. Each set of active outputs with the same delay constitutes an event, and the information includes the number of outputs, output numbers, presence of external fanout, delay, and the signal values of active outputs. It should be noted that for single delay functional devices all the output signal values are received, whereas for a variable delay functional devices, only the signal values of active outputs are received. The 4 bits of device type received by SCHED is just a part of the 12 bits of device type stored in MODEL. These 4 bits specify which one of the eleven groups the device belongs to. Recall that the eleven groups consist of simple and functional devices. Among functional devices, each of the single and variable delay devices are further divided into five groups, depending on the number of inputs and outputs. For each event to

be scheduled, SCHED receives the device ID and device type, followed by the rest of the data of the device. The device type is used as a macro-instruction to jump to the appropriate routine in its control store. This routine then identifies the rest of the data sent by MODEL and also where the data ends. The delay is used to find the element in the time wheel, and the presence of external fanout is used to schedule the event in the appropriate list. For simple devices, the device type, device ID, and the output signal value are stored on the time wheel. For single delay functional devices, the device type, device ID, number of outputs, and the output signal values are stored. In the case of variable delay functional devices, for each event the device type, device ID, number of active outputs, output numbers, and the signal values of outputs are stored. This same stored data of the events, except device type, is sent to MODEL when the element becomes the current time frame. Data of all the events in an element are packed in continuous bits. Looking at the data, it is not possible to separate the end of data of one event and the start of data of another. For all the events, the first 4 bits consist of device type. When information is retrieved from the element and sent to MODEL, for each event the device type is used to jump to a routine in control store. This routine is then used to send all the stored information of the event to MODEL. The 4 bits of device type are primarily used to store the data in an efficient manner.

The problem of multiple input events can be solved by keeping track of where on the time wheel the events of the current time frame are scheduled. When a multiple event is detected, the previous event of that

device is cancelled. The multiple input event is detected by keeping the flags of all the 64K simple devices in the sixth part of the SCHED memory. This part of the memory consists of 64K bits. Before receiving any events of the current time frame, the SCHED sets all the flags to 'unused'. When an event is scheduled on the time wheel, its flag is checked. If it is 'unused', then the event is scheduled, and the flag is set to 'used'. If the flag is found to be 'used', then a multiple input event has occurred. The previous event must be cancelled, and the new event scheduled. If the outputs of the new event have not changed, then the event is discarded and its previous events cancelled.

In the case of simple and single delay functional devices, all multiple input events of the same device occupy a fixed space and element in the time wheel. In addition, for single delay functional devices, some outputs may have external fanout, whereas others may not, so that a multiple input event may have external fanout, whereas the previous event may not, or vice versa. Recall that each element has two lists -- one for devices with external fanouts, and the other for devices with only internal fanouts. For simple devices, an event is cancelled by simply locating the previous event and replacing it by the new multiple input event. In the case of single delay functional devices, if the two fanouts are the same, then the previous event is located and replaced by the new event. If the two fanouts are not the same, then the previous event is cancelled, and the new event is scheduled in the first available space of the other list in the same element. One extra bit is provided following the device ID for all functional devices in the time wheel. When an

event is cancelled, this so-called 'cancel bit' is changed from 'active' to 'cancel'. In order to locate an event for both the simple and functional devices, the address, consisting of the block number and the first bit in the block from where the information of the device is stored, is recorded in the seventh part of the SCHED memory. For single delay functional devices, a bit which specifies whether the data are in internal or external list is also recorded.

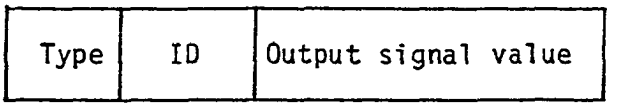
The variable delay functional devices pose a rather complicated problem when events are to be cancelled. From the evaluation of such a device, one or more than one event could evolve. These sets of events are scheduled in various elements of the time wheel. If multiple input events occur, then the new set of events may have delays that are different from the previous set of events. Even if the two events have the same delay, the number of active outputs may be different, or the two may not have the same type of fanouts, so that the new event cannot be replaced by the previous event. Therefore, in order to locate the events of a variable delay device, the seventh part of the memory should have the number of events, and for each event, the delay, presence of external fanout, and the address of the first bit in a block of the time wheel from where the information is stored. If the two events (multiple input event and its previous event) occupy the same amount of space in the same fanout list, then the previous event is replaced by the new event. Otherwise, the previous event is cancelled by changing the cancel bit from 'active' to 'cancel' and scheduling the new event. All cancelled events will, of course, leave dead spaces in the time wheel.

An estimate of the amount of memory required by the seventh part of the SCHED memory should include for each of the 64K simple devices about 20 bits for the block number and the bit in the block from where the data of the device are stored. The total memory required by this part is about 1.3M bits. The 20 bits of each of the 64K devices are stored in a fixed space of the memory, and the data can be retrieved once the device ID is known. There are other methods that could reduce the amount of memory, but only at the expense of processor time.

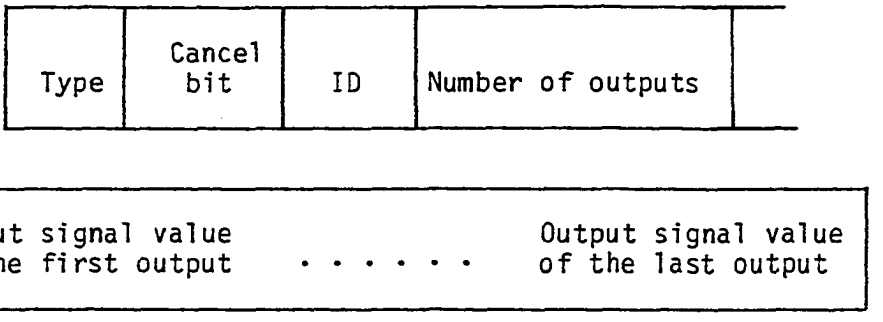
Figure 21 shows the kind of information of a device stored on the time wheel.

In the beginning of the current time frame before any event is scheduled on the time wheel, the 64K flags, used to detect the multiple input events, must be set to 'unused'. To schedule an event, the following operations are performed:

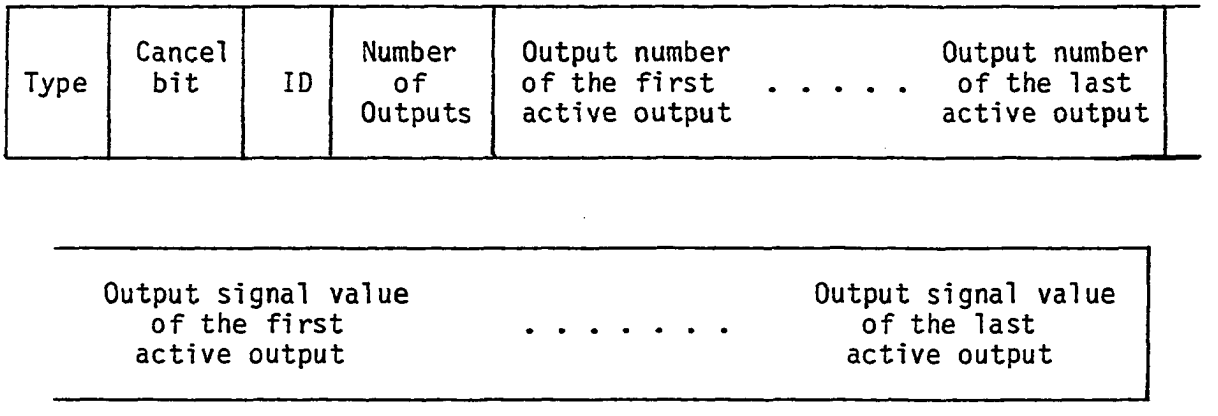
1. Receive the ID and the type of the device from MODEL. Use the type field to jump to the microroutine of the device in the control store of the SCHED processor.
2. Identify and collect the rest of the data of the event from MODEL.
3. From the ID, detect if multiple input event has occurred.
4. (a) If multiple input event has not occurred, then using the data which contain the delay and the presence of external fanout, schedule the event on the time wheel. In the case of functional devices, introduce a 'cancel bit', mark it active, and place it with the rest of the scheduled data.



(a) Simple Device



(b) Single delay functional device



(c) Variable delay functional device

Figure 21. Information of a device stored on the time wheel

Also record the appropriate information so that the previous events of the future multiple input event can be located.

(b) If multiple input event is detected, then cancel the previous event and schedule the new event.

In the case of multiple input devices, the information from MODEL may include more than one event for a device. When a control word is received from MODEL, it marks the end of all events to be scheduled in the current time frame. The current time frame is advanced to the next element of the time wheel, and events stored in the element are sent to MODEL. First the events with external fanout are sent, followed by events with only internal fanout. A control word is sent after all the events in the element have been sent to MODEL. This control word marks the end of all events to be simulated in the current time frame.

The following operations are performed to retrieve the data of an event from the current element of the time wheel and send it to MODEL:

1. Use the type field to jump to a microroutine in the control store of the SCHED processor.
2. For simple devices, send the rest of the data. For functional devices, first check the 'cancel bit'. If it is equal to 'cancel', then the event is not sent. If it is equal to 'active', then the rest of the data are sent to MODEL.

In the beginning of each time frame, all the 64K flags that are used to identify multiple input events must be set to 'unused' before any event can be scheduled. The time required to complete this operation would depend on how many memory locations are written. If the memory

containing the flags is arranged as a 256 x 256 bits memory, and assuming the time required to write into the memory is about 150 ns, then the time required to write into the 256 locations is about 38.4 us. This memory could be separated from the rest of the SCHED memory, and the DMA Address Generator (AM 2940) could be used to perform the above operation while the processor is sending events of the current time frame to MODEL.

Finally, an example will be considered in which a network has simulation delay in the range of 256. Assume that the SCHED memory has 128 data bits and 16 check bits in a word, and the EDC unit consisting of AM 2960s is used in the check-only configuration. It is assumed that the 256 x 256 bit memory consisting of flags which are used to detect multiple input events is separated from the rest of SCHED memory. The 1.3M bits of data memory required to locate the previous events of the multiple input events is arranged as 10K x 128 bits. The rest of the data in the SCHED memory is arranged as 6K x 128. The total memory required by SCHED is 16K x 144 bits. The time wheel consists of 257 elements, and the memory allocated to the elements consists of blocks. There are two types of blocks. The first type holds the data of those source devices that have only internal fanouts. The second type holds the data of those source devices that have external fanouts. In the time wheel memory, each element has four locations or block (0) for events with only internal fanouts and one location or block (0) for events with external fanouts. Since a subnetwork consists of a maximum of 4K source devices with external fanouts, therefore the ratio of source devices with only internal fanouts to external fanouts is 15:1. The number of locations that

will be assigned to each type of blocks will be in this ratio. The free space consists of 254 blocks divided into 127 blocks of each type. Each internal fanout block consists of 32 locations, and each external fanout block consists of 2 locations. Each type of block in free space is numbered from 1 through 127. In the free space, the first 4064 locations are occupied by 127 type 1 blocks, and the next 254 locations are occupied by 127 type 2 blocks. Two locations are assigned to that part of the memory which contains flags for each block in the free space. The flags are used to determine the availability of the block in free space. From the two locations assigned, the first location has 127 flags for type 1 blocks, and the second location has 127 flags for type 2 blocks.

The next part of the memory keeps track of all the blocks that have been assigned to each element. It contains two locations for each of the 257 elements. The first location keeps track of the type 1 blocks, and the second location keeps track of the type 2 blocks. Each location is divided into four fields. The first field of 7 bits contains the current block number that has been allocated to the element, and the second field of 12 bits points to the first available bit in the current block. The third field of 105 bits contains the block numbers of up to 15 blocks assigned to the element. Block (0), which is assigned to each element of the time wheel, is not included. The fourth field of 4 bits contains the number of blocks in the third field. A maximum of 16 blocks of each type can be assigned to each element, which corresponds to a maximum activity of about 9% in the subnetwork. It should be noted that the addresses of the blocks are not stored in the SCHED memory because these addresses can

be formed very easily. The block number can be multiplied by 32 to give an offset from the starting address from where the blocks are stored. The starting address, stored in the control store of the SCHED processor, can be added to the offset to form the starting address of the block in question.

For each device, there is a 19 or 20 bit space packed together in that part of a 1.3M bit memory which is responsible for locating the previous events of the multiple inputs events. Recall that when an event is scheduled, the 20 bit space is loaded with the address of the first bit from where the event is stored. Also, when a multiple input event is detected, the address stored is used to locate the previous event. If an AM 29116 is used as the SCHED processor, then the operations involved in finding the 20 bit space for a given device ID will be very time consuming because the microprocessor does not execute multiplication and division operations. A way to get around this problem is to have a separate 64K x 20 bit memory with its own EDC unit. Then, the device ID can be used as an address of the memory to get to the 20 bit space. Note that functional devices occupy more than one location of MODEL memory. Therefore, they are assigned an equivalent amount of space in this memory also.

Diagnostic/WCS Controller and Host Computer Interface Unit

This unit acts as an interface between the Host computer and the rest of the sub PU. It communicates with the Host computer via a standard bus, and the choice of interface hardware depends on the bus used. Among the standard 32 bit buses, Multibus II and the VME bus are expected to dominate the market. Data rates in these buses are expected to exceed 40 megabytes/sec. Multibus II uses a synchronous timing scheme, whereas VME bus runs asynchronously. IEEE's Futurebus purports to be processor-independent and is attractive for special purpose processors. It provides means for the transfer of information between up to 32 logical modules working over a single backplane.

There are two methods of connecting the sub PUs to the Host computer via a standard bus. In both the methods, the bus is used for communication between the sub PUs and the Host computer. This bus is not used for communication between sub PUs. In the first method, the 256 sub PUs are connected to the Host computer via a shared bus. Before sending data to the Host computer, the sub PU must first be granted the bus. Simulation results for a time frame or control signals are then sent to the Host. Through the bus, the Host can load the memories of the sub PUs with simulation data, send control signals and data to start simulation, write into the WCS of EVAL, or start diagnostics. Another method of communication between the Host and the sub PUs is to connect all the 256 sub PUs through twisted pair wires to a Bus Controller and a Control Processor. The Control Processor is connected to the Host via a standard bus. All

communication between the Host and the sub PUs is done through the control processor.

The key to high speed simulation lies in the partitioning of a network into subnetworks. The partitioning must be done by the Host computer in the most optimal manner. Each subnetwork of (64K-1) simple devices should have no more than 4080 simple source devices with external fanouts. This number corresponds to about 1/16 of the total devices. Figure 22 shows a network partitioned into subnetworks. Due to the nature of the partitioning, most of the source devices have external fanouts in the neighboring subnetworks. Along the breadth of the subnetwork, only the source devices at the two extremes will have external fanouts; others will have only internal fanouts. If a 1Kx64 subnetwork is considered with two devices at each extreme contributing to external fanouts, then a total of 4K devices have external fanouts. With this scenario, all subnetworks which have a breadth greater than 64 can be used. If one device at each extreme contributes to external fanouts, then the subnetwork could be arranged as 2Kx32, and all subnetworks that have a breadth greater than 32 could be used, that is, 1Kx64, 512x128, etc. It should be mentioned that in practice a subnetwork may not have devices arranged in a nice rectangular manner, but the main idea is to have not more than 4080 simple source devices with external fanouts in a subnetwork.

For each sub PU, the Host computer must also assign a 13 bit address of the FDM in COMM to each source device which has external fanouts. This information of each source device, along with its model, must be

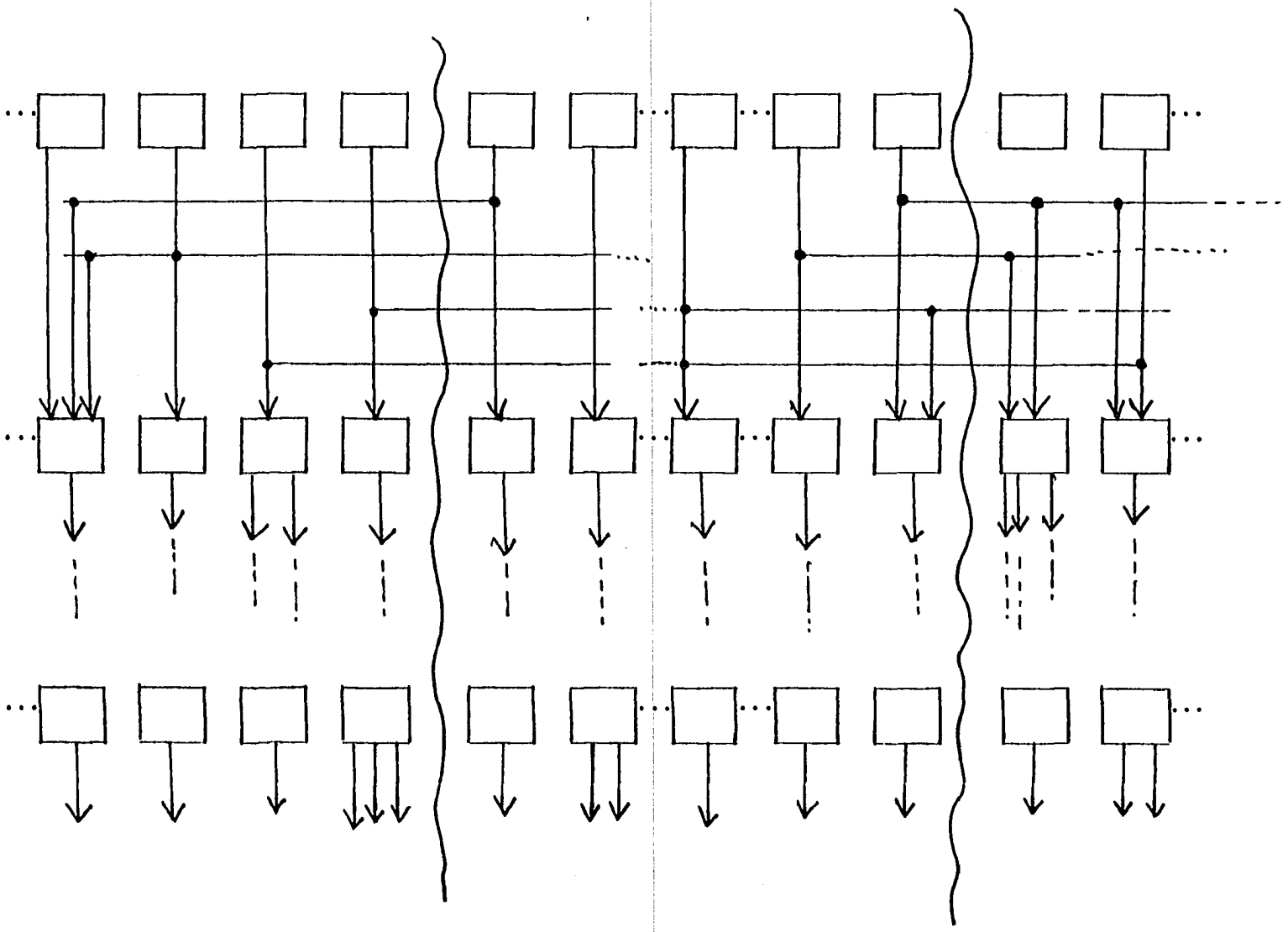


Figure 22. Partitioning of a network into subnetworks

loaded in the MODEL memory. The information required to capture the data of external source devices, and the mapping of captured data to fanout devices, must be loaded in the COMM memory.

In a subnetwork, if the source device is simple, then its first internal fanout device can be assigned any 16 bit ID, but the IDs of other internal fanout devices must be within 32 IDs of the first fanout device. If the source device is functional, then its first internal fanout device can be assigned a 16 bit ID, but the IDs of other internal fanout devices must be within 256 IDs of the first fanout device. The IDs of fanout devices, which have external sources, will be automatically assigned because these devices have at least one internal source. This method of assigning IDs will save some memory in the MODEL of sub PU as compared to assigning arbitrary 16 bit IDs to each device, but it will also put some extra work on the Host computer.

The Host computer should also keep track of the real delays associated with each device in the network. These delays should be converted to simulation delays before the models of the devices are loaded into the sub PUs. This step puts extra work on the Host, but the performance of the PU will be increased. If the real delays are within 256 units of time, then this step could be skipped.

The Host computer should also program the sub PUs for the number of signal values to be used by the network, and the memories of COMM and MODEL in each sub PU must be loaded with the simulation data. All the data to program or load the memories of the sub PU are sent to the HOST INTERFACE and sent on through the FIFO buffers to MODEL, and then to

other units. After system reset, control is transferred to location zero of the control store in each unit. Microprograms could be written starting from location zero so that the units could be programmed and their memory loaded with simulation data.

The amount of memory allocated to each unit was calculated by taking only simple devices into consideration. MODEL memory has (64K-1) words for (64K-1) simple devices. Functional devices are equivalent to tens and hundreds of simple devices. Since they require more memory, therefore, fewer functional devices can be represented by (64K-1) words of MODEL memory, and more memory is available to each functional device in all the units, as compared to simple devices.

HOST INTERFACE is connected to MODEL through FIFO buffers. The amount of data received from MODEL during simulation depends on how much data have been requested by the user through the 'command for the device' field in MODEL memory. Data words between time frames are separated by control words. The data word following the control word consists of the number of the time frame to which the following data belongs. If an uncorrectable error is detected during simulation, then MODEL sends two consecutive control words. Simulation is stopped by sending an error message to the Host, and proper action is taken.

HOST INTERFACE also consists of Diagnostics/WCS Controller, and the same processor can be used to transfer data between MODEL and the Host computer and also for Diagnostic/WCS purposes. During Normal mode, the unit either receives data from the Host and transfers it to MODEL or vice versa. During the WCS mode, the unit receives data from the Host and

loads it serially into the WCS of EVAL. During Diagnostic mode, the unit tests the hardware of the other four units in the sub PU. At any time, only one mode is used. Therefore, the hardware of the unit can be shared among the three modes of operation.

The rest of the discussion deals with Diagnostics/WCS.

Testing of digital hardware has always been costly and time consuming. These problems can be reduced by building diagnostics hardware into the system and testing it via computer-aided environment. A Diagnostics controller could be used to load the test vectors into the system, unload the test results, and compare it to the expected results (result vectors) in order to pin-point hardware related failures in the system. From the cost viewpoint, the additional hardware required for diagnostics must be minimized.

Each of the sub PUs has a Diagnostic/WCS controller to test the hardware of the four units (MODEL, EVAL, COMM, SCHED) and to write into the WCS of EVAL. EVAL contains the evaluation microroutines of all widely used devices in the PROM of the control store. The Host computer can load the evaluation microroutine of a new device in the WCS through the WCS controller. A typical configuration of any unit is shown in Figure 23. The data, address, status, and control registers are replaced by the Diagnostics/WCS pipeline registers (see Appendix B for data sheets of AM 29818). The 29818 consists of two registers. During diagnostics, serial shadow register is an image of the output register. During normal operation, the output register is used, but during diagnostics, the test vector is first shifted in the shadow register before it is finally

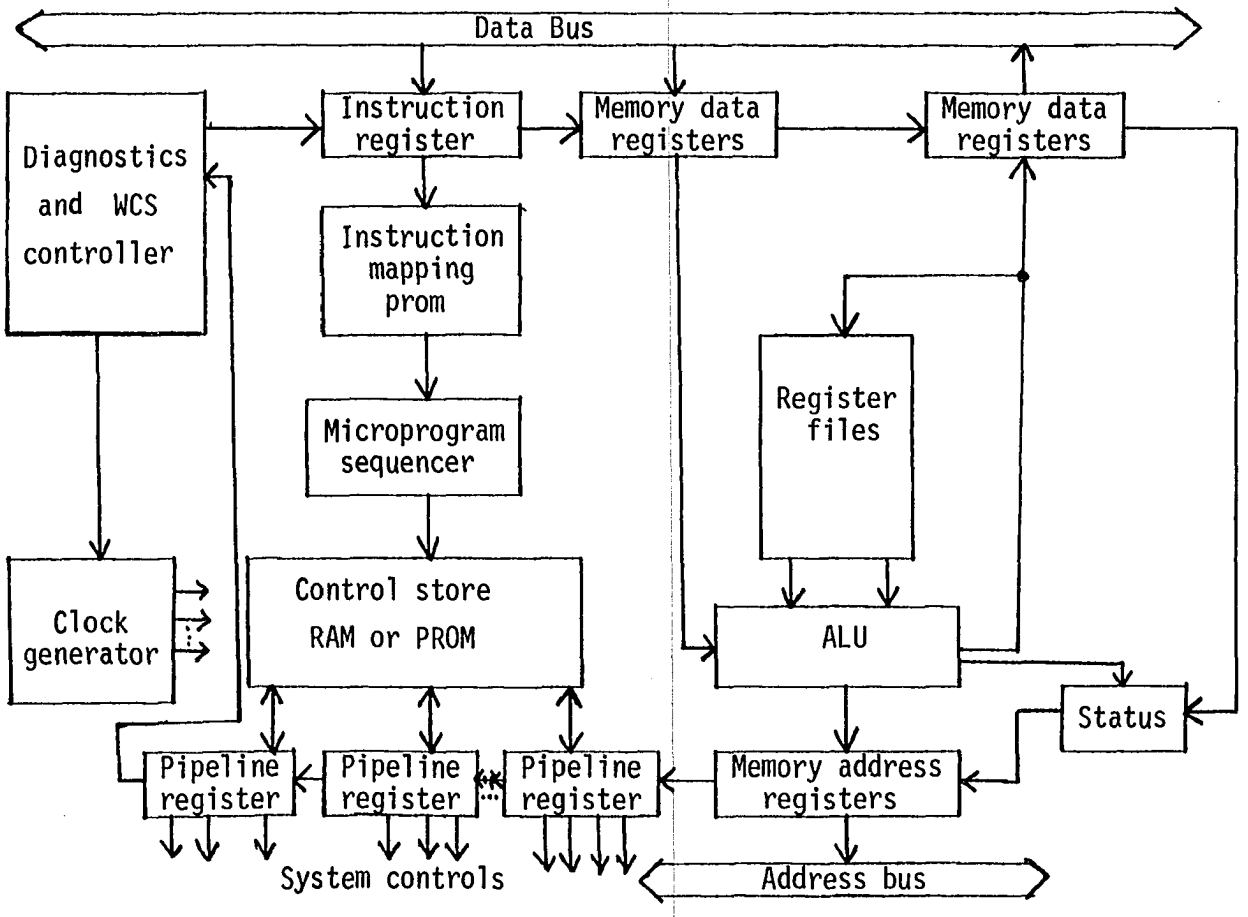


Figure 23. Typical configuration of a unit

loaded into the output register. The use of two registers overcomes the inherent drawbacks in the LSSD where data are shifted directly into the output register, and the unit goes through state changes while the shifting is going on. Since diagnostics are done infrequently, it is costly to provide additional chips, data and control lines just for testing. Therefore, data are shifted in and out serially through the unit as shown in Figure 24. After the diagnostic data have been shifted in, the unit will run with the diagnostic data as though it is in normal operation. After a specified number of microcycles, the test result is shifted out and analyzed. The PCLK of the registers is indirectly controlled by the controller through the WAITREQ and READY inputs of the Clock Generator (see Appendix B for data sheets of AM 2925). When WAITREQ is activated by the Diagnostics controller, the AM 2925 enters a wait state, and all the clocks to the unit, including the PCLK, are disabled. The activation of the READY signal enables all the clocks. Through the control of these two signals, the Diagnostics/WCS controller can output a specified number of microcycles to the unit. Also, the activity of a unit can be synchronized with that of the Controller. During the normal mode, i.e., when Diagnostics/WCS Controller is not operating, the 2925 runs freely, and its microcycle length is controlled by L_{3-1} inputs which are provided by the L_{3-1} field in the microinstruction of the unit.

