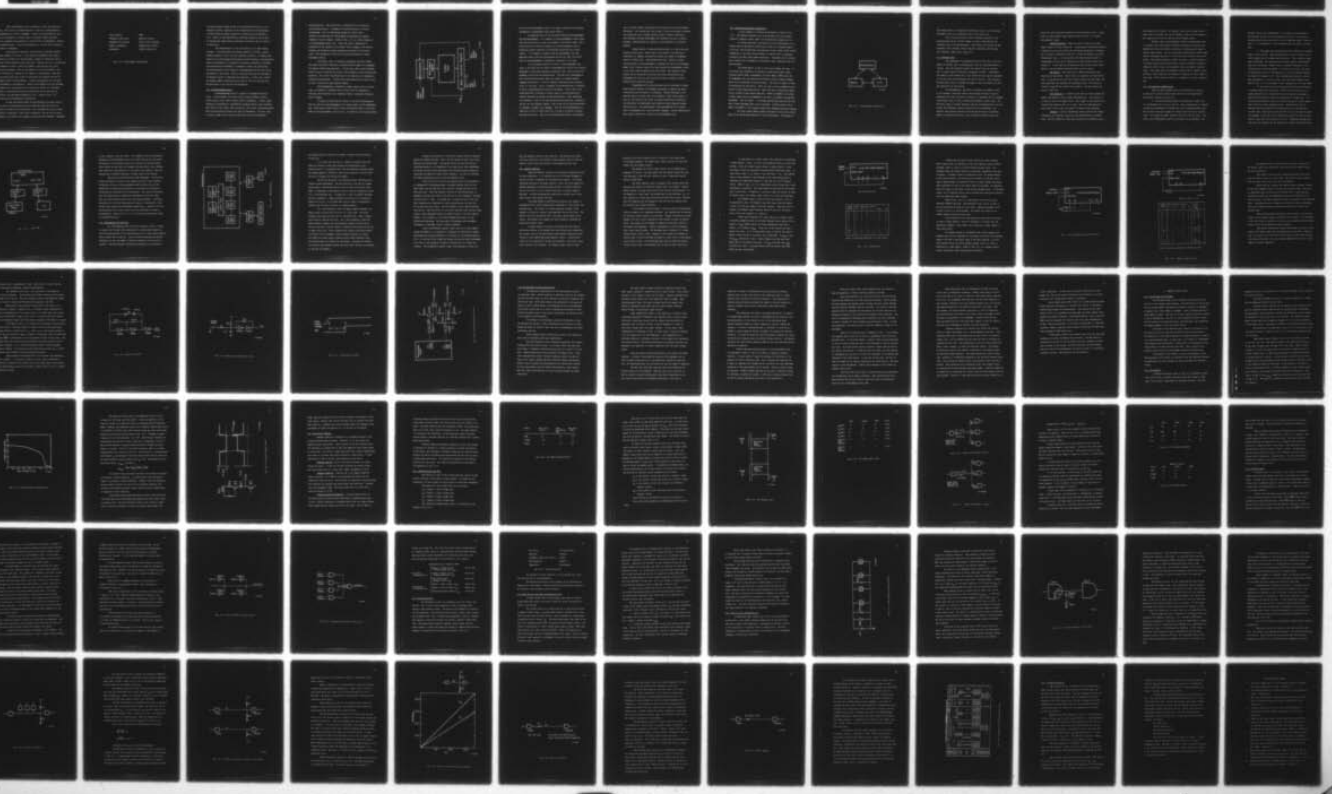
AD-A069 768 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
THE DESIGN OF MICROCOMPUTER SYSTEMS.(U)
JUL 78 T A LANE

UNCLASSIFIED R-817 DAAB07-72-C-0259

NI

1 OF 2
AD
A069768



REPORT R-617 JULY, 1978 LEVEL UILU-ENG 78-2210

CSL COORDINATED SCIENCE LABORATORY

THE DESIGN OF MICROCOMPUTER SYSTEMS

THOMAS ALAN LANE

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) THE DESIGN OF MICROCOMPUTER SYSTEMS.	5. TYPE OF REPORT & PERIOD COVERED Technical Report
(14)	6. PERFORMING ORG. REPORT NUMBER R-817, UILU-ENG-78-2210
7. AUTHOR(s) (10) Thomas Alan Lane	8. CONTRACT OR GRANT NUMBER(s) (15) DAAB-07-72-C-0259
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 178 p.
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program	12. REPORT DATE (11) Jul[redacted]78 13. NUMBER OF PAGES 169
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (9) Master's thesis,	15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessors
Microcomputer Systems
Large-Scale-Integration (LSI)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The nature of integrated circuit technology dictates an upper limit on the number of gates that can be placed on a single chip and still permit production that is economically feasible. This upper limit is constantly being increased by improving technology. The first integrated circuits had a small number of individual gates on a chip. As technology progressed, medium-scale-integrated (MSI) chips were produced with complex functions such as multiplexers, counters and shift registers. Early in the 1970's the maximum number of gates had increased to the point where a complete system or subsystem, such as a microprocessor could be

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 097 709

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

placed on a single chip. This technology is referred to as large-scale-integration (LSI).

UILU-ENG 78-2210

THE DESIGN OF MICROCOMPUTER SYSTEMS

by

Thomas Alan Lane

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

THE DESIGN OF MICROCOMPUTER SYSTEMS

BY

THOMAS ALAN LANE
B.S.E.E., Bradley University, 1973

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1978

Thesis Adviser: Professor Edward S. Davidson

Urbana, Illinois

ACKNOWLEDGMENTS

I would like to express my appreciation to the Electrical Engineering Department of the University of Illinois which provided me the opportunity to do this work. In particular, I would like to express appreciation to my thesis advisor, Professor E. S. Davidson, whose guidance and consultation was instrumental in completing this work. Lastly, a very special thanks is due to my wife Mary, who supported me through this effort.

TABLE OF CONTENTS

LIST OF FIGURES

Page

3. MEMORY

4. INPUT/OUTPUT

1. MICROPROCESSORS AND MICROCOMPUTERS

1.1. A Perspective on LSI Technology

The nature of integrated circuit technology dictates an upper limit on the number of gates that can be placed on a single chip and still permit production that is economically feasible. This upper limit is constantly being increased by improving technology. The first integrated circuits had a small number of individual gates on a chip. As technology progressed, medium-scale-integrated (MSI) chips were produced with complex functions such as multiplexers, counters and shift registers. Early in the 1970's the maximum number of gates had increased to the point where a complete system or subsystem, such as a microprocessor could be placed on a single chip. This technology is referred to as large-scale-integration (LSI).

These highly complex LSI circuits are achieved only with very high development cost. This development cost is due to the large amount of time and effort required for design, layout, initial fabrication and checkout. For any given LSI circuit, the development cost is a fixed cost; it is the same for producing one chip or many chips. To overcome this high cost, LSI circuits are usually developed only for high volume markets in which case a small portion of the development cost can be recovered from the selling price of each chip produced.

If an LSI circuit cannot be sold in sufficient volume, its selling price is high and may not be price-competitive with conventional SSI and MSI implementations.

In some cases a single application has high enough volume to justify development of an LSI chip; e.g., calculator and digital watch chips. In other cases, a more general-purpose LSI chip can be developed which can be adapted to many different applications by the user. It should be pointed out that two of the main components in a microcomputer, the microprocessor and the memory are manufactured to be used in a broad range of applications. This is one reason for their low cost.

1.2. Microprocessors

The microprocessor is an LSI component for use in digital systems. A microprocessor is a single chip or chip set that performs the functions traditionally associated with the central processing unit (CPU) of a computer. That is, it processes data as indicated by a sequence of instructions, called a program, stored in an external memory. Given a microprocessor and memory, a stand-alone computer, called a microcomputer, can be implemented with very few additional components. The low cost of microcomputers has provided a new approach to the design of digital systems which is based on the use of a general purpose computer as an alternative to special purpose hardware.

In order to establish a frame of reference, listed below are some relevant terms along with a working definition of each.

Minicomputer - a small, general purpose computer with a central processing unit (CPU), a memory for storing programs and data, and I/O devices.

Microprocessor - a microprocessor contains registers, arithmetic, and control logic for processing data as specified by instructions stored

in a memory and is always implemented in LSI technology as a single chip or a set of chips.

Microcomputer - a very low-cost, generally low-performance computer which uses a microprocessor as its CPU. A microcomputer consists of a microprocessor, memory, I/O devices and all necessary external logic such as clock generators, buffers and latches. Typically, the memory is also implemented with LSI circuits.

It should be noted that both microprocessor and microcomputer are technology-related terms in that the distinguishing characteristic of each is the technology used to implement the function rather than the function itself.

Microprogrammed Processor - a CPU (usually of a small or medium sized computer) or the heart of a programmable controller which uses a memory-like component called control store for implementing its control logic. Microprocessors, minicomputers and other computers may or may not be microprogrammed.

Microprogrammable Processor - a microprogrammed processor in which the control store is alterable by the user so that each system may have a different control structure (instruction set) or may change easily between several control structures.

The wide variety of available microprocessors provides a spectrum of performance within the class of microprocessor circuits. The range extends from the low-performance 4-bit microprocessors to the high-performance 16-bit microprocessors and to even larger word sizes in multi-chip processors. Word size, speed and instruction set are all factors for performance evaluation.

The 4-bit microprocessors have an architecture which resembles calculator designs. The difference between them and calculator chips is that they are programmable. The 4-bit word size is convenient for performing BCD arithmetic. The primary applications of the 4-bit microprocessors are as programmable calculators and simple controllers. The 8-bit microprocessors are useful for processing alphanumeric data since the word size matches standard character codes such as ASCII and EBCDIC. Typical applications of the 8-bit microprocessors are controllers, character string processors and low precision (or multiple pass) data processors. The microprocessors with 12 or 16 bit word sizes can be conveniently used to implement a general purpose low-performance minicomputer or a high performance controller.

