MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science

Computation Structures Group Memo 164

The Design of an Arbitration Network for a Data-Flow Processor

by

Mary E. McNally

(Thesis submitted in partial fulfillment of the requirements for the Degree of Bachelor of Science at M.I.T.)

July 1978

THE DESIGN OF AN ARBITRATION NETWORK FOR A DATA-FLOW PROCESSOR

by

MARY ELIZABETH McNALLY

ABSTRACT

Contemporary research in the theory of computer architecture has led to developments in parallel computation, and in particular, to the concept of a data-flow processor. Within such a processor, routing networks are used to transport information between processor sections. An arbitration network is a routing network that controls the information flow from the memory section to the functional unit section. In order to enhance the execution speed, this network must exploit the parallelism inherent in data-flow architecture. A logic design is given for an arbitration network which is entirely asynchronous and employs commercially available components.

Thesis Supervisor: David P. Misunas

Title: Staff Member, Division of Sponsored Research

TABLE OF CONTENTS

Chapter 1

1.0 Introduction

1.1 Data-Flow Architecture Research

Recent attempts to achieve high speed data execution have led to the development of a unique type of computer architecture based on the concept of data-flow. A data-flow architecture is controlled by the availability of data, and does not have the inherent limitations of switching and processing delays that conventional computer architectures have to contend with.

This unique architecture relies on the parallelism inherent in the processor design to achieve high speed operation. Several research groups have developed quite different approaches to the development of a parallel processor (1,2,4). This thesis will deal with the research done at M.I.T's Laboratory for Computer Science. The members of this group have developed both a data-flow processor and a data-flow language (3).

The need for high speed computation has arisen from many applications of signal processing, such as modulation, filtering, and fast Fourier Transforms. Signal processing performed on conventional computers has been limited by the delays inherent in synchronous architecture. In contrast, the data-flow concept relies on the availability of data, rather than clocked control, to obtain maximum execution speed and consequently a higher throughput.

1.2 Packet Communication Structures

The data-flow processor referenced in this design utilizes packet communication. All information within the processor is transmitted in discrete information packets. Such a transmission system requires either a well-developed routing network, or the conventional bus or crossbar networks. A description of each of these networks and the tradeoffs involved in selecting the appropriate type of network is discussed by Jacobsen and Misunas (5). The arbitration network for the data-flow processor is an asynchronous routing network. This type of communication structure is better suited to the concurrent transmission of operation packets, and in addition has the benefits of a modular communication structure.

1.3 Arbiters

The main component in the Arbitration Network is the arbiter. The function of the arbiter is to control the access to a shared resource by several users. In this data-flow processor, there are only 4 functional units, yet there are 512 memory cells desiring to use them. The arbiter must allow only one signal to be processed at a given time, restricting subsequent signals until the first signal has been processed to the next stage of the network. The essential criteria for an arbiter is its

ability to handle simultaneous signals on its inputs. When such an event occurs, the problem of glitches and metastable states must be avoided. Much work has been done on the theory of arbiters by Patil (6,7) and others (8). The proper handling of transmission signals by the arbiter is essential for an efficient Arbitration Network.

1.4 Data-Flow Research at M.I.T.

The concepts of data-flow utilized in this design of the Arbitration Network have been developed to a fairly sophisticated level by researchers in the Laboratory for Computer Science at M.I.T. (1,2,3,9). The work done by Amikura (9) is particularly useful for representing the interface of the memory to the Arbitration Network. The initial signally discipline is established by this interface, along with the sizes and structures of the various communication packets.

1.5 Overview of the Thesis

The purpose of this thesis is to present a logic design for an arbitration network which will be fast, reliable, and capable of extending the parallelism of the rest of the processor. Chapter 2 discusses the data-flow processor in some detail. Chapter 3 reviews the theoretical concepts of an arbiter. Chapters 4 and 5 present the actual design of the Arbitration Network along with several ideas regarding synchronous and asynchronous components and circuits. Chapter 6 will present some conclusions, and provide a brief implementation analysis, including network size and cost.

Chapter 2

2 The Data-Flow Processor

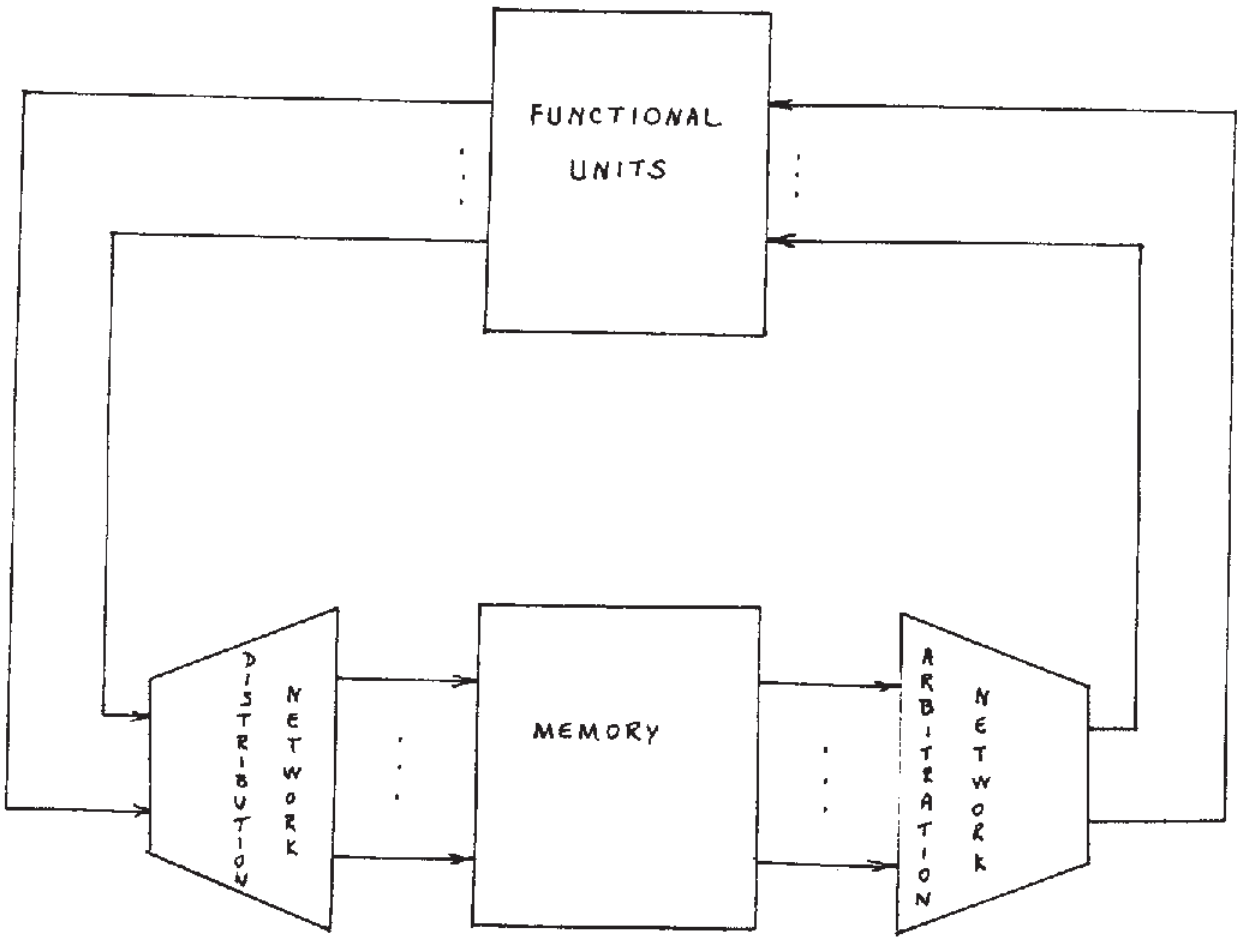
2.0 Overview

The data-flow processor is a stored program computer with the configuration shown in Figure 2.1. Its four main sections include the Memory Section, the Arbitration Network, the Functional Unit Section, and the Distribution Network.

2.1 Memory Section

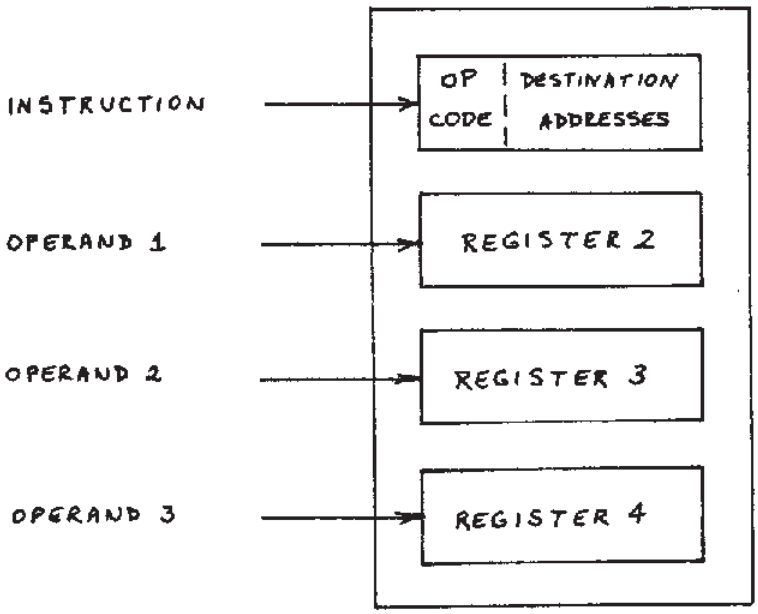
The memory section of the data-flow processor consists of instruction cells, which contain both the instructions and their required operands. The structure of an instruction cell, given in Figure 2.2, shows the four registers that hold the necessary data. Register 1 contains the instruction, which consists of an op code and destination address(es). The op code is used to select the appropriate functional unit to perform the operation, and the destination address(es) specify the locations of those instruction cells in the memory section that are to receive the result packet as one of their operands. Registers 2, 3 and 4 hold the instruction's operands.

The processing of an instruction depends on the arrival of the requisit operands. As soon as all necessary operands are received into the instruction



DATA - FLOW PROCESSOR

FIG. 2.1



INSTRUCTION CELL CONFIGURATION

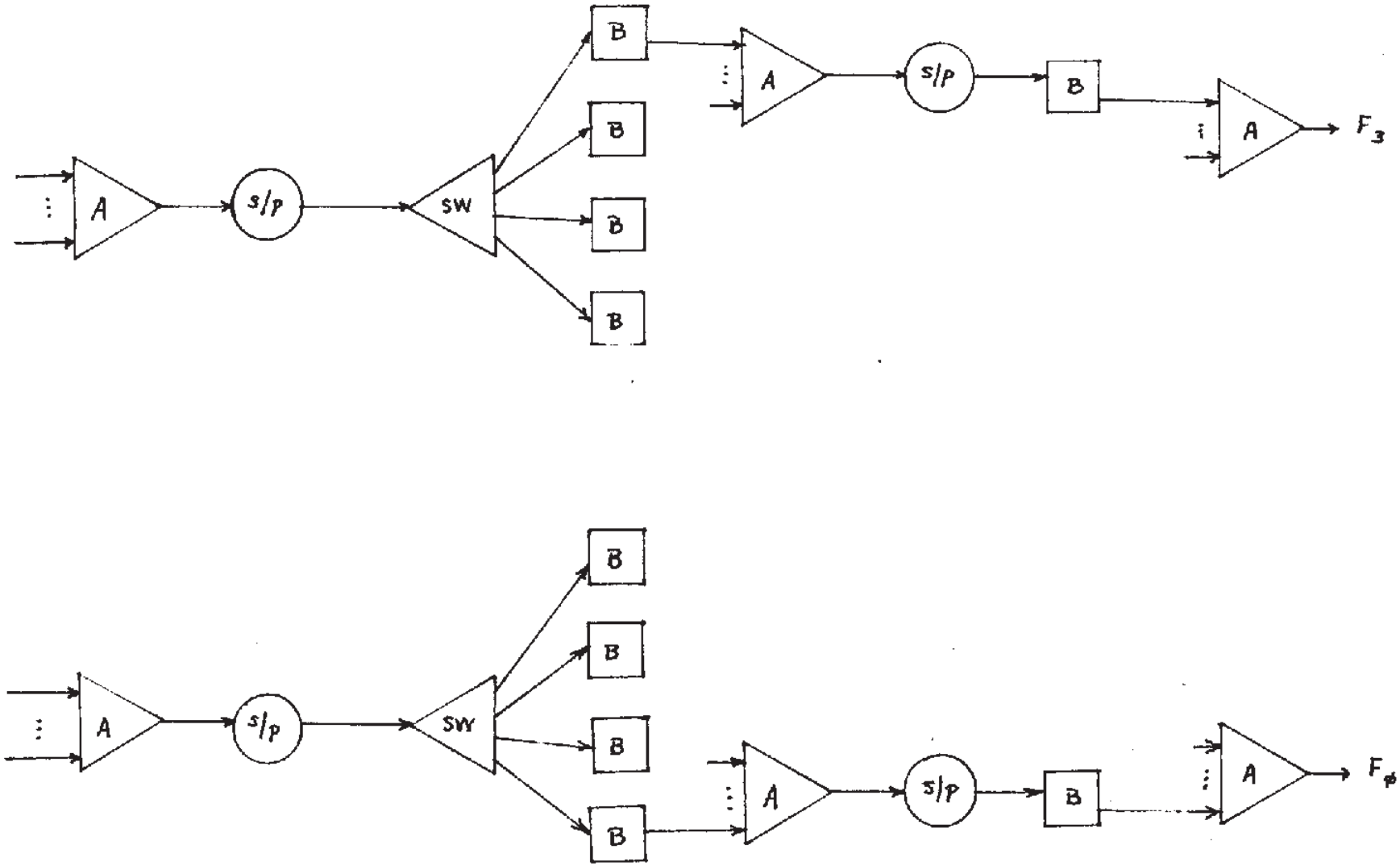
FIG. 2.2

cell, the cell is said to be enabled, and is referred to as an operation packet ready for transmission. An enabled operation packet sends a signal to the Arbitration Network requesting prompt transmission to the appropriate Functional Unit for processing. There are several distinct types of operation packets (9), but for purposes of this design an operation packet (will have the structure shown in Figure 2.3, and) will contain sixteen eight-bit bytes.