As shown in Figure 23, 29818s are used for pipeline registers which contain the current microinstruction. Besides executing microoperations, some of the bits in the microinstruction are used to form the next microinstruction. By breaking this feedback path, the sequential circuit is

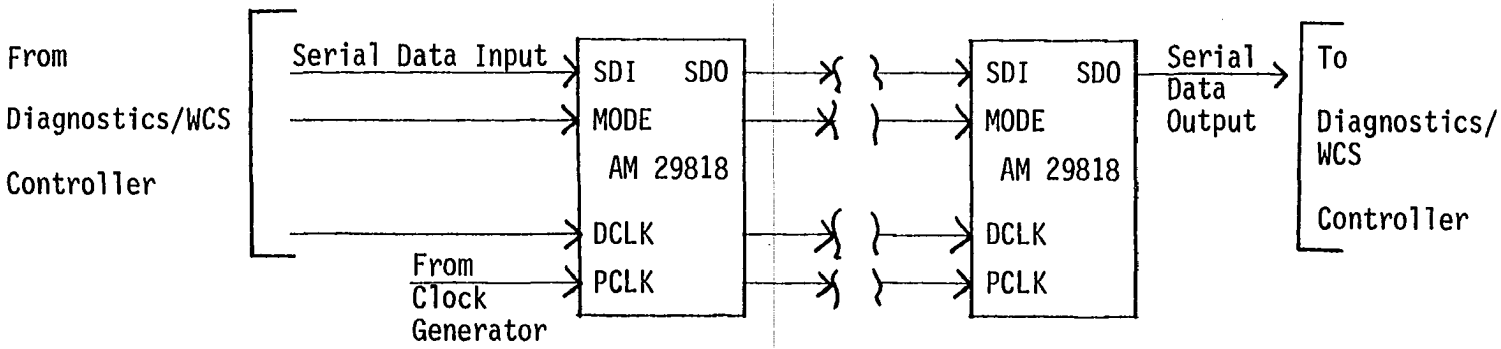


Figure 24. Connections for Diagnostics/WCS registers

converted into a combinational circuit, and all the tests developed for combinational circuits can be used here. Also, this part of the processor controls the hardware in the rest of the unit, and this is the place from where tests must be performed. The following sequence of steps is performed during diagnostics:

1. The test vector is shifted serially into the shadow registers through the SDI input and clocked by the DCLK. Mode is LOW, and PCLK is disabled.
2. After the full test vector has been shifted, the MODE is switched to HIGH, DCLK is disabled, and PCLK is enabled. This loads the contents of the shadow register into the output register, thus allowing the internal state of the system to be set to a desired state.
3. The test result is set up at the inputs of the 29818s. It is clocked into the output registers by switching the Mode to LOW, enabling the PCLK and disabling the DCLK. The test result could be formed as a result of executing one or more microinstructions. The first microinstruction is, of course, provided by the test vector, and the other microinstructions are produced by the control store.
4. The test result is loaded into the shadow register by enabling the DCLK, disabling the PCLK, and switching the MODE to HIGH with SDI as LOW.

5. With Mode switched to LOW, DCLK enabled, PCLK disabled, the test result is shifted out serially through SDO and analyzed while another test vector is shifted in through SDI.

Besides diagnostics, the Diagnostics/WCS controller is also used to load microroutines of new devices from the Host computer into the WCS of EVAL. Normally, a microroutine will be written into continuous locations of the control store. The Microprogram Controller (see Appendix B for data sheets on AM 2910-1) is used to provide the address of the WCS location and increment the address for each subsequent write operation. In addition to the signals for diagnostics, two signals are required to individually control the microrprogram controller and the output registers of 29818s, and one WE signal is required to write into the WCS. Figure 25 shows the connections required for the additional signals. The '2910-1 CLOCK CONTROL' signal is used by the WCS Controller to control the clock (CP) of the 2910-1. When the signal is HIGH, CP is disabled, and the 2910-1 holds its data. When the signal is LOW and the 2925 is not in the wait state, the 2910-1 is used to produce a new address for the control store. Similarly, the '29818 PCLK CONTROL' signal is used by the WCS Controller to control the PCLK of the diagnostic registers. One method to load the WCS is shown below:

1. Before data can be written, the 2910-1 is initialized to output the desired address of the location in the control store. The 12 bit address and an instruction for the controller is shifted serially, at a high speed, into the shadow register. The CJP instruction is made unconditional by making CCEN = 1. The 12-bit

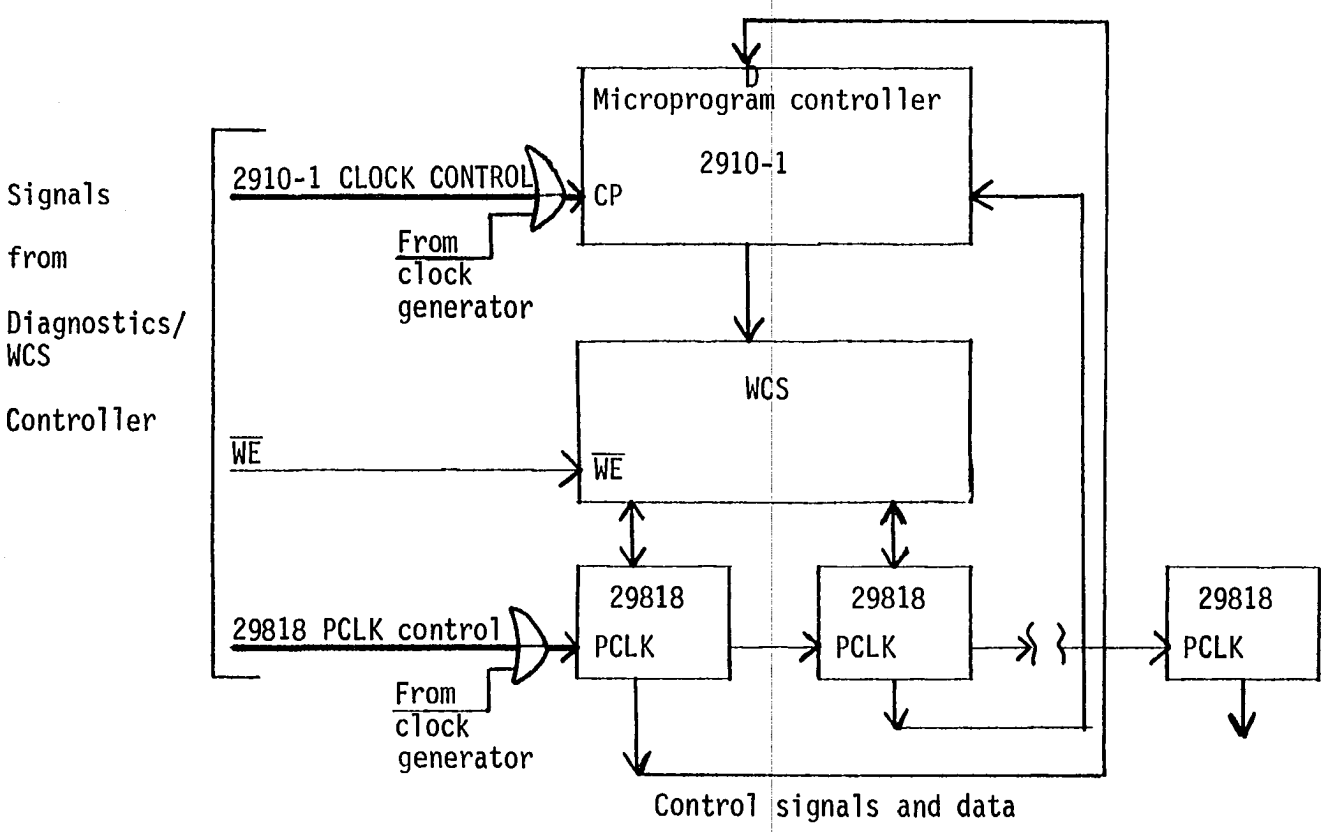


Figure 25. Additional connections for writing into WCS

address is stored in the D_{11-0} field, and the 4 bit instruction in the I_{3-0} field of the microinstruction. The length of the microcycle used during the write operation is selected through the L_{3-1} field in the microinstruction. During this operation, PCLK and DCLK are enabled, MODE is LOW, and CP is disabled. As data are shifted into the shadow register, they are also loaded into the output register.

2. The microinstruction to be written into the control store is shifting serially through the SDI input. MODE is LOW, DCLK is enabled, and PCLK and CP are disabled.
3. Now the data for the WCS are provided by the shadow registers, and the data for the 2910-1 to produce the desired address are provided by the output registers. Data are written into the WCS by switching the MODE to HIGH, SDI to HIGH, WE to LOW, disabling the PCLK, and enabling the DCLK and the CP. WE is HIGH at all times, except during the cycle when data are written into the WCS. WE should go LOW after the address of the WCS has been stable. The CJP instruction with CCEN=1 forces the 2910-1 to output the data in the D_{11-0} field of the microinstruction as the address of the WCS.
4. A new instruction, CONT, for 2910-1 is shifted in the I_{3-0} microinstruction field of the shadow register and loaded into the output register. MODE is LOW, WE is HIGH, PCLK and DCLK are enabled, and CP is disabled.
5. Same as (2).

6. Same as (3) except the 'CONT' instruction forces the 2910-1 to produce the next address as the increment of the previous address.

Steps (5) and (6) are repeated as long as data are written in continuous locations of the WCS. The reason for controlling the PCLK through the WCS controller should now be clear. The PCLK must be disabled in step (6) to hold the 'CONT' instruction in I_{3-0} microinstruction field of the output register. Steps (1) and (2) are used if data are not written in continuous locations.

A thorough discussion of Diagnostics/WCS circuitry is given in Appendix A.

In normal operation, the Diagnostic/WCS controller inactivates the diagnostics/WCS feature by setting the MODE to LOW, WE to LOW, disabling the DCLK, having the 2925 free run by setting both WAITREQ and READY HIGH, and enabling the CP and the PCLK of 2910-1 and 29818s, respectively.

So far, the discussion has been centered around performing diagnostics in a unit. The same Diagnostics Controller can be used to perform diagnostics in all the four units, one at a time, as shown in Figure 26. One unit can be chosen for diagnostics, while the other three units are placed in the wait state. FIFO buffers can be diagnosed by sending data from one unit, receiving the same data from another unit, and comparing it with the expected data. Since diagnostics is done only within a subprocessing unit, the capture bus cannot be diagnosed. However, the EDC

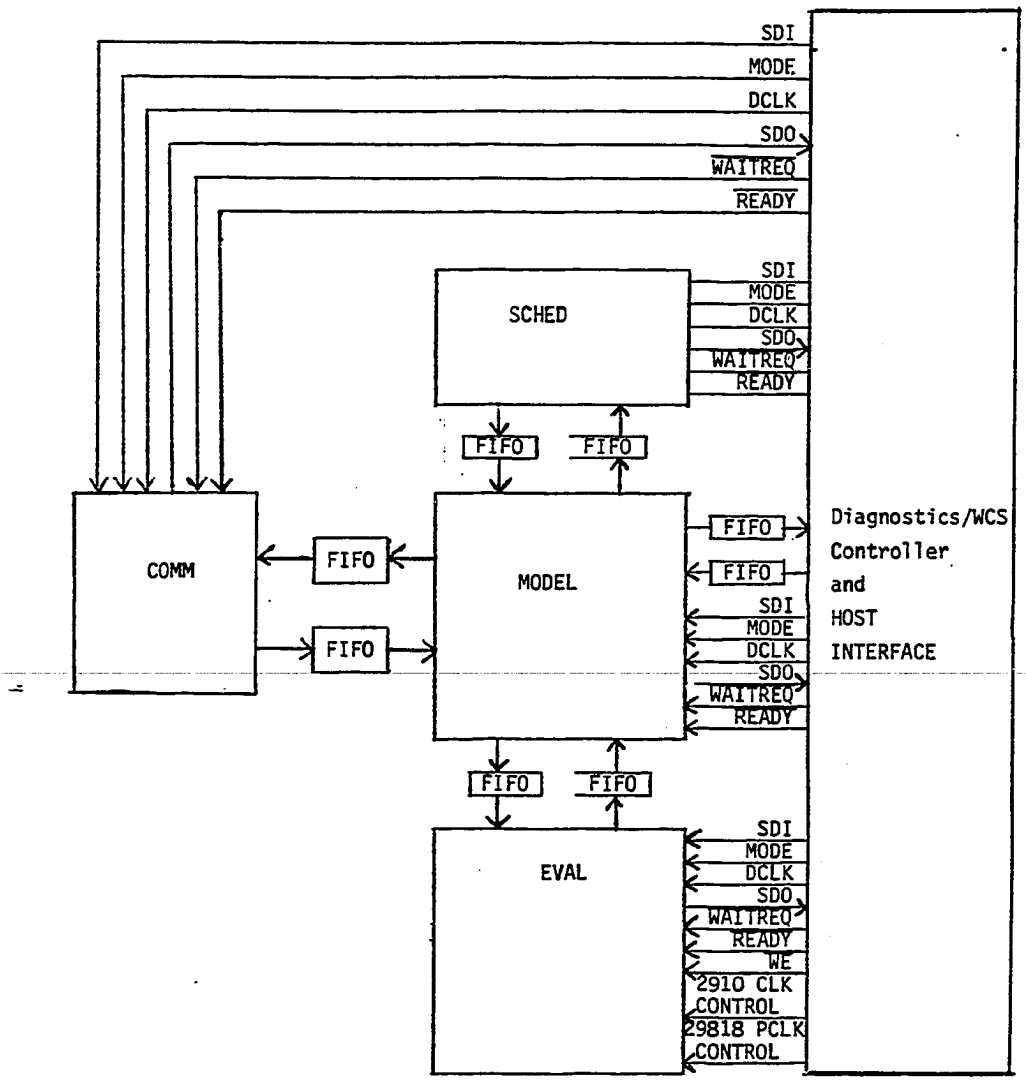


Figure 26. Connections to perform Diagnostics and write into WCS

capability provided for the bus is sufficient to detect any hardware failure.

Finally, a question could be raised for providing diagnostics within the units by expanding the control store to accommodate the diagnostic words. If this method is implemented, the Diagnostics Controller is not required. However, this method has several disadvantages. It does not provide enough support for the two key parameters in diagnostics -- controllability and observability. As a result of an error, there may be an illegal jump, and the address of the next microinstruction is unpredictable. There may be components in the unit that need to be tested with input data besides the microcode. This method provides diagnostic data only for the microinstruction. Because of such obvious problems, this method has not been used.

In the method proposed for diagnostics, the four units are tested by the Diagnostics Controller upon receiving a command from the Host computer. However, if there is hardware-related failure in the Controller, no such direct methods of testing are available. Testing must be done externally if errors are detected. The Controller and the interface for the Host must be designed with diagnostics hardware, and inputs and outputs for diagnostic purposes must be available on the board. These inputs and outputs can then be used to test this unit in a simple, systematic and quick manner.

CONCLUSION

In this dissertation, an architecture of a hardware logic simulator has been presented. Event driven simulation with arbitrary delays and signal values for both simple and functional devices have been described. High speed is achieved through several ways. First, the network is partitioned into subnetworks, and all sub PUs simulate their subnetworks concurrently. Second, the PU is programmed for certain characteristics of the network. This results in an increase in throughput due to a more efficient use of system resources. Third, high speed within the sub PU is achieved by assigning different portions of the simulation algorithm to specialized units in a microprogrammed environment, restricting communication among units through FIFO buffers, and by simulating devices in a pipelined fashion. Fourth, the communication of external data within sub PUs is done in parallel with the simulation of devices. Fifth, by using two buses, intermediate simulation results could be sent to the host computer in parallel with the simulation of devices through the capture bus.

Commercially available inexpensive microprogrammable microprocessors and other standard hardware are used to design the PU. Any future technological developments can be accommodated by a change in microcode.

The highest speed possible, if 256 sub PUs are used, is 128 million simple evaluations/second. Software-based simulators have an average speed of 20,000 evaluations/second. The highest speed, among software

simulators, of 90,200 simple evaluations/second has been reportedly achieved by a CDC Cyber 176 through vector processing.

As technology grows, the CAD tools of today become obsolete. The simulator presented has been developed with the future in mind. With 256 sub PUs, the PU is capable of simulating 16.77 million gates. According to Moore's Law, there will be 10 million components/chip by 1990. More sub PUs can be added to increase the capacity of the PU by modifying the firmware in COMM of each sub PU to accommodate this change. There is a limit to adding sub PUs after which the throughput decreases drastically. Beyond this limit, the capture bus must be made wider, and the hardware and firmware of COMM will need to be changed. Other units are not affected. The traffic on the standard bus during simulation is controlled by the demands of the user.

Methods have been developed so that all similar devices (see Table 1 for different categories for devices) are simulated in a constant time. During a time frame, the communication of external data via the capture bus is done in a constant time regardless of the amount of data. Scheduling of a similar device is done in a constant time. As the number of devices to be scheduled in a time frame increases, the scheduling time increases linearly. The same is true for the other units. Therefore, as the number of devices is increased in the subnetwork, the time required to simulate increases only linearly.

The time wheel has enough elements such that a device with any delay (within a range of 256 units of time) can be scheduled. The memory space for the time wheel is divided into a fixed space and a free space. A

small amount of fixed space is assigned to each element. If more memory is required by the element, blocks from free space are allocated. This is a natural way of organizing memory for elements whose demands are changing dynamically.

A distinction is made between real delays and simulation delays to compress the range of delays in the network. This ensures that active devices will be found for each time frame, thus improving the performance of the PU. Digital devices have been divided into various categories to optimize the system resources.

Methods have been developed so that communication on the capture bus is done at the highest speed possible. There is no communication among sub PUs, and only external data are sent and received for the current frame. No other kind of information is sent, except that required to start the transfer of data, and the overhead incurred if errors are detected.

The reliability of the PU is increased by performing diagnostics at the sub PU level in a computer aided environment and by providing EDC capability for buses and memories. EDC operations are made transparent by performing them in parallel with other operations. There are provisions for loading the WCS of EVAL in each sub PU, with evaluation programs of new devices, through the host computer.

Memory is used efficiently by packing together the data. The model of a device is packed together in MODEL. All data of fanout devices in COMM are packed together, and the data of all devices for the same time frame are packed together in SCHED. Transfer of data, through FIFO buf-

fers, is packed together for each device. Besides saving memory, this method also reduces memory accesses, at the cost of processor time required to manipulate data in order to separate different information packed together.

AM 29116 microprocessor is used because of its excellent field insertion/extraction, bit manipulation power, and high speed. These are the kinds of operations most required during simulation. Also, during simulation, a lot of data are being moved among the processor, memory, and FIFOs. The specialized design of each unit could be optimized for data movement. A lot of data are packed together and stored in memory. In order to retrieve a portion of this data, its address must be first formed. Multiplication and division operations are useful in forming the address. Since AM 29116 does not provide these two operations, it can be combined with AM 29516 (16x16 bit parallel multiplier) which performs 16x16 bit multiplication within 65 ns as the worst case.

For a long time, breadboards and software simulators have been used for design verification. As the complexity of ICs increases, these methods become time consuming. Only recently has research begun to manufacture hardware accelerators for simulation. For future work, the whole area of hardware simulators is open. A comprehensive study on the characteristics of networks needs to be done. Usually, networks have devices from one technology; that is, TTL, CMOS, or ECL etc. The network with devices from different technologies would have very different characteristics. It is important to know how devices in the network are arranged typically, relationship between the length and breadth of the

network, and fanouts of devices. A study on the percentage of source devices with external fanouts produced through partitioning, and how the percentage changes as the number of devices in the network changes, is required. Also, a study on the percentage of activity in the network, and how this activity changes as the number of devices in the network changes and as simulation progresses, is required. In a study reported by Y. M. Levendel et al. (10), an average of 2% activity per time frame was observed during the simulation of a 4000 gate network. The key to high speed simulation lies in partitioning the network in an optimal manner. An algorithm for partitioning needs to be designed.

Design verification has been discussed for the architecture presented. A study could be done to extend it to fault simulation. The simulator can be designed, built and tested by an industry or a university which has a lot of resources, both in terms of engineers and equipment. Each unit in the sub PU should be designed to execute operations on a simple device within 2 us; that is, the time required by EVAL to evaluate a simple device. For functional devices, each unit will take a proportionally longer time. Each unit can be built and tested independently. In order to test the capture bus, copies of COMM must be made. For the HOST INTERFACE, the hardware required to interface to the Host Computer would depend on the standard bus used. Depending on the needs of the installation, identical copies of sub PUs should be made and the sub PU tested in the normal, WCS, and diagnostic mode.

AMD has announced its new 32 bit family of microprogrammable chips which will be out in the market by August 1985. The simulator can be built with the 16 bit chips described in the dissertation, or the new 32 bit family can be used.

BIBLIOGRAPHY

1. R. M. Burger, R. K. Cagin III, W. C. Holton, and L. W. Sunmey. "The impact of ICs on computer technology." Computer, 17, No. 10 (October 1984):88-95.

2. G. Pfister. "The Yorktown simulation engine: Introduction." Proc. 19th Design Automation Conf. 19 (1982):51-54.

3. E. W. Thompson. "Simulation - A tool in an integrated testing environment." Pp. 228-233 in Tutorials on VLSI support technologies: CAD, testing, and packaging. New York, New York: IEEE Computer Society Press, Spring 1982.

4. M. Feuer. "VLSI design automation: An introduction." Proc. of the IEEE, 71, No. 1 (January 1983):5-9.

5. M. Abramovici, Y. M. Levendel, and P. R. Menon. "A logic simulation machine." Proc. 19th Design Automation Conf. 19 (1982):65-73.

6. H. E. Krohn. "Vector coding techniques for high speed digital simulation." Electronic Engineering, 55, No. 642 (September 1980):35-66.

7. M. M. Denneau. "The Yorktown simulation engine." Proc. 19th Design Automation Conf. 19 (1982):55-59.

8. E. Kronstadt and G. Pfister. "Software support for the Yorktown simulation engine." Proc. 19th Design Automation Conf. 19 (1982):60-64.

9. Z. Barzilai, L. Huisman, G. Silberman, D. Tang, and L. Woo. "Simulating pass transistor circuit using logic simulation machines." Proc. 20th Design Automation Conf. 20 (1983):157-163.

10. Y. M. Levendel, P. R. Menon, and S. H. Patel. "Special purpose computer for logic simulation using distributed processing." The Bell System Technical Journal, 61, No. 10 (December 1983):2873-2909.

11. N. Koike, K. Ohmori, H. Kondo, and T. Sasaki. "A high speed logic simulation machine." Digest of Papers COMPCON, March 1983, 446-451.

12. T. Sasaki, N. Koike, K. Ohmori, and K. Tomita. "HAL: A block level hardware logic simulator." Proc. 20th Design Automation Conf. 20 (1983):150-156.

13. R. L. Barto and S. A. Szygenda. "A computer architecture for digital logic simulation." Electronic Engineering, 55, No. 642 (September 1980):35-66.

14. R. L. Barto, S. A. Szygenda, and E. W. Thompson. "Architecture for a hardware simuator." Proc. IEEE Conference on Circuits and Computers, 1980, 891-893.

15. A. V. Pohm and O. P. Agrawal. High speed memory systems. Reston, Virginia: Reston Publishing Co., Inc., 1983.

16. Advanced Micro Devices. Bipolar/MOS Memories data book. Advanced Micro Devices, Sunnyvale, California, 1984.

17. Hitachi. IC Memories data book. San Jose, California, Hitachi, 1984.

18. Advanced Micro Devices. Bipolar Microprocessor logic and interface data book. Advanced Micro Devices, Sunnyvale, California, 1983.

19. N. D. Phillips and J. G. Tellier. "Efficient event manipulation, a key to logic scale simulation." Proc. IEEE 1978 Semiconductor Test Conference, 1978, 266-273.

20. J. G. Vaucher and P. Duval. "A comparison of simulation event list algorithms." Comm. ACM, 18-4 (April 1975):223-230.

21. Texas Instruments. The TTL data book. Vol. 3. Texas Instruments, Dallas, Texas, 1984.

22. TRW Electronic Components Group. VLSI data book. TRW Electronic Components Group, La Jolla, California, 1984.

23. Monolithic Memories. Systems Design Handbook. Monolithic Memories, Santa Clara, California, 1983.

ACKNOWLEDGEMENT

I would like to thank Dr. Arthur V. Pohm and Dr. Terry A. Smay. Their valuable help, constant encouragement, and personal kindness are deeply appreciated and gratefully acknowledged.

I am thankful to Dr. R. Brown, Dr. D. Grosvenor, and Dr. S. Marley for serving on my committee.

I would like to express my deep appreciation to Dr. J. Kopplin, Chairman of Electrical and Computer Engineering, for providing the financial support during my graduate program.

Finally, I wish to thank my parents for their encouragement and support.

APPENDIX A: PROPOSED IMPLEMENTATION OF THE EVALUATION UNIT

EVAL has been designed with standard, commercially available, chips. The data sheets of all the chips used in the design have been included in Appendix B, and the data books used are listed in the bibliography (16, 17, 18, 21, 22, 23). The implementation has been divided into three sections. Section 1 describes the hardware of EVAL. Section 2 describes the microinstruction, and Section 3 describes the microprograms of the device to be evaluated.

Hardware

EVAL has three modes -- Normal Mode (NM), Diagnostics Mode (DM), and WCS Mode (WCSM). During NM, the unit is evaluating devices. During DM, the Diagnostics Controller is used to pin-point hardware related failures, and during WCSM the WCS Controller is used to load the control store with device microprograms from the host computer.

Normal mode

Figure 27 shows the block diagram of EVAL in NM. The microprogram controller (2910-1) produces the address of the next microinstruction while the current microinstruction in the pipeline register is being executed. Data from MODEL are read from the Read FIFO, evaluated, and the results are written into the Write FIFO to be used by MODEL. The external RAM is used when the device data and intermediate results require

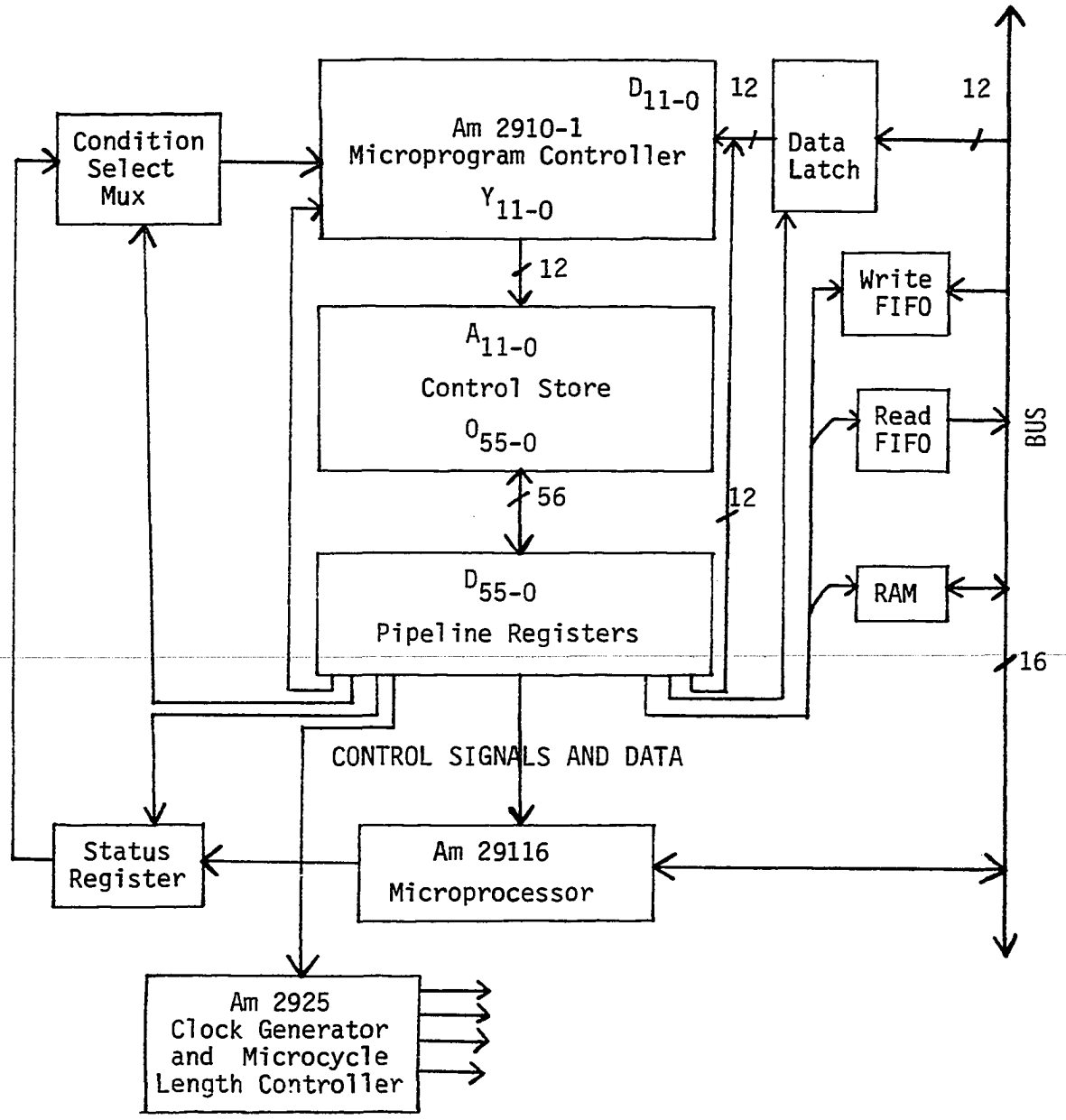


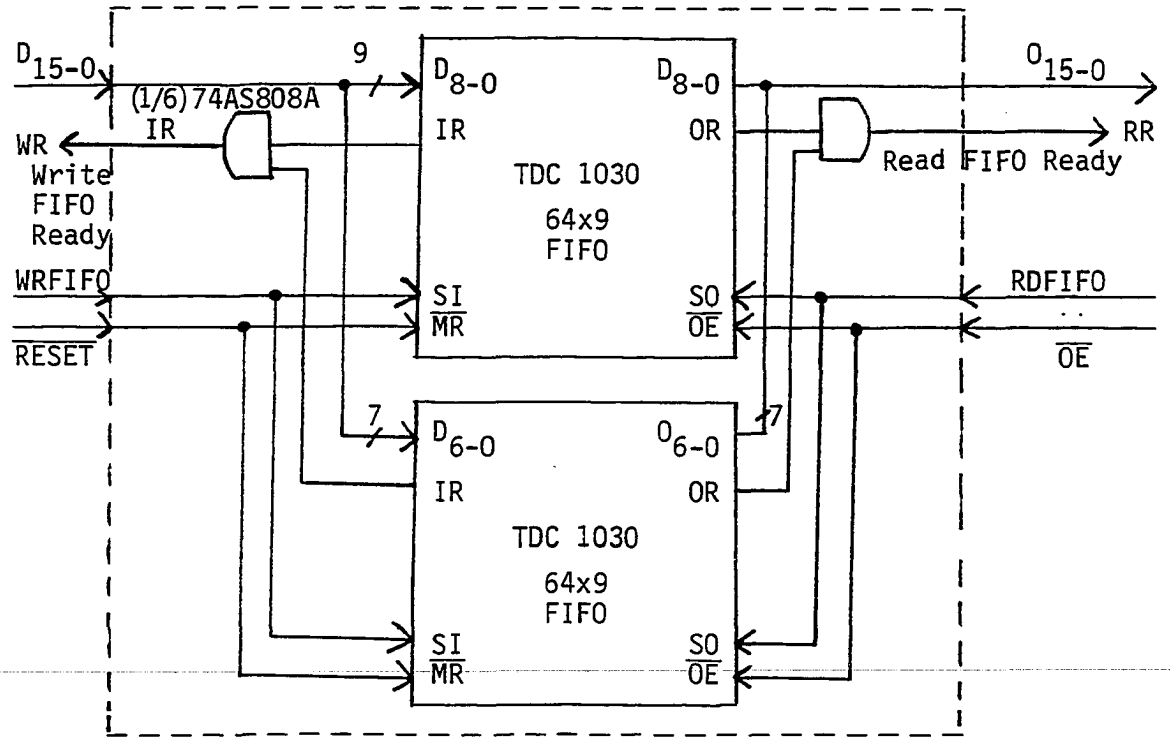
Figure 27. Block diagram of EVAL

more memory than is present in the internal RAM of the microprocessor (AM 29116).