1.3. Applications of Microcomputers

There are two design approaches to implementing small digital systems. One is to design special purpose hardware which uses flip-flops to store and define a state, and gates to define the next state and decode output signals. This approach usually yields a system which is very fast, but inflexible. The other approach is to use a microcomputer which is programmed to perform the specified control function. This approach usually yields a system that is slow compared to a special purpose hardware implementation, however it is very flexible in that the system function is determined by a program. When using a microcomputer, changes in system functions can be done by changing programs rather than by alteration of system hardware. In other words, in a microcomputer system, a fixed hardware configuration is adapted to many different applications by programming.

Thus a microcomputer user is required to write and debug programs. This could be a disadvantage for a user who is inexperienced in programming with low-level languages. Often, once microprocessors have been on the market for a while, there is software support available, such as assemblers and simulators to ease the job of programming. Specialized hardware tools, such as system emulators, are also often available to assist in debugging.

The concept of designing a system around a dedicated, general purpose computer is not new. It has long been recognized that using a computer as the heart of a system offers a degree of flexibility that no special purpose hardware design could have. However, the cost of traditional minicomputers severely limited the number of applications that could afford this approach. With the introduction of the microcomputer, this cost restriction is removed in vast numbers of applications. Many small system applications which could not previously use computers may now be designed around the microcomputer. Current microcomputers are not equal in performance to minicomputers. However, microcomputers are not designed to compete with minicomputers but to extend the concept of a dedicated computer to applications where the minimization of cost and size are important, but high-speed performance is not. Listed in Fig. 1.3.1 are some typical microcomputer applications.

In many applications where the microprocessor has been successfully used, the microprocessor has had speed and capability which far exceeded the need for the application. Thus, although the use of a microprocessor is sometimes a clear case of "overkill", the low cost of microprocessors still make it the cheapest solution for many problems. Therefore,

Cash registers	Games
Peripheral controllers	Numerical control
Automatic test systems	Point of sale terminals
Digital instruments	Communication systems
Automobiles	Traffic controllers

Fig. 1.3.1 Microcomputer applications

the microprocessor should be used in any application where it is the cheapest solution, regardless of how inefficiently the microprocessor is used because the unused capability is obtained for no additional cost. Often, in such applications, additional capability can be used to provide more sophisticated processing or ancillary jobs at little or no extra cost.

The microprocessor is not the solution to all logic design problems. In systems where the hardware function is trivial, special purpose hardware is still the most economical solution. In systems that require 50 or more packages using special purpose hardware, microprocessors should be considered as a potentially attractive alternative, providing the microprocessor performance is adequate to handle the application. The microprocessor is well-suited for implementing functions that can be described by a flow chart. That is, functions which can be described by a sequential series of operations and decisions. In some cases, either the program length, instruction cycle time, or both are such that the microprocessor is too slow for the application.

1.4. Microprogrammed Control

Microprogramming refers to a method of implementing control logic. In this method, the control section contains memory, usually called control store, which contains control information. Control signal sequences are generated by continuously accessing control store locations. Each access to the control store retreives a word called a microinstruction. Each microinstruction contains two types of information. The first type is control signals which setup the data paths for the corresponding

microinstruction. The second type is information for selecting the next microinstruction. A sequence of microinstructions is called a microprogram. Thus, by specifying appropriate control store contents, the execution of a microprogram can generate any complex sequence of control signals. Shown in Fig. 1.4.1 is a block diagram of a microprogrammed control unit. Since the control information is determined by the contents of the control store, changes to the control sequence can be made by simply modifying the contents of the control store. This flexibility is one of the attractive features of microprogrammed control.

The control store is typically implemented with ROM or PROM. This means the control structure is defined when the memory is programmed and remains fixed during execution. In some cases, RAM (called writeable control store) is used and the control store contents can be altered during execution. This allows the architecture of the machine to be dynamically altered during execution. In most cases, high speed bipolar control store is used to produce the high execution speeds.

Microprogramming is considered a highly regular form of control logic, as compared to hardwired control units which are implemented with gates and flip-flops in an irregular fashion to generate timing and control signals.

It should be noted that the "micro" in the word microprocessor does not stand for microprogrammable, but rather simply for LSI implementation. While some microprocessors, such as the National GPC/P or the Intel 3000 are microprogrammable, most are not. In addition, most microprocessors

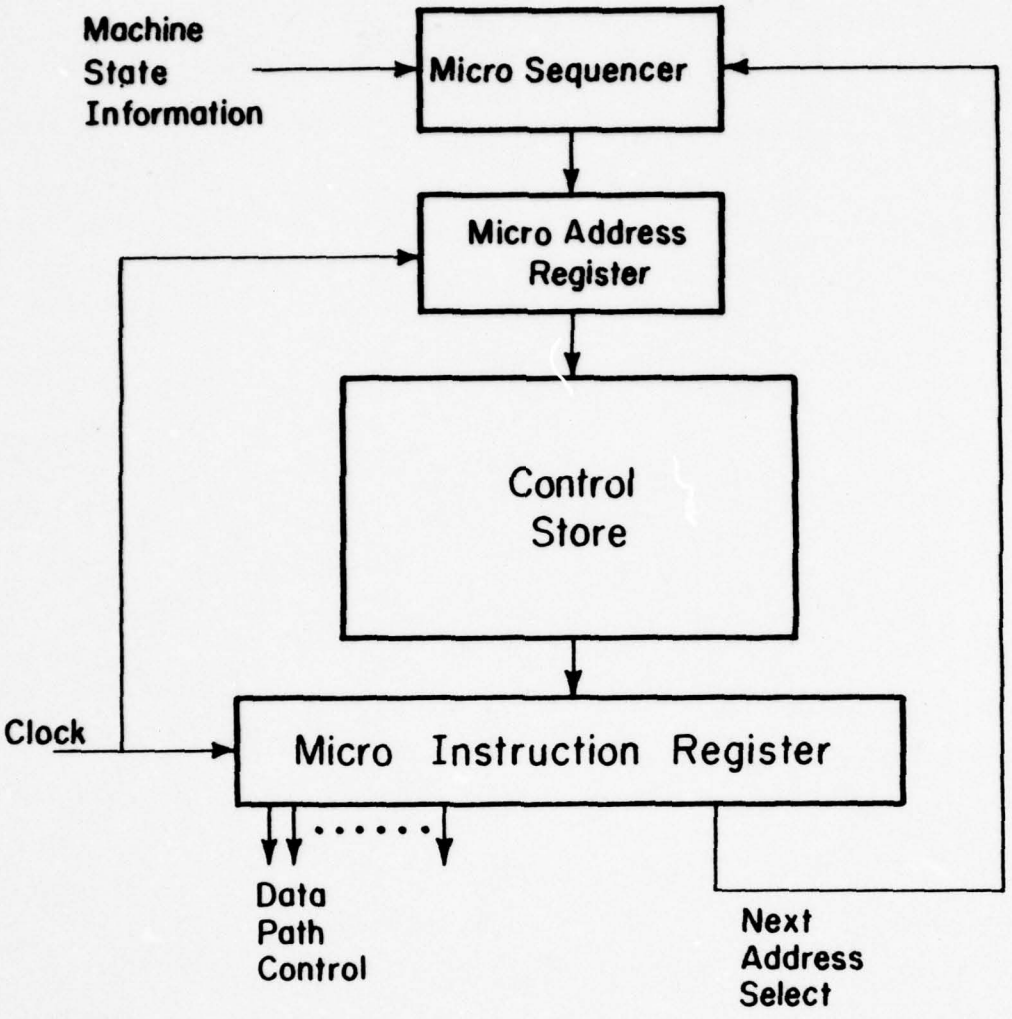


Fig. 1.4.1 Microprogrammed control unit

do not use microprogrammed control, but rather classical state machines implemented in programmable logic arrays (PLA's).

An important point is the distinction between microprogrammed and microprogrammable. If a computer is microprogrammable, there exists some way for the user to alter the contents of the control store. This allows the user to write and execute his own microprograms, thus to define the control structure of the machine instruction set (within limits). If a computer is simply called microprogrammed, the user generally cannot alter the microprogram and the fact that the control logic is microprogrammed is essentially transparent to the user.

In the classical sense of the word, a microprogrammed computer is one which has two levels of programmability and both are directly meaningful to the system hardware at execution time. The low level programming is done with microinstructions. The high level programming is done with macroinstructions, which are similar to standard machine language instructions. In such a computer, macroinstruction execution occurs in two steps. First, a sequence of microinstructions is executed to fetch a macroinstruction from the user memory. The op code of the macroinstruction is decoded to cause a jump to the microinstruction sequence necessary to execute the macroinstruction. After completing this sequence, a jump to the instruction fetch microroutine is executed and the two step sequence repeats. This is one use of microinstructions: to implement a higher level language. This use of microprogramming is important in that it allows each user, in effect, to define a personal macroinstruction set. Thus for any specified application, the designer

can, in theory, define instructions that accomplish the job with maximum efficiency. This feature may also be used to allow one machine to execute the instruction set of another machine, which is known as emulation. This minimizes software development and eliminates the need to rewrite programs. Emulation may not, however, be as efficient as a rewritten program.

Another method of using microinstructions is to write the user program at that level. Notice that in this context, the term microinstruction is somewhat contradictory because two levels of programmability no longer exist. Some manufacturers have referred to their microprocessors as microprogrammable, although they are not microprogrammable in the classical sense of the word because the programming is done at one level (the microinstruction level) only. This apparent contradiction can be reconciled by the fact that many microprocessors are programmed with very low level instructions which resemble the microinstructions of previous machines.