2.2 Arbitration Network

2.2.1 Function

An Arbitration Network is an asynchronous routing network that accepts packets at its input ports and transmits them concurrently to its output ports. In this implementation packets arrive at the input ports from the Memory Section and are routed to the output ports for passage to the Functional Unit Section. The arbitration network performs several different tasks in the entire routing process and therefore consists of several different units. The substructure of an Arbitration Network is given in Figure 2.3. It is composed of arbiters, switch units, serial-to-parallel converters, and buffer units.



ARBITRATION NETWORK

FIG. 2.3

The arbiter is the most crucial component of the Arbitration Network. The arbiter must accept all enable signals generated by the operation packets in the Memory Section. Once it has selected the appropriate packet for transmission to its output port, it must inhibit subsequent packet transmissions until the first one is completed. This control over the routing network is extremely important for both speed and reliability of packet routing. The arbiter is a rather complex module and is described in much greater detail later in this thesis (Chapter 3).

2.2.2 Switch Unit

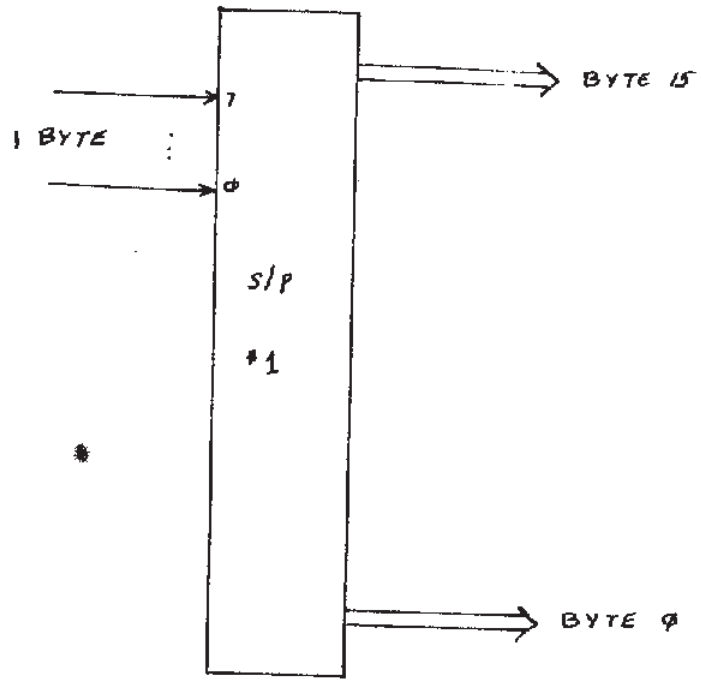
The switch unit has a single input port and several output ports. Its function is to assign a packet at its input port to the output port selected by information contained within the packet. The information required for this switch selection is the op code of the instruction, which is located in the two low order bits of the operation packet for the purposes of this design. The switch unit is entirely responsible for getting the operation packet to the appropriate Functional Unit.

2.2.3 The Serial-to-Parallel Converter

The serial-to-parallel conversion modules accept byte serial data at their input ports and convert it to a more parallel form at the output ports. As shown in Figure 2.4 there are two distinct serial-to-parallel units that differ only in the number of their input and output ports. The effect of the two modules is to convert byte serial data to entirely parallel data. The reason for the conversion is to increase transmission speed. By separating the process of conversion into two steps, the increase in overall transmission speed is achieved without tying up too much time in the actual conversion process.

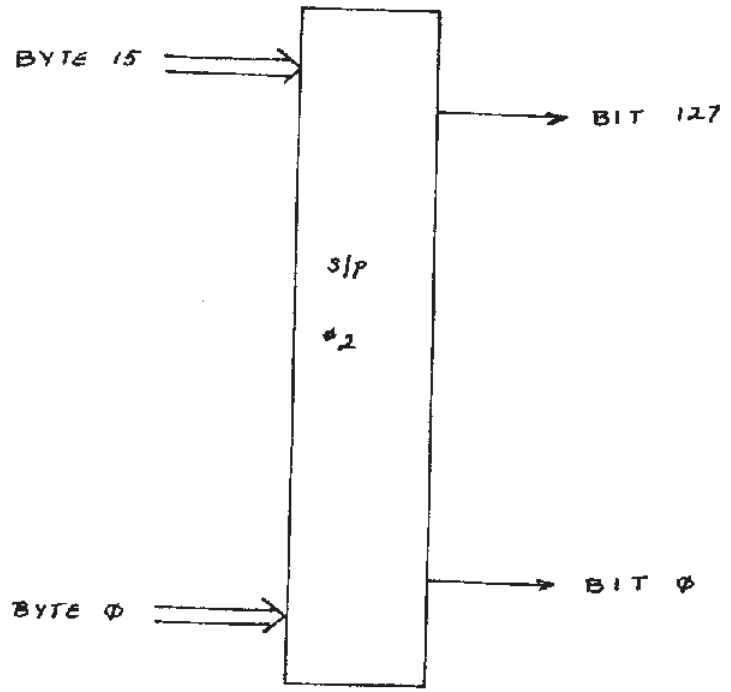
2.2.4 Buffer Unit

The buffer units are temporary storage areas for the operation packets. the idea of buffering the packets at various stages in the network allows for more parallelism in execution. As soon as a packet is stored in a buffer unit, a new packet is free to enter the arbitration network without loss of data. In this way, packets can be buffered so that the entire network can be in use at any given time. This is important since idle time in an Arbitration Network slows down the transmission process and defeats the effects of parallelism.



STAGE 1

SERIAL BYTE DATA → PARALLEL BYTE DATA



STAGE 2

BYTE PARALLEL DATA → BIT PARALLEL DATA

S/P CONVERSION UNITS

FIG 2.4

2.3 Functional Unit Section

The Functional Unit Section consists of the processing units which handle all computations within the data-flow processor. Data presented at its input ports in the form of operation packets is processed according to the instruction specified in each packet, and a result packet is generated. The result packet contains the result of the computation performed and the destination address(es), which specify the instruction cells that are waiting for this data.

For the purposes of simplifying this design, the Functional Unit is assumed to have four processing units, one for each of the four basic arithmetic operations. This structure helps to achieve faster execution by allowing concurrent processing, and by reducing idle time. Since instructions and operands arrive simultaneously from the Arbitration Network, the processing units (ALU's essentially) do not have to sit idle during memory acquisitions. As a result, execution speed and efficiency are obtained.

2.4 Distribution Network

The function of the Distribution Network is analagous to that of the Arbitration Network. It must accept result packets at its input ports, and channel the data in these packets to the instruction cell(s) specified as destination

address(es). This is the way in which results from one operation are used as operands in a second operation. The first instruction is processed in the Functional Unit Section, and a result packet is generated. The destination address(es) in the result packet specify the address of an instruction cell in memory. The data in the result packet is then placed into the appropriate operand register in this instruction cell. When all operands have been received, this instruction cell is enabled, and the cycle begins again.

2.5 Parallelism and Concurrency

There have been several allusions to the concepts of parallelism and concurrency. These concepts are intrinsic to the nature of data-flow and deserve a more in-depth explanation.

The idea of parallel computation suggests that more than one data computation can be performed at a given time. This is not the case in conventional architecture with its single arithmetic processing unit. The conventional computation scheme requires several memory access cycles to retrieve instructions, operands, and destination addresses. The delays inherent in such a scheme reduce the execution speed of the processor. In addition, the arithmetic logic unit, and other sections of the processor, remain idle during this time. It was

this type of idle time that led to the concept of data-flow. In the data-flow architecture there are several functional units, and therefore, several computations can be processed simultaneously. The instructions are selected for processing by the availability of data, and need not be invoked sequentially. (There actually is a particular sequence of operation, but it is also dictated by the availability of operands and is controlled by the data-flow language (3). By adding several stages of arbitration between the Memory and Functional Units, it becomes obvious that instruction execution becomes highly parallel. The crucial point in the control of such data-flow lies with the arbiter, which is a component in the Arbitration Network. Problems arise when more than one enable signal arrives at the input port of the arbiter. Given that they arrive at least far enough apart to be discerned as two distinct signals, then the arbiter must process the first one that arrives and queue the second one. The resolution of ambiguity will be addressed again in the actual description of the arbiter design. (Chapters 3, 5).

2.6 Modularity

The configuration of the Arbitration Network given in Figure 2.3 reflects a multi-level structure. The major reason for this scheme is to maximize the transmission of packets through the network. The buffering, serial-to-parallel conversion, and switching units are configured so as to allow several packets to pass through the network concurrently, but at different stages. This ensures that the network has less idle time and a faster throughput.

It must be noted that the specific configuration given is not the only one that can be used. The separate units are designed to be modular, and therefore interchangeable to a certain degree. Specifically, it is possible to buffer the packet either before or after the switch unit, or to combine both serial-to-parallel units into a single conversion process, etc. The actual configuration can be determined by the system implementer. There are advantages to certain configurations which will become more apparent after the design has been presented (Chapter 5).

Chapter 3

3.0 Arbiters

3.1 The Role of Arbiters in Parallel Processing

The allocation of shared resources in a circuit is controlled by signal arbitration. When several users desire access to the shared resource, an arbiter must be employed to ensure that only one user is granted access at a given time. In the data-flow processor, the Functional Units are shared resources, being used by all of the instruction cells in the memory. The role of an arbiter is to allocate these functional units to the instruction cells as they become available. It must also insure that the arrival of simultaneous requests will not result in glitches at the output or metastable states. A considerable amount of work on asynchronous arbiters has been done by Patil (6, 7) and others (8).

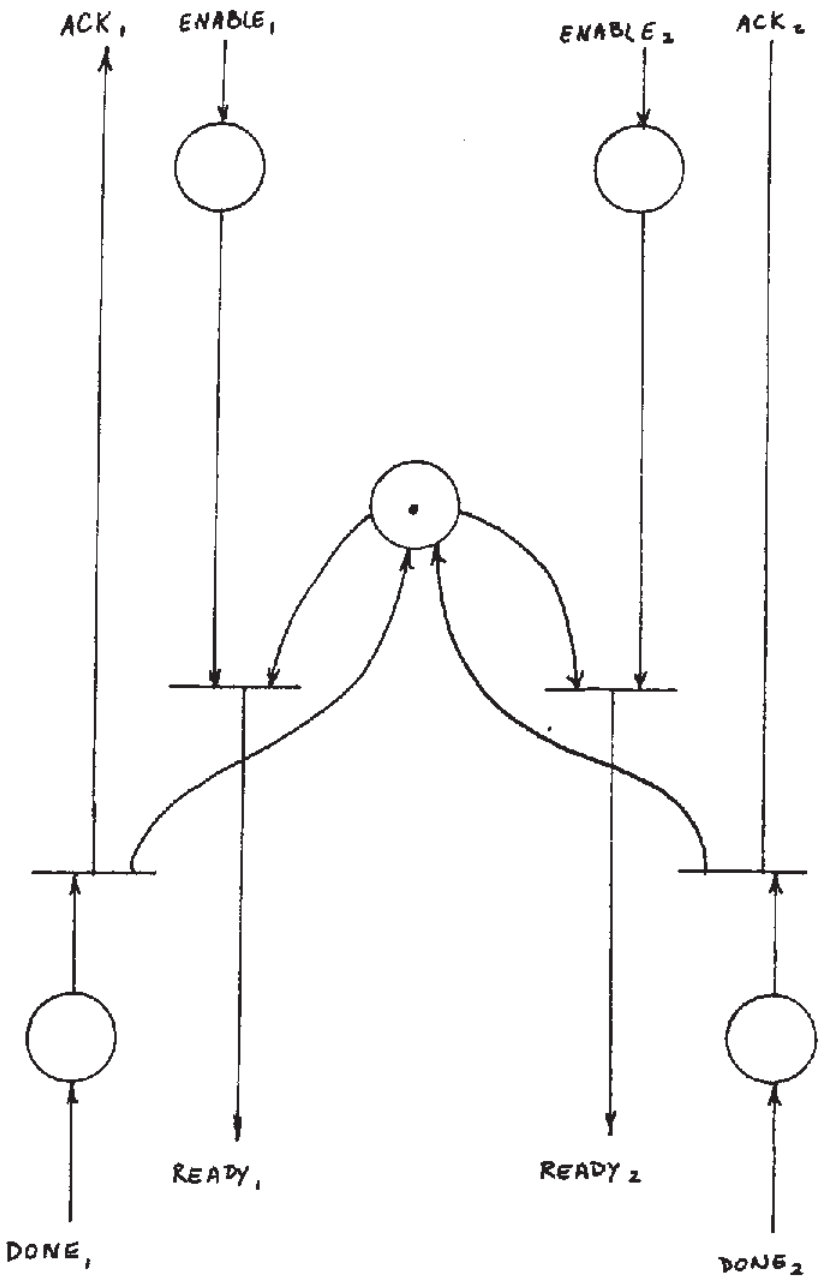
3.2 Petri Nets

A convenient method of describing an arbiter is by the use of Petri Nets. A Petri Net is a graphical means of system representation. It can be used to model information flow in a circuit, and is especially useful for modelling speed independent circuits such as the arbiter. The Petri Net was developed by Carl Petri (11) and was applied to speed independent circuits by Misunas (12).