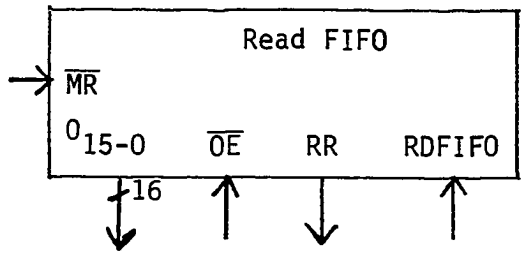
In describing the hardware of this unit, first the hardware required to design each block in Figure 27 is described. Finally, these blocks are connected together to form the EVAL. The chips required to design a 64x16 bit high speed FIFO are shown in Figure 28, and the design for the 256x16 bit RAM is shown in Figure 29. The function table for reading the RAM, writing into the RAM, and loading the address into address counters is shown in Table 7.

The total microaddress space available when AM 2910-1 is used is 4K. The amount of PROM and RAM required depends on the application. In this design, 2K of PROM and 2K of RAM is used arbitrarily, but the access time of the control store must not be greater than 35 ns. The evaluation programs of heavily used simple and functional devices are built into the PROM. Those functional devices that have not been used before, or are seldom used, have their evaluation programs loaded by the host. The design of the control store is shown in Figure 30. Seven AM 27S191A PROMs and twenty-eight AM 2149 RAMs are used to design a 4Kx56 bit control store. PROMs occupy the lower 2K of microaddress space because the reset routine starts at zero, and the WCS occupies the upper 2K.

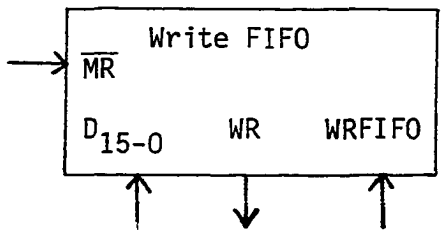
The design of EVAL is shown in Figure 31. In designing this unit, the primary aim is to build a very high speed processor which is also very flexible to microprogram. The Clock Generator and Microcycle Controller (AM 2925), connected to a 46 MHz oscillator, provides cycle time of 21.74 ns at F_0. Clock outputs with cycle times of 108.7 ns, 130 ns,



(a) Design of FIFO



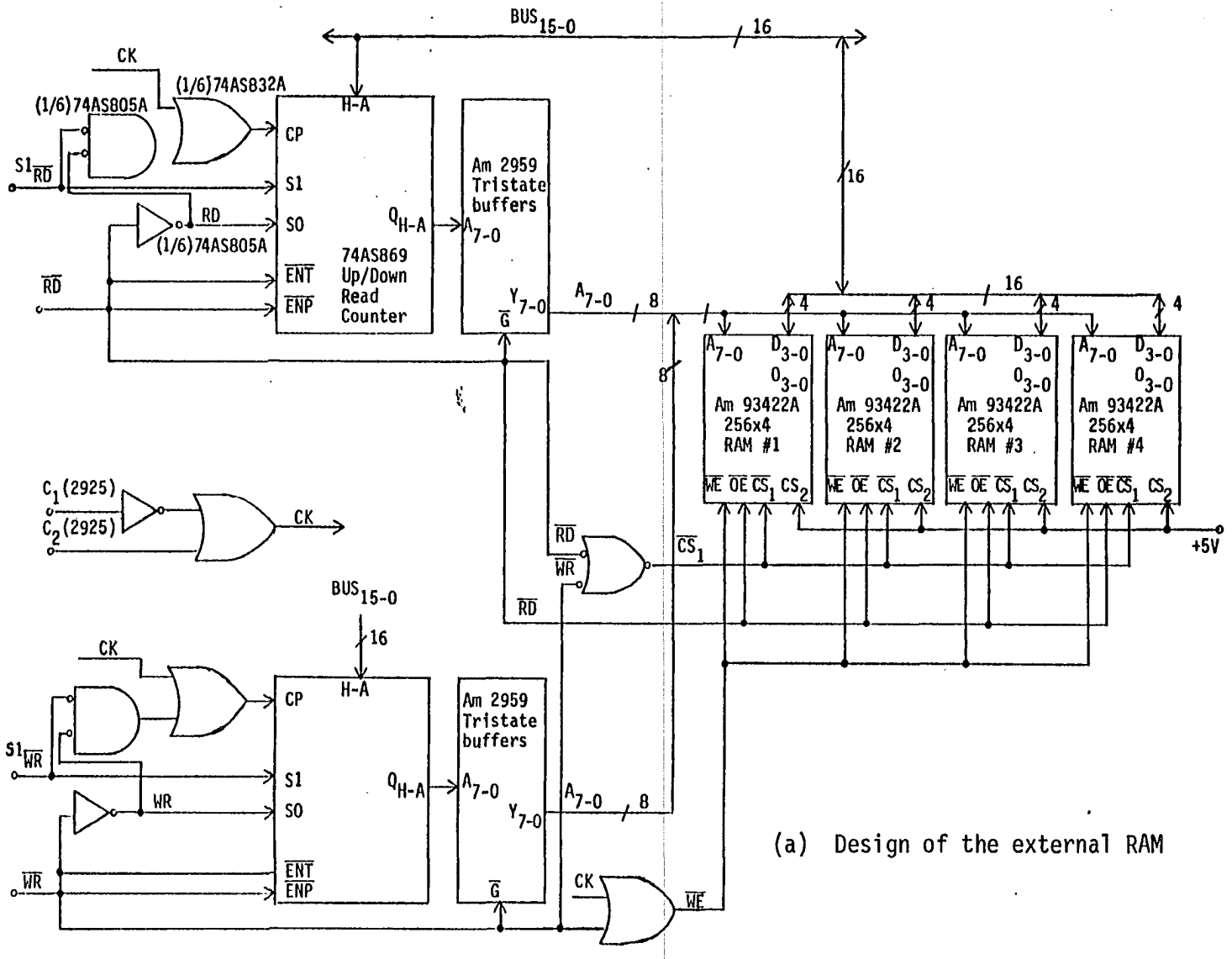
(b) Symbol for Read FIFO



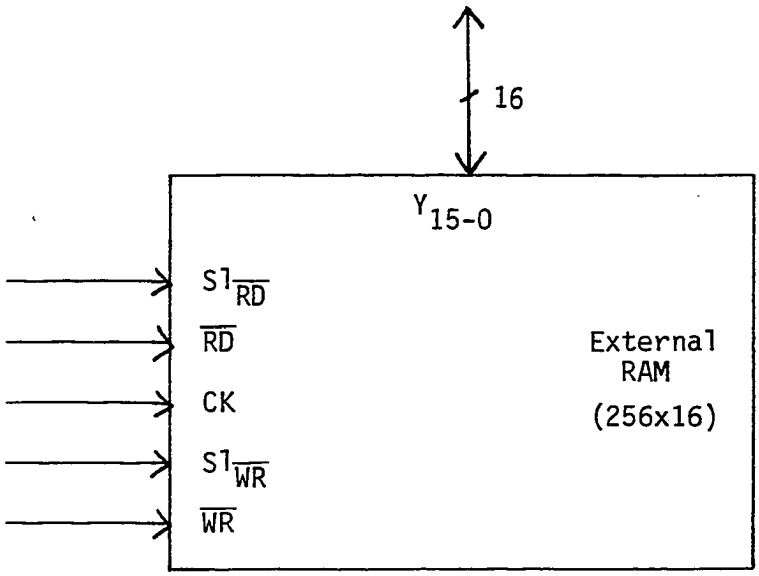
(c) Symbol for Write FIFO

Figure 28. FIFO

Figure 29. 256 x 16 bit external RAM



(a) Design of the external RAM



(b) Symbol for the external RAM

Table 7. Function table for external Ram

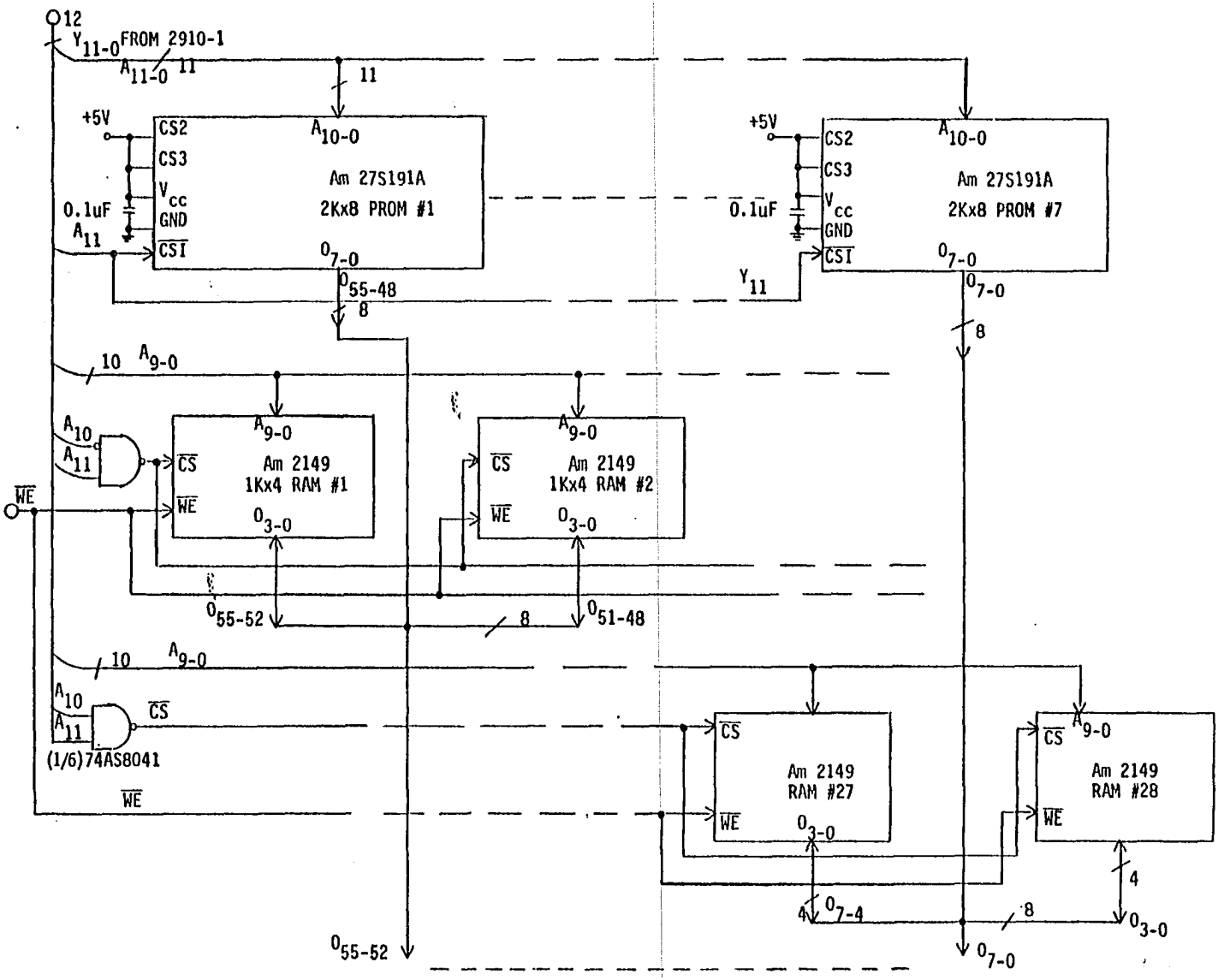
(a) Function table for reading the external RAM

RD	$S1_{RD}$	Function
0	0	Read from RAM and decrement address in Read Counter
0	1	Read from RAM and increment address in Read Counter
1	0	No operation. Y_{15-0} at high impedance
1	1	Load address in Read Counter from Y_{7-0}. Y_{15-0} at high impedance

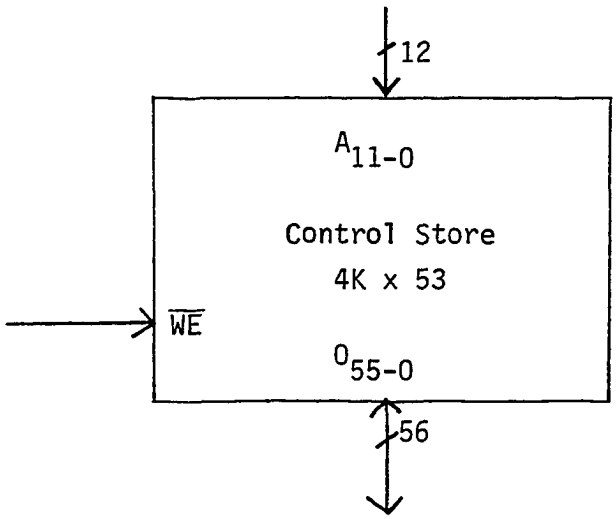
(b) Function table for writing into external RAM

WR	$S1_{WR}$	Function (Y_{15-0} at high impedance)
0	0	Write into RAM and decrement address in Write Counter
0	1	Write into RAM and increment address in Write Counter
1	0	No operation
1	1	Load address into Write Counter from Y_{15-8}

Figure 30. Control store

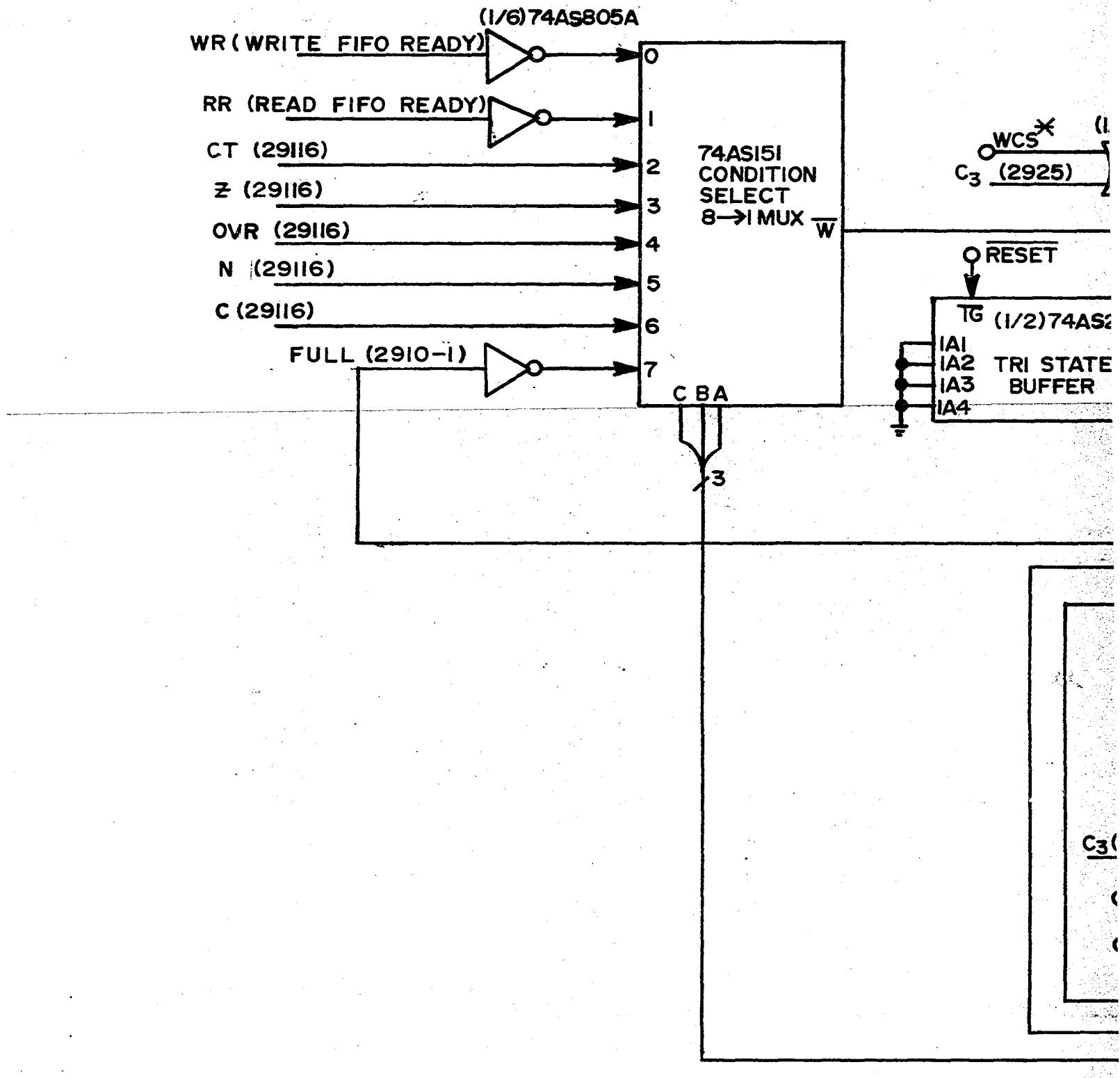

12
Y_{11-0} FROM 2910-1
A_{11-0} 11
11
+5V
CS2
CS3
V_{cc}
GND
$\overline{CS1}$
0.1uF
A_{11}
A_{10-0}
Am 27S191A
2Kx8 PROM #1
O_{7-0}
O_{55-48}
8
Y_{11}
Am 27S191A
2Kx8 PROM #7
O_{7-0}
8
10 A_{9-0}
A_{10}
A_{11}
$\overline{CS}$
$\overline{WE}$
A_{9-0}
Am 2149
1Kx4 RAM #1
O_{3-0}
Am 2149
1Kx4 RAM #2
O_{55-52}
8
O_{51-48}
$\overline{WE}$
10 A_{9-0}
A_{10}
A_{11}
$\overline{CS}$
(1/6)74AS8041
$\overline{WE}$
Am 2149
RAM #27
Am 2149
RAM #28
4
O_{7-4}
8
O_{3-0}
O_{55-52}
O_{7-0}