Programming with microinstructions has the advantage of higher speed execution because the overhead time needed to perform the macroinstruction fetch and decode is eliminated. Another advantage is the detailed level of control that can be achieved. This direct control of built-in CPU functions can result in greater parallelism of execution and elimination of unnecessary steps caused by using a general purpose instruction set. The disadvantage of microprogramming is the tedious task of programming with a very low-level language. Several manufacturers have support software for easing the microprogramming task.

1.5. Typical Microcomputer Organization

A block diagram of a typical microcomputer is shown in Fig. 1.5.1. The diagram illustrates the three sections that traditionally comprise a computer system, namely the microprocessor, memory and I/O.

As stated previously, the function of the microprocessor is to process data as specified by instructions stored in an external memory. To do this, the microprocessor must generate control signals which provide information to the other system elements to indicate what is happening internally and what should happen externally. The memory and I/O devices must interpret this information to synchronize their operations with the microprocessor.

The system memory is used to store user programs and data. It can contain any combination of read/write random-access memory (RAM), read-only memory (ROM), programmable read-only memory (PROM) which can be programmed once, and electrically-alterable read-only memory (EAROM) which can be repeatedly written and erased. The choice of memory is made by considering the application. ROM's are used to store programs and constants in high-volume applications. RAM's are used in applications where data is altered during program execution. PROM's and EAROM's are similar to ROM's except they are programmed by the user, as opposed to being mask programmed. They are useful in low-volume applications where the one time cost of mask programmed ROM's is not justified. Another typical use of PROM's and EAROM's is for program storage while debugging prototype systems.

The total primary memory capacity available to the user is determined by the addressing capability of the microprocessor. The design of

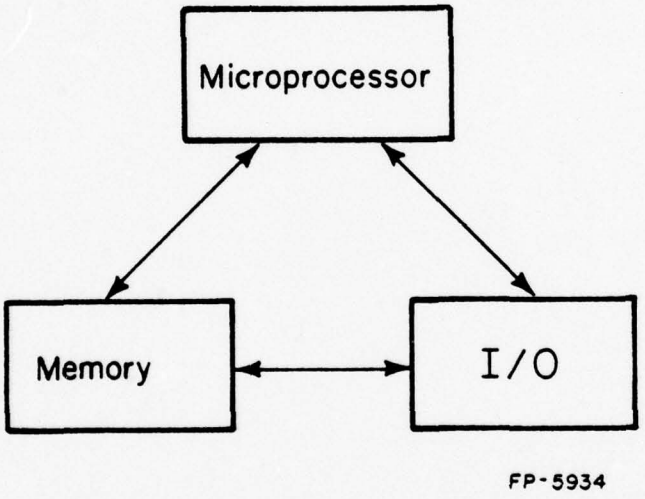


Fig. 1.5.1 Microcomputer organization

the system memory is an important consideration since in a real system, the memory costs are a significant part of the system cost.

The I/O section refers to the I/O devices connected to the microprocessor and the circuits needed to control these devices and interface them to the microprocessor. Some typical I/O devices are CRT terminals, small printers, switch registers, keyboards, LED displays, A/D converters, small discs, tapes, etc.

1.6. External Logic

One consequence of implementing circuits with LSI is that the number of external pins on microprocessor chips for interconnections is limited. The wire connection between the silicon chip and the metal pin is a major cause of failures in integrated circuits. Furthermore, the number of pins required for a chip rather than the logical complexity of the chip most often determines the size of the chip. Thus the manufacturers have tried to minimize the number of pins to minimize the size and probability of chip failure.

In microcomputers, the effort to minimize the number of pins on LSI packages, along with several other design constraints (such as minimizing chip area) has made it necessary to supplement the LSI packages with external logic to form a working microcomputer in most cases. The functions likely to require external logic are clock generation, initialization, bus buffering, interrupt control and I/O interfacing. The external logic usually consists of TTL MSI and SSI with an increasing number of specialized LSI parts, such as general purpose parallel and

serial I/O chips along with terminal and disk controller chips. Listed below are some external logic functions that can occur in a typical microcomputer system.

Input Multiplexers - These are used to select one of many signal sources to drive a given pin. Using external multiplexers permits a reduction in pin count and has the additional advantage of allowing a pin to be used at different times for a variety of purposes (so-called "time-shared" pins). In some systems this multiplexing function is accomplished with the use of three-state or open collector logic (see Chapter 2). In any case the microprocessor must provide control information to the multiplexer indicating which input to select.

Bus Buffers - The output pins of MOS microprocessors have limited drive capability. The DC fanout is typically one standard TTL load. If the MOS output must drive long PC traces with many loads, the load capacitance may increase the propagation delay. The problem is solved by using TTL bus drivers which connect to the MOS outputs and drive all the loads.

Data Registers - A problem arises when two output signals are needed simultaneously from a single time-shared pin. The solution is to send out one of the signals early. This signal is then stored in an external data register until it is used. Thus, the time-sharing of pins trades off pin count (and speed) for external data registers.

Decoders - Control, addressing, device selection and timing information are sometimes supplied by the microprocessor in encoded form. TTL MSI decoders are then used to decode this information into a

form useable by the system. For example, seven control signals which never occur at the same time can be encoded in three bits and decoded to one signal per wire with an external three-to-eight decoder.

External logic in microcomputer systems in undesireable since it requires extra logic design, additional packages and increased cost. Manufacturers have tried several approaches to minimizing the amount of external logic needed. One is to design new microprocessors which supply control information and data in a form that can be directly used by the system. This approach implies a higher pin count. A second approach is to design a family of specialized chips, such as peripheral interfaces, timing and control chips, clock generator chips, and memory interface chips. Since these chips are designed to operate with a specific microprocessor, their design can be optimized to minimize the total number of chips used. Often, they are useful in a limited way in other systems as well.

1.7. Microcomputer Configurations

There are many configurations of microcomputers available. Listed below are three configurations which can be distinguished by application, package count and cost.

1) The self-contained family of microprocessor, memory and I/O, implemented with a chip or chip set. This configuration is intended to produce microcomputers at minimum cost with a minimum package count. The LSI chips have been designed to interact directly with no external logic. All timing and signal protocols are built into the chips. Typically, this configuration results in relatively low performance. The

ultimate case of this configuration is the single chip microcomputer. The extremely low cost of this configuration allows it to be used in dedicated applications where cost is the all-important factor. Shown in Fig. 1.7.1 is an example of a self-contained chip set family, the Intel 4004.

2) The single chip microprocessor which interfaces to standard memory and I/O devices. These microprocessors utilize a unified bus to talk to memory and I/O through simple protocols. This is done to permit easy interfacing. This configuration typically requires some external logic for interfacing. However, there is greater flexibility in the interface which allows any memory or I/O device to be used. The ease of interfacing allows this configuration to be easily expanded through the addition of memory or I/O. This configuration produces relatively high performance at the cost of extra logic and higher package count. Shown in Fig. 1.7.2 is an example of this configuration, the Intel 8080.

3) The multi-chip, bit-slice microprocessor which interfaces to standard memory and I/O devices. The bit-slice architecture is a useful approach when a microprocessor with the desired word length or operating characteristics cannot be fabricated on a single chip. In a bit-slice microprocessor there are typically two types of chips, namely data slices and control sequencer slices. The data chips are all identical and partitioned as a bit-slice parallel to the data paths. In other words, each chip has data paths for an N bit word and the chips are designed so that they can be connected in parallel to form an arbitrarily large word with a multiple of N bits. Connections between data path chips are necessary for the execution of certain instructions such