A Petri Net is a directed graph with two types of nodes: transitions and places. Places are drawn as circles and transitions as straight lines. An input place has a branch from a place to a transition, while an output place has a branch from a transition to a place. Information flow within the net is handled by tokens. When every input place to a transition has a token in it, the transition is enabled and may fire at any time. When it fires, a token is removed from each of the input places and a token is put into each output place. There is no conservation of the number of tokens.

There is a way to block information flow in a Petri Net by using shared input places. An input place can have branches to more than one transition. When one of the transitions fires, the token is removed from the input places, including the shared input. Once the token is removed, the other transitions are no longer enabled, and must wait for a token to be placed in the shared input place.

A Petri Net representation of an arbiter is shown in Figure 3.1. With a token originally in the shared input place, the first enable signal to come in will cause the transition to fire. Once one of the transitions fires, the token is removed from the inputs, thereby disabling the other transition. The ready signal will cause the desired operation to be performed. When processing is complete, an acknowledge signal is returned, which acknowledges the enable signal (see transition signalling in Chapter 4), and enables all transitions again.



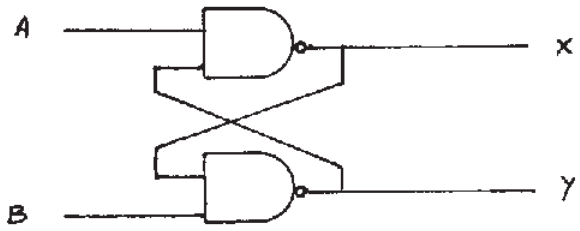
PETRI NET REPRESENTATION OF
AN ARBITER

FIG. 3.1

3.3 Metastable States

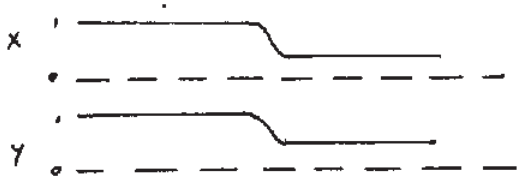
One of the major problems with an asynchronous arbiter is the proper handling of simultaneous changes in its inputs. If this situation occurs, and the arbiter is incapable of resolving the conflict, the arbiter may possibly enter a metastable state. This state is an unstable equilibrium state which bistable components can experience in addition to their two stable states of on (1) or off (0).

All bistable devices pass through a metastable state in the transition from a high to low or a low to high logic level. Consider the basic arbiter circuit shown in Figure 3.2a. If the inputs A and B are initially 0, the outputs will of course be 1. If both inputs change simultaneously to 1, the outputs may enter the metastable state shown in Figure 3.2b. In this unstable state, the output voltage does not reach the requirements of the threshold voltage needed to cause the transition from a high to a low logic level. The metastable state is shown graphically in the transfer characteristic shown in Figure 3.2c. While the device remains in this unstable state, its output is uncertain. It will eventually return to one of its stable states, but the delay incurred until this happens, and the possibility of glitches on the outputs, are undesirable. Care must be taken to ensure that the arbiter cannot enter a metastable state.



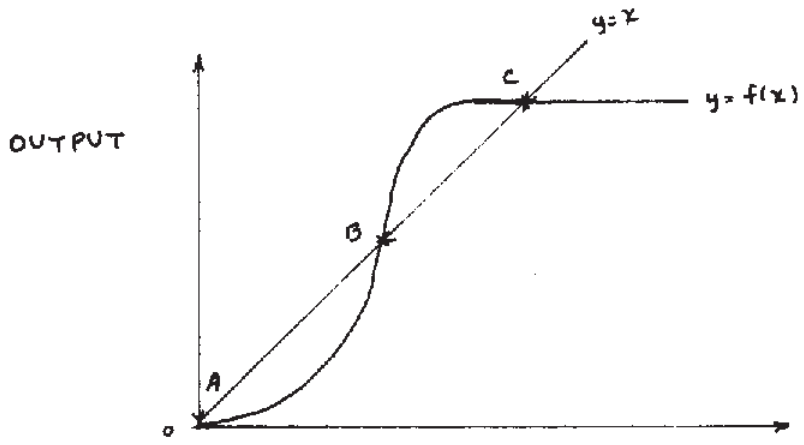
SIMPLE ARBITER CIRCUIT

FIG. 3.2 a



METASTABLE STATE

FIG. 3.2 b



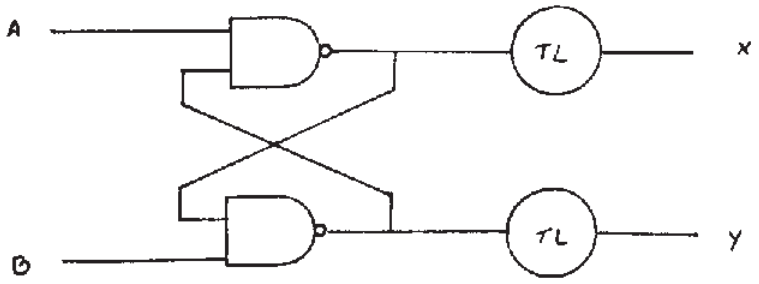
LOW TO HIGH TRANSITION FOR A BISTABLE DEVICE

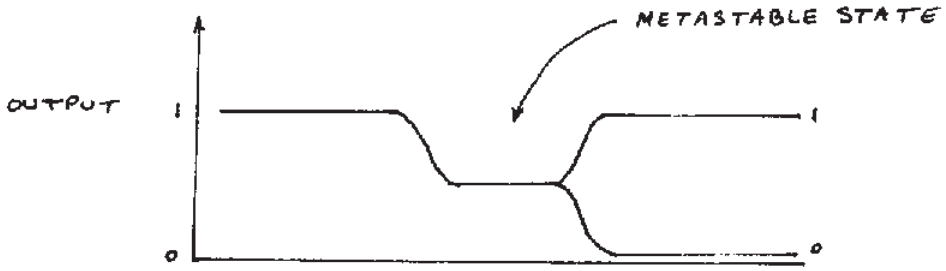
FIG. 3.2 c

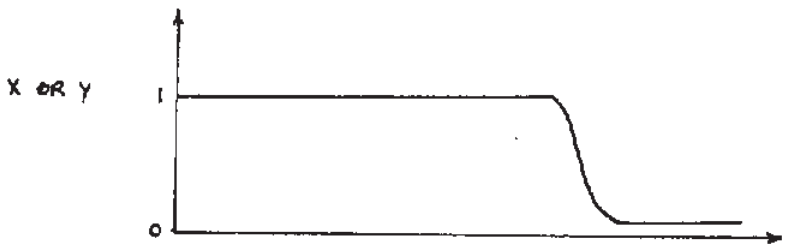
3.4 A Design for an Asynchronous Arbiter

A method of avoiding the metastable state is by connecting some threshold logic to the simple arbiter circuit. The resulting circuit is shown in Figure 3.3. The circuit now behaves as follows: initially both inputs are zero. The outputs are stable at the high logic level (1). If a single input changes, then the corresponding output will pass directly to a low logic level. There is no problem entailed in this particular transition. However, the situation is quite different if both inputs change simultaneously. In this case, the output levels of both gates will fall to an intermediate value between 1 and 0. The threshold logic will be preset either high for 0 to 1 transitions, or low for 1 to 0 transitions. This will prevent the output from changing until the circuit comes out of the metastable state and assumes a stable logic level, either high or low. The effect of the additional threshold logic on the arbiter is shown in Figure 3.3. It shows the original high output level, the metastable state, and the final output.

A realization for the threshold logic is the differential amplifier as shown in Figure 3.4. The output is produced by the differential amplifier when the outputs of the circuit differ by an amount larger than the level set by the offset voltage.

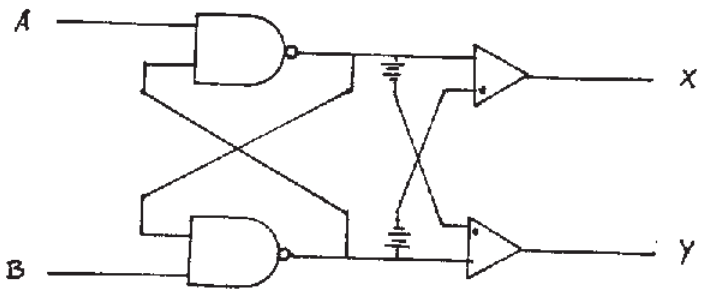






EFFECT OF THRESHOLD LOGIC ON
SIMPLE ARBITER CIRCUIT

FIG. 3.3



ARBITER CIRCUIT

FIG. 3.4

The arbiter circuit described in this chapter was developed by Patil (6). Other works by Patil on arbiters include (7).

Chapter 4

4.0 The Use of Synchronous Components in an Asynchronous Design

The Arbitration Network is an asynchronous circuit. Most of the current commercially available logic devices were designed for use in synchronous circuits. The use of such devices in asynchronous design require particular attention on the part of the designer to propagation delays and signal control.

The main criteria for the selection of components for this design was that they do not inhibit the execution speed of the processor, and that they are commercially available and economically priced. The reason for the first limitation is intrinsic to the concept of data-flow computation, wherein data availability, and not clock cycles, controls the flow of data within the circuit. The second limitation is merely sound engineering practice.

The inherent limitation in synchronous circuits is the need to wait for a clock pulse even after data is ready at the input and/or output ports of the component. The concept of data-flow decrees that this idle time should be eliminated. By doing so, data-flow speed can be increased, but proper signal control becomes much more difficult.

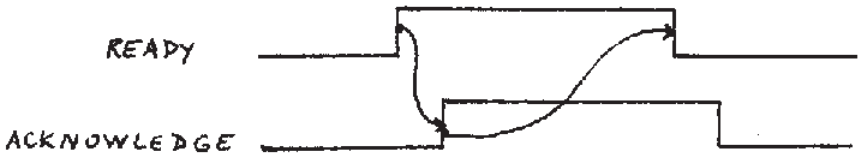
4.1 Transition Signalling

The signalling discipline selected for this design is transition signalling. A modular unit is invoked by means of a READY signal, and signals completion of its required task with an ACKNOWLEDGE signal. Not all modules will generate an ACKNOWLEDGE due to the structure of the network. For instance, the first level of the network will acknowledge each byte as it comes in, and the end of the entire packet. Therefore, the serial-to-parallel unit and the buffer will issue ACKNOWLEDGE signals back to the memory, but neither the arbiter nor the switch will do so.

Transition signalling is shown in Figure 4.1. The 0 to 1 transition on the READY line activates each module. Internal control then takes over and ensures that the proper data handling occurs. When this is completed, an ACKNOWLEDGE signal is returned, which resets the READY line. The ACKNOWLEDGE line then falls back to its low level. Both lines will remain low until the next READY signal arrives, and the cycle resumes again.

4.2 Propagation Delays

Clocked-input devices are not the sole source of time delays. Even the more asynchronous components, such as multiplexors and even SSI gates, are subject to propagation delays of varying length, depending on the



TRANSITION SIGNALLING

FIG. 4.1

TTL family of circuits considered. Of the five TTL series of components currently available, a wide variety of parameters allow for trade-offs in system design. For maximum speed, the 54S/74S series has been selected for this design, in spite of its fairly high level of power dissipation.

Even with asynchronous circuits, timing was a crucial element in this design. With conventional synchronous circuits, clock pulses can control data flow, but in asynchronous circuits, the designer must pay scrutinizing attention to propagation delays. The selection of 54S/74S TTL logic provided the basis for faster propagation times. In addition, however, each functional device must meet the criteria for fast switching times, or an alternate solution must be found. In this manner, some of the component selections for this design may not seem very sophisticated, but in a speed-independent circuit, simplicity is fast. For this reason such devices as FIFO'S, RAM'S, etc. were rejected as being much too slow for this network.