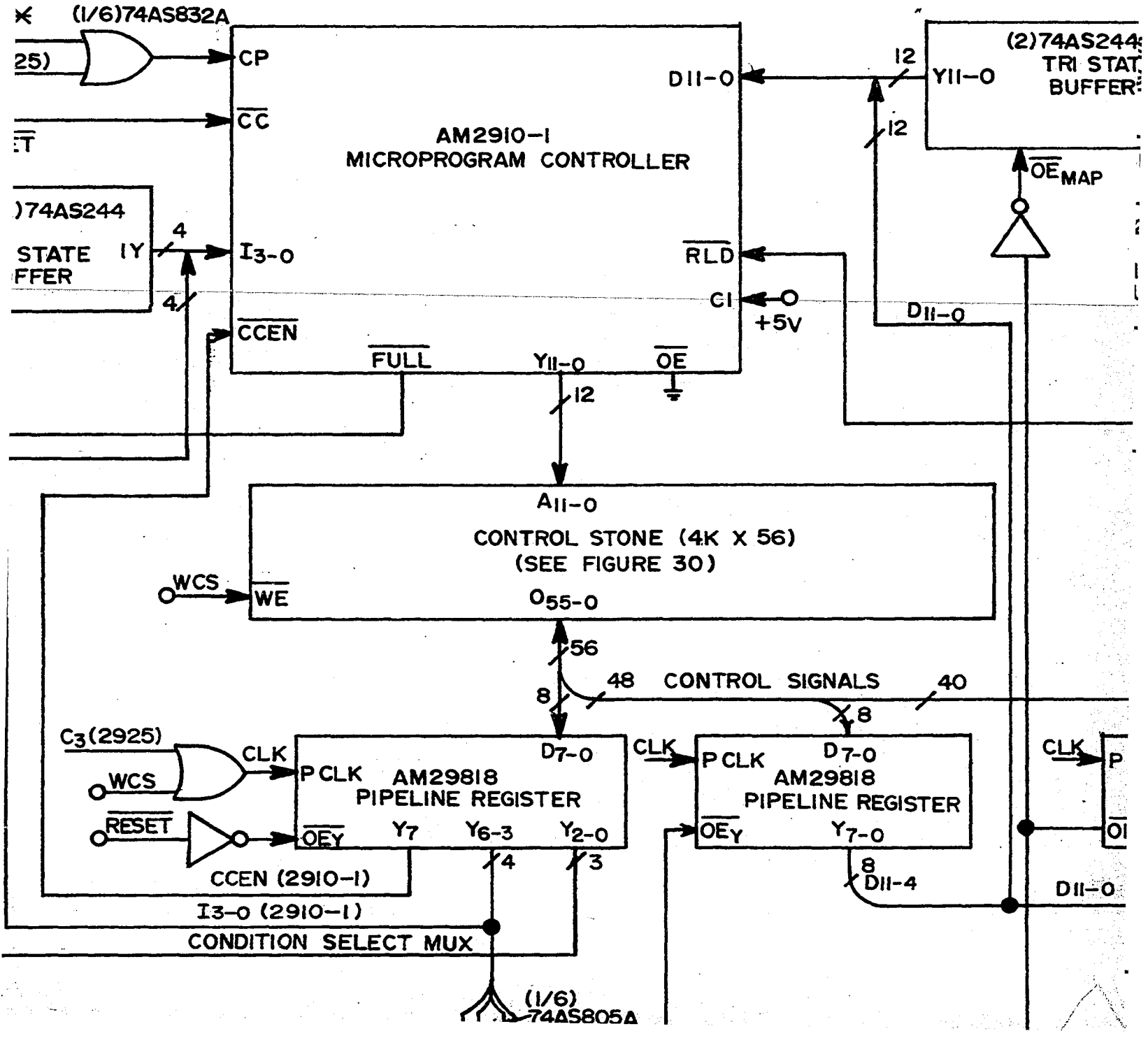
(a) Design

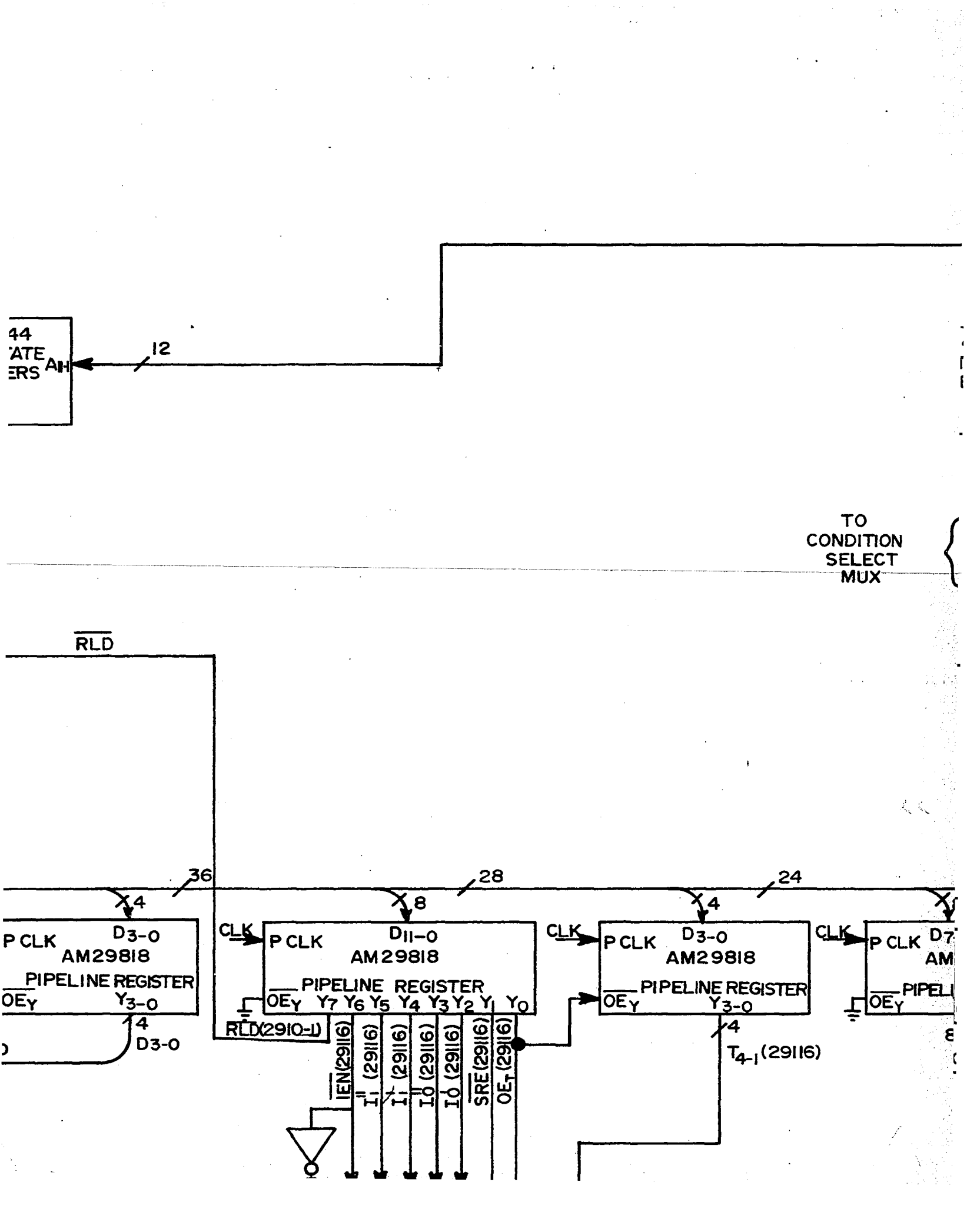


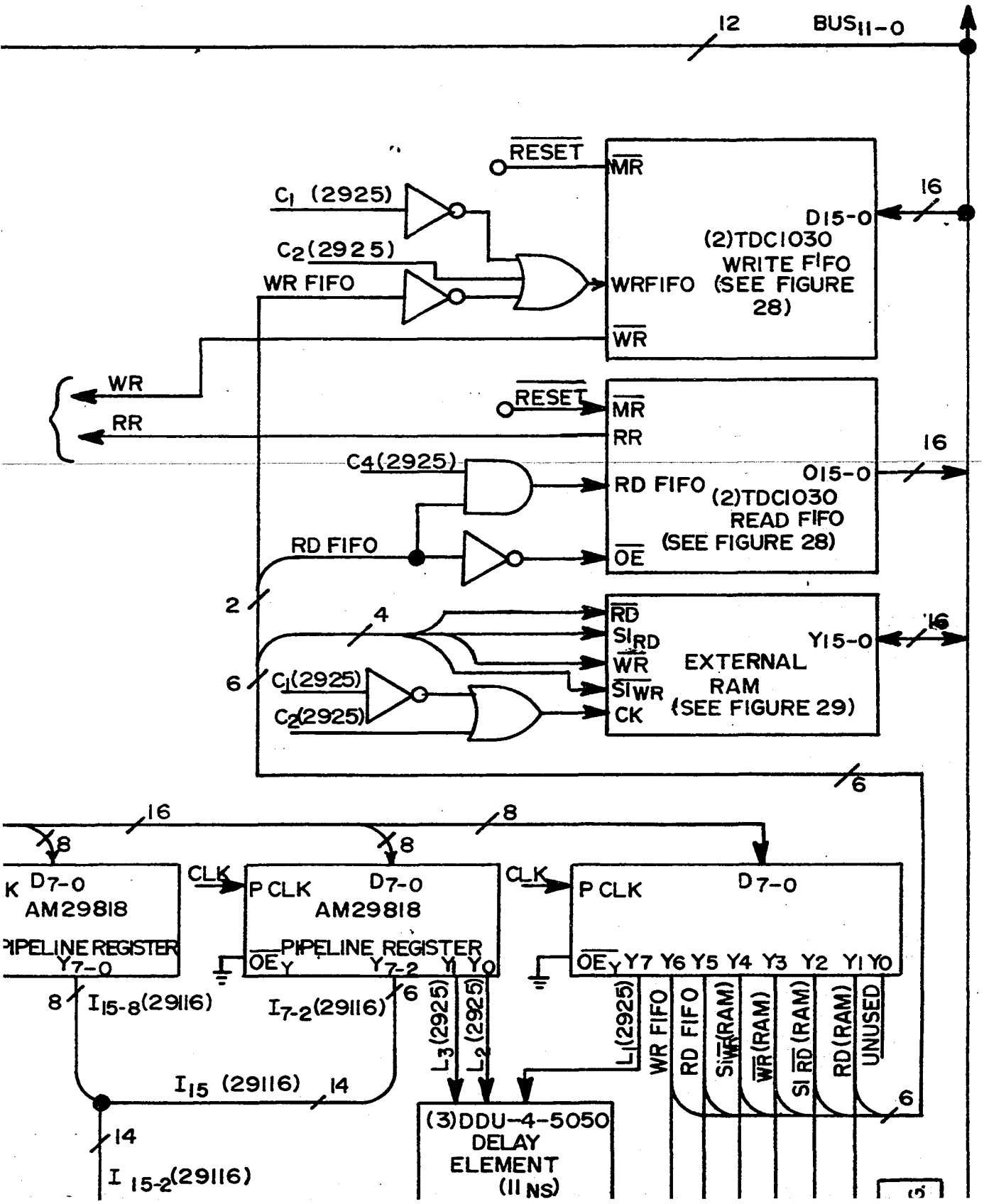
(b) Symbol for the control store

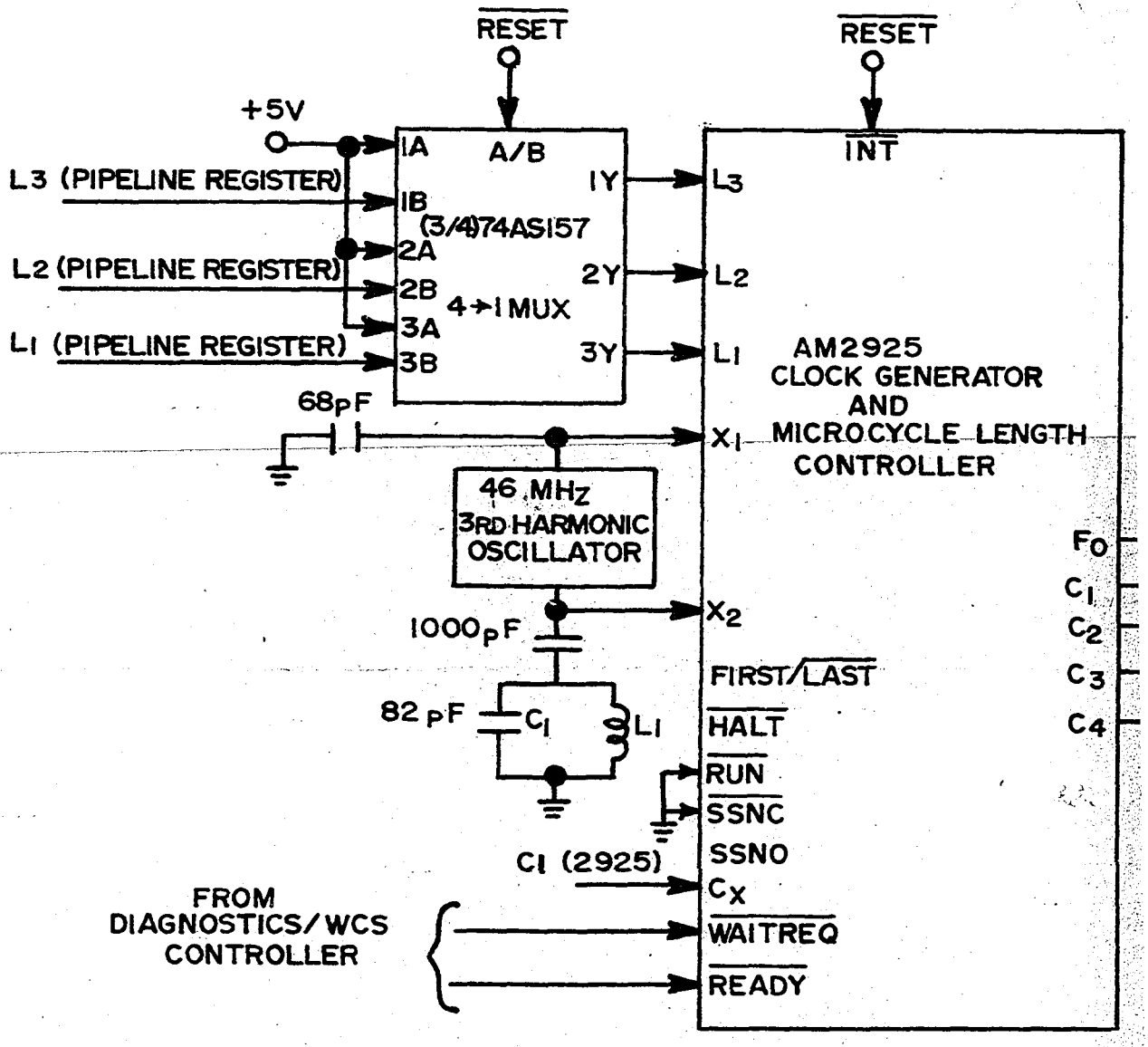
Figure 31. Design of EVAL in Normal mode

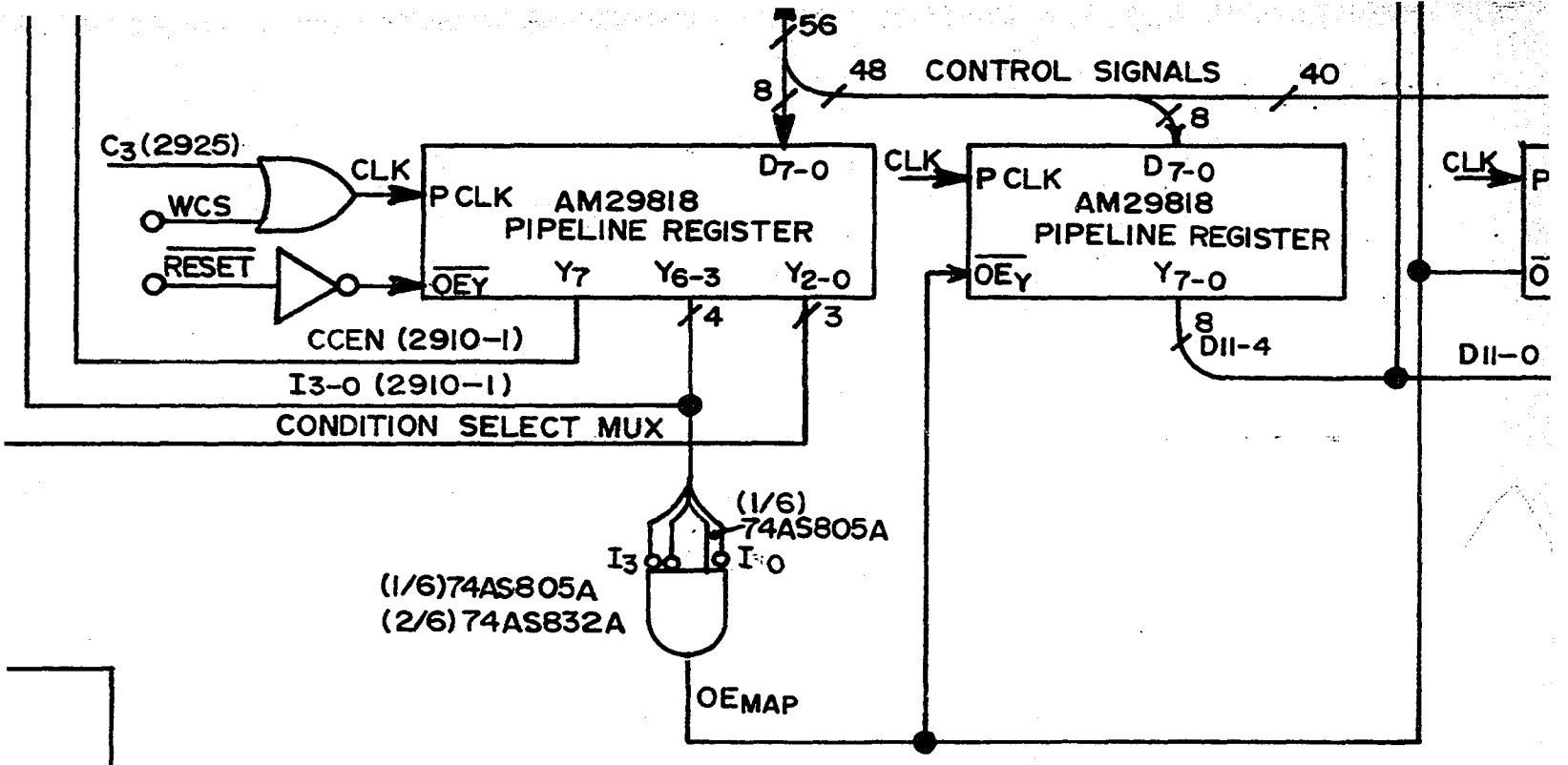

(1/6)74AS805A
WR (WRITE FIFO READY)
RR (READ FIFO READY)
CT (29116)
Z (29116)
OVR (29116)
N (29116)
C (29116)
FULL (2910-1)
0
1
2
3
4
5
6
7
74AS151
CONDITION
SELECT
8→1 MUX
W
C B A
3
WCS
C3 (2925)
RESET
TG
(1/2)74AS2
1A1
1A2
1A3
1A4
TRI STATE
BUFFER


(1/6)74AS832A
CP
CC
AM2910-1
MICROPROGRAM CONTROLLER
D11-0
Y11-0
12
OE MAP
I3-0
RLD
CI
+5V
CCEN
FULL
OE
A11-0
CONTROL STONE (4K X 56)
(SEE FIGURE 30)
WCS
WE
O55-0
56
48
CONTROL SIGNALS
40
8
C3(2925)
CLK
WCS
RESET
P CLK
AM29818
PIPELINE REGISTER
D7-0
OEY
Y7
Y6-3
Y2-0
Y7-0
CCEN (2910-1)
I3-0 (2910-1)
CONDITION SELECT MUX
D11-4
(1/6)
74AS805A


44
ATE
ERS
12
TO
CONDITION
SELECT
MUX
RLD
36
28
24
4
8
4
4
P CLK
D3-0
AM29818
PIPELINE REGISTER
OEY
Y3-0
CLK
D11-0
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
RLD(2910-1)
IEN(29116)
I1 (29116)
I0 (29116)
SRE(29116)
OET(29116)
T4-1(29116)


12
BUS11-0
RESET
MR
C1 (2925)
C2(2925)
WR FIFO
16
D15-0
(2)TDC1030
WRITE FIFO
(SEE FIGURE 28)
WRFIFO
WR
WR
RR
RESET
MR
RR
C4(2925)
RD FIFO
16
O15-0
(2)TDC1030
READ FIFO
(SEE FIGURE 28)
RD FIFO
OE
2
4
6
RD
SI RD
WR
SI WR
CK
C1(2925)
C2(2925)
EXTERNAL
RAM
(SEE FIGURE 29)
Y15-0
16
6
16
8
8
8
K D7-0
AM29818
PIPELINE REGISTER
Y7-0
CLK
P CLK
D7-0
AM29818
PIPELINE REGISTER
OEY
Y7-2
Y1
Y0
CLK
P CLK
D7-0
OEY
Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
8
I15-8(29116)
I7-2(29116)
6
L3(2925)
L2(2925)
L1(2925)
WR FIFO
RD FIFO
SI WR (RAM)
WR (RAM)
SI RD (RAM)
RD (RAM)
UNUSED
I15 (29116)
14
6
14
I 15-2(29116)
(3)DDU-4-5050
DELAY
ELEMENT
(11 NS)


RESET
RESET
+5V
1A
A/B
1Y
L3
L3 (PIPELINE REGISTER)
1B
(3/4)74AS157
2A
2Y
L2
L2 (PIPELINE REGISTER)
2B
4→1 MUX
3A
3Y
L1
AM2925
CLOCK GENERATOR
AND
MICROCYCLE LENGTH
CONTROLLER
L1 (PIPELINE REGISTER)
3B
INT
68pF
X1
46 MHz
3RD HARMONIC
OSCILLATOR
F0
C1
C2
C3
C4
X2
1000pF
FIRST/LAST
82 pF
C1
L1
HALT
RUN
SSNC
SSNO
C1 (2925)
CX
FROM
DIAGNOSTICS/WCS
CONTROLLER
WAITREQ
READY



*WCS = INPUT IS ACTIVE DURING WRITABLE CONTROL STO

R

IGTH

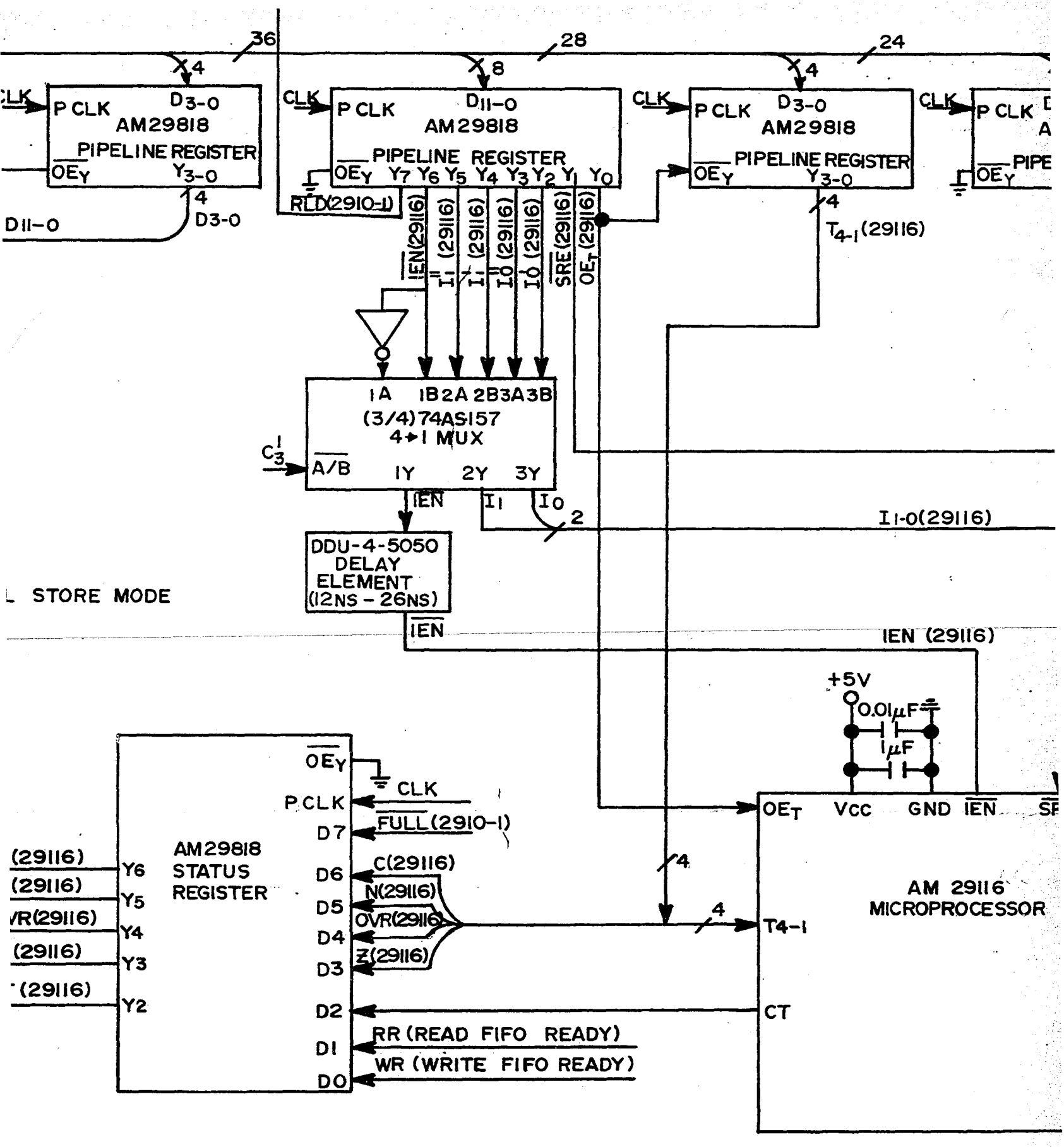
F0
C1
C2
C3
C4

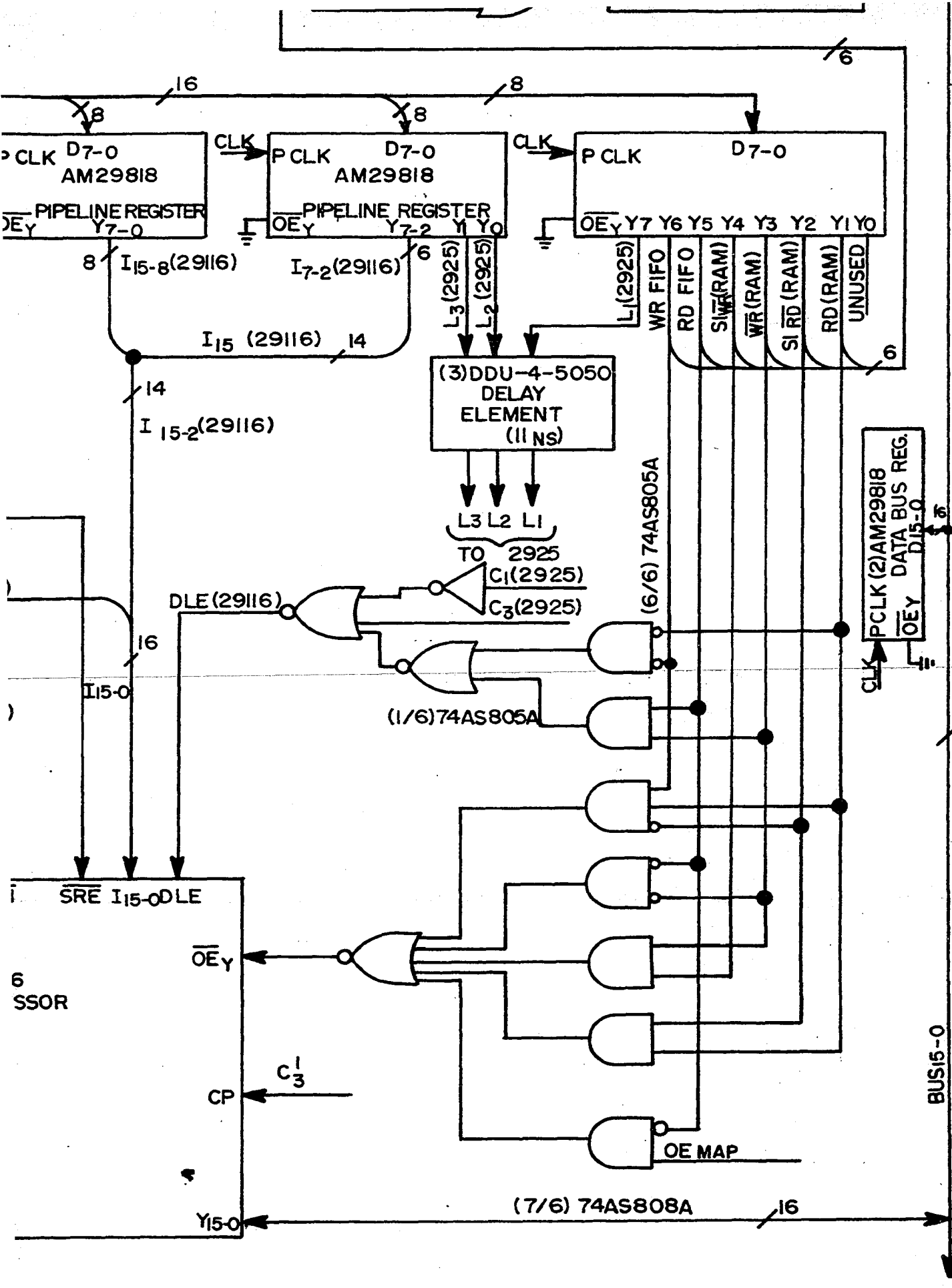
C3 (2925)

C 1/3

TO CONDITION SELECT MUX

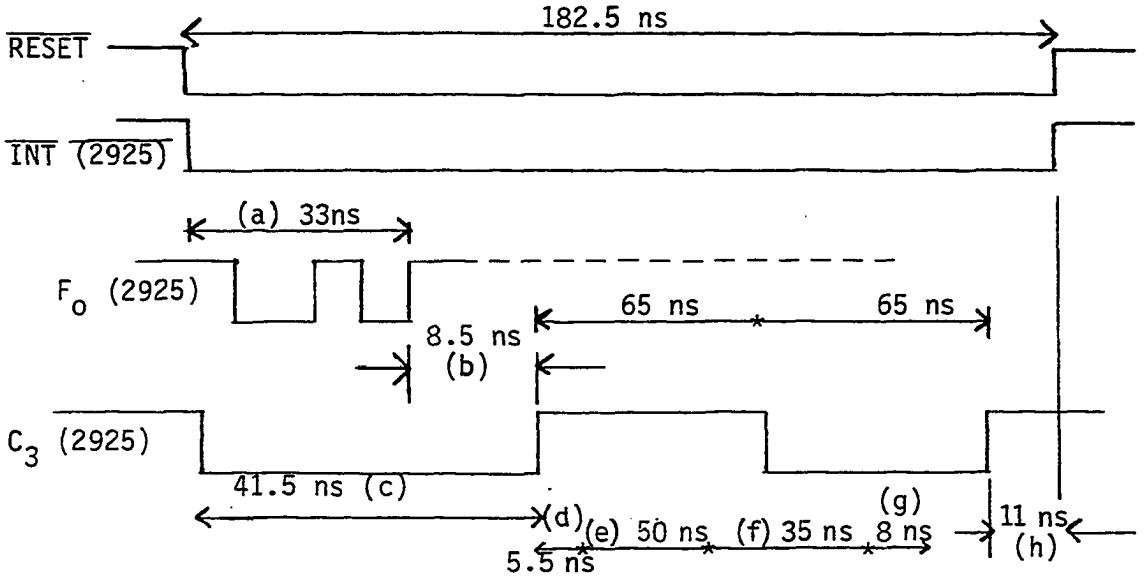
C(2911
N(2911
OVR(291
Z (2911
CT (291


36
28
24
4
8
4
P CLK
D3-0
AM29818
PIPELINE REGISTER
OEY
Y3-0
D11-0
CLK
D11-0
OEY Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
RLD(2910-1)
IEN(29116)
I1 (29116)
I1 (29116)
I0 (29116)
I0 (29116)
SRE(29116)
OET(29116)
T4-1(29116)
1A 1B 2A 2B 3A 3B
(3/4)74AS157
4→1 MUX
C3
A/B
1Y 2Y 3Y
IEN
I1
I0
I1-0(29116)
DDU-4-5050
DELAY
ELEMENT
(12NS - 26NS)
L STORE MODE
IEN (29116)
+5V
0.01μF
1μF
OEY
CLK
PCLK
FULL(2910-1)
D7
AM29818
STATUS
REGISTER
D6
D5
D4
D3
D2
D1
D0
C(29116)
N(29116)
OVR(29116)
Z(29116)
RR (READ FIFO READY)
WR (WRITE FIFO READY)
Y6
Y5
Y4
Y3
Y2
(29116)
(29116)
VR(29116)
(29116)
(29116)
OET
Vcc
GND
IEN
T4-1
CT
AM 29116
MICROPROCESSOR


16
8
8
8
CLK
D7-0
AM29818
PIPELINE REGISTER
OEY
Y7-0
P CLK
D7-0
AM29818
PIPELINE REGISTER
OEY
Y7-2
Y1
Y0
P CLK
D7-0
OEY Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
8
I15-8(29116)
I7-2(29116)
6
L3(2925)
L2(2925)
L1(2925)
WR FIFO
RD FIFO
SI WR (RAM)
WR (RAM)
SI RD (RAM)
RD (RAM)
UNUSED
6
I15 (29116)
14
14
I 15-2(29116)
(3)DDU-4-5050
DELAY
ELEMENT
(11 NS)
L3 L2 L1
TO 2925
C1(2925)
C3(2925)
DLE(29116)
(6/6) 74AS805A
PCLK (2)AM29818
DATA BUS REG.
D15-0
OEY
16
CLK
16
I15-0
(1/6)74AS805A
16
SRE I15-0 DLE
OEY
6
SSOR
C3
CP
OE MAP
BUS15-0
(7/6) 74AS808A
16
Y15-0

152 ns, and 174 ns, are used. The reason for using such an oscillator will be apparent later in this section. The pipeline registers (AM 29818s) are used to hold the current microinstruction while the next microinstruction is being produced at the control store. Upon system reset, the pipeline register, which contains the I_{3-0} (2910-1) field in the microword, is tristated and four zeros are forced into the I_{3-0} input of 2910-1. This represents a jump to zero (JZ) instruction which selects location zero in the control store as the starting of the reset routine. The System Reset resets the Read FIFO, the Write FIFO, and forces the clock outputs to free run by activating the INT input of AM 2925. The LOW pulse width of the RESET must be between 183 us and 243 us, as shown in the timing diagram of Figure 32.

The Evaluation unit contains eight status flags. The RR (Read FIFO Ready) flag is active when the Read FIFO is ready with the data. Similarly, the WR flag (Write FIFO Ready) is active when the Write FIFO is ready to accept data. These two signals are connected directly to the Condition Code Multiplexer (74AS151) because they may change in the beginning of the next cycle if the FIFOs are read or written in the previous cycle. All the eight status flags are clocked into the Status Register (Am 29818) at the end of C_3, but only the five status flags of the microprocessor (29116) are connected from the status register to the Condition Code Multiplexer. The other three flags (RR, WR, FULL) are clocked merely for diagnostic purposes because the status register is used as a diagnostic register in DM.



(a) $\overline{INT}$ to F_o set up time is 33 ns min.

(b) F_o to C_3 offset is 8.5 ns max.

(c) Within 41.5 ns, the output of that pipeline register which contains the D_{11-0} field is tristated, all zeros are available at I_{3-0} input of 2910-1 and L_3, L_2, L_1 inputs at 2925 are equal to 111. $^{3}C_3$ will be 130 ns.

(d) Max. delay from C_3 to CP is 5.5 ns.

(e) Max. delay from I_{3-0} to Y_{11-0} at 2910-1 is 50 ns. Microaddress is zero.

(f) Control Store access time is 35 ns max.

(g) Pipeline Register (29818s) setup time before C_3 is 8 ns.