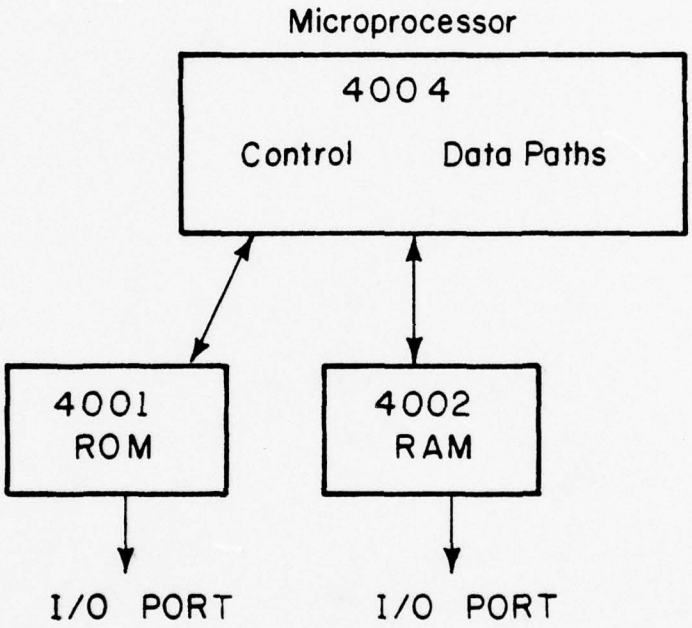


Fig. 1.7.1 Intel 4004

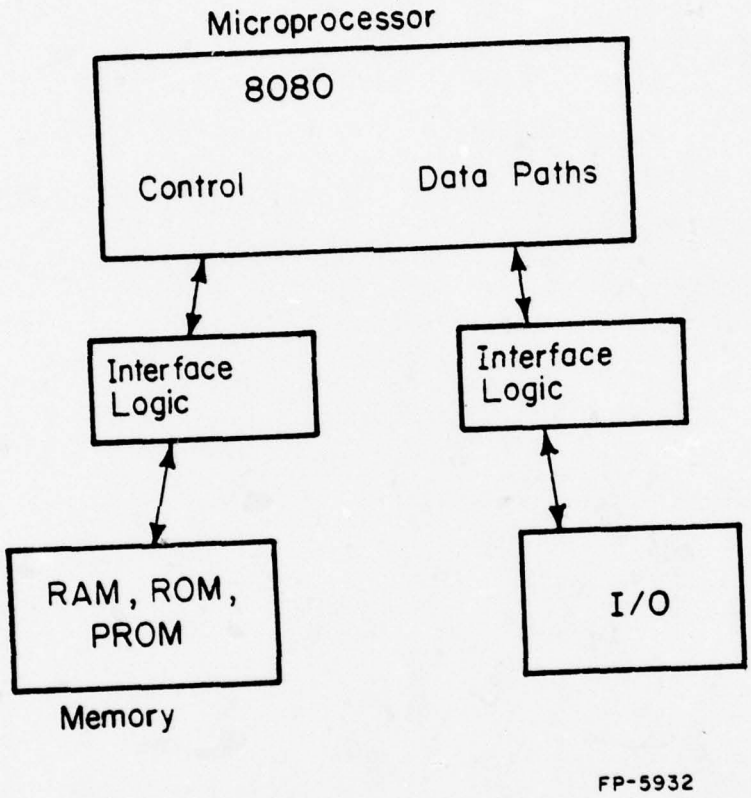


Fig. 1.7.2 Intel 8080

as add, subtract, shift and rotate. The sequencer chips are designed to implement a microprogrammed control unit which controls the data paths. The function of the sequencer chips is to provide the current control store address and the logic to determine the next control store address. These chips are sliced parallel to the control store address, thus connecting them in parallel generates a larger control store address.

The bit-slice architecture has many advantages. The user can define the machine in terms of word length and control store capacity. In addition, the use of microprogrammed control allows the user the flexibility to define the architecture of the machine by creating an instruction set optimized for the application. Bit slices are typically very fast because they are generally implemented with bipolar logic. Thus, bit-slice microprocessors have very high performance. The implementation of a bit-slice microprocessor requires a large amount of SSI and MSI logic along with considerable design effort. However, bit-slice microprocessors can be built which rival the performance of today's minicomputers. Shown in Fig. 1.7.3 is a bit-slice microprocessor based on the AMD 2900 family.

1.8. Microcomputer Bus Structure

All microcomputers have an I/O bus structure, that is, a group of signal lines through which communication is established between the data processing section of the microprocessor and all other system elements such as memory and I/O devices. The bus structure organization permits flexibility in that the number of devices connected to the bus can be variable. The microprocessor architecture determines the structure of

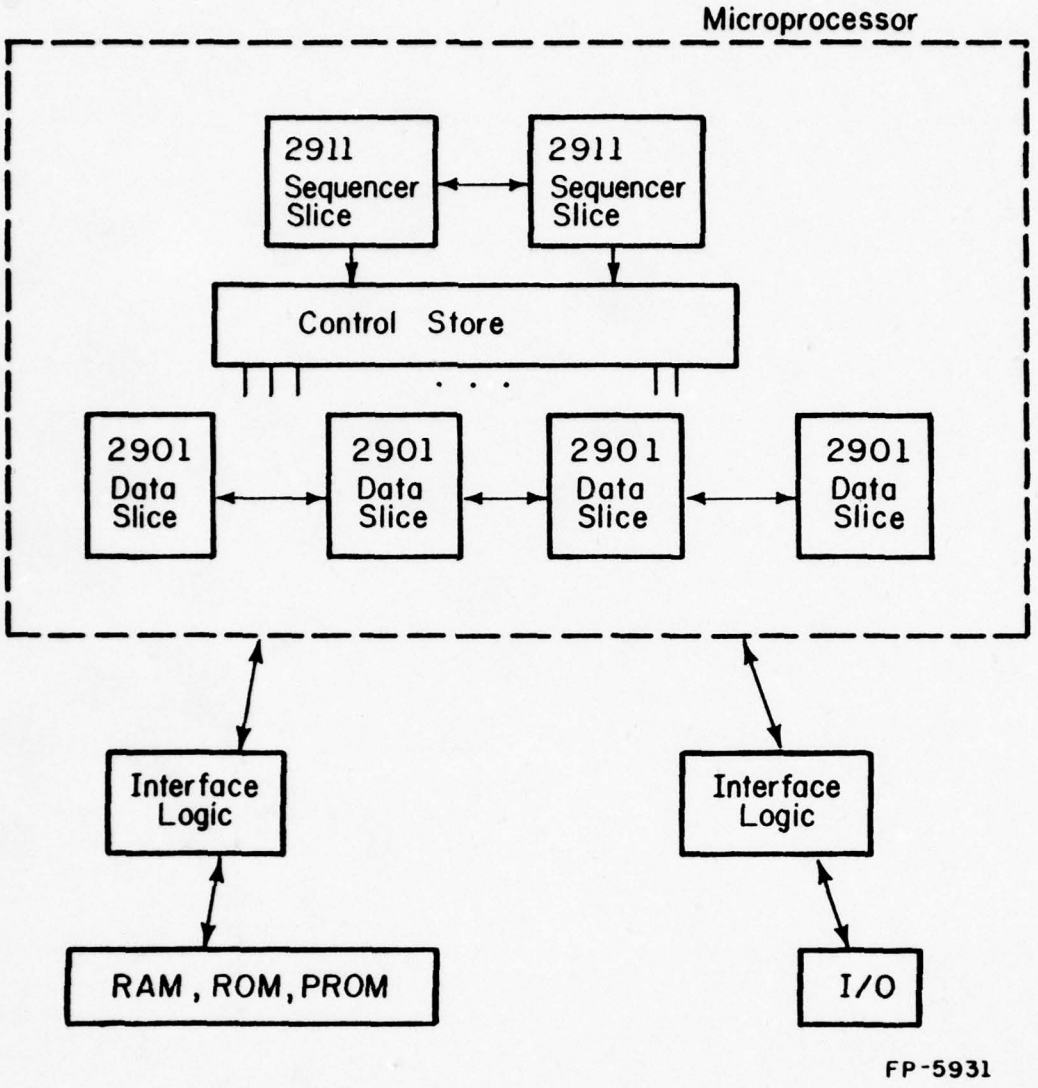


Fig. 1.7.3 AMD 2900 Family

the bussing system in terms of the number of busses and the structure of each bus.

It is clear that the use of a common bus requires that some means be provided to steer data between the microprocessor and a selected device. To accomplish this, the microprocessor supplies control and timing signals to indicate to each device connected to the bus when and how to interpret the various bus signals.

The I/O bus structure is an important factor affecting microcomputer system performance. One bus structure is the use of a single bidirectional, time-multiplexed bus, such as that used with the Intel 8008. An advantage of this method is that it allows a minimum pin count on the microprocessor chip. A single bus is used to transfer addresses and data alternately. Memory operations are done by first sending an address on the bus. At the same time, control signal is generated to indicate that the data on the bus is a memory address.

For a read operation, after the memory address transfer, the memory places the read data on the bus at a later time. The microprocessor then reads the data bus at a specified time. Thus, a memory read typically requires one cycle. For a write operation, after the memory address transfer, the microprocessor places the write data on the bus along with a control signal to indicate that the data on the bus is memory write data. Thus, a memory write requires the microprocessor to initiate two bus transfers, which typically requires two cycles. The single bus system usually requires output buffer registers since the microprocessor bus outputs are time-shared. Typically the address is latched in the external devices and stored until the data is available at the next bus transfer.

Another bus structure is the two bus system, which has separate busses for address and data. Such a two bus system is found in the Intel 8080 and the Motorola 6800. This method allows one data word and its associated address to be transferred in the same cycle, thus eliminating the delays and extra logic associated with time-multiplexed bussing and resulting in potentially faster output transfers. The use of a multiple bus system requires more pins on the microprocessor chip.

Since the 4 to 8 bit word size commonly used in microcomputers is inadequate for addressing memory, the memory address word size was made larger than the data word size, usually a integer multiple of the data word size. Since the internal circuitry of the microprocessor is designed to process data words, often the bus is also designed to transfer one data word at a time. In systems with a single bus such as the Intel 4004, addresses are transferred to other system devices over the data bus in data word size pieces. The full address is thus time-multiplexed onto the bus. When the device or the device interface logic receives these pieces it must reassemble them in an address buffer to form the complete address. This procedure is very time consuming and results in very inefficient addressing. The logic used to disassemble and reassemble the address is sometimes called a "bundle interface".

Since a microcomputer spends a major part of its time communicating with memory, it is important to maximize the efficiency of memory addressing when system performance is of concern in the intended application. In the case of the single bus system, this can be done by designing a bus that is wide enough to transfer an address word in a single bus transfer. This essentially implies using a microprocessor in which the

data and addresses words are the same size. The multiple bus system voids the fixed word size problem by using separate buses for data and address, each of which are matched to the appropriate word size.

1.9. Support Circuitry

Many microcomputer systems rely on externally generated clocks for their operation. It is normally the job of the logic designer to develop circuits to generate the specified clock signals. In systems requiring one-phase or two-phase external clocks, the clock signals are commonly generated by interconnecting one-shots if timing tolerances are not too strict, or by a crystal oscillator if precise timing is required. In cases requiring more complicated clocking, such as four-phase clocks, other methods are used.

One common method of generating multiphase clock signals is implemented with a counter whose states correspond to clock phases, and decoding logic. The counter is clocked repeatedly through a fixed sequence of states by a master clock generated with an oscillator. The master clock runs continuously and is the basic timing signal of the system. The desired clock signals are then obtained by decoding the appropriate counter states.