4.3 Allowance for Johnson Counters

The serial-to-parallel conversion units in this design employ clocked Johnson Counters, which are described more fully in Chapter 5. The inclusion of these synchronous counters does not detract too heavily from the execution

speed of the unit because of the way in which they are implemented. A Johnson Counter could have been designed with fully asynchronous SSI gates, but the gatings required to control the signalling, both with the serial-to-parallel unit and the adjoining buffer, would have resulted in longer propagation delays than the MSI Johnson Counter does. This is one instance where the use of asynchronous circuits would not have increased execution speed.

Chapter 5

5.0 Overview of the Arbitration Network

The Arbitration Network is a modular, multi-level routing network. The four basic modules are the arbiter, the serial-to-parallel converter, the buffer, and the select switch. As shown in the block diagram in Figure 2.3, each of these modules handles different forms of data at its input/output ports. The control scheme for similar units is the same, but the data lines differ, depending on the degree of parallelism which has been attained at that point in the circuit.

In order to control the flow of data between modules, tri-state devices have been used. The size of the resulting data bus differs also with the level of parallelism. At the first level arbiter the data is byte-serial, as there are 8 data lines. At the second level arbiter the data has been expanded to 4 bytes, thus there are 32 data lines. Finally, the third level of arbitration controls 128 data lines. Appropriate modifications have been made for each of the other modules in the three levels.

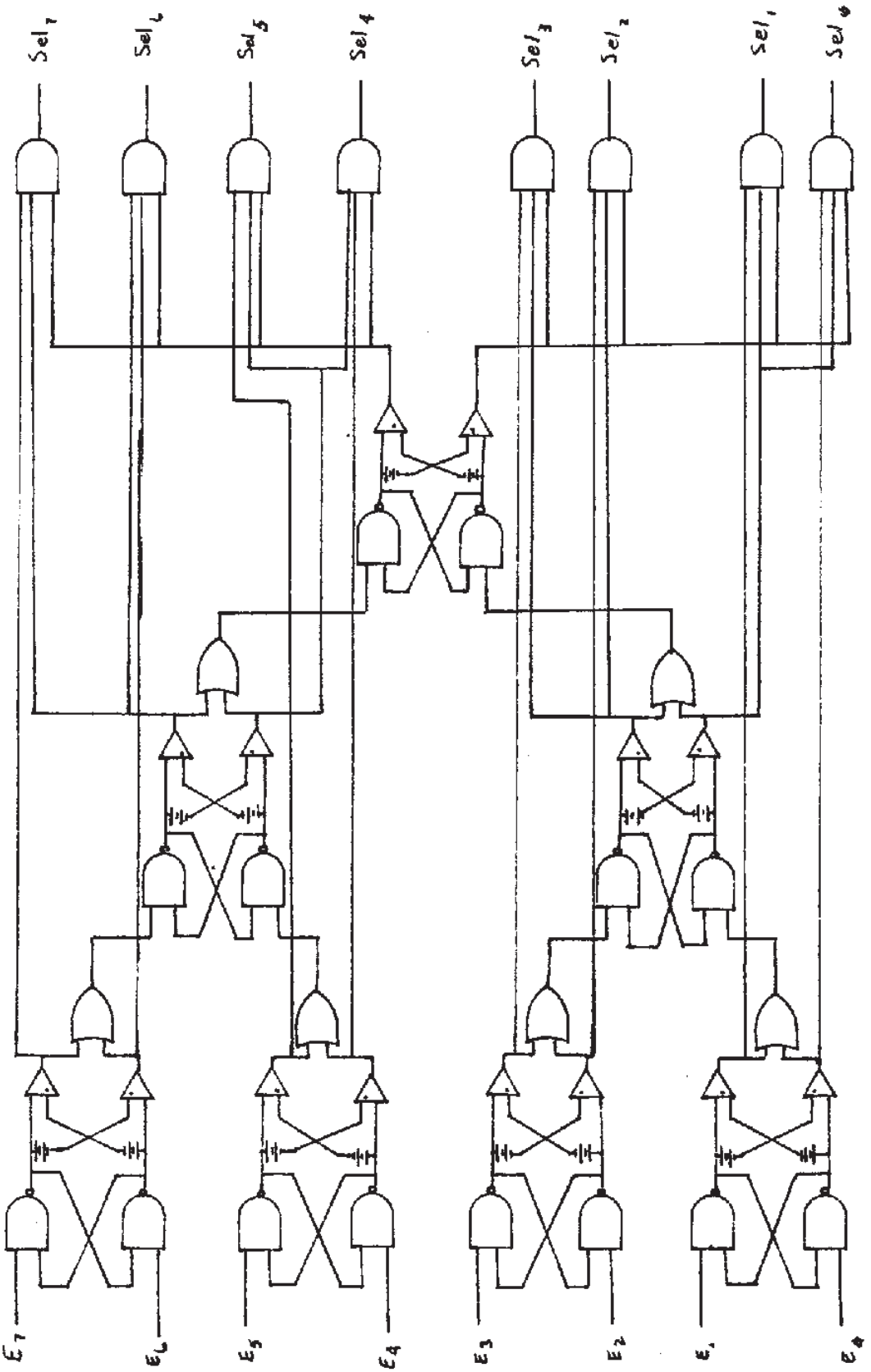
5.1 The Arbiter

To reiterate, the purpose of the arbiter is to select one of eight signals at its input ports on a first come, first serve basis. In a sense, it acts as a selector, and as such, merely allows the appropriate data signals onto

the bus at a given time. The basic eight-input arbiter is a three-level structure composed of the two-input arbiters discussed in Chapter 3. Additional gating is necessary, however, to determine which of the eight input lines actually got passed through the arbiter. This gating is in the form of AND gates and is shown as part of the arbiter in Figure 5.1. Note also in this figure that the use of a tri-level structure as opposed to a single level of arbiters permits a limited amount of queuing. This is a desirable feature since it helps to insure that signals are processed in the order in which they arrive. For instance, if two signals enter simultaneously only one will ultimately be passed through the structure, but the second will be passed through to the second level. As soon as the first signal is processed, and all inputs re-enabled, the second signal will pass through the third level and be processed.

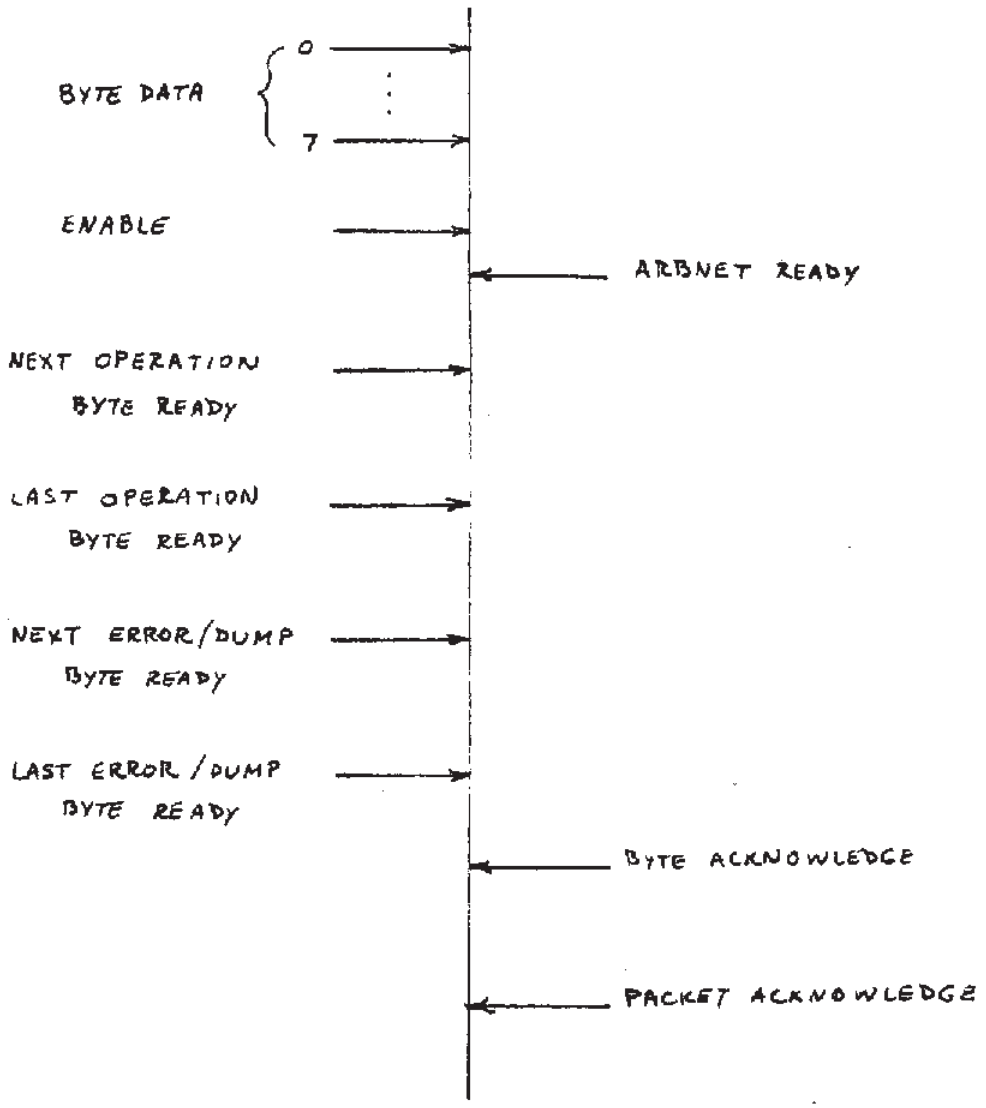
5.1.1 First Stage Arbiter

The first arbiter handles the interface signals from the memory section, which are shown in Figure 5.2. Its chief function is to select the set of signals to put onto the bus. Therefore, the signals are passed through a set of octal buffers with tri-state outputs. The schematic given in Figure 5.3 shows the select line control. When an enable signal gets passed through the arbiter, it selects the 8 lines of data and the 4 control