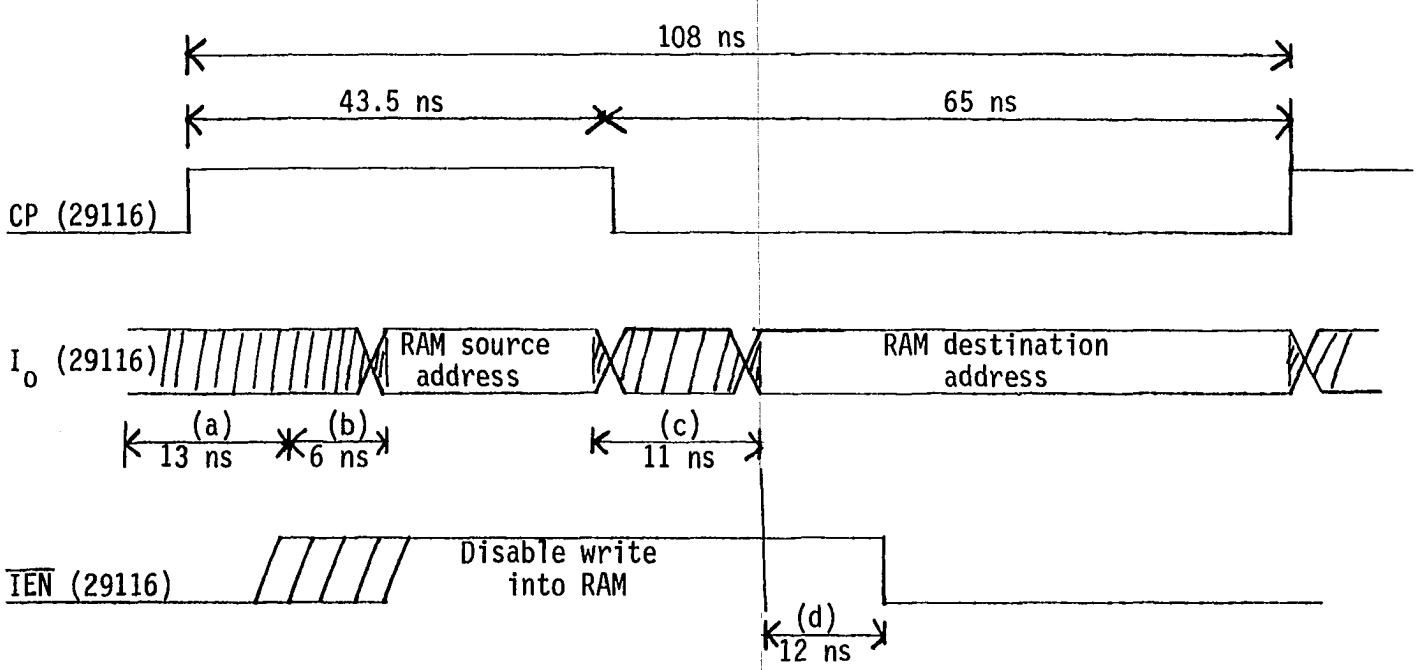
(h) Hold time for L_3, L_2, L_1 input of 2925 is 11 ns max. The cycle time of the microinstruction at location zero is forced to be 130 ns. The $\overline{RESET}$ could remain active 70 us after the next C_3. So range of LOW pulse width of $\overline{RESET}$ is between 183 ns to 243 ns.

Figure 32. Timing diagram of $\overline{RESET}$

For every device, the first word sent by MODEL is the device ID, followed by, along with other information, a 12 bit macroinstruction or microaddress which points to the starting location of devices microroutine. When the macroinstruction is to be read from the Read FIFO, the I_{3-0} (2910-1) field of the microword should contain the JMAP instruction which enables the Tristate buffers (74AS244), disables the pipeline register containing the D_{11-0} (2910-1) field in the microword, and the macroinstruction is sent to D_{11-0} inputs of the 2910-1 as the next microaddress. The MAP, PL, and VECT outputs from 2910-1 are not used, so that the speed of EVAL is increased tremendously. To accommodate for Diagnostics/WCS hardware, the clock output, C_3, from 2925 is delayed before it is the CP input of 2910-1 and the CLK input of 29818s. This delay produces certain timing problems, and a similar gate is used to produce an equal delay at the CP input of 29116. Within one clock cycle, different source and destination addresses for the microprocessor registers can be specified by multiplexing I_1'', I_i' and I_0'', I_0' fields in the microword before producing it to the I_1, I_0 inputs of 29116, as shown in the timing diagram of Figure 33. The bidirectional T_{4-1} pins of the 29116 can be used in two ways. When used as an output, the OE_T (29116) from the microword enables the T_{4-1} output on the 29116 but disables the T_{4-1} field in the microword. In this way, the four status flags from the 29116 are clocked into the status register at the end of the cycle. When used as an input, the OE_T (29116) from the microword disables the T_{4-1} output of 29116. The data from the T_{4-1} field of the microword is enabled and used to select one of twelve conditions while simultaneously

the ALU is executing an instruction. The result of the selected condition is produced at the CT output and clocked in the status register at the end of the cycle, as shown in the timing diagram of Figure 34. Since T_{4-1} is used as input, these values will be clocked in the status register instead of C, N, Z, OVR from the 29116. In this case, the SRE bit in the microword must be high, so that the internal status register of 29116 clocks the status of the ALU because it is not available to the external status register. Immediate data from the I_{15-0} fields of the microinstruction can be sent to 29116 in two cycles, as shown in Figure 35. During the first cycle, the 29116 captures the instruction and executes the captured instruction in the second cycle when the I_{15-0} (29116) field in the microword contains the data. Also, during the first half of the first cycle, the IEN must be LOW.

When writing to the peripherals, the 29116 does not hold the data after the end of the microcycle, which is a low to high transition of the clock (CP). Hence, all the data from 29116 should not only be written before this transition but should also satisfy the setup and hold times of these peripherals. Timing diagrams for writing and reading from external RAM and FIFO are shown in Figures 36, 37, 38, and 39. It is also possible to generate a microaddress in 29116 and use it as the next microaddress by sending it to 2910-1, using the JMAP instruction of 2910-1, as shown in Figure 40. While 29116 is doing internal operations, it is possible to simultaneously transfer data between the external RAM and the FIFOs. The timing diagrams to perform such operations are shown in Figures 41 and 42. The output, Y_{15-0}, pins of 29116 are enabled only



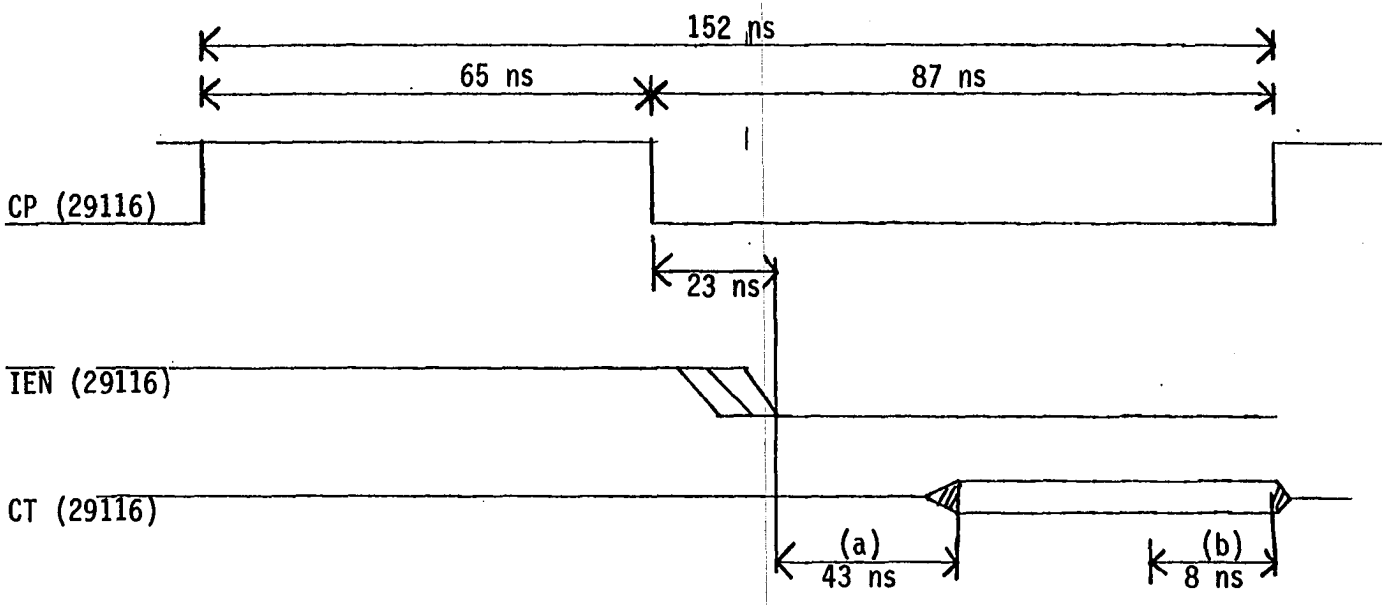
(a) Clock to output delay at pipeline register of 13 ns max.

(b) Input to output delay of 6 ns at 74AS157 Mux. Note that the setup time of 24 ns for I_o has been satisfied.

(c) Select to output delay of 11 ns max. at 74AS157 Mux.

(d) An exact delay of 12 ns is produced to avoid accidental write into 29116s' RAM. $\overline{IEN}$ should go LOW at least 10 ns after I_o is stable. Set up time of 22 ns for $\overline{IEN}$ has been satisfied.

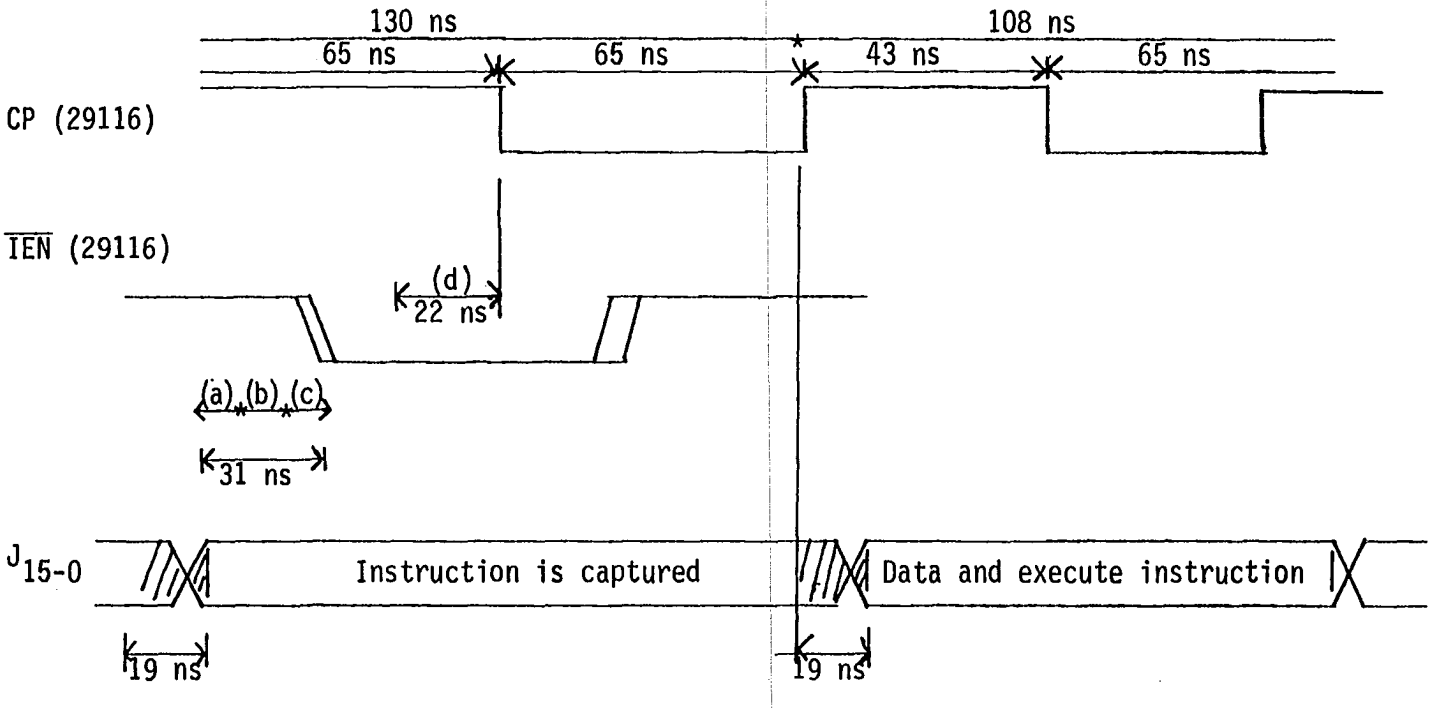
Figure 33. Timing diagram for reading and writing into 29116s' RAM within one cycle



(a) $\overline{\text{IEN}}$ to CT disable is 43 ns max.

(b) Status Register (29818) setup time is 8 ns.

Figure 34. Timing diagram for using the CT output of 29116



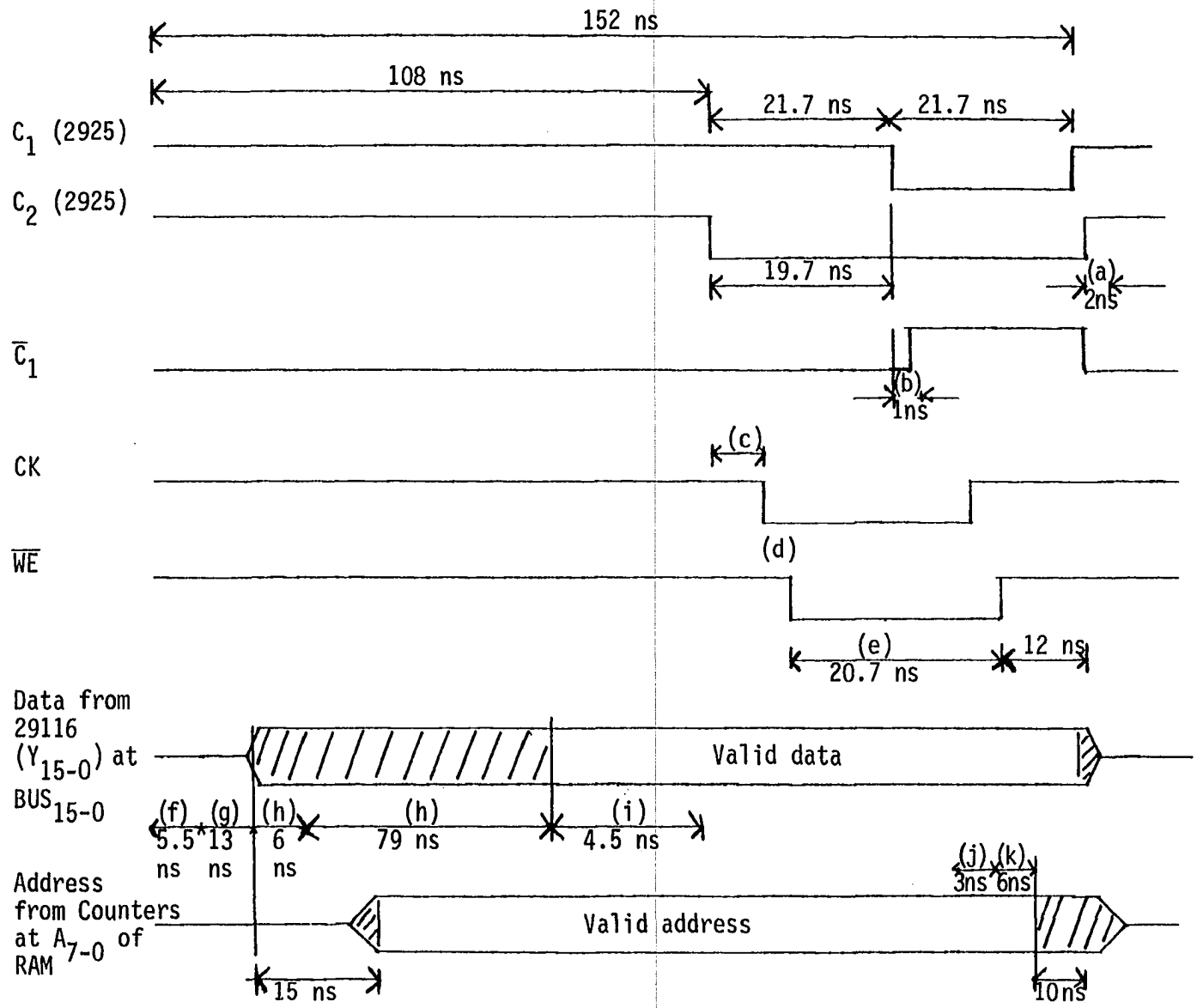
(a) Clock to output delay of 13 ns max. at pipeline register.

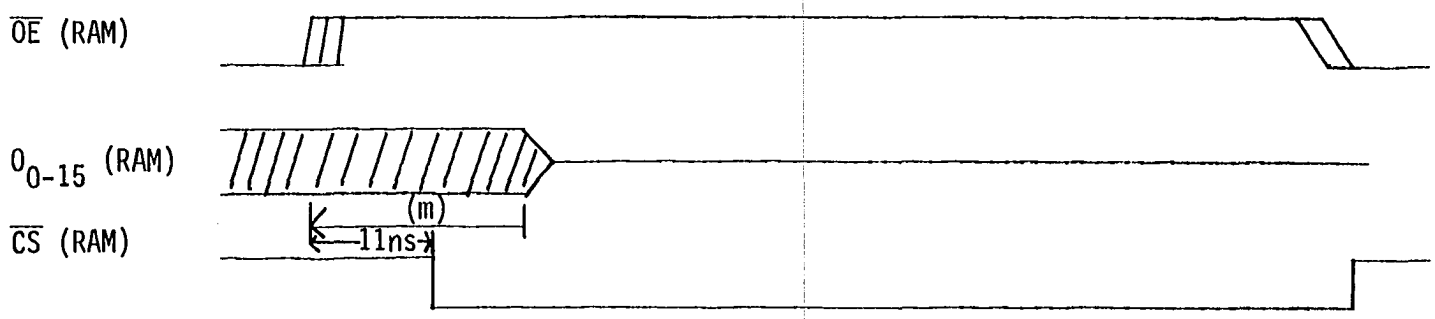
(b) Input to output delay of 6 ns at 74AS157 max.

(c) 12 ns delay at delay element.

(d) 29116 requires that $\overline{IEN}$ be LOW at least 20 ns before CP makes a transition from High to Low during the first cycle.

Figure 35. Immediate instruction cycle timing


152 ns
108 ns
21.7 ns
21.7 ns
C1 (2925)
C2 (2925)
19.7 ns
(a)
2ns
C̄1
(b)
1ns
(c)
CK
(d)
WE
(e)
20.7 ns
12 ns
Data from 29116 (Y15-0) at BUS15-0
Valid data
(f)
5.5 ns
(g)
13 ns
(h)
6 ns
(h)
79 ns
(i)
4.5 ns
(j)
3ns
(k)
6ns
Address from Counters at A7-0 of RAM
Valid address
15 ns
10ns



(a) C_1 to C_2 skew of 2 ns max.

(b) Min. delay at inverter is 1 ns.

(c) Delay at OR gate of 5.5 ns max.

(d) Delay at OR gate of 5.5 ns max.

(e) The RAM requires that the minimun $\overline{WE}$ low pulse be 20 ns. This condition will be met even in the worst case.

(f) C_3 to CLK delay of 5.5 ns max.

(g) PCLK to output delay of 13 ns.

(h) I_{15-0} to Y_{15-0} delay of 79 ns at 29116. Also, 6 ns max. delay of I_1,I_0 from input to output of Max.

(i) Note that all the constraints to set up and hold the data have been satisfied in order to load the 74AS869 counters.

(j) CK to output delay at 74AS869 counter of 3 ns min.

(k) Input to output delay at tristate buffer (2959) of 6 ns typical. Note that the address is not valid 10 ns max. before the Low to High transition of C_1.

(l) Delay to enable output at tristate buffers of 15 ns max.

(m) $\overline{OE}$ high to output disable of RAM is 30 ns max. Note that all the constraints for writing into the RAM have been satisfied.

Figure 36. Timing diagram to write into the external RAM and load counters from 29116

Figure 37. Timing diagram to read from the external RAM by 29116

(a) PCLK to output delay at pipeline register of 13 ns max.

(b) Output disable to enable delay of 15 ns max. at tristate buffer (2959).

(c) Address is not valid 15.5 ns before Low to High transition of CP.

(d) $\overline{OE}$ to output enable delay of 25 ns.

(e) 29116 requires the data to be valid at least 44 ns before the Low to High transition of CP, if it is to be used in the same cycle.

(f) Since DLE goes Low at least 17 ns before the Low to High transition of CP, therefore data are latched before it changes.

(g) O_{15-0} will be disabled 43 ns max. after the Low to High transition of CP.

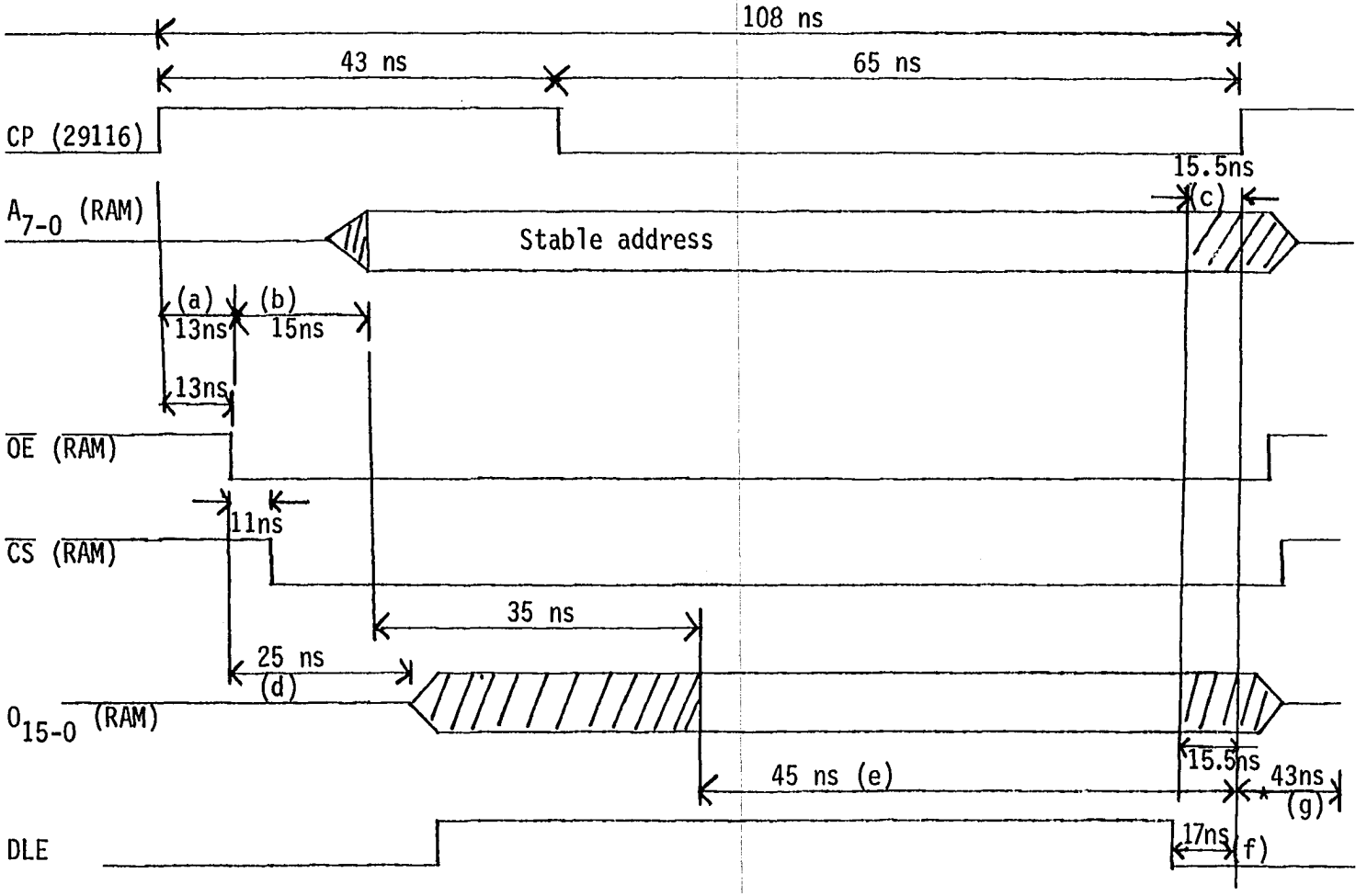

108 ns
43 ns
65 ns
CP (29116)
15.5ns
(c)
A_{7-0} (RAM)
Stable address
(a)
13ns
(b)
15ns
13ns
$\overline{OE}$ (RAM)
11ns
$\overline{CS}$ (RAM)
35 ns
25 ns
(d)
O_{15-0} (RAM)
15.5ns
45 ns (e)
43ns
(g)
17ns
f)
DLE

Figure 38. Timing diagram for reading from FIFO into 29116

(a) Delay from High to Low transition by C_4 to Low to High transition of CLK. The C_3 to C_4 skew of 11 ns max. will not contribute to worst case.

(b) Delay at pipeline register of 13 ns. max.

(c) Delay at AND gate of 5 ns max.

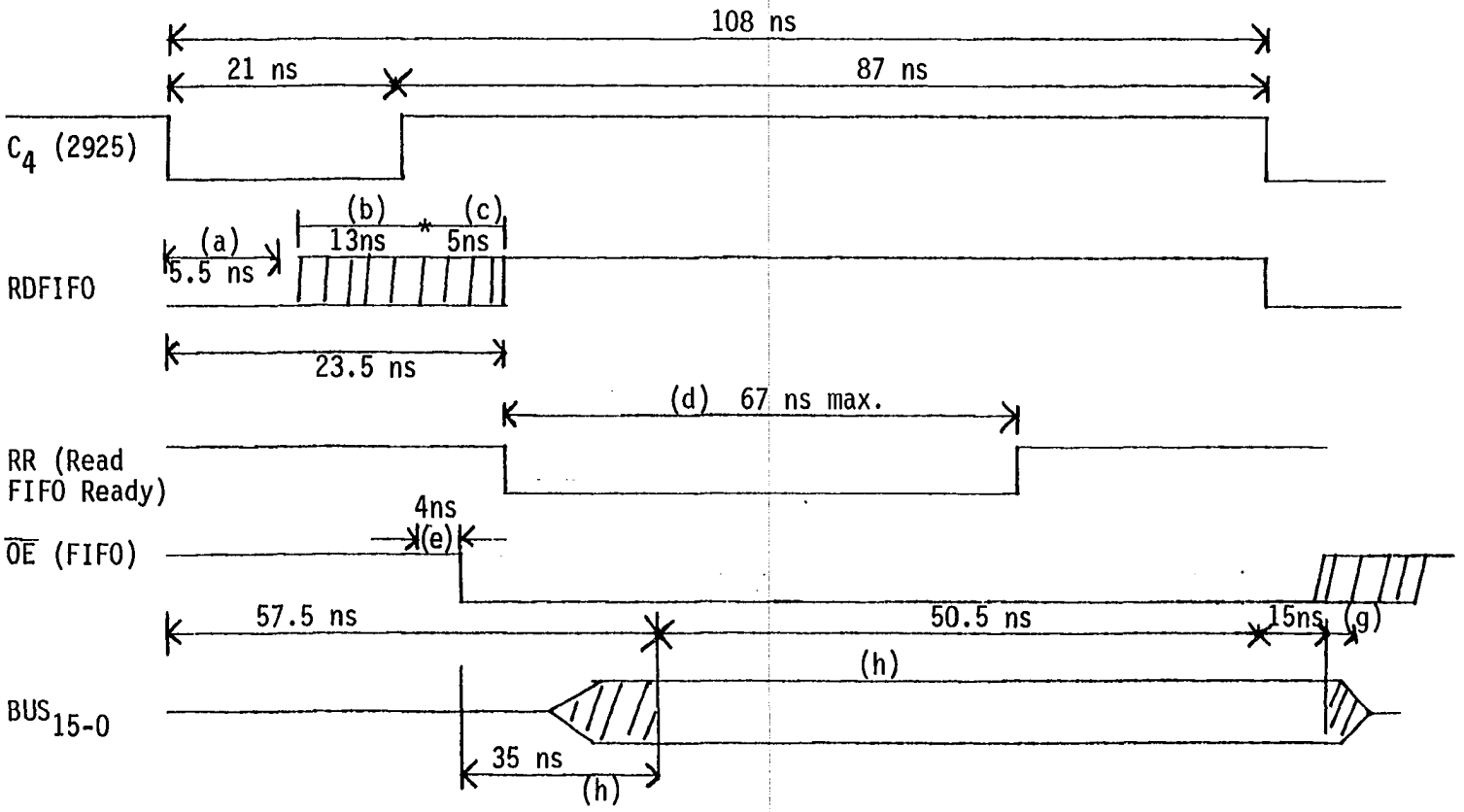
(d) RR will be active before the Read FIFO is read again.