A binary counter is usually not suitable for this type of application because of the hazards that occur during state transitions. The counters used in this application should execute hazard-free transitions to avoid generating false momentary outputs. Counters well-suited for this application, called shift counters, can be built using shift registers with feedback. The feedback signal, which is some

function of the shift register state, is applied to the serial input of the shift register. Two common shift counter circuits are the ring counter and the Johnson counter.

The outputs of an N-bit binary counter can be decoded into a maximum of 2^N states. The ring counter and the Johnson counter are both less efficient in that an N-bit version of each of these has N and 2N states, respectively.

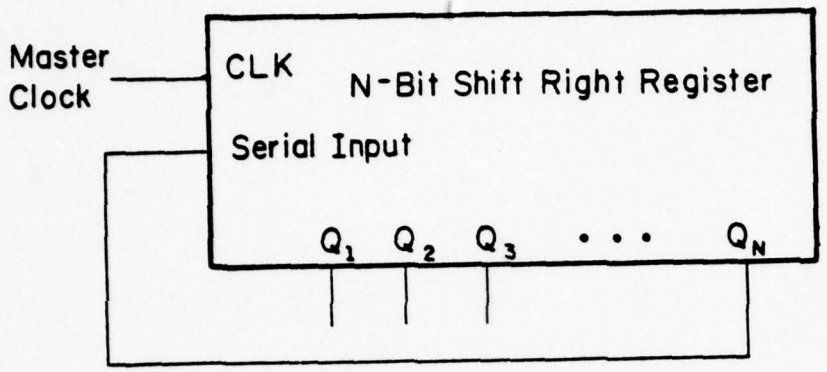
The shift registers used to implement shift counters must have parallel data outputs to allow the counter state to be decoded. Another useful feature is a Clear input which can be used for initialization. In the absence of a Clear input, the counter must be self-starting, i.e., it must sequence to a valid state from any arbitrary initial state. Furthermore, the computer must be designed so that the start-up sequence does not interfere with proper operation.

In the design of shift counters, there are several considerations which are common to all sequential machines. One consideration is when and how the shift counter should be initialized. Another consideration is what happens if an error occurs causing the shift counter to enter an invalid state. The counter can be designed to hang-up or self-start (return to the normal state sequence). Another consideration is how the decoding logic treats invalid states. The decoding logic is minimal if it treats invalid states as don't cares. However, if the counter ever reaches an invalid state it could be decoded as being in a valid state or even several valid states at once. If the decoding logic does not treat the invalid states as don't cares, the decoding logic is said to reject false data.

A ring counter is a shift counter that operates by circulating a single logical 1 level. An N bit shift register forms an N state ring counter. Since the counter outputs contain a single logical 1 level, the counter states are essentially decoded without additional logic. In other words, State 1 = Q_1, State 2 = Q_2, and State i = Q_i. The elimination of state decoding is the advantage of the ring counter which is otherwise very inefficient because it uses only N of the 2^N possible states. Shown in Fig. 1.9.1 is a ring counter circuit along with a table describing its operation. This ring counter circuit must be initialized into a valid state. In addition, if an error causes the counter to reach an invalid state, the counter cannot recover (it is not self-starting).

The feedback logic is used to insert the logical 1 level at the appropriate time. The feedback logic must detect when the logical 1 level is in the last stage of the shift register. When this state is reached, the feedback logic functions should be such that the serial input of the shift register is a logical 1.

Every state transition of a ring counter has a hazard, unless the output propagation delay of the shift register from a logical 0 to a logical 1 (t_{PLH}) exactly equals the output propagation delay from a logical 1 to a logical 0(t_{PHL}). Since this is not usually the case, a hazard will occur. If it is known which delay is shorter, the type of hazard, either overlap or isolation, can be predicted. If t_{PLH} is less than t_{PHL}, overlap will occur. Overlap implies that the next state begins before the present state ends. If t_{PLH} is greater than t_{PHL}, isolation will occur. Isolation implies that the present state ends before the next state begins.



FP-5930

Ring Counter Circuit

CLOCK PULSE	Shift register outputs Q_1	Q_2	Q_3	Q_{N-1}	Q_N
→1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
N-1	0	0	0	1	0
-N	0	0	0	0	1

Table illustrating operation of a ring counter

Fig. 1.9.1 Ring Counter

Consider how the type of hazard affects the state decoding. Since single states are available at the shift register outputs without decoding, there is a built in hazard between adjacent states. The adjacent states are either isolated or overlapped depending on the type of hazard. To decode a group of consecutive states, the method depends on the type of hazard. If overlap occurs reliably, an "OR" gate can be used to decode groups of consecutive states by simply "ORing" the outputs which correspond to each of the single states in the group. No momentary zero will appear at the output of the OR gate between states. If isolation occurs, an R-S flip-flop can be used by setting and clearing the flip-flop with the appropriate state outputs.

Shown in Fig. 1.9.2 is a ring counter circuit which uses a different feedback function. One advantage of this circuit is that it can be initialized by using the Clear input. Another advantage of this circuit is that it is self-starting. The counter can return to its normal sequence from any invalid state.

This discussion of the ring counter has been based on counters in which a single logical 1 level is circulated. Of course, with the appropriate changes, a ring counter that circulates a single logical 0 level can be built.

The Johnson counter is implemented with a shift register and feedback such that the compliment of the output of the last shift register stage is fed back to the serial input of the shift register. An N bit shift register forms a 2N state Johnson counter, thus it is twice as efficient as a ring counter. Shown in Fig. 1.9.3 is a Johnson counter circuit along with a table describing its operation.

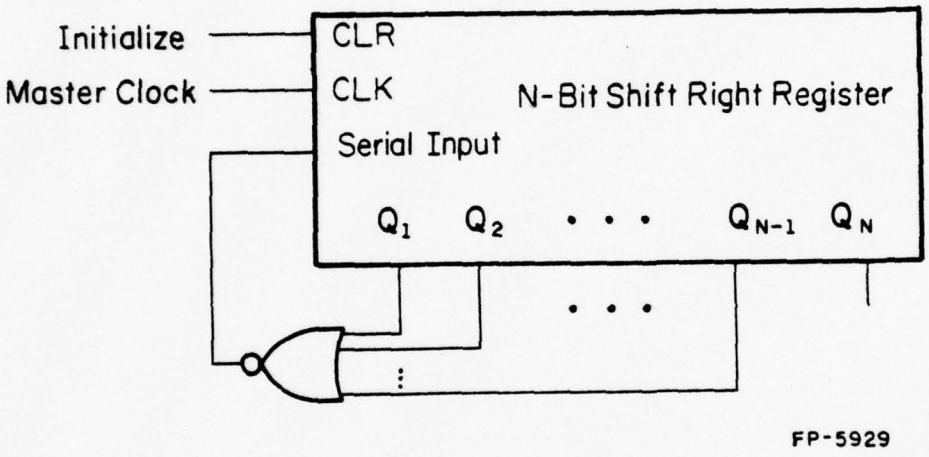
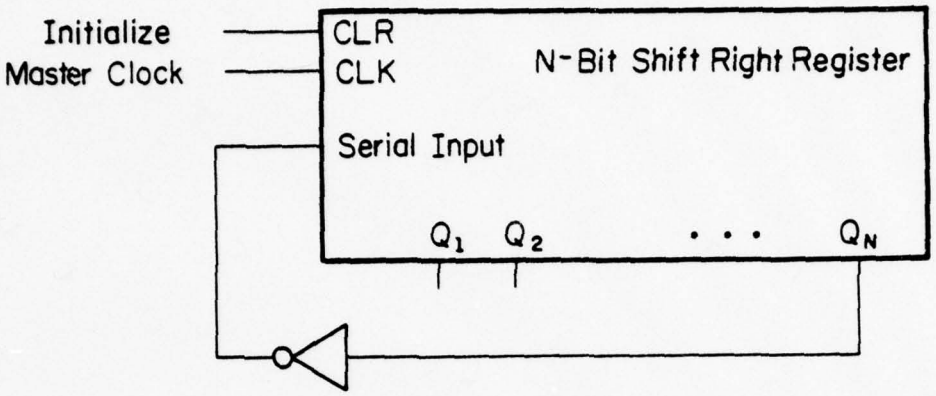


Fig. 1.9.2 Modified ring counter circuit



FP-5928

Johnson counter circuit

CLOCK PULSE	Shift register outputs Q_1	Q_2 . . . . Q_{N-1}	Q_N	Decode Logic (AND)
1	0	0 0	0	$\overline{Q}_1 \cdot \overline{Q}_N$
2	1	0 0	0	$Q_1 \cdot \overline{Q}_2$
3	1	1 0	0	$Q_2 \cdot \overline{Q}_3$
.	.	. . .	.	. .
.	.	. . .	.	. .
.	.	. . .	.	. .
N	1	1 1	0	$Q_{N-1} \cdot \overline{Q_N}$
N+1	1	1 1	1	$Q_1 \cdot Q_N$
.	.	. . .	.	. .
.	.	. . .	.	. .
.	.	. . .	.	. .
2N	0	0		$\overline{Q_{N-1}}\ Q_N$

Table illustrating operation of Johnson counter

Fig. 1.9.3 Johnson counter circuit

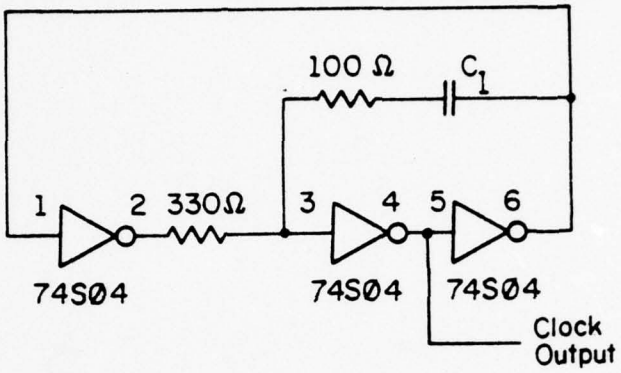
The operation table illustrates that the state transitions of the Johnson counter are hazard-free, since only one bit changes during any state transition.