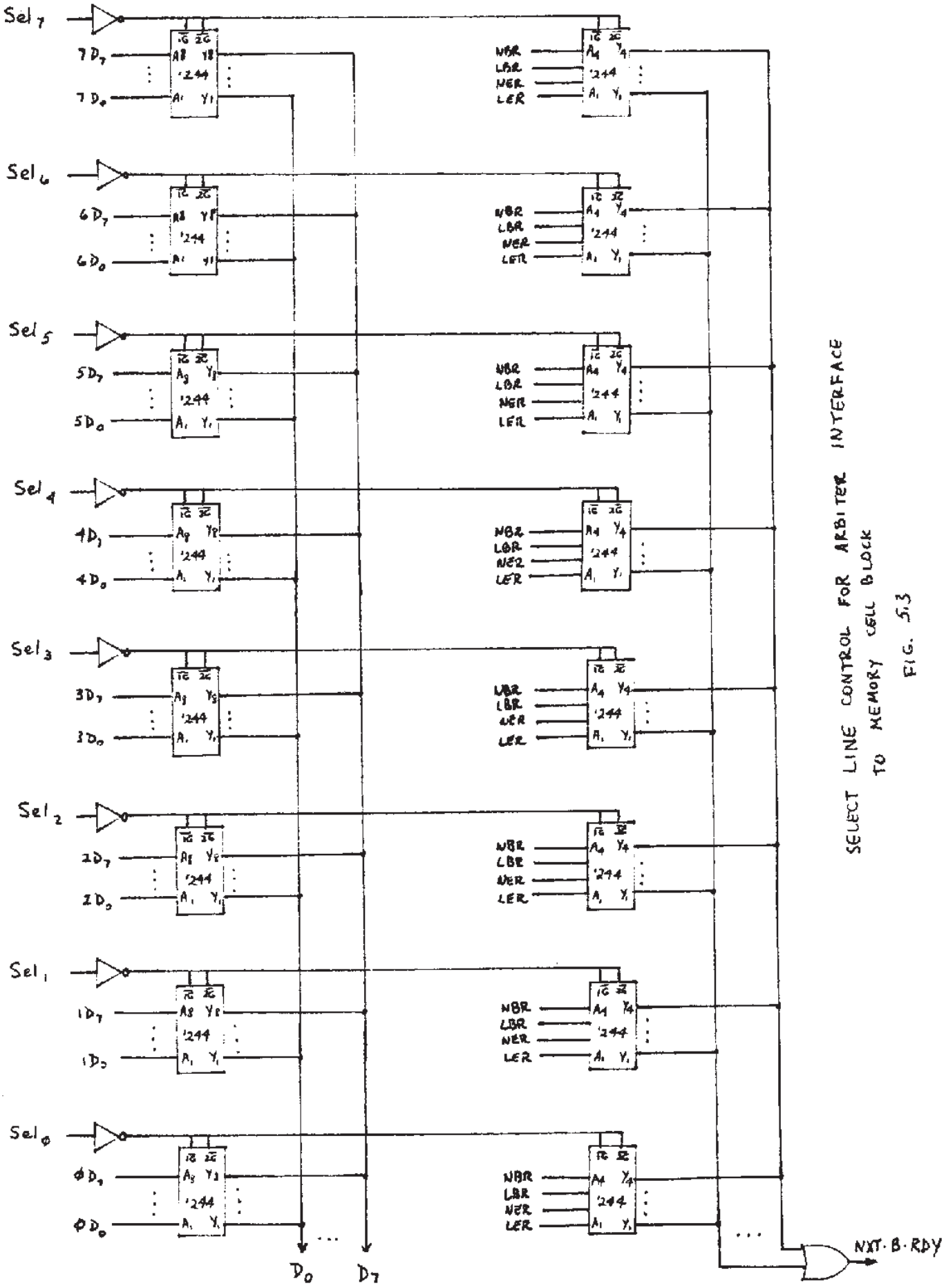
EIGHT INPUT ARBITER

FIG. 5.1



INTERFACE SIGNALS BETWEEN CELL BLOCK MODULE AND ARBITRATION NETWORK

FIG. 5.2



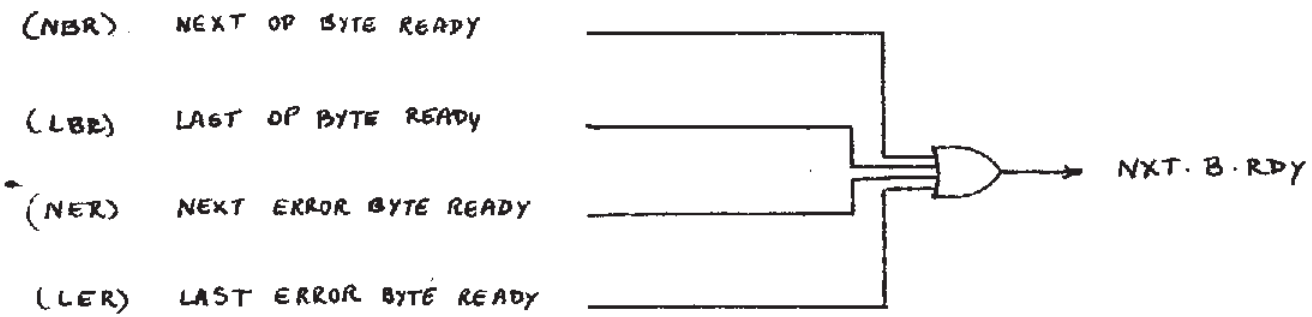
SELECT LINE CONTROL FOR ARBITER INTERFACE TO MEMORY CELL BLOCK

FIG. 5.3

lines to pass onto the bus. All other buffers maintain their high impedance outputs. This arbiter serves no additional function. Once allowed onto the bus, the control signals will direct the data flow to the next module.

There is one additional note about the control signals. The Arbitration Network passes 16 bytes of data as an operand packet to the functional units. This operand packet can be valid data, or it can be an error packet (9). The arbiter makes no distinction between these types of packets, however, as their contents are of no use to it. Therefore, a single control line is generated by OR'ing the four control lines, as shown in Figure 5.4.

As stated in Chapter 4, the arbiter itself does not issue an ACKNOWLEDGE itself. There are two ACKNOWLEDGE signals which go back to the memory, one of which is generated after each byte is received in the serial-to-parallel unit, the other after the entire packet has been stored in the buffer. These signals are described later in this thesis with regard to the respective modules from which they are generated.



CONTROL SIGNALS FROM CELL BLOCK MODULE

FIG. 5.4

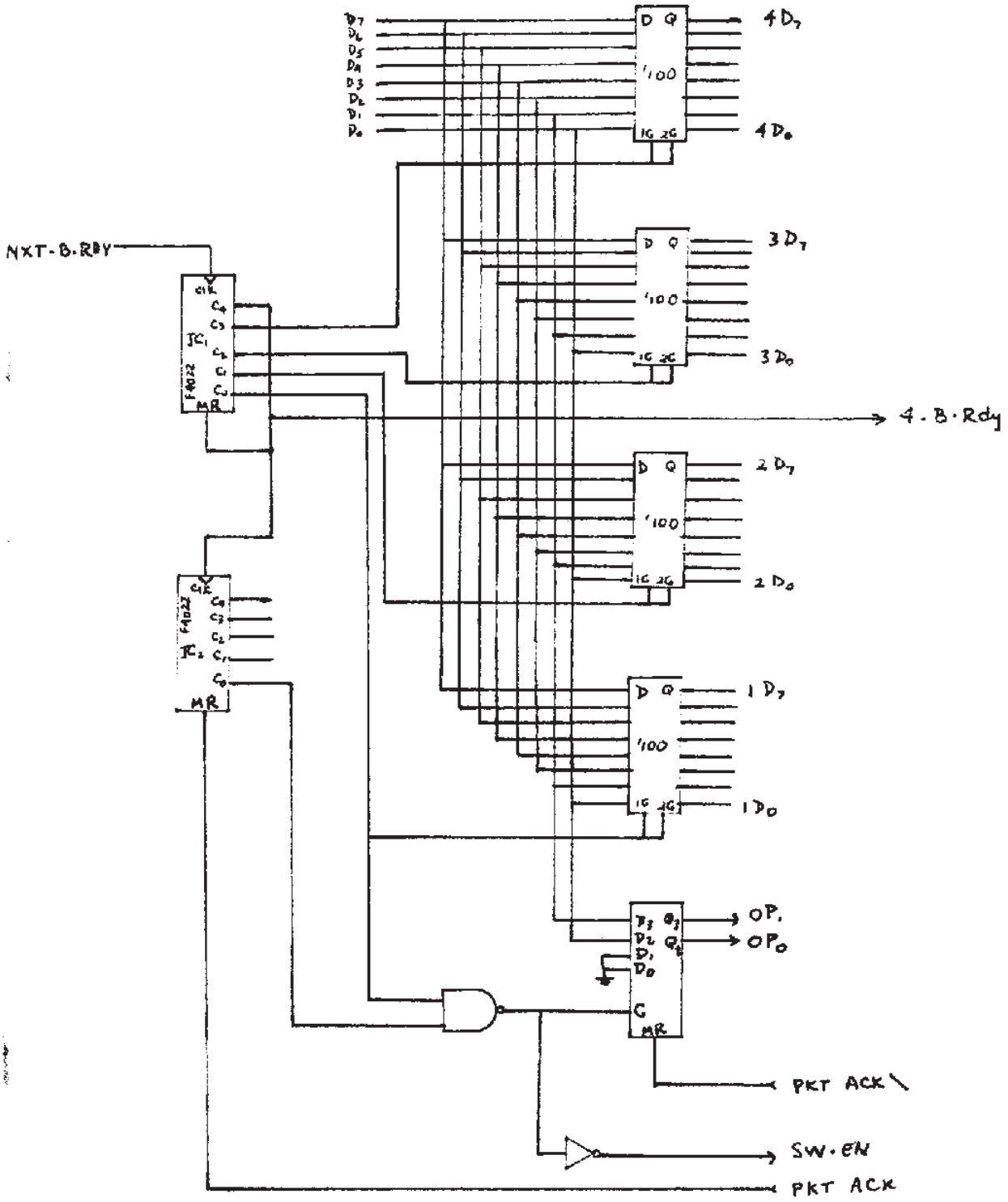
5.1.2 Second and Third Stage Arbiters

The control for the other two arbiters is generally the same as for the first, but the data is handled quite differently. With these latter stages, the eight input lines are READY signals generaged by the buffers to indicate that a packet is ready for transmission into the next stage. The arbiter processes these signals as they are received, and generates a single output, which is a select line as in the first stage. Since the buffers have tri-state outputs, this select line merely enables the outputs of one set of buffers.

The structure of these latter stage arbiters is the same as that of the first, i.e. a tri-level structure of 2-input arbiters and the AND gates to generate the select lines. In addition, however, the PKT ACK signals must be AND'ed with the enable lines to insure that the arbiter is not invoked until the previous data has been transmitted to the next stage. The PKT ACK from the second stage is generated from the buffer unit, while that from the third is generated from the functional unit.

5.2 Serial-to-Parallel Unit 1

The first S/P unit is shown in Figure 5.5. The module is activated by the Nxt·B·Rdy signal from the arbiter. As previously stated, this signal is obtained by OR'ing the four control signals selected by the arbiter. The main purpose of this module is to accept 8-bit bytes serially, and to transmit them to the buffer in a 4-byte parallel form.



SERIAL-TO-PARALLEL CONVERTER UNIT

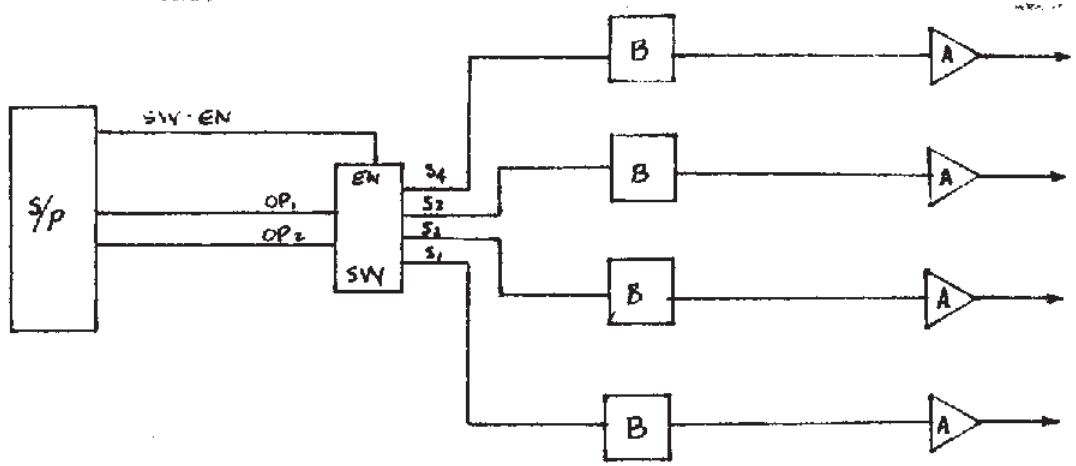
FIRST STAGE

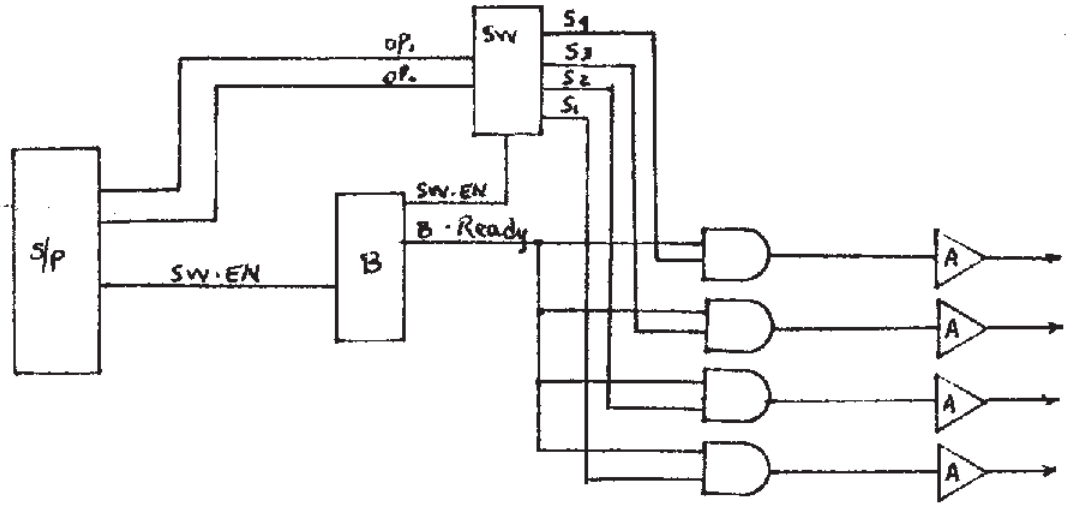
FIG. 5.5

The Johnson Counters, in the initial state, have a high output at C_o and low outputs on C_1 to C_4. When it is counting up, this counter will set each succeeding line high while dropping the previous one to a low level. Thus there will be only one high output at a time.

The arbiter selects the data and control signals to place on the bus. When the first Nxt·B·Rdy signal comes in, the first byte is stored in the latch labelled 1D. Also, the two low order bits of this byte, which have been defined as the opcode, are latched as OP_1 and OP_o. These signals will select the appropriate functional unit to which the packet will be transmitted. If the module which follows this S/P module is the switch, OP_1 and OP_2 will serve as inputs and SW·EN will enable the switch. Alternatively, if the next module is a buffer (the only other choice) then SW·EN will serve as the select line on the buffer. See Figure 5.6 for a clearer description of this interchangeability of modules.