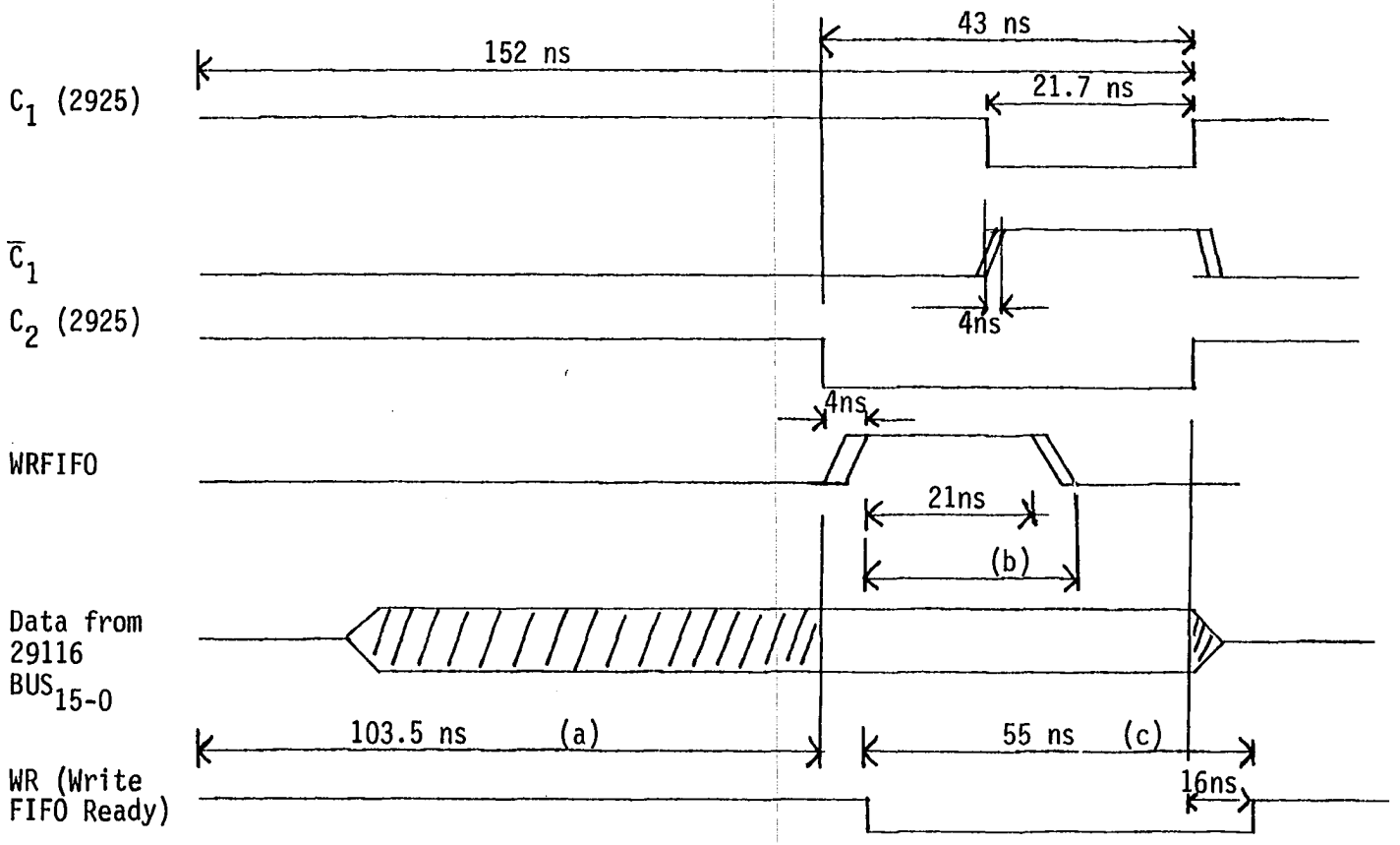
(e) Delay at OR gate of 4 ns max.

(f) Delay from $\overline{OE}$ to enabled output of 35 ns max.

(g) The data are held for 15 ns min. after RDFIFO goes Low.

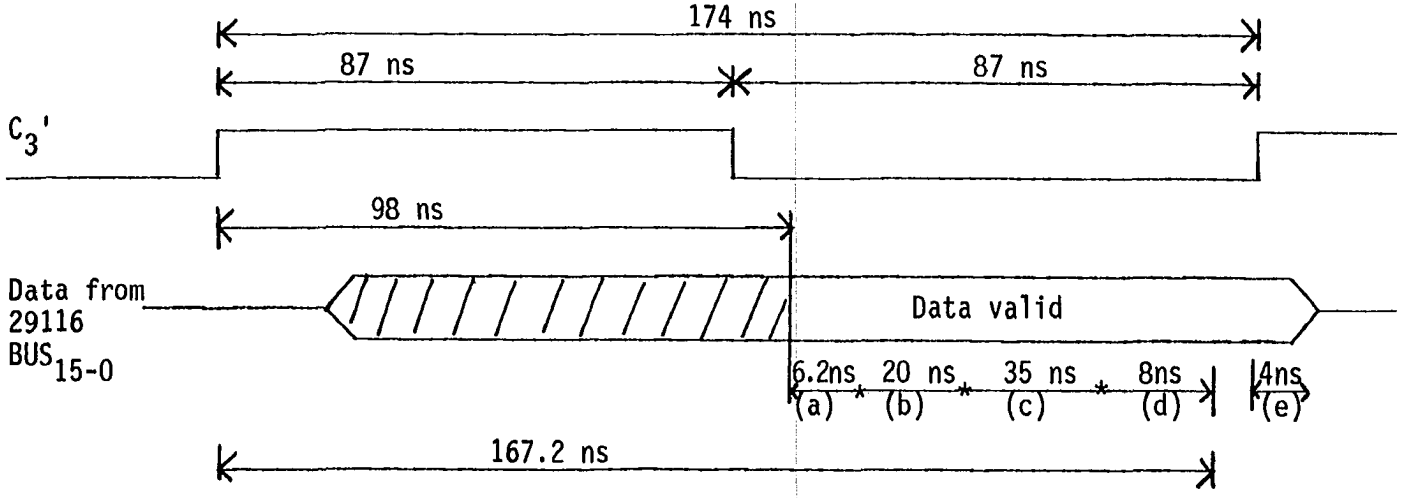
(h) 29116 requires the data to be valid at least 44 ns before CP↑. This condition is satisfied.


108 ns
21 ns
87 ns
C_4 (2925)
(a)
5.5 ns
(b)
13ns
(c)
5ns
RDFIFO
23.5 ns
(d) 67 ns max.
RR (Read
FIFO Ready)
4ns
(e)
$\overline{OE}$ (FIFO)
57.5 ns
50.5 ns
15ns
(g)
(h)
BUS_{15-0}
35 ns
(h)



(a) Data from 29116 are valid within 103.5 ns.
(b) Write FIFO requires the data to be valid 25 ns min. after Low to High transition of WRFIFO. This condition is satisfied.
(c) WR will be Low 55 ns max. after WRFIFO . Hence, WR is Low 16 ns after $C_1\uparrow$.

Figure 39. Timing diagram for writing into FIFO from 29116



(a) Delay of 6.2 ns at 74AS244 (Tri-State Buffers) between Y_{15-0} (29116) and D_{11-0} (2910-1).

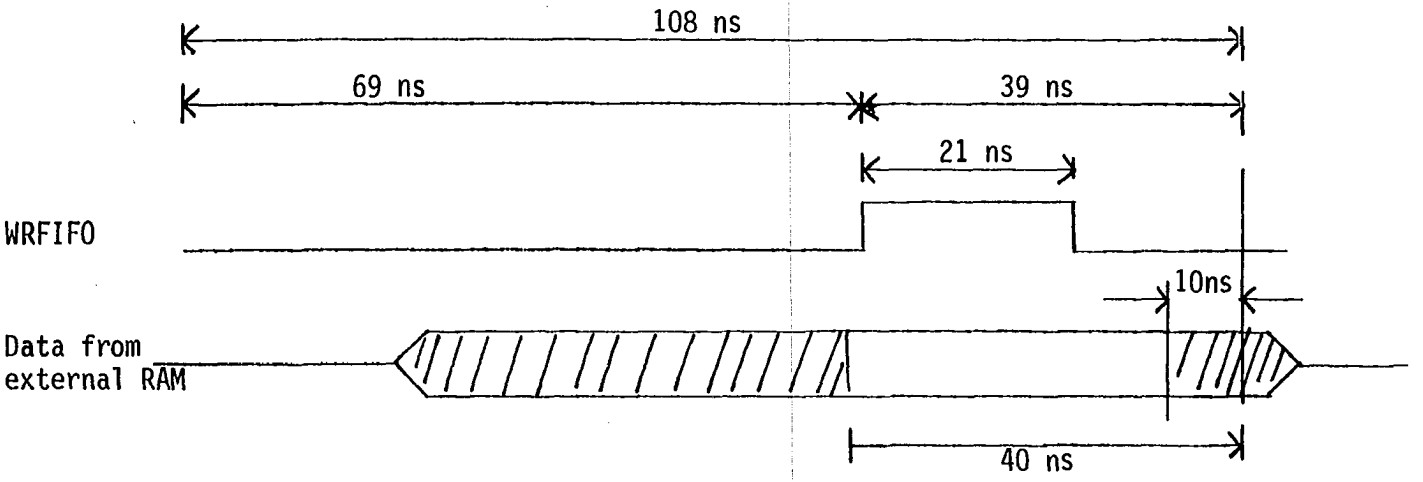
(b) Delay of 20 ns from D_{11-0} to Y_{11-0} of 2910-1.

(c) Access time of control memory is 35 ns.

(d) Setup time of 8 ns for the pipeline registers.

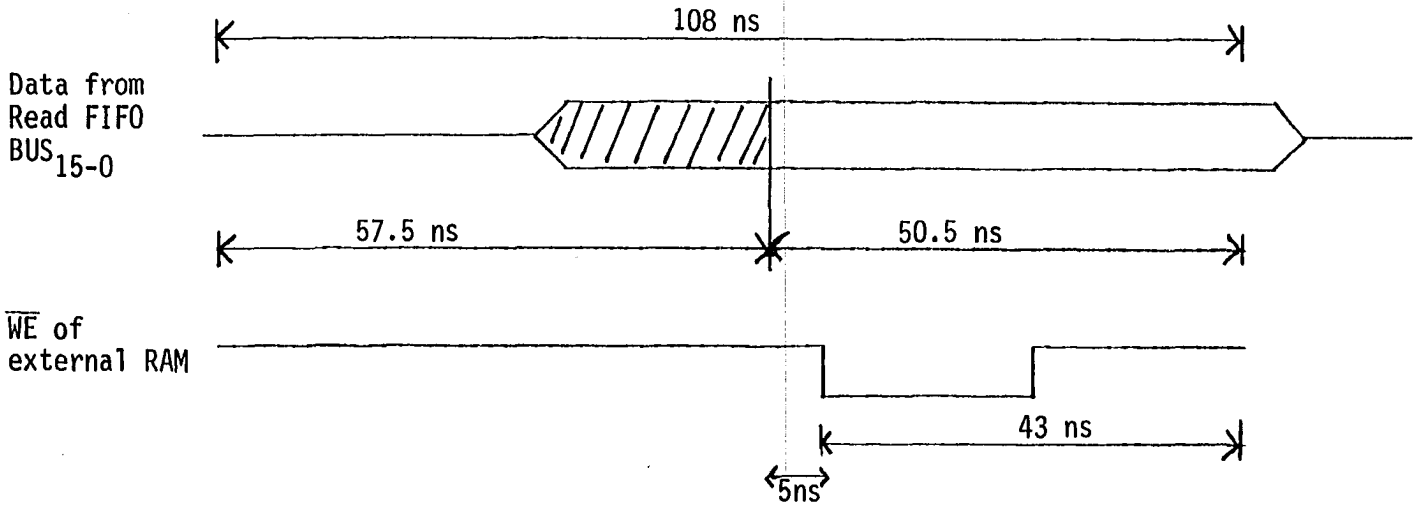
(e) 2910-1 requires a hold time of 4 ns after a Low to High transition of the clock.

Figure 40. Timing diagram for using 29116 to generate the next microaddress



Notes: Data from external RAM is valid 40 ns min. before Low to High transition of C_3, and it is not valid 10 ns max. before Low to High Transition of C_3. Since FIFO requires the data to be valid 25 ns after WRFIFO, hence it will be written even in the worst case.

Figure 41. Timing diagram for writing into FIFO and reading from the external RAM



Notes: The external RAM requires the data to be set up 5 ns min. before $\overline{WE}$ goes Low and 5 ns min. after $\overline{WE}$ goes High. Both of these conditions are met.

Figure 42. Timing diagram for reading from FIFO and writing into the external RAM

when the microprocessor is writing to an external chip, as shown in the following equation:

$$\overline{OE_y} = \overline{(WRFIFO.RD.\overline{SI_{\overline{RD}}}) + (\overline{WR}.\overline{RDFIFO}) + (WR.SI_{\overline{WR}} + RD.SI_{\overline{RD}}) + OE_{MAP}.\overline{RDFIFO})}$$

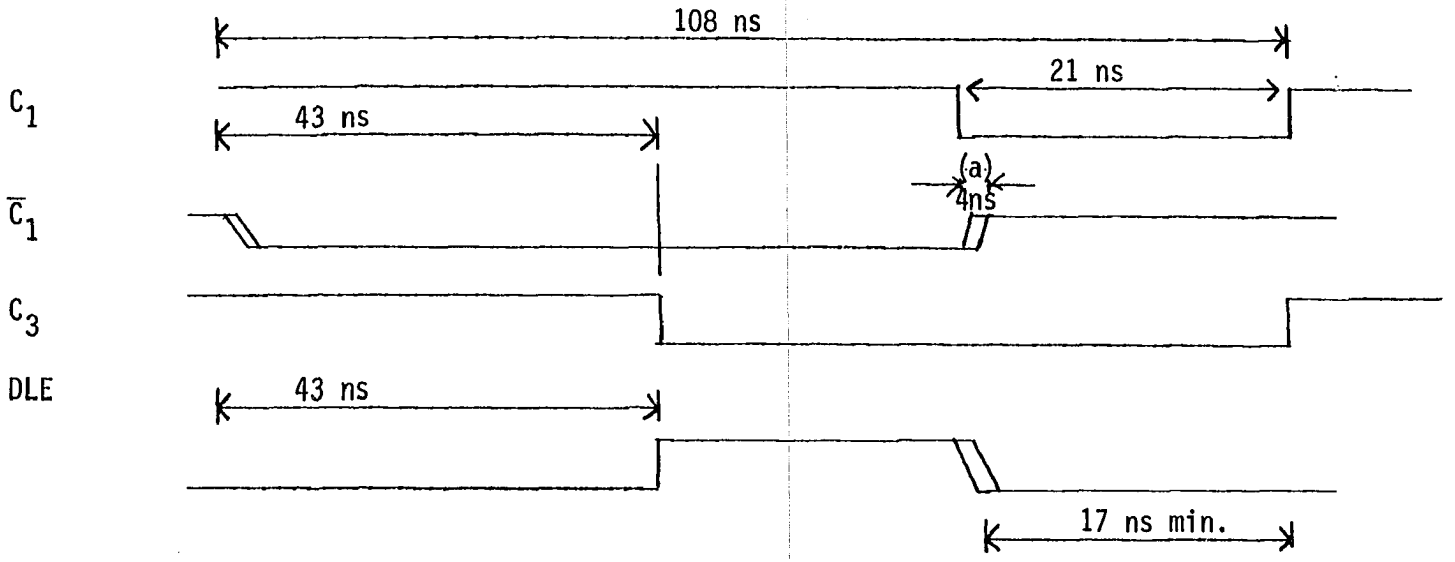
When the bidirectional Y_{15-0} pins of 29116 are disabled, Y_{15-0} acts as an input, but the data from the peripherals will be latched into the D-latch only when 29116 is reading from the peripherals, as shown in the equation below:

$$DLE = \overline{C_1}.C_3.\overline{(\overline{RD}.\overline{WRFIFO} + RDFIFO.WR)}$$

This way, data read from the peripherals will remain in the D-latch as long as another read is not initiated by the microprocessor. Thus, 29116 can perform an internal operation and simultaneously read from an external source in one cycle. From the above equation, it is apparent that the data will also be latched in the D-latch if the macroinstruction is read from the Read FIFO for 2910-1. The reason for using the clock outputs C_1 and C_3 for latching data in the D-latch is shown in the timing diagram of Figure 43.

The two data bus registers are provided for diagnostic purposes only. Under control of the Diagnostics Controller, data from the BUS can be clocked into these registers.

So far, the timings for the data path of the evaluation unit have been analyzed. In analyzing the control path, it should be noted that a



Notes: Data from Read FIFO are valid 50.5 ns min. before the Low to High transition of C_3, and data from the external RAM are valid 15.5 ns max. before the Low to High transition of C_3, but it is not valid 15.5 ns max. before the Low to High transition of C_3. It is required that in order to latch the data, the DLE must be High at least 42 ns before the Low to High transition of C_3 and can go Low almost 10 ns after data is valid. Both of these restrictions have been satisfied. Also, during the first 32 ns after the Low to High transition of C_3, the data are changing but the input at DLE must remain Low. Otherwise a High to Low pulse may latch some unwanted data in the D-latch.

Figure 43. Timing diagram for latching data in the D-latch of 29116

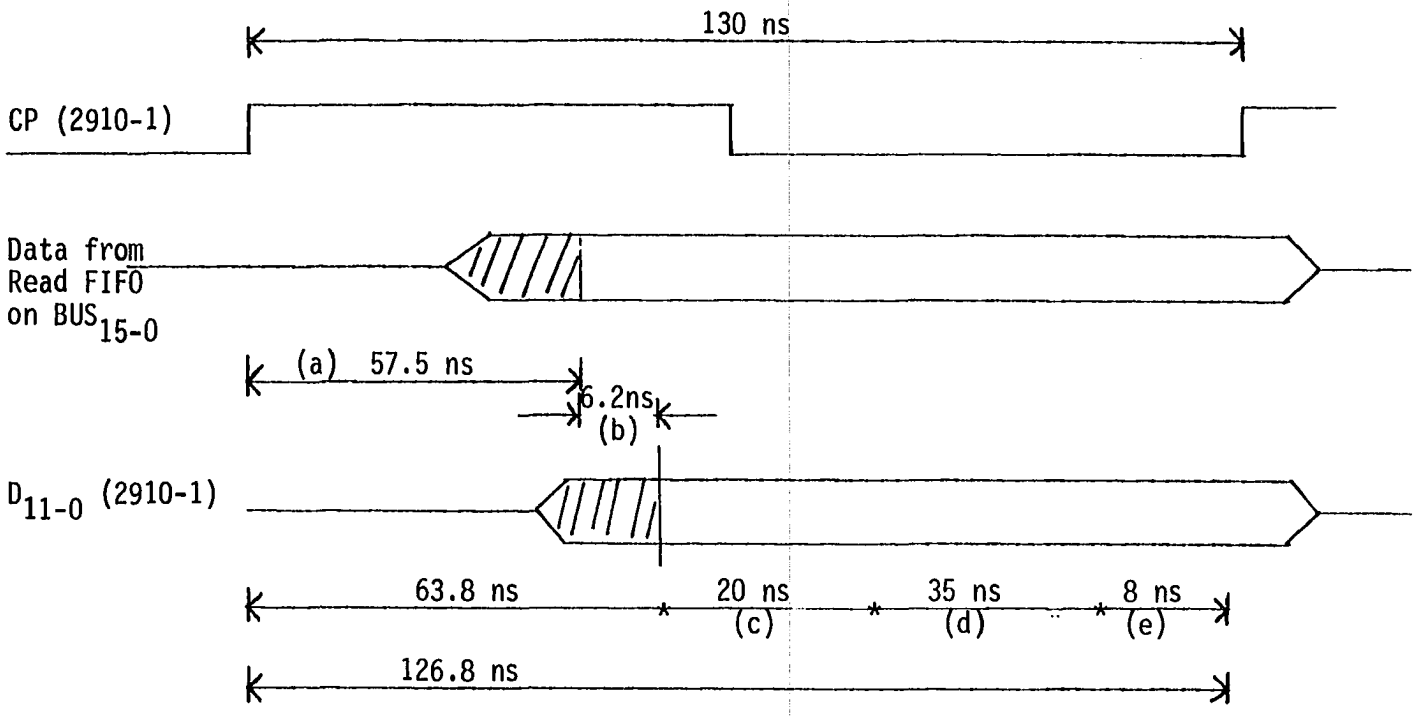
constant delay of 56 ns will be included in all control paths, as shown in Table 8.

Table 8. Constant delay in control path

Pipeline Register: PCLK to Output = 13 ns max.
Control Store: Access Time = 35 ns max.
Pipeline Register: Setup Time = 8 ns max.
Constant Delay = 58 ns max.

The minimum delay at Am 2910-1 is 50 ns (I_{3-0} to Y_{11-0}) except for instructions 8, 9, and 15 which have a minimum delay of 75 ns/85 ns (dependent on the previous instruction). The conditions in the conditional instructions can be tested in parallel, within the minimum delays of 2910-1 that have been specified. The JMAP instruction, as shown in the timing diagram of Figure 44, can be executed in 126.8 ns. All other instructions, except 8, 9, and 15, can be executed in 106 ns, whereas instructions 8, 9, and 15 can be executed in 131/141 ns (dependent on the previous instruction).

The Clock Generator and Microcycle Length Controller (Am 2925) is connected to a 46 MHz oscillator to produce a clock output of 21.74 ns at F_0. Crystals are not generally available with fundamental frequencies above 20-25 MHz. However, an overtone oscillator can be designed to oscillate at one of its odd harmonic frequencies by choosing values of C and L, as shown in Figure 31.



(a) Max. delay of 57.5 ns before data are valid.

(b) Delay at tristate buffers of 6.2 ns.

(c) D_{11-0} to Y_{11-0} delay of 20 ns at 2910-1.

(d) Access time of control store is 35 ns max.

(e) Set up time of pipeline registers is 8 ns max.

Figure 44. Timing diagram for the JMAP instruction in 2910-1

As shown in Table 9, the L_{3-1} inputs of 2925 can be controlled through the microinstruction to produce the four desired microcycle lengths. The L_{3-1} inputs of 2925 can be connected either directly from the control store or from the pipeline register. In the former case, it is easier to microgram because the entry in the L_{3-1} field controls the microcycle length of that microinstruction, whereas in the latter case, the L_{3-1} field of the current microinstruction controls the next microinstruction. The former case has the disadvantage of being slower because in order to satisfy the hold time of the L_{3-1} inputs, the delay at the multiplexer and the OR gate delay between the C_3 and CP inputs of 2910-1 must be considered. In the worst case, a delay of 9 ns must be added to the microcycle lengths, as shown in Table 9. For this reason, the latter case has been implemented in this design.

Table 9. Length of clock outputs

L_3	L_2	L_1	Length of clock cycle
H	L	H	108.7 ns
H	H	H	130.44 ns
L	H	H	152.17 ns
L	H	L	173.91 ns

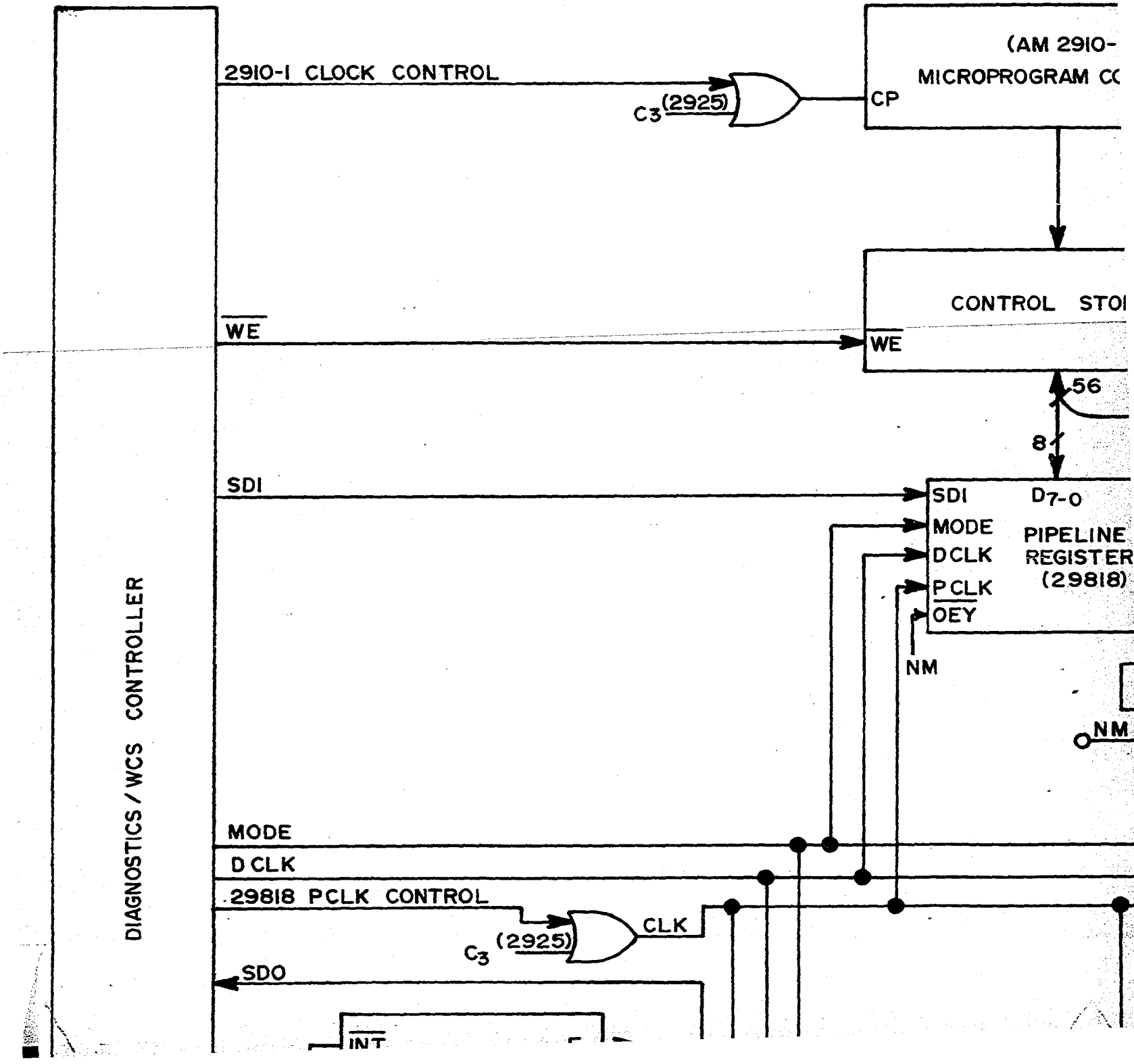
The hardware of EVAL has been designed for the worst case, and it should work under all commercial temperatures. If EVAL is used to simulate only gates, then this unit can be made more specialized by excluding the 256 x 16 bit external RAM. The 32 x 16 bit internal RAM of 29116 is sufficient to evaluate any gate.

Diagnostics and writable control store mode

Figure 45 shows the connections required in EVAL to support Diagnostics/WCS. Almost no extra chips are required to support Diagnostics/WCS. The only constraint is to replace all normal registers, with which diag nostics is to be done, by diagnostics registers (29818s). The Data Bus Registers are used entirely for diagnostic purposes to latch the data on the data bus. The status register is used to control and observe the status of the system.