This counter circuit can be initialized with the Clear input, since the 000...0 state is a valid state. This circuit cannot recover from an invalid state, thus it is not self-starting.

Unlike the ring counter, the Johnson counter requires decoding of the shift register outputs to obtain any particular state. Any state or consecutive group of states can be decoded with a 2-input AND gate provided that both true and complemented outputs are available on the shift register. The state decoding functions are indicated in the operation table. Although there are no hazards in the Johnson counter transitions, there can be hazards between the decoded outputs due to delays in the decoding logic.

While the preceding discussions illustrate how to use the ring counter and the Johnson counter to generate clock signals, it is important to note that these two methods can also be used to generate complex timing signals by the use of appropriate state decoding.

Many digital applications require the designer to supply a basic system clock signal, which must be generated by an oscillator. Shown in Figs. 1.9.4 and 1.9.5 are two oscillator circuits, one RC controlled and one crystal controlled. Both use a single inexpensive TTL IC and a small number of discrete components.



FP-5927

C_1	Frequency
200 pf	5 MHz
1600 pf	1 MHz
.018 μF	100 KHz
.18 μF	10 KHz

Fig. 1.9.4 RC oscillator

The upper limit is approximately 5 MHz. This circuit is taken from the "TTL Applications Handbook", Fairchild Semiconductor.

The frequency of the Fig. 1.9.5 oscillator is determined by the crystal frequency. This provides more stable operation for the added expense of the crystal. The upper frequency limit is approximately 20 MHz. This circuit is taken from "The Electronic Engineer", May 1969.

When power is first applied to any circuit, the state of all flips-flops, counters, registers, etc., is random. Before operation can begin, everything must be in a known state, or in other words, the circuit must be initialized. Shown in Fig. 1.9.6 is a circuit which will provide an initialization signal every time power is applied or the switch is closed.

The signals in Fig. 1.9.7 illustrate the operation of the initialization circuit. The power supply is turned on and then the power supply voltage reaches a valid level at t_1. From t_1 to t_2, the supply voltage is valid and the initialize signal is present. During this time, the initialize signal ($\overline{INIT}$) is being used to set all flip-flops to their correct state. Also, during this time self-starting state machines are being clocked to their correct sequence. At t_2, the initialization period is over and system operation begins. The duration of the initialization period, t_d, is determined by the RC product in the initialization circuit.

Many teletypes now operate with a 20 ma current loop convention. Thus a logical 1 = 20 ma and a logical 0 = 0 ma. Thus, a converter is required to convert the TTL levels used by the system logic to the current loop representation used by the teletype. Shown in Fig. 1.9.8 is a typical conversion circuit.

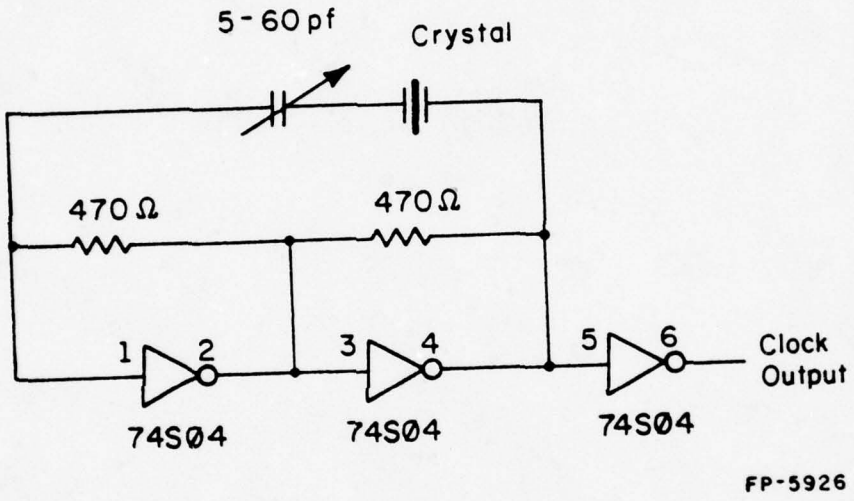


Fig. 1.9.5 Crystal oscillator

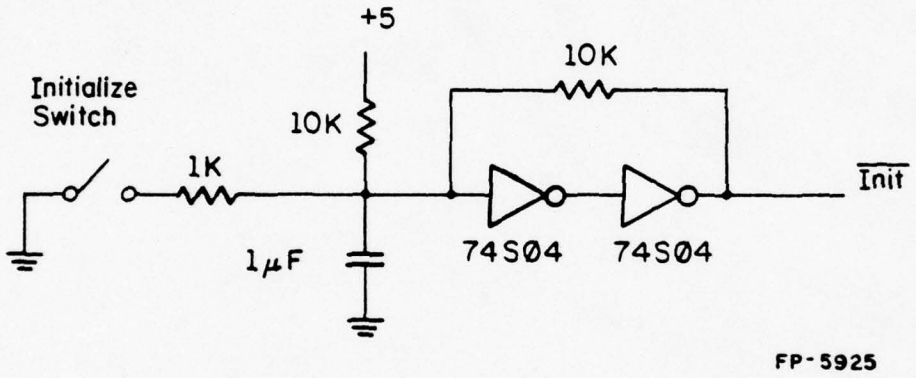


Fig. 1.9.6 Power-up initialization circuit

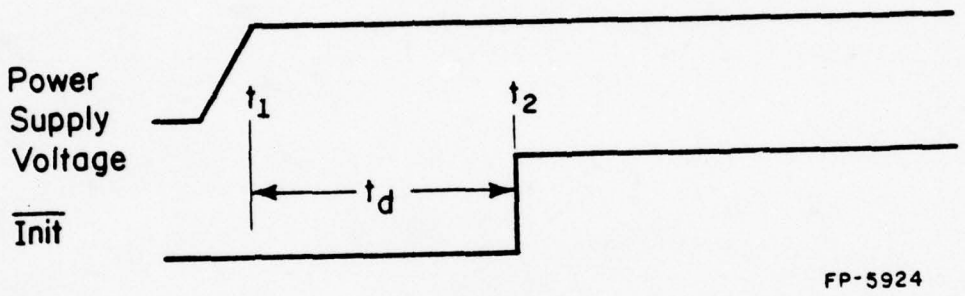


Fig. 1.9.7 Initialization timing

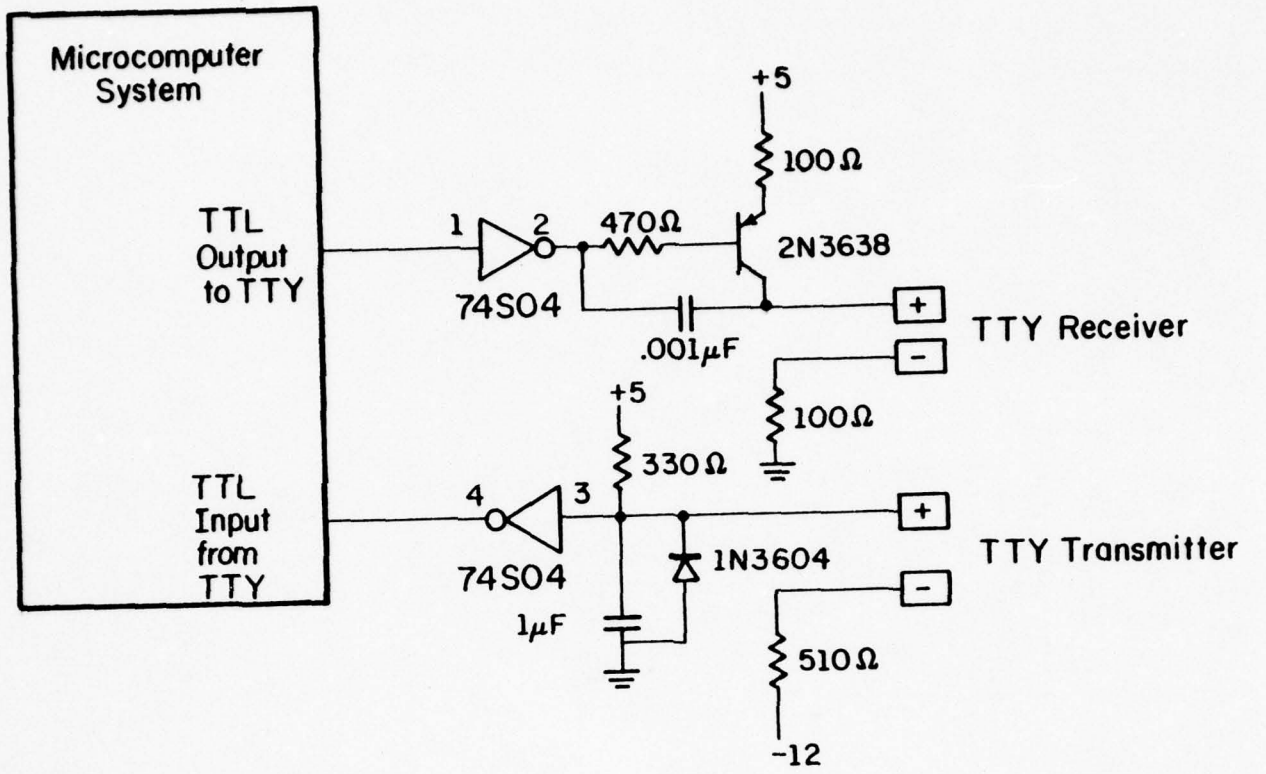


Fig. 1.9.8 20 ma TTY interface circuit

1.10. Microcomputer Design Considerations

One important characteristic of any microcomputer system is its execution speed. Several measures of execution time that are commonly used are the state time, the cycle time and the register-to-register add instruction time. While these provide some indication of the execution speed, the only reliable method of determining how fast a microcomputer can perform a given task is to write a program which performs the task. From the execution times of the various instructions, the execution time of the program can be determined.