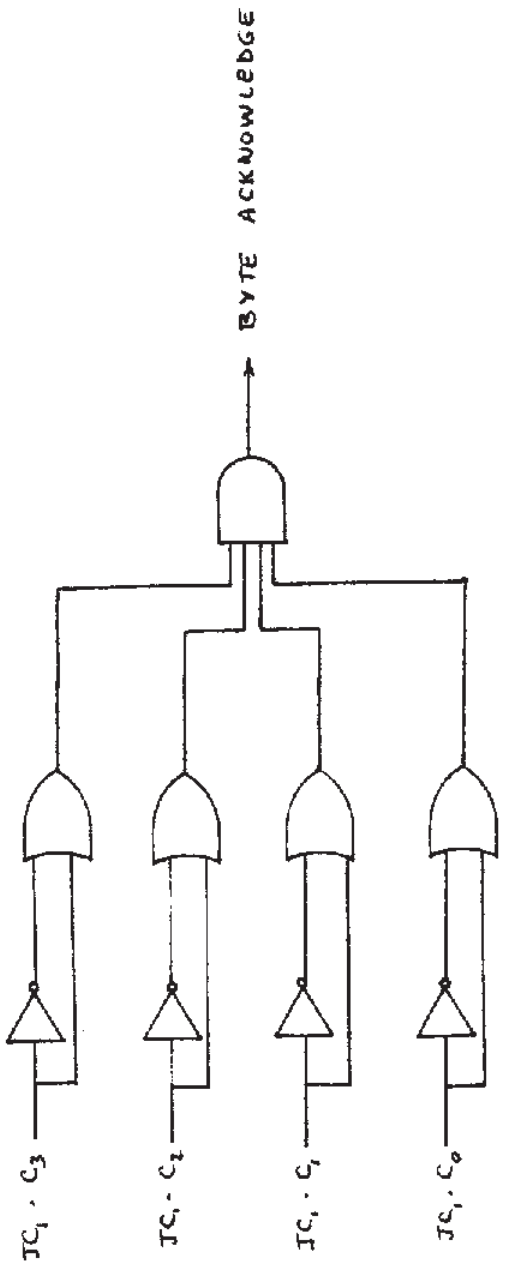
As the first Johnson Counter is incremented, successive bytes are latched into latches 2D, 3D, and 4D, each time returning a BYTE ACK signal to the memory. (see Figure 5.7) When four bytes have been received, the counter has a high output at C_4 which is the 4-B·Rdy signal. This signal notifies the buffer that four bytes are ready for storage. It also resets this Johnson Counter and increments the second Johnson Counter. This second counter will make sure that





INTERCHANGE OF BUFFER AND SWITCH MODULES

FIG. 5.6



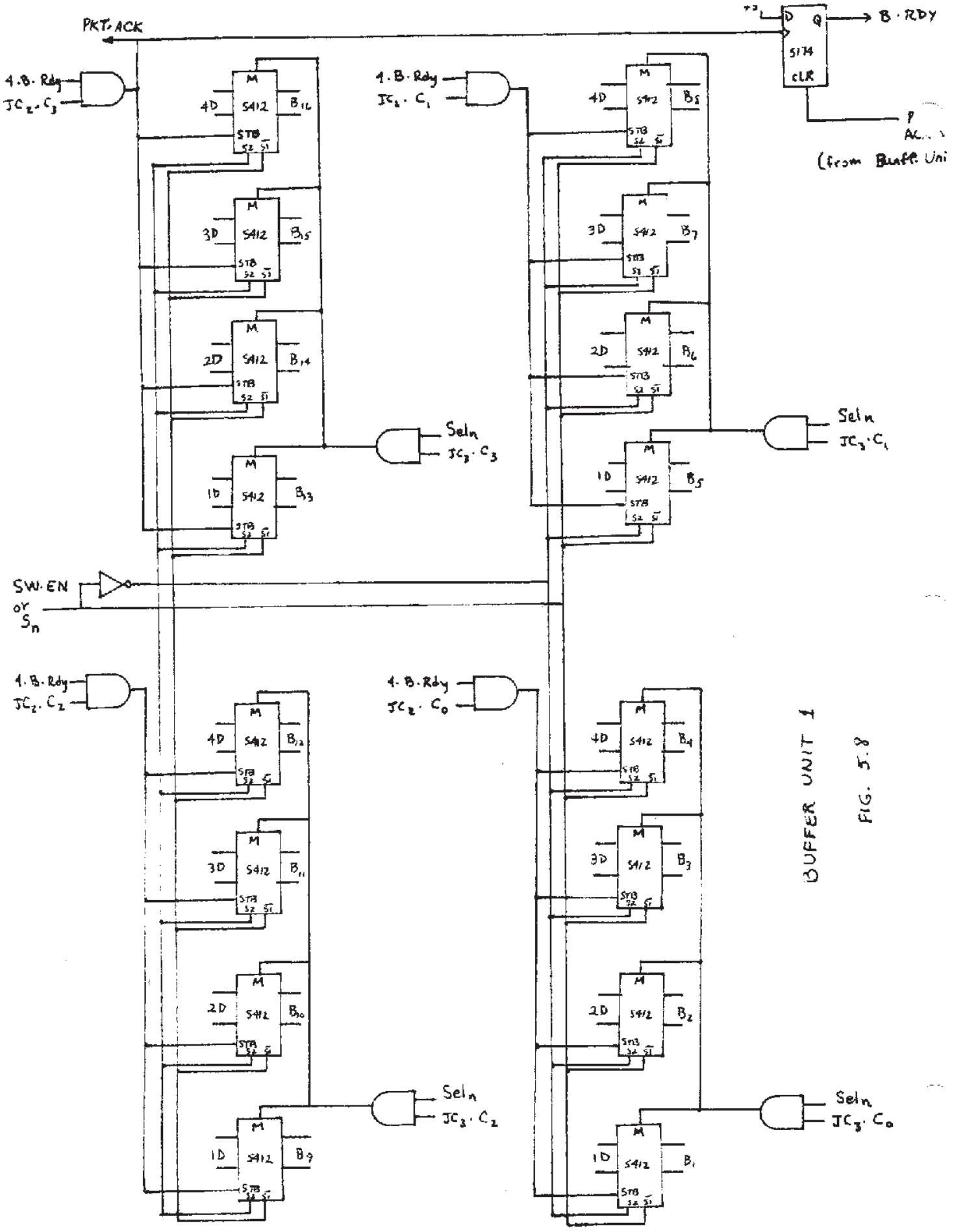
BYTE ACKNOWLEDGE FROM S/P UNIT TO MEMORY CELL BLOCK MODULE

FIG. 5.7

four sets of four bytes each are stored in the buffer. The outputs of this second counter will select the appropriate bank of latches in the buffer in which to store the data. When all four sets of bytes have been transmitted and stored, the buffer will issue a PKT ACK signal which will reset the S/P unit.

5.3 Buffer Unit 1

The first buffer unit, whether it comes before or after the switch module, serves as the end of the first stage of the arbiter. The schematic for this module is given in Figure 5.8. It consists of four sets of four 8-bit latches with tri-state outputs. The buffer unit is selected by either the output of the switch module (S_n) or the select line from the S/P unit (SW·EN). Initially the outputs are all at a high impedance state. When selected, (SEL), the latches become transparent, presenting whatever is on the inputs at the outputs. Then, when the STROBE is taken low by the gated signal, the input data is latched into the appropriate bank of four latches. The Johnson Counter in the S/P 1 unit controls the latching. The latching of the fourth set generates a PKT ACK signal, which indicates that the entire operand packet has been stored in the buffer. It also generates a BUFF·RDY signal to the second arbiter. When the buffer is deselected, the



BUFFER UNIT 1

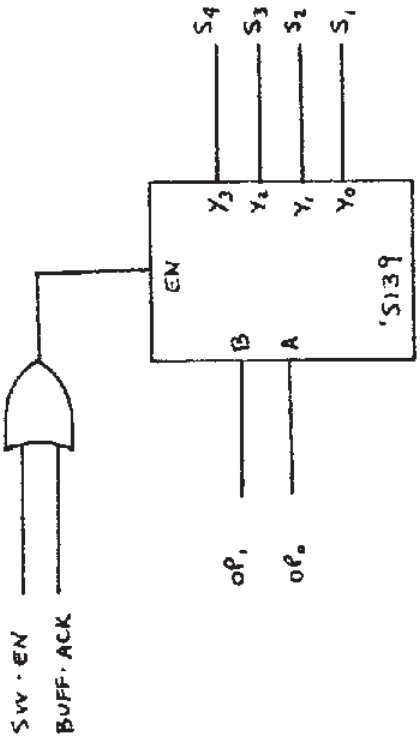
FIG. 5.8

outputs return to their high impedance states. The PKT ACK signal also generates a ready signal to the second arbiter. The data will remain in the latches until selected by the arbiter for further transmission.

5.4 Switch Module

The switch module is solely responsible for ensuring that a given packet is routed to the proper functional unit. The switch module is shown in Figure 5.9. It is a fairly simple module, and is controlled by the first S/P unit and one of the buffers.

The two low order bits in the first byte of the operand packet are designated as the op code in this design. As such they represent which of the four functional units must be used to handle the packet. The actual switching process consists of using these two bits to select one of four buffers or second stage arbiters, depending on the system configuration (see Figure 5.6). When the first byte of the packet enters the first S/P unit, it is latched, and the op code bits are also latched. No data transfer occurs until all four bytes have been latched, therefore there is more than enough time for the switch to stabilize its outputs. The outputs will remain enabled until the switch is no longer needed. In the case where the switch is placed before the buffer units, the BUFF·ACK signal will reset the switch. If, however, the data is



SWITCH MODULE

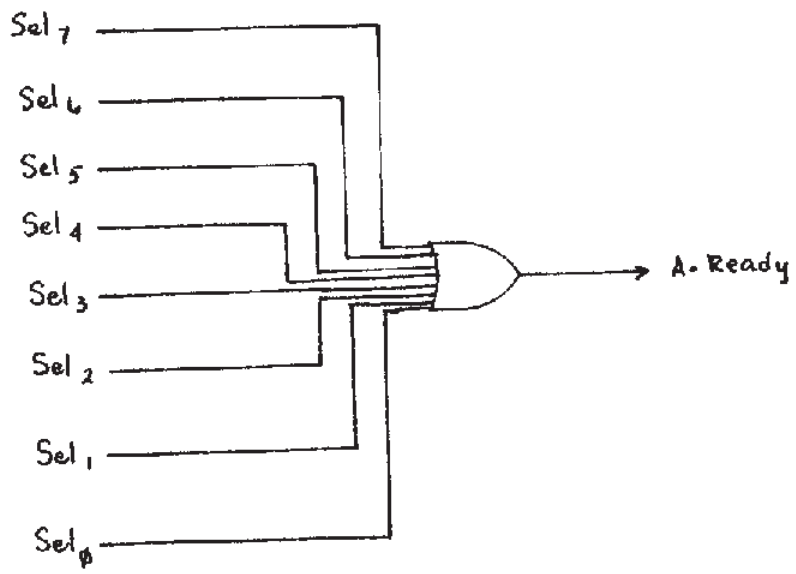
FIG. 5.9

buffered before the switch, the switch will select the appropriate arbiter and remain selected until the BUFF ACK from the second stage buffer is received. Note that in the second configuration in Figure 5.6 there must be additional gating in order to use the switch to select the arbiter. The only inputs to an arbiter are ready signals which in this case are generated by the buffer. In order to select the arbiter, the switch select lines must be inverted (to an active high level) and AND'ed with the BUFF.Rdy signal.