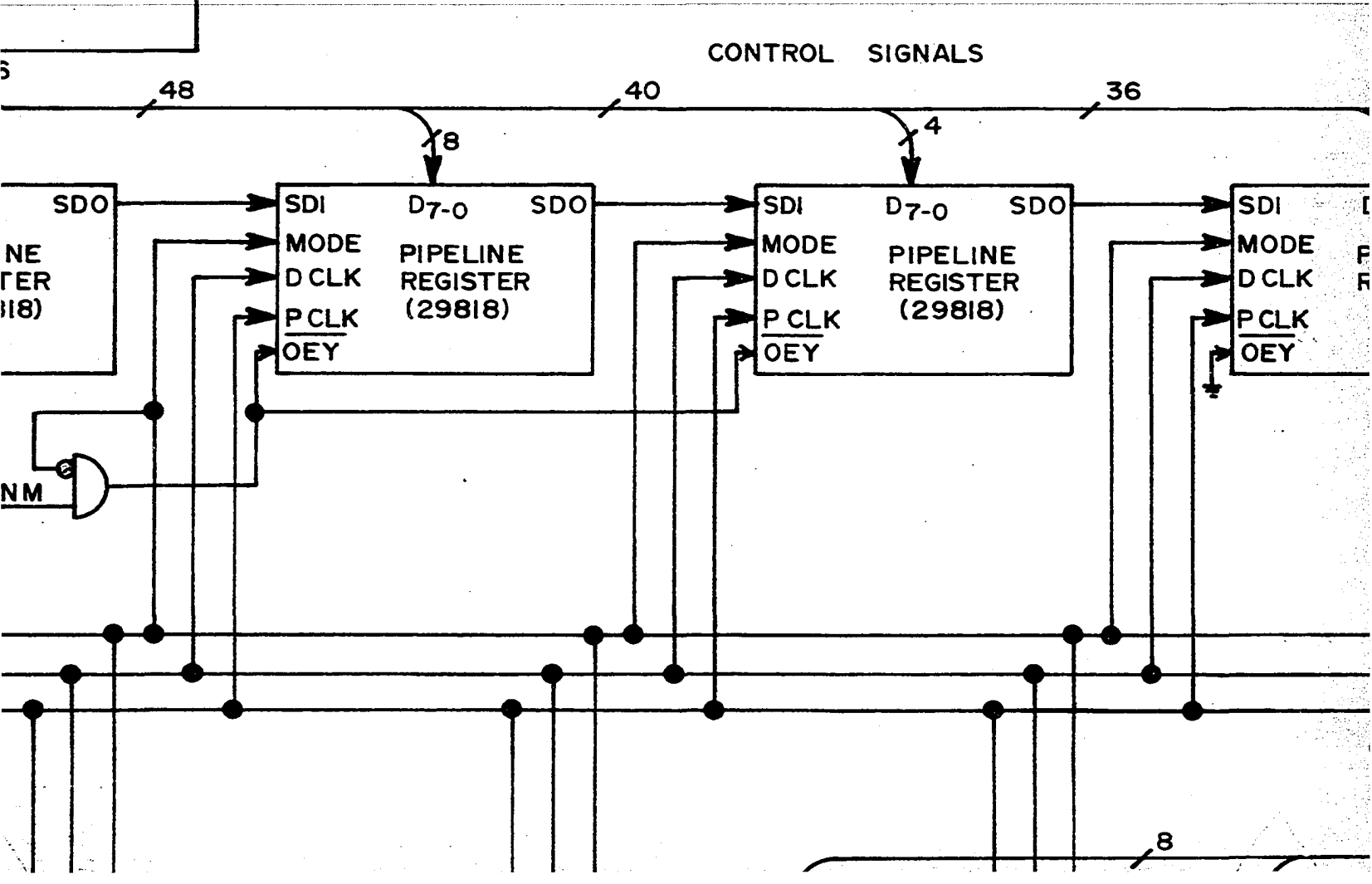
Since the next microinstruction is a function of the current microinstruction and the status of the unit, the unit can be controlled by loading the status and the current microinstruction through the test vector. Due to the nature of the design, if data are written into the FIFO, the Write FIFO Ready (WR) signal may or may not change from LOW to HIGH until the beginning of the next cycle. Therefore, a LOW or a HIGH signal may be clocked into the status register. Due to the unpredictability of this signal, its true value can only be observed in the next microcycle if another write into FIFO is not initiated. In contrast, the WR signal in the Normal mode, connected directly to the Condition Code Multiplexer, produces the true value of the signal at CC input of 2910-1 during all

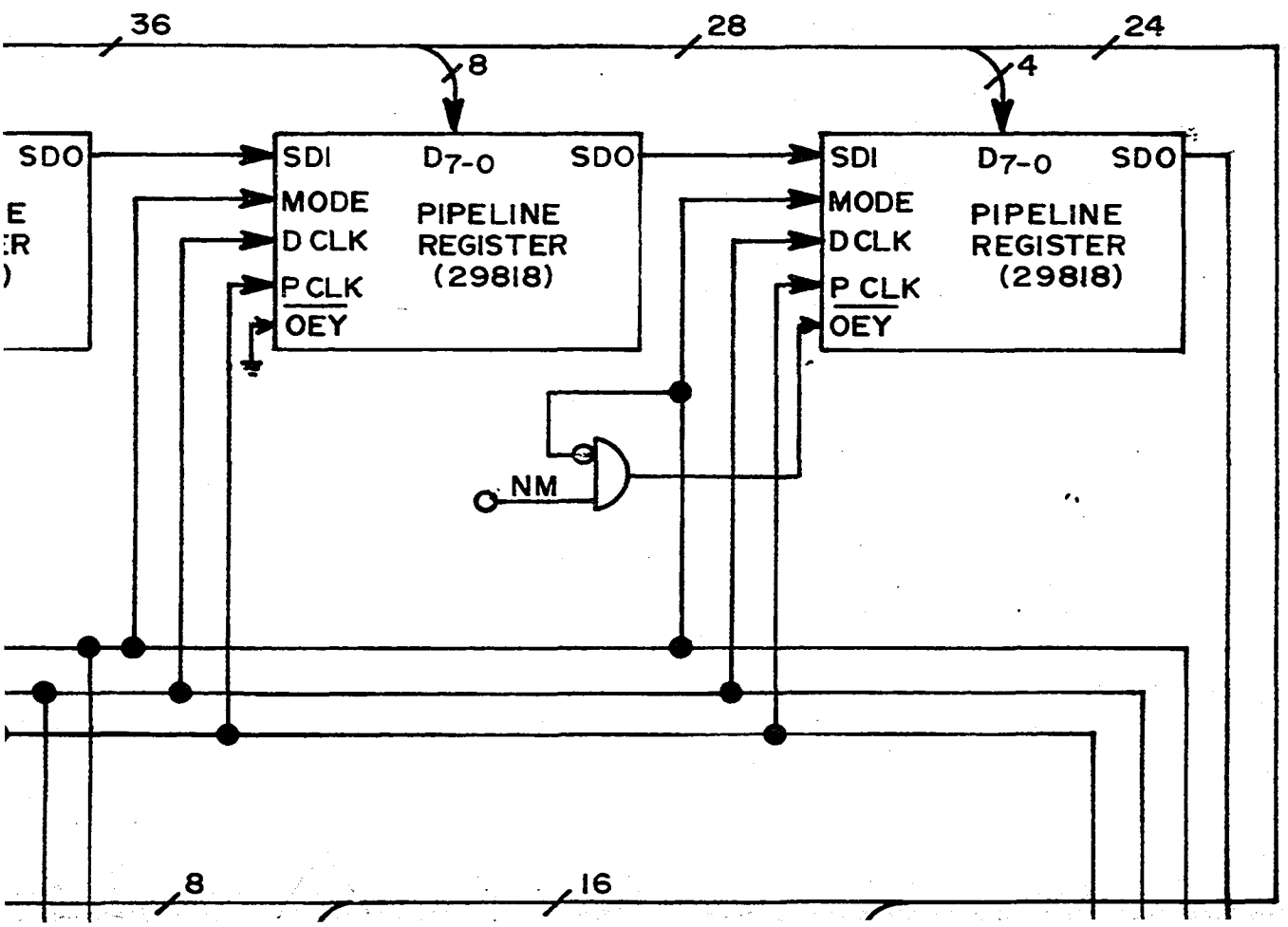
Figure 45. Connections in EVAL to support Diagnostics/WCS

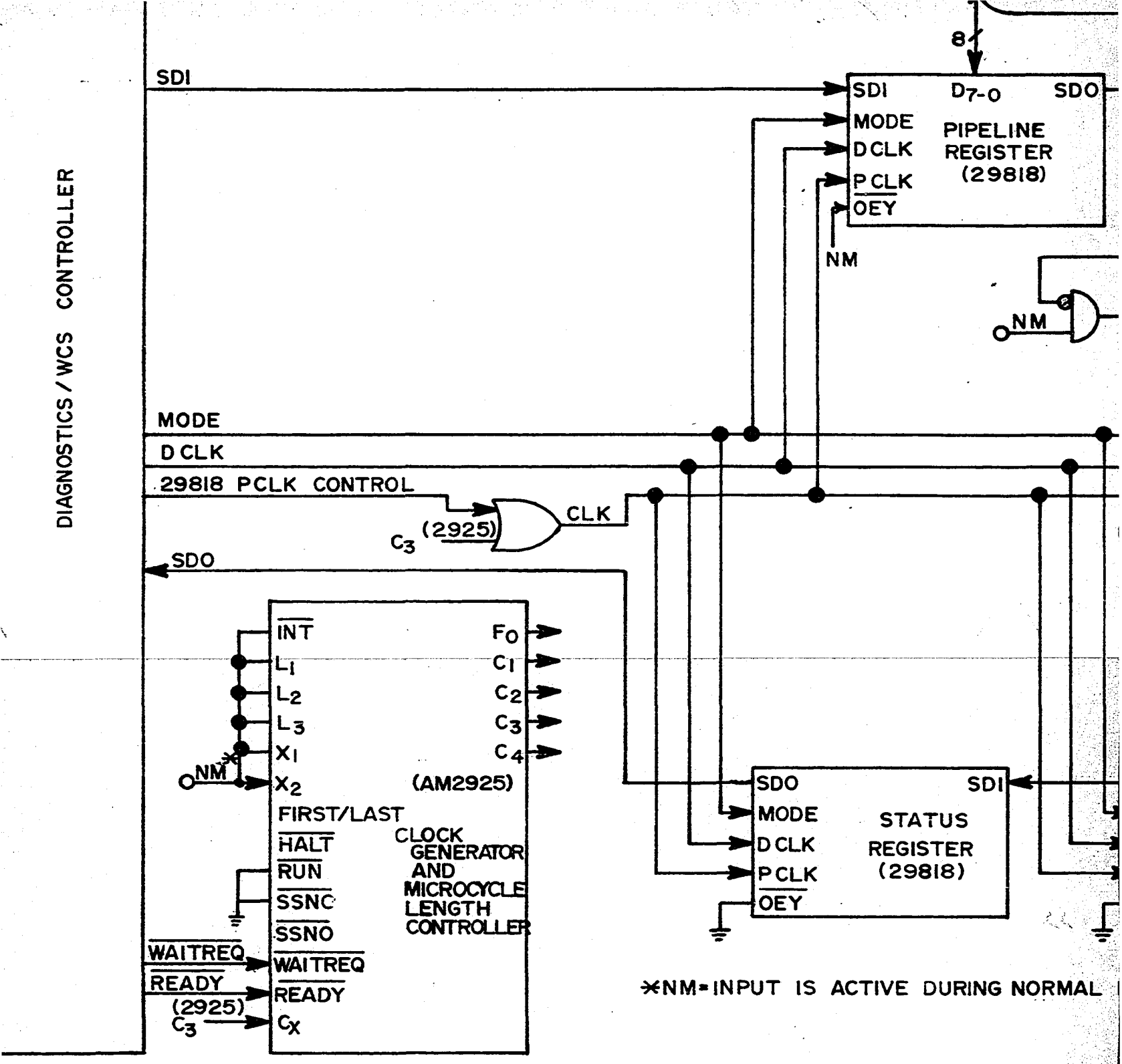

DIAGNOSTICS / WCS CONTROLLER
2910-1 CLOCK CONTROL
C3 (2925)
CP
(AM 2910-
MICROPROGRAM CC
CONTROL STOI
WE
WE
56
8
SDI
SDI
D7-0
MODE
DCLK
PCLK
OEY
PIPELINE
REGISTER
(29818)
NM
NM
MODE
DCLK
29818 PCLK CONTROL
C3 (2925)
CLK
SDO
INT

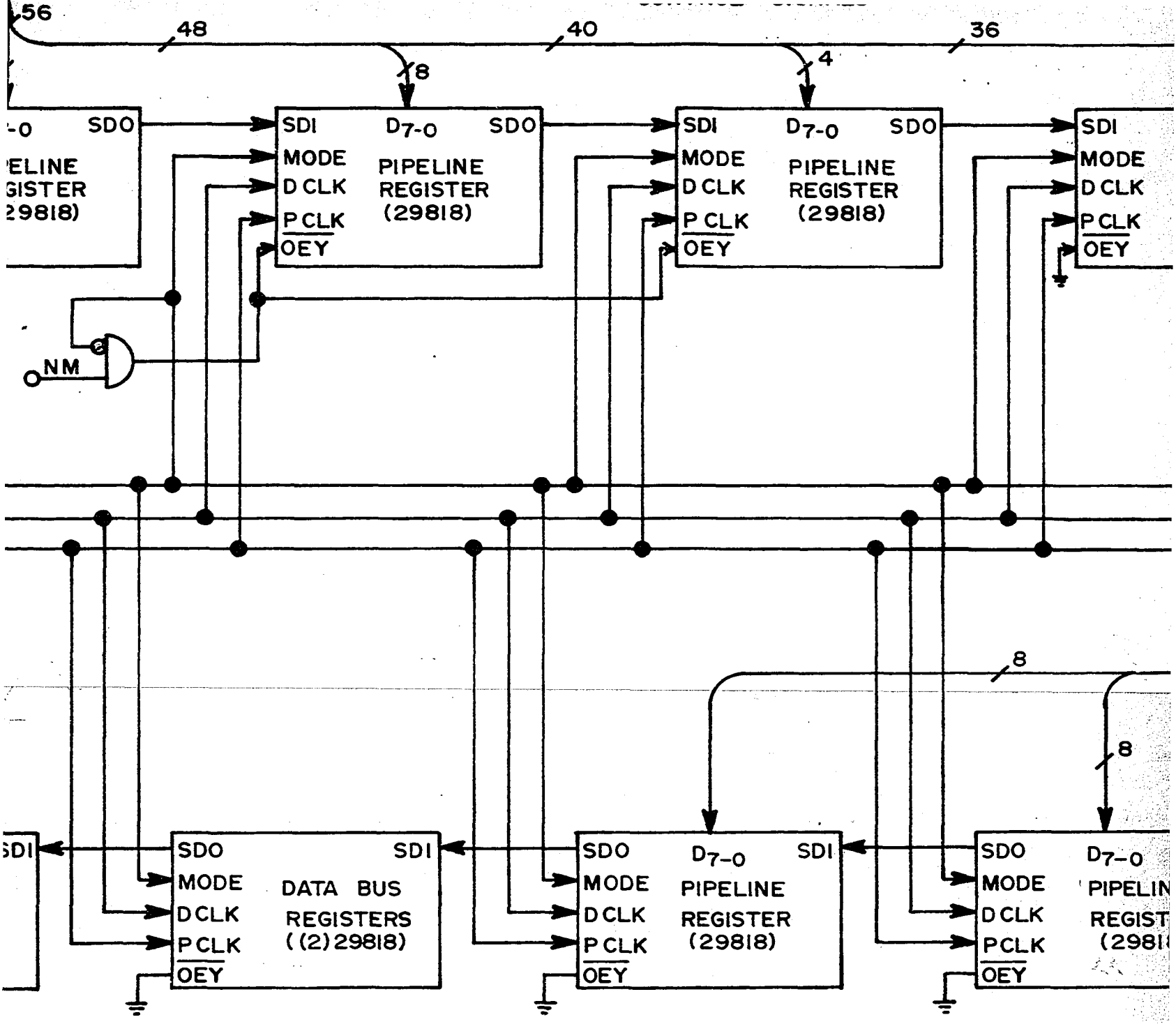
0-1)
CONTROLLER

TORE

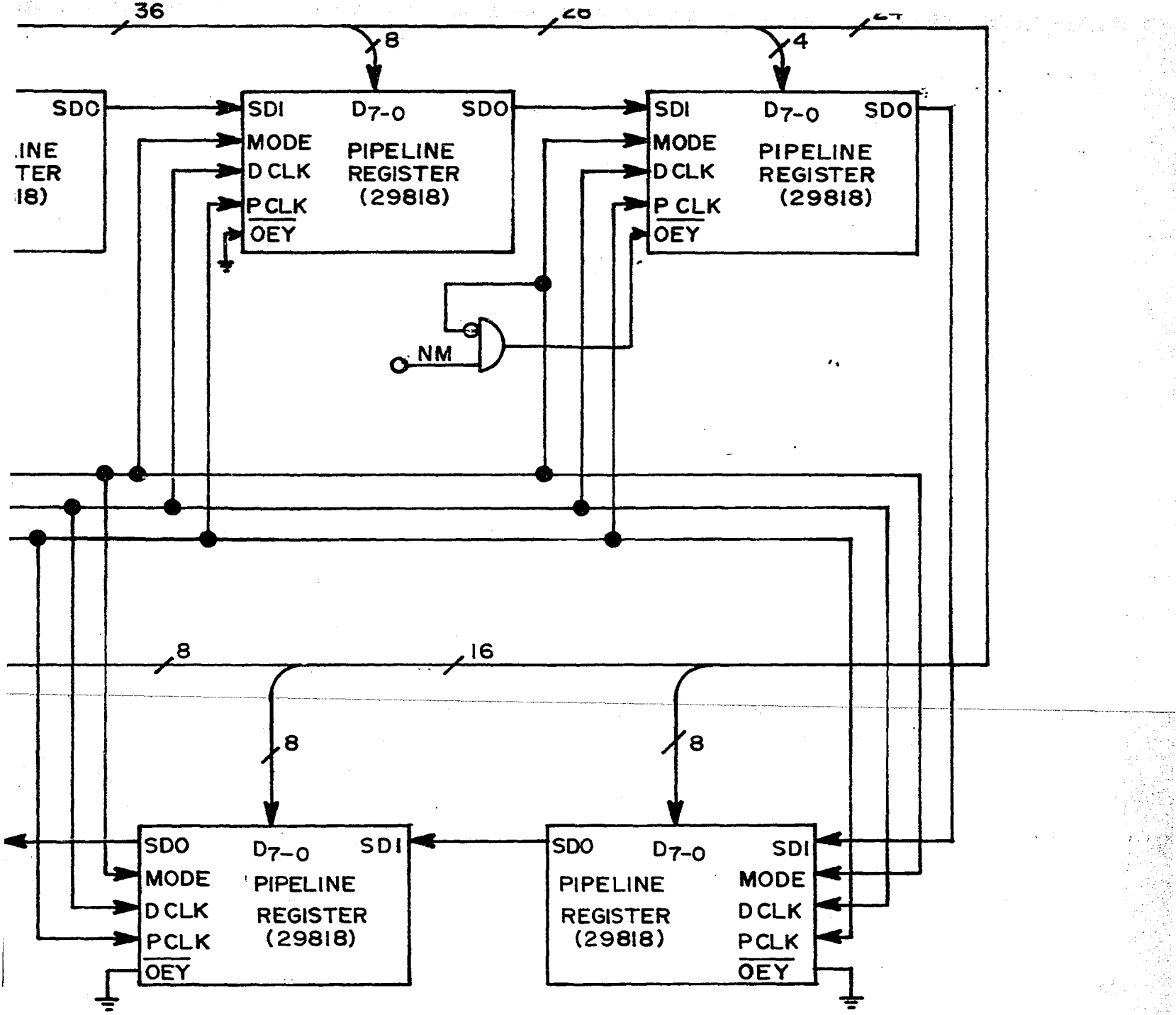



36
28
24
8
4
SDO
SDI
D7-0
SDO
MODE
D CLK
P CLK
OEY
PIPELINE
REGISTER
(29818)
NM
8
16


DIAGNOSTICS / WCS CONTROLLER
8
SDI
SDI
D7-0
SDO
MODE
DCLK
PCLK
OEY
PIPELINE REGISTER (29818)
NM
NM
MODE
DCLK
29818 PCLK CONTROL
C3 (2925)
CLK
SDO
INT
L1
L2
L3
X1
X2
NM
FIRST/LAST
HALT
RUN
SSNC
SSNO
WAITREQ
READY
(2925)
C3
CX
F0
C1
C2
C3
C4
(AM2925)
CLOCK GENERATOR AND MICROCYCLE LENGTH CONTROLLER
SDO
SDI
MODE
DCLK
PCLK
OEY
STATUS REGISTER (29818)
*NM=INPUT IS ACTIVE DURING NORMAL



NG NORMAL MODE


36
8
4
SDO
SDI
D7-0
SDO
MODE
D CLK
P CLK
OEY
PIPELINE
REGISTER
(29818)
NM
8
16
8
8
SDO
D7-0
SDI
MODE
D CLK
PCLK
OEY
PIPELINE
REGISTER
(29818)

microcycles.

Some of the outputs (OEy) of the diagnostic registers are controller in the normal mode by the contents of the microinstruction in the pipeline register. During diagnostics, the result vector needs to be loaded from the pipeline register to the shadow register, and the OEy must be enabled. Since the MODE is HIGH during this loading operation, it is used to enable the OEy regardless of the contents of the pipeline register (see Figure 45). During normal operation, the MODE is LOW, and the OEy is controlled by the contents of the microinstruction in the pipeline register.

Description of the Microinstruction

The microinstruction consists of 56 bits with one unused bit. The control store consists of the lower 2K locations of PROM and the upper 2K locations of RAM, and the Reset routine starts at location zero. Each of the fields in the microword will now be described as follows:

1. **CCEN (2910-1) (Bit 55):** When this bit is high, the condition of the status flag is ignored, and all conditional instructions are made unconditional.
2. **I_{3-0} (2910-1) (Bits 54-51):** This field selects one of the sixteen available instructions. Since the CJV instruction serves the same purpose as the CJP, it can be ignored. The JMAP instruction is used to either select the macroinstruction from the Read FIFO, or the microaddress generated by the 29116 as the next microaddress.

3. **Condition Select Mux (Bits 50-48):** This field is used to select one of eight status flags, shown in Table 10, whose condition could be used to select the next microaddress.

Table 10. Status flags

Microword Bits 50	49	48	Status flag selected	cc input fo 2910-1 when status flag is true
0	0	0	WR from Write FIFO	1
0	0	1	RR from Read FIFO	1
0	1	0	CT from 29116	0
0	1	1	Z from 29116	0
1	0	0	OUR from 29116	0
1	0	1	N from 29116	0
1	1	0	C from 29116	0
1	1	1	FULL from 2910-1	0

Before writing data into the Write FIFO, the WR (Write Ready) flag must be checked. If this flag is true, the Write FIFO is ready to accept data. Otherwise the FIFO is full.

Before reading from the Read FIFO, the RR (Read FIFO Ready) flag must be checked. This flag is true if the FIFO contains data. Otherwise the FIFO is empty.

It is very important to note that when RR and WR are true, the CC input of 2910-1 is 1, which is a false condition for the microprogram controller. This feature is helpful in microprogramming. The Z (Zero), OVR (Overflow), N (Sign), and C (Carry) flags of the 29116 give the status of the ALU. Besides these four flags, the 29116 also

provides facilities to test a total of 12 different conditions and output the result on the CT output. If the stack in 2910-1 is to be used, the FULL status flag could be checked to determine whether the stack is full.

4. **D_{11-0} (2910-1) (Bits 47-36):** This field contains a 12 bit data which could be used to load the register/counter of 2910-1, or provide the next microaddress.

5. **RLD (2910-1) (Bit 35):** When this bit is active, the register/counter of 2910-1 is loaded with the data in D_{11-0} field, regardless of the instruction and condition.

6. **L_{3-1} (2925) (Bits 9-7):** This field is used to control the microcycle length of the next microinstruction; that is, the L_{3-1} field in the current microinstruction controls the microcycle length of the next microinstruction. Table 9 shows the different cycle lengths that can be obtained by changing this field. Table 11 will be helpful in choosing the correct cycle length. It is divided into five parts, and the microcycle length of the part that requires the highest microcycle should be used.

7. **IEN, I_1'-I_1', I_0''-I_0', I_{15-2} (29116)(Bits 34, 33-32, 31-30, 23-10):** When the instruction in 29116 specifies the internal RAM registers as a source or destination, the I_{4-0} part of the instruction addresses the 32 registers. I_1 is divided into I_1'' and I'_1 and I_0 is divided into I_0'' and I'_0, where I_{4-2}, I_1', I_0' specifies the source address, and I_{4-2}, I_1'', I_0'' the destination address. For a single address

Table 11. Microcycle length requirements for various fields in the microword

Microword bits	Functions	Microcycle length
54-51, 35	Instruction 2 of 2910-1 (used to read macroinstruction from Read FIFO)	130 ns
	Instructions 8, 9, and 15 of 2910-1	130/152 ns. If the previous executed instruction was 4 or 12 or RLD was low, then use 152 ns
28	OE_T=0 (test status bits of ALU and produce result at CT)	152 ns
34	IEN=0 (during the first cycle of immediate instruction)	130 ns
54-51, 33-30 23-10, 6-1	29116 writes into external RAM	152 ns
	29116 loads the address counters of external RAM	152 ns
	29116 writes into Write FIFO	152 ns
	29116 provides the next microaddress	174 ns
	All other combinations	108 ns

case, any 29116 register can be used as the source or the destination, but in the double address case, the source and the destination addresses differ in only the two least significant bits I_1 and I_0. If an immediate instruction is used, the first microcycle consists of the instruction in I_{15-0} with IEN=0, and the second microcycle consists of data in I_{15-0} with IEN=1. In fact, IEN=1 in all the instructions, except the first microcycle of the immediate instruction.

8. **SRE, OE_T, T_{4-1} (29116) (Bits 29, 28, 27-24):** The SRE controls the internal status register of 29116. When SRE=0, the internal status register is latched at the end of every microcycle with the eight status bits of the ALU except when NO OP, Save Status, and Test Status instructions are used. When SRE=1, the latched status bits will be held in the register.

   When OE_T=0, the T_{4-1} field can be used to test any one of the twelve possible test conditions in parallel with the execution of the instruction by the ALU. The result of the test condition is produced at the CT status output and clocked in the external status register. However, the status of the ALU will not be clocked in the external status register, but could be saved in the internal status register by setting SRE=1.

9. **WRFIFO; RDFIFO; SI_{WR}, WR; SI_{RD}, RD (Bits 6, 5, 4-3, 2-1):** When WRFIFO is active, data on BUS_{15-0} will be written into the Write FIFO. When RDFIFO is active, data will be read from the Read FIFO onto the BUS_{15-0}. The FIFOs should be written or read only if their

corresponding flags are true. The functions of WR, $S1_{WR}$ and RD, $S1_{RD}$ are described in Table 7.

The 29116 can perform an operation on internal data (an operation is internal if 29116 does not use the Read FIFO, Write FIFO, and the external RAM) and write the result out in the external RAM is one microcycle of 152 ns if the address of the external RAM is in the Write Address Counter. Otherwise, one extra cycle of 152 ns is required to load the Write Address Counter with the correct address. Similarly, the result of a 29116 operation can be written in the Write FIFO in one microcycle of 152 ns.

29116 can read from the Read FIFO or the external RAM, perform an operation, and store the result internally in one microcycle of 108 ns. The Read Address Counter must point to the correct location in the external RAM, otherwise one extra microcycle of 152 ns is required to load the counter. All external data read by 29116 will be latched in its D-latch and can be used anytime later on, as long as another external read is not performed. In this way, data could be prefetched while the ALU is performing a different function.

It should be pointed out that both Read and Write Counters can be loaded in one microcycle of 152 ns. Also, to use the external RAM efficiently, the microroutine should load both the counters in the beginning, and then all accesses should be from successive locations of RAM.

While 29116 is performing internal operations, data from the Read FIFO can be transferred to the external RAM, or data from the external RAM can be transferred to the Write FIFO in 108 ns.

Firmware

When the system is Reset, the microinstruction at microaddress zero is selected with a microcycle time of 130 ns. The Reset routine consists of four microinstructions in the first four locations of the control store, as shown in Table 12.

At microaddress 0, the RR flag is checked repetitively until it is true. EVAL waits until MODEL starts sending information. At microaddress 1, the device ID is read from FIFO and stored in R31 (register 31 of 29116 RAM). Actually, the device ID must be checked to ascertain that it is not a control word. The control word should be sent to MODEL. This Reset routine does not check for control words. At microaddress 2, both the address counters in the external RAM are reset to zero. This microinstruction is not required by simple devices. At microaddress 3, data from Read FIFO is sent to R0. If the device is simple, then this data contain both input and output signals of the device. For functional device, it may contain the input signals, or the output signals, or both. At microaddress 4, the macroinstruction is read from the FIFO and sent to the 2910-1 as the next microaddress which points to the starting of the device's microroutine. The first microinstruction in the microroutine will be executed in 108 ns.

Microinstructions at addresses 5 through 10 consist of three subroutines which are accessible from any device microroutine. A jump to the subroutine at addresses 5 and 6 is made when the Read FIFO is empty. This subroutine checks the RR flag repetitively until it is true. An

Table 12. Reset routine and subroutines

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	J_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_0^1, I_0^{11}$	WR FIFO	RD FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
0	0	CJP	RR	0	1	108	1	0	1	X	NO OP	0	0	0 NO	1 OP	0	1
1	X	CONT	X	X	1	152	1	0	1	X	SOR,MOVE,SODR,R31	0	1	0	1	0	1
2	0	CJS	RR	5	1	108	1	0	1	X	SONR,MOVE,SOZ,NRY	0	0	1	1	1	1
3	0	CJS	RR	7	1	130	1	0	1	X	SOR,MOVE,SODR,RO	0	1	0	1	0	1
4	X	JMAP	X	X	1	130	1	0	1	X	NO OP	0	1	0	1	0	1
5	0	CJP	RR	5	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	--
6	1	CRTN	X	X	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	--
7	0	CJP	RR	7	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	--
8	1	CRTN	X	X	1	130	1	0	1	X	NO OP	---	---	NO	OP	---	--
9	0	CJP	WR	9	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	--
10	1	CRTN	X	X	1	152	1	0	1	X	NO OP	---	---	NO	OP	---	--

unconditional return from the subroutine is then made, and the next microinstruction is executed in 108 ns. The subroutine at addresses 7 and 8 is similar to the previous subroutine except that the next microinstruction of 130 ns is executed when a return from this subroutine is made. A jump to the subroutine at addresses 9 and 10 is made when the Write FIFO is full. A return is made when the WR flag indicates that the FIFO is ready to accept data. The next microinstruction after the return is assumed to be a write into FIFO, which is 152 ns.