The design of a microcomputer requires consideration of both the hardware and software. These considerations are important in determining the amount of external logic, the ease of programming, the system performance and cost.

Major hardware design considerations include clock generation, power supply requirements and logic compatibility.

Most microcomputers require externally generated clock signals, although some have on-chip clocks. For on-chip clocks, the user need only connect a frequency controlling device such as a crystal or RC network. Some microprocessors require external multiphase clocks. Other clock considerations involve required frequency, signal rise and fall times, clock signal voltages and other specified timing relationships. Another issue is whether the microprocessor can tolerate much variance of these requirements from the nominal specifications. When signals must meet tight specifications, the clock circuitry design can become complicated.

The power supply voltages required are important because each power supply represents a significant cost in small systems. Most systems contain a +5 volt supply to power the TTL logic. Systems using MOS microprocessors often require one or two additional supply voltages. When selecting memory chips, the supply voltages used by the memory chips should be considered since often they can require another different supply voltage, at potentially large currents.

Logic compatibility refers to the capability of directly interconnecting components of different logic families within a single system. Such direct connections require that the signals from each device be compatible with the others to which it may be connected, in terms of both logic voltage levels and signal currents. In most cases, logic compatibility is referenced to TTL. In other words, a device is referred to as TTL compatible or not TTL compatible. Logic compatibility is desireable because it eliminates the need to use voltage level translators and current drivers. Most MOS microprocessors and memories available are designed to be compatible (or nearly compatible) with TTL logic voltage levels.

There are several system characteristics that affect the system software. In effect, they define the system as the programmer sees it. Major software characteristics of microprocessors include the data word size, the addressing modes, the instruction set and the number of registers.

The small data word size typically used in microcomputers has several effects on the programmer. When the real data is such that it must be stored in several microprocessor data words, operations on real data require multiple-word microcomputer operations. This type of

operation is similar to conventional multiple precision arithmetic. Another result of small data words is that addresses require a larger register size than data for efficient operation. This separation of address and data handling decreases the number of possible register-to-register transfers and can cause complications when manipulating addresses.

The instruction set affects the system flexibility, the amount of program memory needed, the ease of programming and the execution time. One characteristic of the instruction set is the addressing modes. Typical addressing modes are direct, immediate, relative, indexed and indirect. A variety of addressing modes is necessary to allow the programmer to select among the methods throughout his program for efficient execution. It should be noted that many microcomputer programs are stored in ROM and such programs cannot be modified during execution. The addressing modes must then provide adequate capability for using and manipulating addresses in registers.

Another important hardware feature of the microcomputer from the programmer's point of view is the number of registers available. Generally, a larger number of registers increases the efficiency of the computer in terms of execution time. This is because the registers can be used for temporary data storage and the relatively slow and cumbersome references to the system memory can be avoided. There are several types of registers. General purpose registers can be used as temporary storage for addresses, operands and results. Other special purpose registers may be used for special addressing functions or chip input/output.

There are several other system features which are present in some microcomputers: stacks, interrupt capability and DMA.

Many microprocessors use a last-in-first-out stack for saving status during subroutine calls and interrupt processing. Before leaving the main program, the program counter contents and other status information are saved by "pushing" them onto the stack. The return to the main program is done by "popping" the information off the stack and into the appropriate registers, thus restoring the original status condition. The stack is convenient for handling nested subroutine calls and interrupts in which a sequence of status savings and restorations is done. On some microprocessors, the stack may also be used for temporary storage by the programmer.

There are two methods used to implement a stack. In one method, the complete stack or part of the stack is implemented within the microprocessor chip. In the other method, a register within the microprocessor is used as a stack pointer and part of the system RAM is used for storing the stack. The stack pointer register contains a memory address which is the current top-of-stack. To push data onto the stack, the stack pointer is incremented and the data is written into the memory at the address then indicated by the stack pointer. To pop data off the stack, the data is read from memory at the address indicated by the stack pointer. The stack pointer is then decremented. Several minor variants of this scheme are commonly used as well.

The use of the on-chip stack is faster because stack operations are accomplished with no memory references. Many microprocessors have been designed with on-chip stacks so they can be used in applications where the only system memory used is ROM.

When using stacks that are implemented with RAM, the depth of the stack is essentially unlimited. However, when using an on-chip stack, the depth of the stack is limited to some small number, typically 16 or less. The programmer must make sure that the stack usage due to data storage, subroutine calls and interrupt requests that occur during the program execution do not exceed the capacity of the stack. When this happens, the stack overflows resulting in a loss of information. In microprocessors which have on-chip stacks, some are designed so that the stack is accessible to the programmer. On-chip stacks may also have associated stack full and stack empty signals so that the depth of the stack can be extended by software into the system RAM.

Interrupt capability is a feature that allows some external device to signal the computer that it needs immediate servicing. Microprocessors which can be interrupted have an input called the interrupt request line. All the external devices which may want to interrupt are connected to this line. The microprocessor tests this line periodically. When it sees a request, it starts the interrupt process. This begins by saving the status of the machine, consisting of the program counter and any other needed registers. The status saving can be done in hardware, software or a combination depending on the particular machine being used. After saving the machine status, the interrupt service routine is started. After servicing the interrupting device, the original status is restored and the main program processing resumes. Interrupt capability is needed only in situations that require fast real-time response from the microcomputer. However, in many applications the interrupt capability is

a great convenience. In some cases the external conditions are slow enough that they can be tested periodically with software by a polling technique, thus eliminating the need for interrupts.

Direct memory access (DMA) refers to a system capability which allows high speed peripherals to transmit data directly to the memory without going through the microprocessor. This feature allows large blocks of data to be transferred at high speed and causes minimal interference with normal processing. Some microprocessors, such as the Intel 8080, have been designed with features that allow easy implementation of DMA. Most microprocessors can be made to perform DMA or a "DMA-like" operation, however, some may require extensive additional hardware while those designed for DMA require less.

A DMA request disables the processor and allows the requesting peripheral to control the memory busses directly. The DMA feature is particularly critical for slow processors with fast peripheral devices such as disks, on the system. Even the relatively slow disk systems which are used with microprocessors are often too fast to be handled by interrupt routines. DMA capability is then essential.

2. COMPUTER CIRCUIT DESIGN

2.1. Logic Design in LSI Systems

The widespread usage of LSI integrated circuits has altered the role of the computer logic designer. The availability of LSI circuits such as microprocessors and memories have made it possible to implement a computer with a small number of packages. Each LSI package contains a large number of gates. Thus much of the logic design problem has shifted to the IC manufacturer. The job of a logic designer building a system with LSI circuits has become selecting, interconnecting, and interfacing the needed packages to form the desired system.

The nature of the interface logic required is dictated by the particular LSI circuits used and the desired system characteristics. The flexibility in the hardware design of a system utilizing LSI circuits is in theinterfacing logic. In most cases, the interface is implemented with SSI gates and MSI functions. Currently TTL is used because of the large number of low-cost functions available. Thus a typical microcomputer system consists of LSI packages interfaced with TTL logic.

The purpose of this chapter is to give the designer a basic understanding of TTL circuits, an overview of the MSI functions available and guidelines for circuit design, all which are needed for effective design.

2.2. TTL Circuits

Transistor-transistor logic, or TTL, is an integrated circuit logic family which is currently the most widely used because of high speed, low cost and a large number of functions available. Most TTL

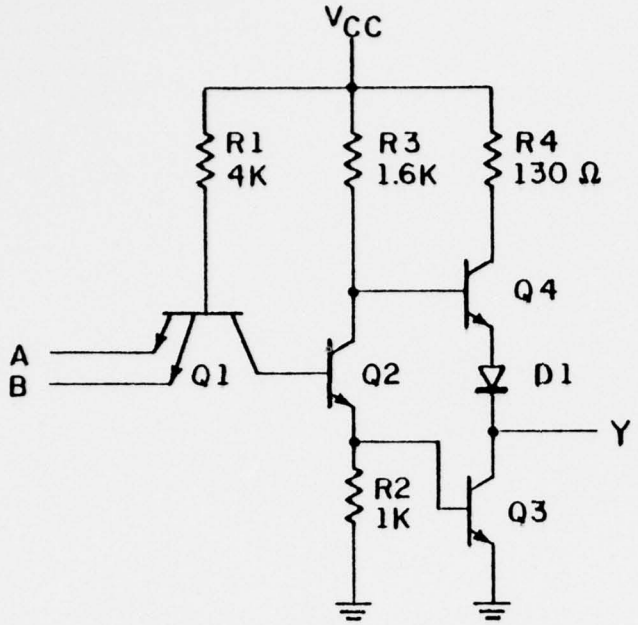
integrated circuits are identified by a 74XX Series number, which identifies the circuit function.

The basic TTL NAND gate circuit is shown in Figure 2.2.1, along with the NAND gate symbol and truth table.

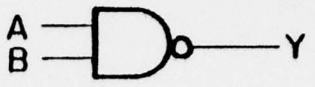
The gate circuit contains the multiemitter input transistor (Q1) and the totem-pole output stage (Q3 and Q4) characteristic of TTL gates. The multiemitter input transistor performs the AND function on the inputs. The remainder of the circuit acts as an inverter. The totem-pole output gives the gate good drive capability because of its low output impedance in both logic states. This makes it effective for driving capacitive loads.