5.5 Serial-to-Parallel Unit 2

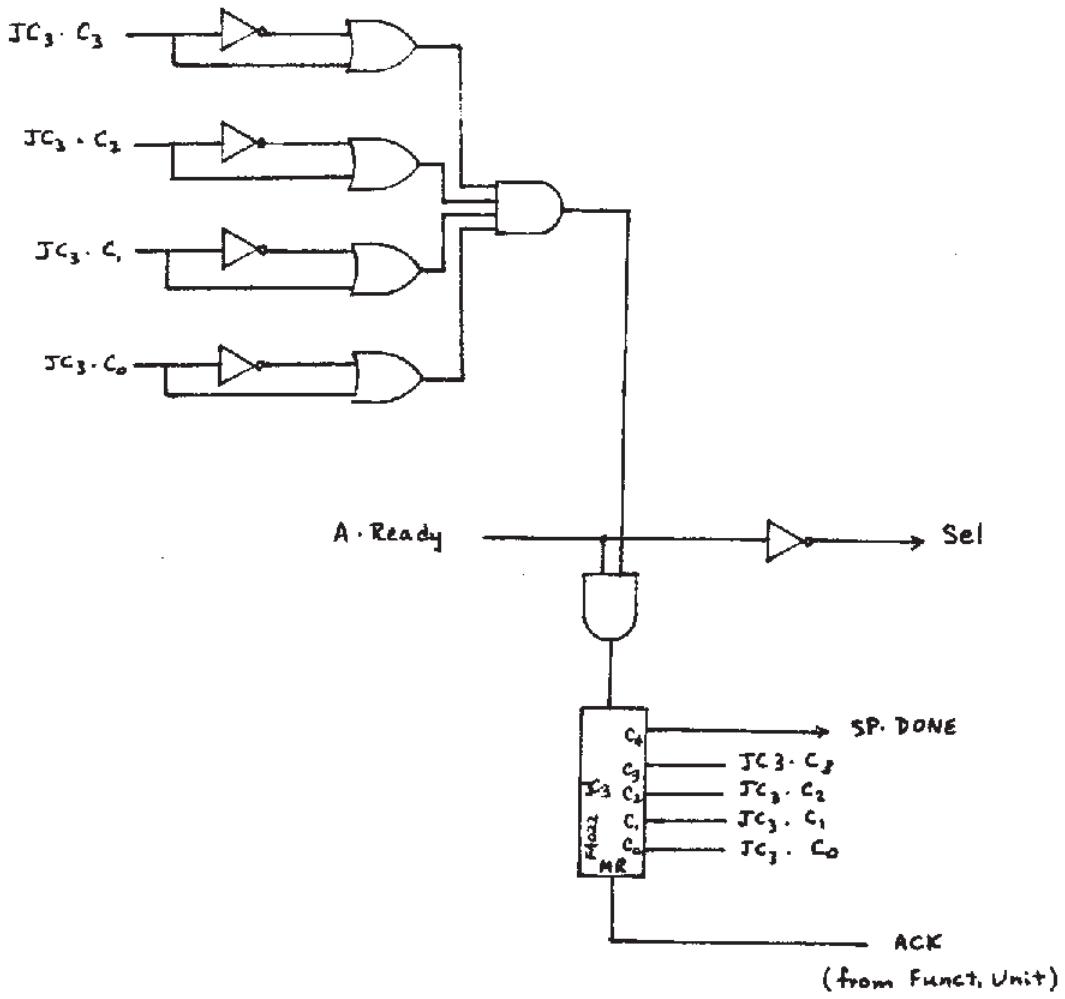
The second S/P unit, unlike the first, does not need temporary storage. The input data is four bytes in parallel, and there are four sets of input. The data is latched into the second buffer as it comes in, thereby making the buffer outputs entirely parallel data on a 128 bit bus.

The control for this S/P unit is similar to that for the first unit. However, since there are only four sets of input, as opposed to sixteen, there is need for only one Johnson Counter. The second S/P unit and buffer unit cannot be separated, so the data lines are not shown in Figure 5.10. The S/P unit is invoked by a READY signal from the arbiter. This signal provides the first clock pulse and latches the first set of four bytes into the buffer. Subsequent clock pulses are generated by the



SECOND AND THIRD STAGE ARBITER
READY SIGNAL

FIG. 5.10 a



SERIAL - TO - PARALLEL CONVERTER UNIT
STAGE 2

FIG. 5.10 b

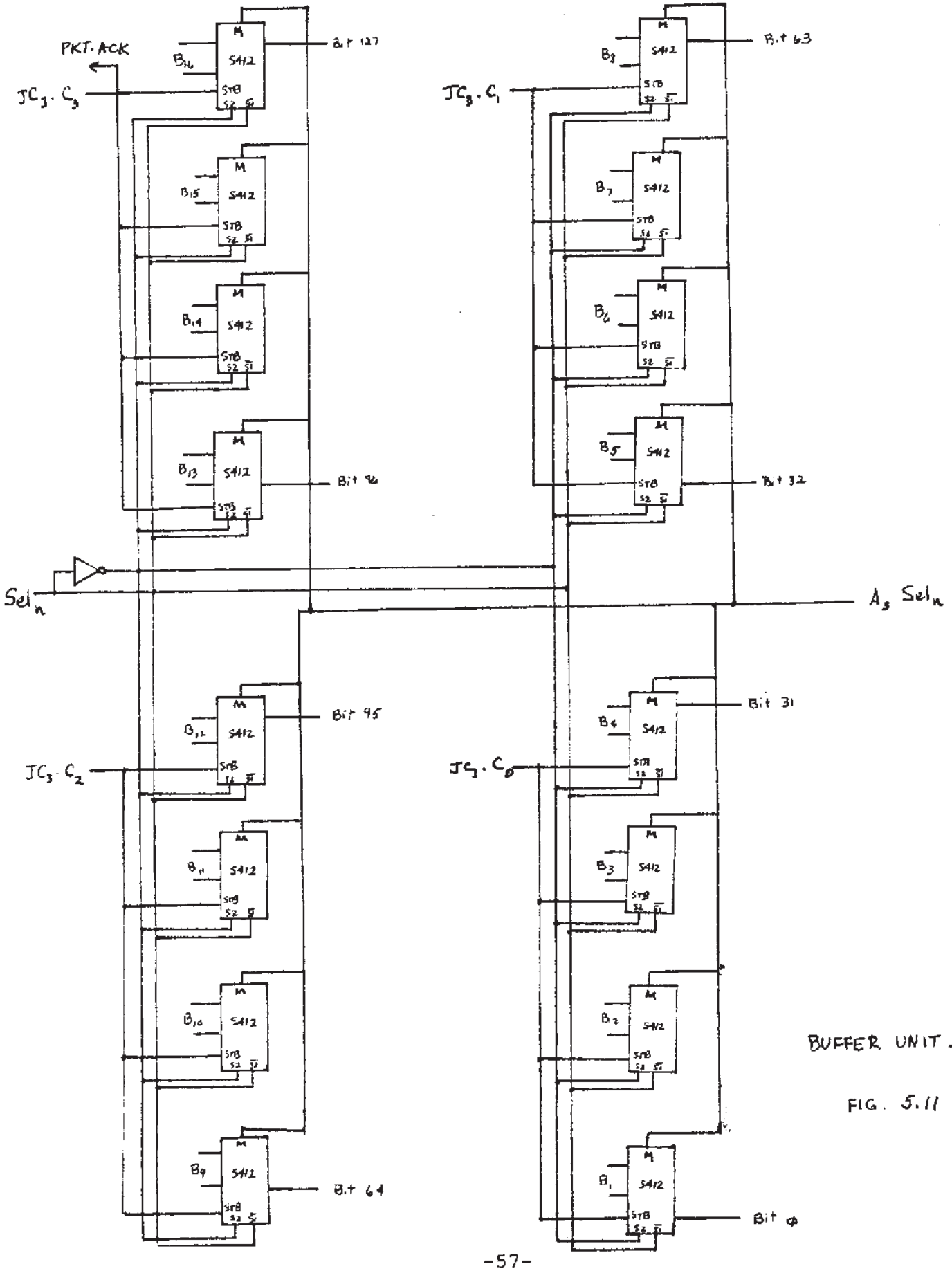
counter itself. Due to the inner operation of the Johnson Counter, and the fact that the output must go through four levels of gating before it can issue the next clock pulse, there is ample time for this type of clocking. As in the first S/P units, the outputs C_o through C_3 latch data, and C_4 serves as the READY line. In this case, however, once C_4 goes high, the last section of data will have been stored, so the C_4 signal is also an ACKNOWLEDGE back to the first buffer. This allows the first buffer to accept new data now that its original data has been stored in the second stage of the network.

This second stage of the network must keep this data until it receives an ACKNOWLEDGE from the functional unit.

5.6 Buffer Unit 2

The second buffer unit employs tri-state latches just as the first one does. The input to the buffer is four bytes in parallel, which means a 32 line data bus. The output of the buffer is the 128 line bus which supports the operand packet in its totally parallel form. As stated in section 5.5, the second S/P unit cannot be separated from this buffer. This is also true of the first stage of the network, and exists merely because the data must be stored after it is converted. There is no way that the S/P unit should be required to store the data that it converts, that's what the buffers are for. The S/P unit determines the structure of the data that goes in the buffer.

The structure of this buffer unit is given in Figure 5.11. The latching scheme is similar to that in the first buffer. The latches have high impedance outputs when not selected. The select line is generated by the arbiter. Once selected, the latches become transparent, displaying at the outputs whatever is at the inputs. The data is latched by the appropriate transition of the counter. For instance, the C_o output of the counter is initially high. When counted up, the first four bytes are stored in the latch whose outputs are lines 0 to 31 on the bus. The second set of data



BUFFER UNIT 2

FIG. 5.11

gets latched into the buffer at lines 32 to 63 etc. The completion of the S/P and buffer units is signalled by output C_4 of the counter. This will deselect the buffer, returning the outputs to high impedance dates.

When the third arbiter selects a buffer from which to get the data for transmission to the functional unit, the output enable line on the buffer is taken low, and the 128 bits are put on to the bus. The data is processed in the appropriate functional unit, and the functional unit sends an ACKNOWLEDGE signal to reset the S/P unit.

5.7 Overview of Network Signal Control

The schematics presented within this chapter show the transition signalling discipline that has been selected to control the information flow. The Arbitration Network was designed on a modular basis so as to simplify the signalling control and to provide as much flexibility as possible in the network configuration. With such a scheme, two types of signal control become important. The inter-modular signals must ensure that data flows sequentially through the routing network at the fastest possible speed; and intra-modular signals must control internal data flow. Transition signalling is ideally suited to the modular concept. Treating the module as a complete unit, and placing data on a common data bus, each module is invoked by means of a READY signal. This signal initializes the module and starts

its internal operation. When the module has completed its specified function, it returns an ACKNOWLEDGE signal. The module then remains inactive until the next READY signal arrives.

Due to the nature of this network, care must be taken to ensure that the proper READY and ACKNOWLEDGE signals are connected to each module. As pointed out earlier in this chapter, not all modules return ACKNOWLEDGES to their immediate predecessors. The network has three stages, and signals actually control stages as opposed to individual modules. As an example, recall that the PKT·ACK signal generated by the first stage buffer returns to the Memory Cell Block, and merely passes by the remaining modules in the first stage. The control signals required for each module have been specified, and therefore all connections can be easily handled by the network integrator.

5.8 Intra-Modular Signal Control and Timing

The most obvious approach to handling information in a data-flow processor would be to detect the arrival of data at a particular point in the network, and to use this signal to control latching and data transfer. Theoretically, this would provide for the fastest execution speed. However, the gating required to detect the arrival of new data is also subject to propagation

delays. With a limited selection of SSI and MSI devices, this detection circuit becomes costly, in terms of both time and space. As an alternative scheme, it becamse apparent that by allowing for minimum and maximum propagation delays in the devices, that an acceptable execution speed could be attained at a lower cost than the detection scheme. The main idea behind the delay-controlled signalling scheme is to allow ample time for the data to be latched or transferred. The first stage of the arbiter is controlled by the NXT·B·RDY signals generated by the Memory Cell Block. Once the data has been stored in the buffer, it can be manipulated by the circuit since it is no longer arriving at arbitrary times. This simplifies control considerably.

The following descriptions of the modules are supported graphically by timing diagrams in the Appendix.

The first arbiter's operation is fairly straight-forward. Any delays incurred in the gating occur before the drivers are selected. Once the appropriate select line enables the output lines of one of the drivers, the interface signals assume control. When the NXT·B·RDY signals come in, data is already stabilized on the input lines. The arbiter merely allows the data to get on to the bus to the S/P unit.

When the data reaches the first S/P unit, the Johnson Counter assumes internal control. The NXT·B·RDY clocks the counter. The Memory Cell Block sets up the data on the bus before issuing the NXT·B·RDY signal. The propagation delay for the latches in the S/P unit ranges from 14-25 ns. The propagation delay for the Johnson Counter is minimally 35 ns for high-to-low transitions, and 45 ns for low-to-high transitions. Thus, when the counter is clocked, there is sufficient time for the data to be latched before the next byte comes in.

The switch module requires no time of its own. It is enabled when the first byte is latched, and remains selected until the entire packet has been stored in the buffer. Since the data is not transferred out of the S/P unit until the fourth byte has been received, the switch module is already selected and its outputs stablized. Therefore there is no time delay generated by the switch itself.

The first buffer is selected by either the switch or the S/P, depending on the network configuration. The select lines require 21-35 ns to enable the latch inputs. Since this also is selected with the first byte, the buffer is selected in plenty of time to accept the data. It will remain selected until the 16 bytes have been stored. The four byte sections are latched on the high-to-low transition of the second Johnson Counter. The gated

STROBE (7 ns AND gate maximum delay) and the latch (25 ns maximum delay) are still within the 35 ns minimum delay of the Johnson Counter. This means that data will be latched before the next NXT·B·RDY signal can come in. The buffer is deselected by either clearing the switch or the S/P select line, both of which are cleared by the PKT·ACK signal.

The second arbiter is self-contained, as opposed to the S/P and buffer units which share signals, and presents no timing problems.