One may be tempted to combine microinstructions at microaddresses 0 and 1 together to form one microinstruction, but this may not work for the simple reason that EVAL and MODEL do not run with the same clock. To visualize this problem, assume that the FIFO is empty and the combined microinstruction is repetitively checking the RR flag and also reading data from the empty FIFO. Assume that MODEL inputs the data in the FIFO, and after the fall through time of the data, the RR flag is true exactly 30 to 35 ns nefore the end of the microcycle. This time is enough for the 2910-1 to select the next address as the increment of the previous address instead of the address in D_{11-0}, but this time is not sufficient for the 29116 to latch the data read from the FIFO in its D-latch or to operate on the data. For this reason, the RR flag must be first checked before the FIFO is read. In a similar manner, the WR flag must be first checked and the data written in the FIFO in the following microcycle if the WR flag is true.

When a microroutine has completed execution, control is transferred to microaddress 0 where the reset routine starts. For every device, simple or functional, the device ID will be stored in R31 of 29116's RAM, both the address counters of the external RAM will be reset, and the first data word will be stored in R0 of 29116's RAM before control is transferred to the device's microroutine. The first microinstruction in the microroutine must be executed within 130 ns because the microinstruction at address 4 demands such a cycle time. This kind of arrangement saves some memory in the control store because the same common microinstructions are not stored with each device's microroutine. However, the speed is reduced because some microoperations need not be performed for simple devices (ex., resetting the counters), and in the case of functional devices, it may be better to store the device ID in the external RAM. This trade-off between saving control memory space and increasing the speed will have to be dealt with by the microprogrammer. All the different factors must be weighed to make a decision.

As an example, the evaluation of a 2 input NAND gate, through a table look-up scheme, is given. Since each input can have any one of the three signals (0, 1, unknown), the truth table will contain 9 entries (3^2), as shown in Table 13. It should be noted from the table that the input combinations of 3 and 7 do not have an entry. This is because only three signal values have been considered, and these decimal values correspond to the fourth signal value.

Table 13. Truth table for a 2 input NAND gate

Decimal inputs	Input combination A	Input combination B	Output A.B
0	0	0	1
1	0	1	1
2	0	2	1
3	X	X	X
4	1	0	1
5	1	1	0
6	1	2	2
7	X	X	X
8	2	0	1
9	2	1	2
10	2	2	2

The output of the truth table is stored in the external RAM at locations 245_{10} through 255_{10}. Actually, all tables of constants should be stored in an external ROM (not included in the evaluation unit). The two leftmost bits in every location of the external RAM contain the output signal value. The Reset routine would have stored the device ID in R31 of 29116, and the input and output signal values of the gate in R0 of 29116 before transferring control to the NAND microroutine. The signal values of the 2 inputs are in the 4 leftmost bits of R0, and the output signal values are in the 2 rightmost bits, as shown in Figure 46. Bits 13-4 are don't cares.

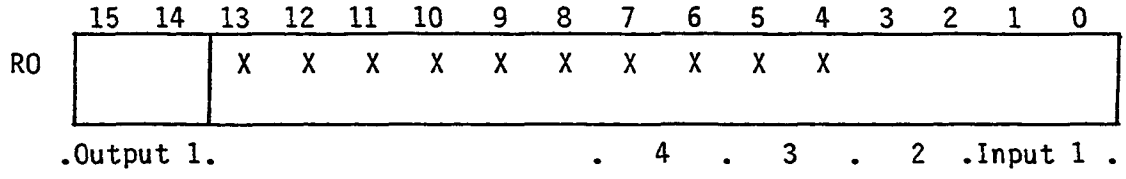


Figure 46. Arrangement of input and output signal values

The microroutine of the NAND gate starts from location A of the control memory, as shown in Table 14. The idea is to use the signal values of the two inputs as an offset from the beginning of the truth table and read the output signal value at that address. This new output signal value is then compared to the old one, and an event is generated if the two outputs differ. At locations A and (A+1) of the control memory, the microinstructions change bits (15-4) of RO to zero and store the result in R1. At locations (A+12) and (A+3), 245_{10} is added to R1, and the result is stored in the Read Address counter of the external RAM. This result acts as an address to the new output signal value. At location (A+4), the new output is read from the external RAM and stored in the D-latch of 29116. At location (A+5), the old output in RO is compared to the new output in the D-latch by masking all but the two leftmost bits in RO and the D-latch. The Z flag gives the result of the comparison. At location (A+6), the Z flag is checked. If Z=1, then the two outputs are the same; no event is generated, and control is transferred to the Reset routine. If Z=0, then the two outputs are different and an event is generated. Before writing to the FIFO, the WR flag is first checked at

Table 14. Microroutine for a 2 input NAND gate using a table look-up scheme

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	J_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_o^1, I_o^{11}$	WR FIFO	$\overline{RD}$ FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
A+1	X	CONT	X	X	1	130	1	0	1	X	0000,0000,0000,1111	---	---	NO	OP	---	-
A+2	X	CONT	X	X	1	152	0	0	1	X	TOR1,TORI4,ADD,R1	---	---	NO	OP	---	-
A+3	X	CONT	X	X	1	130	1	0	1	X	245_{10}	0	0	0	1	1	1
A+4	X	CONT	X	X	1	108	0	0	1	X	ROTC,0,CDRI,RO,R1	0	1	0	1	0	1
A+5	X	CONT	X	X	1	108	1	0	1	X	0011,1111,1111,1111	---	---	NO	OP	---	-
A+6	0	CJP	Z	0	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	-
A+7	0	CJS	WR	9	1	152	1	0	1	X	NO OP	---	---	NO	OP	---	-
A+8	0	CJS	WR	9	1	152	1	0	1	X	50R,MOVE,SOR4,R31	1	0	0	1	0	1
A+9	1	CJP	X	0	1	108	1	0	1	X	SONR,MOVE,SOD,NR4	1	---	NO	OP	---	-
A+0	X	CONT	X	X	1	108	0	0	1	X	TOR1,TORIR,AND,RO,R1						

location (A+7). At location (A+8), the device ID is sent to the Write FIFO, and at location (A+9), the new output signal value is sent to the Write FIFO and control is transferred to the Reset routine. The reset routine requires 608.5 ns to be executed before control is transferred to any microroutine. The reset routine and the NAND microroutine altogether require 1479 ns if an event is not generated, and 1891 ns if an event is generated. During this total time, the data are read from the FIFO, the output of the gate is evaluated, and the result is written into the FIFO if an event is generated.

A 3 input NAND gate can be evaluated similarly to a 2 input NAND gate by storing its truth table in the external RAM (actually, it should be stored in a ROM). Its truth table will contain 27 (3^3) entries, but it will occupy a total of 43 locations out of which 16 locations will not be used. The total time required to evaluate is the same as the 2 input NAND gate, and the same microinstructions are used with different masks and different starting address of the truth table in the RAM.

Similarly, a 4 input NAND gate will require exactly the same amount of total time and the same microinstructions (with different masks and different starting address of the truth table in RAM) as a 2 input NAND gate, if its truth table is stored in the external memory. The truth table will require a total of 171 locations, out of which only 81 locations will contain the output values, and the rest will be unused.

From the discussion, it is clear that any simple device whose input-output relationship can be derived with a truth table can be evaluated in 1891 ns/1479 ns (dependent on whether an event is generated or not) and

the same microinstructions, as the 2 input NAND gate can be used with different masks and different starting address of truth table.

Going back to the discussion of the 2 input, 3 input, and 4 input NAND gates, some external memory can be saved if only the truth table of the 4 input NAND gate is stored and all the other NAND gates use the same truth table. This can be done as follows: (1) The 2 input NAND gate is assigned a macroinstruction equal to some control store address B, and the 3 input NAND gate has a macroinstruction equal to B+4. Assume that the microroutine of the 4 input NAND gate starts at microaddress A, so this gate has a macroinstruction equal to A. (2) When any of the three gates is to be evaluated, after the reset routine has completed, control will be transferred either to the microinstruction at microaddress B, or microaddress B+4, or to microaddress A. (3) A 4 input NAND gate can be made out of a two input NAND gate by forcing signal values of 1 at inputs 3 and 4 (i.e., bits 7-4 of RO) (see Table 13). This is done by microinstructions at microaddresses B to B+3. Also, the microinstruction at (B+3) jumps to location A where the truth table of the 4 input NAND gate lies. (4) Similarly, the microinstructions from microaddresses (B+4) to (B+7) convert the 3 input NAND gate to a 4 input NAND gate by forcing a signal value of 1 at input 4 (i.e., bits 7-6 in RO) and then jumping to the microroutine of the 4 input NAND gate (see Table 15). In this way, the external memory required can be reduced at the expense of speed. Both the 2 input and the 3 input NAND gates will require 2369/1957 us to be evaluated. A generalization can be made for a lot of simple devices by noting that the truth table of the simple device which has the highest

Table 15. Microroutines to convert a 2 and a 3 input NAND gate into a 4 input NAND gate

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	J_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_o^1, I_o^{11}$	WR FIFO	RD FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
B	X	CONT	X	X	1	108	0	0	1	X	SONR,MOVE,SOI,NRA	----	----	NO	OP	----	--
B+1	X	CONT	X	X	1	130	1	0	1	X	XXXX,XXXX,0101,XXXX	----	----	NO	OP	----	--
B+2	X	CONT	X	X	1	108	0	0	1	X	ROTM,0,MARI,RO,RO	----	----	NO	OP	----	--
B+3	1	CJP	X	A	1	130	1	0	1	X	0000,0000,1111,0000	----	----	NO	OP	----	--
B+4	X	CONT	X	X	1	108	0	0	1	X	SONR,MOVE,SOI,NRA	----	----	NO	OP	----	--
B+5	X	CONT	X	X	1	130	1	0	1	X	XXXX,XXXX,01XX,XXXX	----	----	NO	OP	----	--
B+6	X	CONT	X	X	1	108	0	0	1	X	ROTM,0,MAR1,RO,RO	----	----	NO	OP	----	--
B+7	1	CJP	X	A	1	130	1	0	1	X	0000,0000,1100,0000	----	----	NO	OP	----	--

number of inputs can be stored in the ROM, and all other similar logic devices with fewer inputs can be converted to this device.

So far in the examples that have been considered, a large amount of external memory has been wasted because of the way the truth table is stored. Taking the example of a 4 input NAND gate, 171 words of external memory are required to store a truth table with 81 entries. Also, in every word only 2 bits are used to store one output signal value. A better method to save memory space is to store the 171 output signal values in 22 (171/8) locations (word 0 to 21) of the external memory by packing 8 output signal values in one memory word. Figure 47 shows this arrangement for an arbitrary memory word 4. Note that the smallest input combination value corresponds to the two MSBs and the larger input combination corresponds to the two LSBs. Again, in this method memory is allocated to all the 171 output signal values, when only 81 are needed.

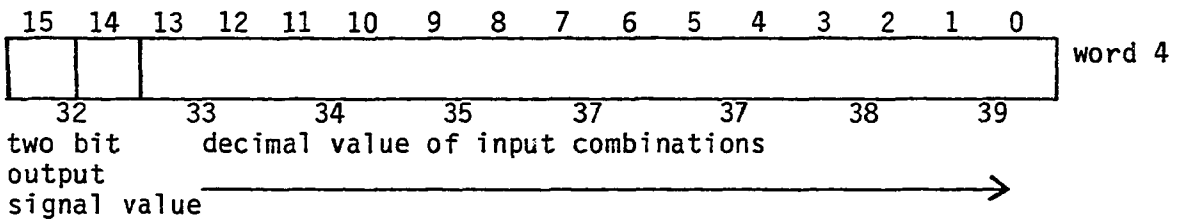


Figure 47. Arrangement of output signal values packed in one memory word

This 22 word truth table is stored in the external memory in locations 234_{10} through 255_{10} (for example), and the microroutine starts from location 'C' of the control store (see Table 16). The following steps are required to read the table and evaluate the gate:

1. After the Reset routine has completed, R0 of 29116 contains the input signal values of the 4 inputs in bits 7-0. Bits 15-8 are forced to zero, and the result is stored in R1. Then, an arithmetic shift right of R1 is performed three times, and the result is stored in R2. These operations correspond to a division of the number in R1 by 8 and the result in R2, which is rounded to the smaller integer value, points to the memory word where the output signal value is stored. Microinstructions at microaddress C_1 (C+1), (C+2), (C+3), and (C+4) execute these operations.
2. Since the truth table is stored in external memory, starting from location 234_{10}, the offset calculated in steps 1 and stored in R2 is added to the starting address of the table and stored in the Read address counter of the external RAM. Microinstructions at addresses (C+5) and (C+6) perform these operations.
3. Microinstruction at address (C+7) reads the memory word from the external memory containing the desired output signal in the D-latch of 29116. Simultaneously, the data in R2 are rotated left three times, and the result is stored in the accumulator. This rotation is equivalent to an arithmetic shift left three times which corresponds to the multiplication of the number in R2 by 8. This number in the

Table 16. Microroutine of a 4 input NAND gate using a table look-up scheme

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	J_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_o^1, I_o^{11}$	WR FIFO	RD FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
C+0	X	CONT	X	X	1	108	0	0	1	X	TOR1,TORIR,AND,RO,R1	---	---	NO	OP	---	--
C+1	X	CONT	X	X	1	108	1	0	1	X	0000,0000,1111,1111	---	---	NO	OP	---	--
C+2	X	CONT	X	X	1	108	1	0	1	X	SHFTR,SMRR,SMDNZ,R1,R2	---	---	NO	OP	---	--
C+3	X	CONT	X	X	1	108	1	0	1	X	SMFTR,SMRR,SHDNZ,R2	---	---	NO	OP	---	--
C+4	X	CONT	X	X	1	130	1	0	1	X	SMFTR,SMRR,SMDNZ,R2	---	---	NO	OP	---	--
C+5	X	CONT	X	X	1	152	0	0	1	X	TOR1,TORI4,ADD,R2	---	---	NO	OP	---	--
C+6	X	CONT	X	X	1	108	1	0	1	X	234_{10}	---	NO	OP	---	1	1
C+7	X	CONT	X	X	1	108	1	0	1	X	ROTR1,3,RTRA,R2	---	NO	OP	---	X	0
C+8	X	CONT	X	X	1	130	1	0	1	X	TOR1,TORAA,SUBS,R1	---	---	NO	OP	---	--
C+9	X	CONT	X	X	1	174	0	0	1	X	TONR,TOAI,ADD,NRY	---	---	NO	OP	---	--
C+10	X	JMAP	X	X	1	130	1	0	1	X	C+11	---	---	NO	OP	---	--
C+11	1	CJP	X	C+19	1	108	0	0	1	X	ROTC,0,CDRI,RO,R1	---	---	NO	OP	---	--
C+12	1	CJP	X	C+19	1	108	0	0	1	X	ROTC,2,CDRI,RD,R1	---	---	NO	OP	---	--
C+13	1	CJP	x	C+19	1	108	0	0	1	X	ROTC,4,CDRI,RO,R1	---	---	NO	OP	---	--
C+14	1	CJP	X	C+19	1	108	0	0	1	X	ROTC,6,CDRI,RO,R1	---	---	NO	OP	---	--
C+15	1	CJP	X	C+19	1	130us 101	0	0	1	X	ROTC,8,CDRI,RD,R1	---	---	NO	OP	---	--
C+16	1	CJP	X	C+19	1	101	0	0	1	X	ROTC,10,CDRI,RD,R1	---	---	NO	OP	---	--
C+17	1	CJP	X	C+19	1	101	0	0	1	X	ROTC,12,CDRI,RD,R1	---	---	NO	OP	---	--

Table 16. Continued

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	J_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_0^1, I_0^{11}$	WR FIFO	RD FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
C+18	1	CJP	X	C+19	1	108us 101	0	0	1	X	ROTC,14,CDRI,RD,R1	---	---	NO	OP	---	---
C+19	X	CONT	X	X	1	108us 101	1	0	1	X	0011,1111,1111,1111	---	---	NO	OP	---	---
C+20	0	CJP	Z	0	1	108us 101	1	0	1	X	NO OP	---	---	NO	OP	---	---
C+21	0	CJS	WR	9	1	152us 011	1	0	1	X	SOR,MOVE,SORA,R1						
C+22	0	CJS	WR	9	1	152us 011	1	0	1	X	SOR,MOVE,SOR4,R31	1	---	NO	OP	---	---
C+23	1	CJP	X	0	1	108us 101	1	0	1	X	TOR1,TORA4,EXOR,RO	1	---	NO	OP	---	---

accumulator is the smallest input combination (see Figure 47) of that memory word.

4. The smallest input combination in the accumulator is subtracted from the input combination in R1 and the result, with some modification, points to the desired output signal in the D-latch of 29116. Microinstruction at address (C+8) performs this operation and stores the result in the accumulator.
5. The old output signal value is in bits 15-14 of R0. The new output signal value in the accumulator must be rotated so that it occupies bits 15-14 in order to be compared to the old signal value. The number of bit positions to be rotated would depend on where the output signal value is. If it occupies the two leftmost bit positions, then it has to be rotated left 14 times before it can be compared to the old output signal value. A table of all possible rotations is stored in the control memory from addresses (C+11) to (C+18). The microinstructions at addresses (C+9) and (C+10) add the starting address of the rotation table to the value in accumulator and sends the result to 2910-1 as the next microaddress.
6. The next microaddress points to the correct microinstruction in the rotation table, which together with the microinstruction at address (C+19) rotates the new output signal value in the D-latch, compares it to the old output in R0, and stores the result in R1.
7. The microinstruction at address (C+20) checks the Z flag. If both the output signal values are the same, then control is transferred to the reset routine. If the outputs are not the same, then an event is

generated, and the control is transferred to microinstructions at addresses (C+21), (C+22), and (C+23). These microinstructions first send the device ID to the Write FIFO, followed by the output signal value, and finally control is transferred to the Reset routine.

Using this method, the amount of time required to evaluate a 4 input NAND gate is 2022 ns if an event is generated, and 1610 ns if the event is not generated. Again, this example can be generalized in two ways if the same microroutine is used with different masks and different address of the truth table. Firstly, any simple device whose input-output relationship can be expressed through a truth table, will require 2022/ 1610 ns to be evaluated. Secondly, the truth table of the simple device, which has the highest inputs, is stored in the external memory. All other similar simple devices with fewer inputs can then use the same truth table once they are converted to the simple device which has the highest number of inputs.

In the last example, the NAND gate will be evaluated by inspection of the NAND logic and not by storing the truth table in the external memory, as has been done in the previous examples. A seven input NAND gate is considered even though only four input gates will be evaluated. Table 17 contains the microroutine. The gate will be first checked for a zero input. If any one of the inputs has a signal value of 0, then the output is a 0. If none of the inputs is 0, and any one input has a signal value of 2, then the output is a 2. Finally, if none of the inputs has a signal value of 0 or 2, then all the inputs have a signal value of 1, and the output is 1. The microroutine starts from microaddress D and ends at

Table 17. Microroutine for a 7 input NAND gate evaluated without tables

Micro-address	Am 2910-1					Am 2925	Am 29116					Write FIFO, Read FIFO, RAM					
	$\overline{CCEN}$	D_{3-0}	$\overline{cc}$ Mux	D_{11-0}	$\overline{RLD}$	L_{3-1} in ns.	$\overline{IEN}$	$\overline{SRE}$	OE_T	T_{4-1}	$I_{15-2}, I_1^1, I_1^{11}, I_o^1, I_o^{11}$	WR FIFO	RD FIFO	$S1_{\overline{WR}}$	$\overline{WR}$	$S1_{\overline{RD}}$	$\overline{RD}$
D	X	CONT	X	X	1	108	1	0	1	X	ROTR1,I,RTRA,RO	---	---	NO	OP	---	---
D+1	X	CONT	X	X	1	130	1	0	1	X	TOR1,TORAA,EXNOR,RO	---	---	NO	OP	---	---
D+2	X	CONT	X	X	1	108	0	0	1	X	PRTNR,PRI,PRTA,NRA	---	---	NO	OP	---	---
D+3	X	CONT	X	X	1	108	1	0	1	X	1101,0101,0101,0101	---	---	NO	OP	---	---
D+4	0	CJP	Z	D+11	1	130	1	0	1	X	SONR,MOVE,SOZ,NRX	0	0	1	1	1	1
D+5	X	CONT	X	X	1	108	0	0	1	X	ROTC,O,CDRT,RO	---	---	NO	OP	---	---
D+6	X	CONT	X	X	1	108	1	0	1	X	0011,1111,1111,1111	---	---	NO	OP	---	---
D+7	o	CJP	Z	0	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	---
D+8	0	CJS	WR	9	1	152	1	0	1	X	NO OP	---	---	NO	OP	---	---
D+9	0	CJS	WR	9	1	152	1	0	1	X	SOR,MOVE,SOR4,R31	1	---	NO	OP	---	---
D+10	1	CJP	X	0	1	108	1	0	1	X	SONR,MOVE,SOD,NR4	1	---	NO	OP	---	---
D+11	X	CONT	X	X	1	108	0	0	1	X	PRT3,PRI,PR3R,RO,R1	---	---	NO	OP	---	---
D+12	X	CONT	X	X	1	108	1	0	1	X	1101,0101,0101,0101	---	---	NO	OP	---	---
D+13	0	CJP	Z	D+15	1	108	1	0	1	X	NO OP	---	---	NO	OP	---	---
D+14	1	CJP	X	D+5	1	130	1	0	1	X	BONR,15,SETN,D	---	---	NO	OP	---	---
D+15	1	CJP	X	D+5	1	130	1	0	1	X	BONR,14,SETN,D						

(D+16). Microinstructions at microaddresses D through (D+4) test if any one of the input of the gate is 0. If the test passes, control is transferred to microinstruction (D+5). Otherwise, microinstructions (D+11) through (D+13) test if any one input has a signal value of 2. If the test passes, then control is transferred to (D+5) via (D+4). Otherwise, all the inputs are 1, and control is transferred to (D+5) via (D+15). Microinstructions (D+5) and (D+6) compare the old and the new outputs, and a test is made by microinstruction (D+7). If an event is not generated, then control is transferred to the Reset routine. However, if an event is generated, microinstructions at microaddresses (D+8) through (D+10) transfer the device ID and the new output to the FIFO, before transferring control to the Reset routine. The minimum time required to evaluate the gate is 1543 ns if the output is 0 and an event is not generated. If an event is generated, then the time required is 1956 ns. However, if the output is ether 1 or 2, then 2000 ns are required if an event is not generated, and 2413 ns are required if an event is generated. By using the same microroutine, with different masks, NAND gates with any number of inputs can be evaluated in the time mentioned above. Some control memory can be saved at the expense of evaluation time if the same microroutine for the seven input NAND gate is used for all other NAND gates, instead of having one microroutine for each NAND gate with different inputs. This can easily be done by first converting the NAND gate to a seven input NAND gate, and then jumping to the NAND microroutine. The conversion will require an additional 478.2 ns.

Table 18 gives the evaluation times for all the three methods described to evaluate a NAND gate. Clearly, there is a trade off between evaluation time and external memory space. Method 1 uses the most external memory and has the minimum evaluation time, whereas Method 3 does not use the external memory and has the maximum evaluation time. The worst case evaluation time of 2891 ns will occur if an NAND gate with fewer inputs is first converted to a NAND gate with the highest inputs, whose microroutine is in the control store, and Method 3 is used to evaluate the gate. The best case of 1479 ns will occur by using Method 1. Hence, a NAND gate can be evaluated anywhere from 1.5 us to 3 us. This figure is true for other gates too, and it is safe to use this range for all simple devices.

Table 18. NAND gate evaluation time for the three methods

Method	Description	Evaluation time in ns	External memory	Control memory
1	One entry of truth table is stored in one word of external memory	1891/1479	171 words	10 words
2	Eight entries of truth table are stored in one word of external memory	2022/1610	22 words	24 words
3	Truth table is not used. Evaluation is done by inspection of NAND logic	1956/1543 best case 2413/2000 worst case	0 words	16 words

APPENDIX B: DATA SHEET

Cascadable 16 Bit Error Detection and Correction Unit (AM 2960) . 172
SSR Diagnostics/WCS Pipeline Register (AM 29818) 178
Clock Generator and Microcycle Length Controller (AM 2925) . . . 180
Microprogram Controller (AM 2910-1) 183
16 Bit Bipolar Microprocessor (AM 29116) 191
2048 x 8 Bit Bipolar PROM (AM 27S191A) 214
1024 x 4 Bit Static RAM (AM 2149) 216
256 x 4 Bit TTL Bipolar RAM (AM 93422A) 220
First-In First-Out Memory (TDC 1030) 224
Synchronous 8 Bit Up/Down Counter (74AS869) 228

Also available as the AmZ8160 for AmZ8000 Systems

Am2960 • Am2960-1 Am2960A

Cascadable 16-Bit Error Detection and Correction Unit

DISTINCTIVE CHARACTERISTICS

- **Boosts Memory Reliability**
  Corrects all single-bit errors. Detects all double and some triple-bit errors. Reliability of dynamic RAM systems is increased more than 60-fold.
- **Very High Speed**
  Perfect for MOS microprocessor, minicomputer, and mainframe systems.
  - Data in to error detect: 32ns worst case.
  - Data in to corrected data out: 65ns worst case.

  High performance systems can use the Am2960 EDC in check-only mode to avoid memory system slowdown.
- **Replaces 25 to 50 MSI chips**
  All necessary features are built-in to the Am2960 EDC, including diagnostics, data in, data out, and check bit latches.
- **Handles Data Words From 8 to 64 Bits**
  The Am2960 EDC cascades: 1 EDC for 8 or 16 bits, 2 for 32 bits, 4 for 64 bits.
- **Easy Byte Operations**
  Separate byte enables on the data out latch simplify the steps and cuts the time required for byte writes.
- **Diagnostics Built-In**
  The processor may completely exercise the EDC under software control to check for proper operation of the EDC.

GENERAL DESCRIPTION

The Am2960 Error Detection and Correction Unit (EDC) contains the logic necessary to generate check bits on a 16-bit data field according to a modified Hamming Code, and to correct the data word when check bits are supplied. Operating on data read from memory, the Am2960 will correct any single bit error and will detect all double and some triple bit errors. For 16-bit words, 6 check bits are used. The Am2960 is expandable to operate on 32-bit words (7 check bits) and 64-bit words (8 check bits). In all configurations, the device makes the error syndrome available on separate outputs for data logging.

The Am2960 also features two diagnostic modes, in which diagnostic data can be forced into portions of the chip to simplify device testing and to execute system diagnostic functions. The product is supplied in a 48 lead hermetic DIP package.

4

TABLE OF CONTENTS

FUNCTIONAL DESCRIPTION
- Block Diagram 4-6
- Architecture 4-7
- Pin Definitions 4-8
- 16-Bit Configuration 4-11
- 32-Bit Configuration 4-14
- 64-Bit Configuration 4-17

ELECTRICAL SPECIFICATIONS

APPLICATIONS
- Byte Write 4-36
- Diagnostics 4-36
- Eight-Bit Data Word 4-36
- Other Word Widths 4-36
- Single Error Correction Only 4-36
- Check Bit Correction 4-37
- Multiple Errors 4-37

SYSTEM DESIGN CONSIDERATIONS
- High Performance Parallel Operation 4-39
- EDC in the Data Path 4-39
- Scrubbing Avoids Memory Errors 4-39
- Correction of Double Bit Errors 4-39
- Error Logging and Preventative Maintenance 4-40
- Reducing Check Bit Overhead 4-41
- EDC Per Board vs. EDC Per System 4-41

FUNCTIONAL EQUATIONS 4-42

Am2960 BOOSTS MEMORY RELIABILITY 4-46

ADVANCED INFORMATION

Am2960-1
- Speed selected version of Am2960 on the critical data paths
- Plug-in replacement for Am2960

Am2960A
- Second generation of Am2960 EDC internal ECL circuitry and state-of-the-art process technology combined to provide fastest version of popular Am2960.
- Plug-in replacement for Am2960
- Improved speed
  25 – 30% speed improvement on the critical paths versus the Am2960

Am2960-1
- Speed selected version of Am2960 on the critical data paths.
- Plug-in replacement for Am2960.

Am2960A
- Second generation of Am2960 EDC internal ECL circuitry and state-of-the-art process technology combined to provide fastest version of popular Am2960.
- Plug-in replacement for Am2960.
- Improved speed
  25 – 30% speed improvement on the critical paths versus the Am2960.