The second S/P unit operates similar to the first. The A·READY signal is generated by the arbiter to start the Johnson Counter and to select the buffer. The data transfer involved is merely to accept 4-byte input data and load it sequentially into the buffer, producing a totally parallel output. The high-to-low transition of each of the Johnson Counter outputs latches the data into the buffer. The data which is on the input lines was selected by the arbiter also. The A·READY signal was generated before clocking the counter. The summation of this gated delay, and the propagation delay of the counter total at least 42 ns, whereas the output of the first buffer is enabled within 35 ns (maximum). Therefore, the data will be stabilized on the bus in time to latch properly. Once started, the Johnson Counter uses its own outputs to clock itself. Although this would possibly

generate problems in other implementations, the additional gating (i.e. the edge-catcher) generates a delay of at least 22 ns. This delay can be increased to ensure proper operation by merely using chips other than 74S series TTL. The ACK from the functional unit will clear the counter.

Chapter 6

6.0 Conclusion

This completes the description of the Arbitration Network. The assumptions made to restrict the scope of this thesis can be eliminated, and the design described herein expanded, to achieve a more functionally complete routing network. These assumptions include the size of the packet (16 bytes), which is actually variable in the real data-flow processor; the size and location of the op code; and the size of the Arbitration Network itself. The actual Memory Cell Block would contain considerably more than the assumed 512 instruction cells. This would necessitate an expansion of the Arbitration Network using the same basic modules.

The actual cost of implementing this design would be very costly in terms of space. Employing the assumption that there are 512 Memory Cells, the number of each type of module in the Arbitration Network is as follows:

	Stage			
Type	1st	2nd	3rd	Total
Arbiters	64	32	4	100
S/P 1	64			64
S/P 2		32		32
Switches	64			64
Buffers	256	32		288
				548

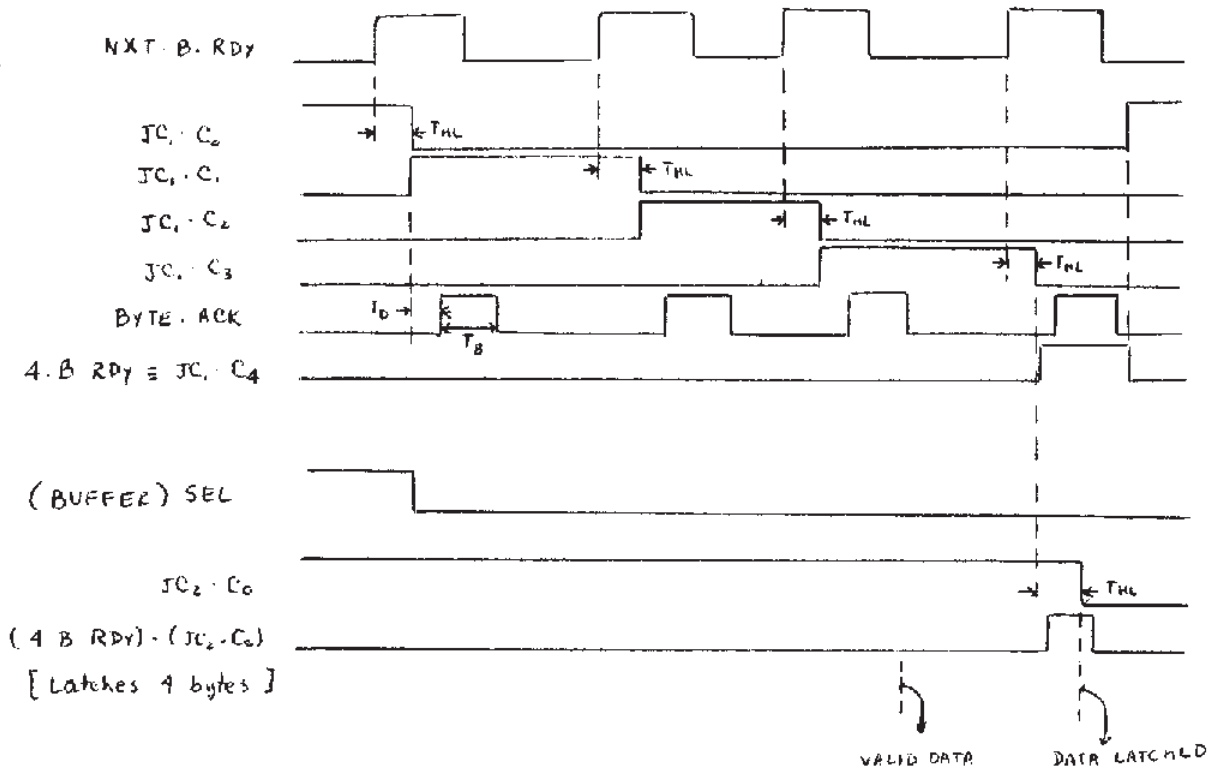
This figure states that there are approximately 548 modules in this structure. When considered in terms of IC's, there are more than 6700. This represents an incredible amount of board space. Even if built in a single rack, transmission line delays and noise become significant problems. These problems suggest the need for an alternative implementation. One alternative worthy of consideration is custom-designed LSI chips. In order to handle the large number of bus lines, an LSI design would have to put an entire path, or section of a path, on a single chip, as opposed to placing individual modules on the chip. A second alternative would be to eliminate, or reduce the extent of, the serial-to-parallel conversion. However, this will decrease execution speed and place additional constraints on the Functional Units.

The design presented in this thesis is an attempt at the realization of a fairly complex routing network. It serves to illustrate the nature of the problems involved and proposes some ideas for their solution. With the commercially available components utilized in this design, this network cannot be built economically. Advanced technology and alternative schemes must be considered in making this into a feasible project.

APPENDIX

Timing Diagrams

Acceptance and Latching of First Four Bytes



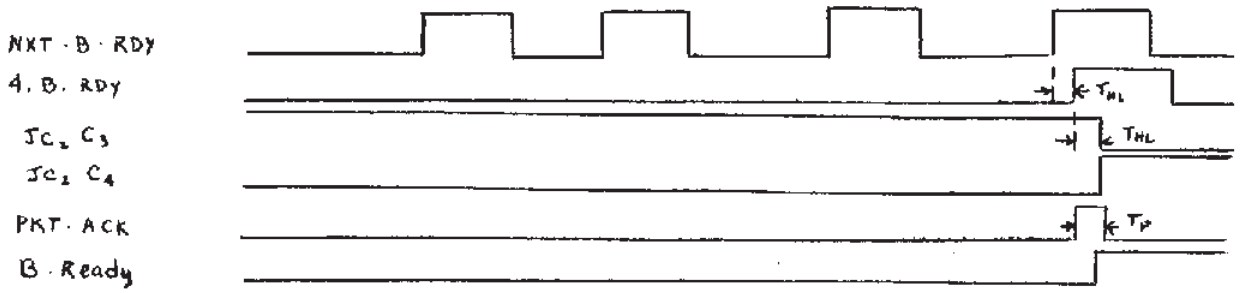
T_{HL} - Counter Propagation Delay 35-75 ns

T_B - Duration of BYTE · ACK signal ~ 30 ns

T_D - Delay from counter output to BYTE · ACK 14 ns

Total delay from NXT · B · RDY to BYTE · ACK 49 ns minimum

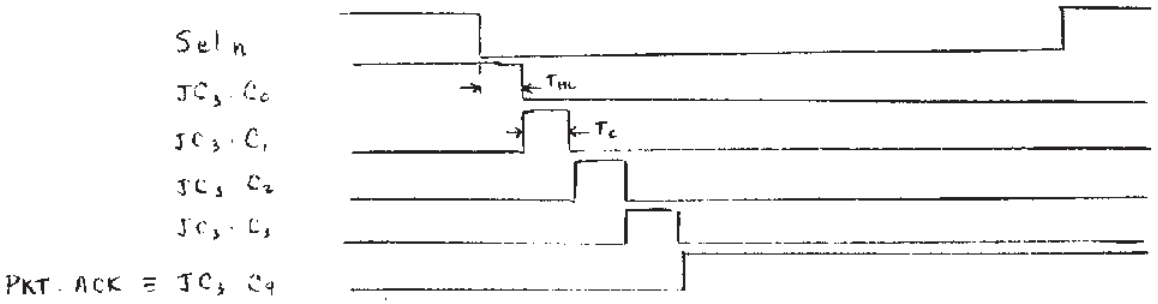
Generation of Packet Acknowledgement



T_{HL} - Counter Propagation Delay (35-75 ns)

T_P - Duration of PKT·ACK signal 35 ns minimum

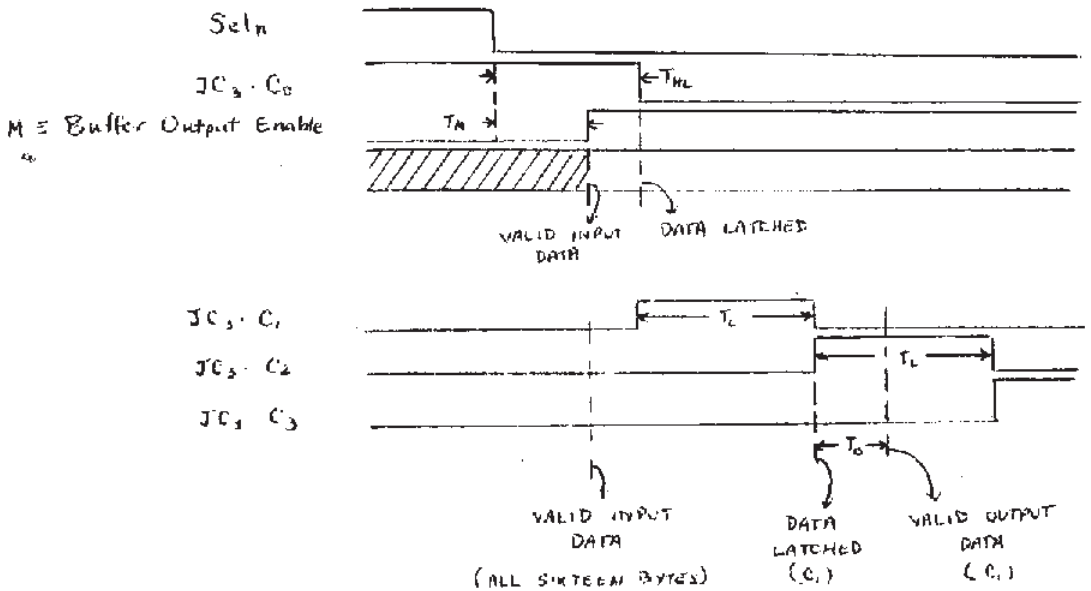
SERIAL - TO - PARALLEL UNIT 2 CLOCKING



T_{HL} - Counter Propagation Delay (35-75 ns)

T_c - Clock period ~ 30 ns

INDIVIDUAL FOUR BYTE TRANSFERS



T_L - Length of buffer transparency before data is latched ~ 60 ns

T_o - Propagation Delay of Buffer

T_M - Buffer output enable time 33 ns maximum

BIBLIOGRAPHY

1. Dennis, J.B., and Misunas, D.P., "The Design of a Highly Parallel Computer for Signal Processing Applications," Computation Structures Group Memo 101, Laboratory for Computer Science, M.I.T., Cambridge, MA., August 1974.

2. Dennis, J.B., and Misunas, D.P., "A Preliminary Architecture for a Basic Data-Flow Processor," IEEE Proceedings of the Second Annual Symposium on Computer Architecture, pp. 126-132, 1975.

3. Dennis, J.B., Misunas, D.P., and Leung, C.K., "A Highly Parallel Processor Using a Data Flow Machine Language," Computation Structures Group Memo 134, Laboratory for Computer Science, M.I.T., Cambridge, MA., January 1977.

4. Misunas, D.P., "Report on the Workshop on Data Flow Computer and Program Organization," Laboratory for Computer Science, M.I.T., Cambridge, MA. July 1977.

5. Jacobsen, R.G., and Misunas, D.P., "Analysis of Structures for Packet Communication," Proceedings of the 1977 International Conference on Parallel Processing, August 1977.

6. Patil, S.S., "Synchronizers and Arbiters," Computation Structures Group Memo 91, Laboratory for Computer Science, M.I.T., Cambridge, MA., October 1973.

7. Patil, S.S., "Forward Acting nxm Arbiter," Computation Structures Group Memo 67, Laboratory for Computer Science, M.I.T., Cambridge, MA. June 1972.

8. Plummer, W.W., "Asynchronous Arbiters," IEEE Transactions on Computers, Vol. C-21, No. 1, January 1972.

9. Amikura, K., "A Logic Design for the Cell Block of a Data Flow Processor," Master's Thesis, Laboratory for Computer Science, M.I.T., Cambridge, MA. December 1977.

10. Course notes, 6.032 Computation Structures. M.I.T., Cambridge, MA., Spring 1977.

11. Petri, C.A., Communication with automata. Rome Air Develop. Center, Suppl. I to Tech. Rep. No. RADC-TR-65-377. Reconnaissance-Intelligence Data Handling Branch, Rome. Air Develop. Center, Griffin AFB, New York, January 1966.

12. Misunas, D.P., "Petri Nets and Speed Independent Design," Communications of the ACM, Volume 6, No. 8, August 1973.