The operation of the NAND gate of Fig. 2.2.1 is illustrated by the following discussion. Shown in Fig. 2.2.2 is the transfer characteristic of a TTL gate.

When either input voltage is low, Q1 is in saturation, Q2 and Q3 are off. Q4 is on. The output voltage is two diode drops below V_{cc}. As the input voltage rises, the base of Q2 also rises. When the input voltage is .5 volts, Q2 turns on and its collector voltage drops as its emitter voltage rises. Q1 goes from saturation to reverse active mode, as Q2 goes from off to active. As the collector of Q2 drops, the output voltage drops, since the emitter of Q4 follows its base. The gate is now in the transition region of the curve. As the emitter of Q2 rises, Q3 turns on and becomes active. A further increase in input voltage causes Q2 to saturate, which turns off Q4 and drives Q3 into saturation. The output voltage is now $V_{ce_{sat}}$, namely the collector to emitter saturation voltage of Q3. The diode D1 is present to assure that Q4 turns off when Q3 saturates.



Circuit



Symbol

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

FP-5922

Fig. 2.2.1 TTL NAND gate circuit

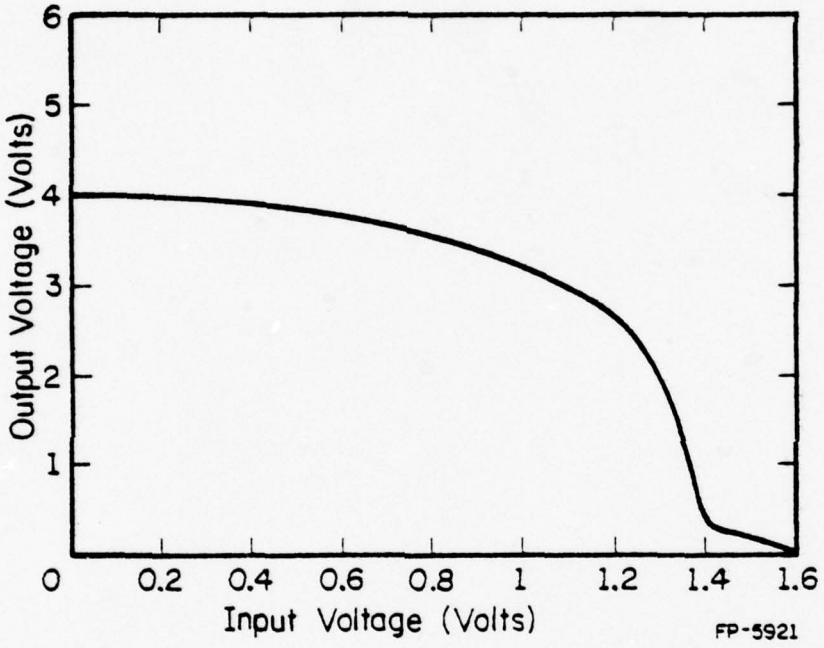


Fig. 2.2.2 TTL gate transfer characteristic

The totem pole output shown in the NAND gate circuit is used in almost all TTL logic, both SSI and MSI. The drive capability of the totem pole output is an important factor in achieving high TTL switching speeds. However, the totem pole output has an inherent problem which causes it to generate a current spike every time the output goes from low-to-high. Consider the totem pole circuit shown in Fig. 2.2.3. Assuming a logical 0 output, Q3 is on and saturated. Q4 is off. When the gate switches, Q4 rapidly goes from cut-off to active, while Q3 is attempting to turn off. The turn off requires a transition from saturation, through the active region, to cut-off. Since Q3 is initially saturated, there is a charge-storage delay while it tries to turn off. This results in a low-resistance path between V_{cc} and ground for the short interval that both Q3 and Q4 are on, which produces a current spike in the V_{cc} line. The maximum value of the current spike, $I_{cc_{max}}$, is given by:

$$I_{cc_{max}} = \frac{V_{cc} - V_{D1} - Vce_{Q3} - Vce_{Q4}}{R4}$$

The current spike previously described occurs during the logical 0-to-logical 1 output transition. A similar action occurs during the logical 1-to-logical 0 output transition. However, since the transition does not usually involve the turn off of a saturated transistor, the period of the conduction overlap is small and the current spiking effect is negligible in this direction.

Any current spikes generated appear as noise in the system and since many gates can be switching simultaneously this noise could result in signal errors. Thus the totem-pole output circuit, which is itself part of the logic circuitry is a source of noise in the system. The

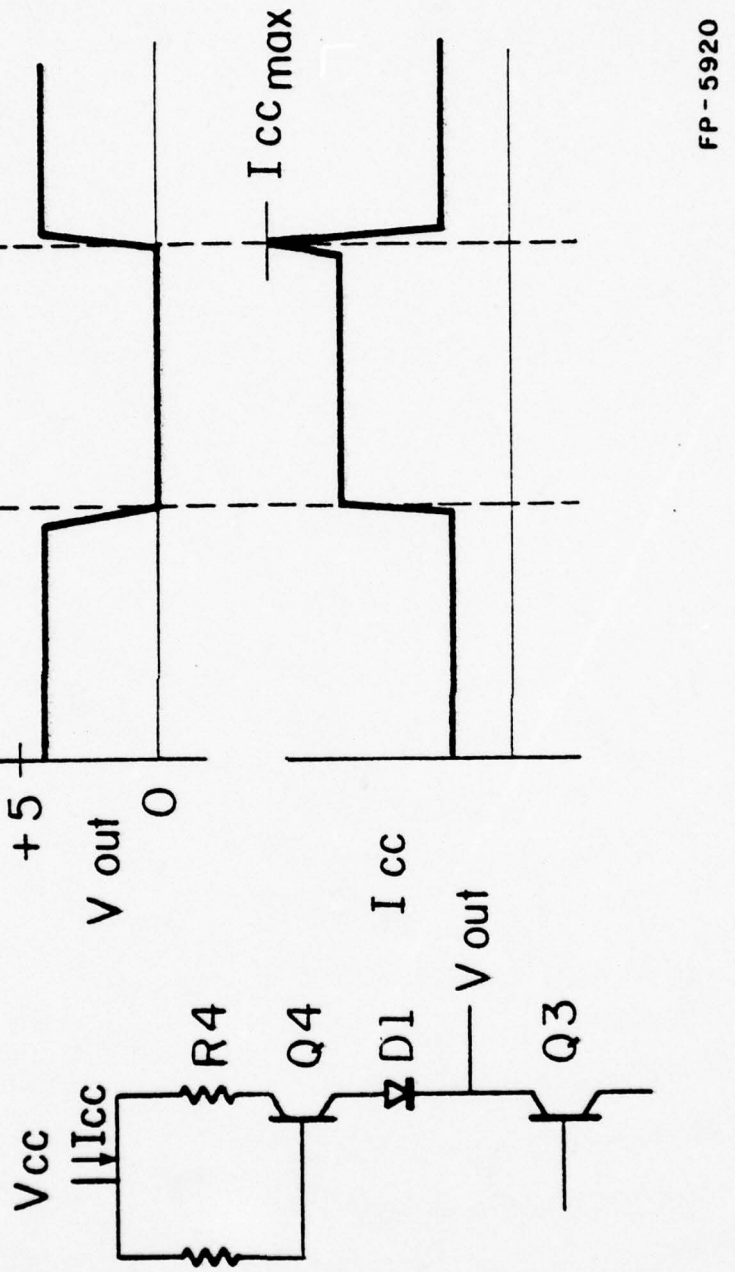


Fig. 2.2.3 Totem-pole output circuit

rather high noise immunity of TTL circuits minimizes the problems of such noise spikes. However, more serious problems exist in systems with mixed logic families. Although this current spiking cannot be eliminated, there are methods to reduce the effects of it on the rest of the system.

2.3. TTL Circuit Families

Standard 74XX TTL is designed for a reasonable balance in the trade-off between power and speed. Therefore, it is well suited for general purpose applications. Other families of TTL are available which are variations of standard TTL in that they emphasize either higher speed or lower power. The result is high speed logic that consumes significantly more power or low power logic which operates at slower speeds. A brief description of each of the TTL logic families is given below.

Standard 74XX TTL - This is the widest available and lowest priced TTL family. It also has the most functions and second sources. The typical gate delay is 10 nsec with a power consumption of 10 mW.

Schottky 74SXX TTL - Schottky TTL is the highest speed TTL family available. It is made by utilizing Schottky diodes with the transistors inside the gate. This prevents the transistors from saturating and eliminates the storage time delays within the transistors. Schottky parts are designated 74SXX. The typical gate delay is 3 nsec with a power dissipation of 19 mW.

Low-power Schottky 74LSXX TTL - Low-power Schottky TTL is a TTL family which utilizes Schottky technology to implement gates which utilize Schottky technology to implement gates which are slightly faster than standard TTL but require only 20% of the power. This is done by

utilizing Schottky transistors within the gate to gain speed and then using larger resistors which slows down the gate but also requires less power. Low-power Schottky parts are designated 74LSXX. The typical gate delay is 9.5 nsec with a power dissipation of 2 mW. Low-power Schottky is a relatively new technology. As more functions appear and second sources develop, low-power Schottky will challenge standard TTL in medium speed applications.

The wide range of performance capabilities allows the designer to optimize all portions of a system according to varying requirements. In the future, the advantages of Schottky technology will make Low-power Schottky the choice in medium speed applications and Schottky the choice in high speed applications. All families are compatible and interface directly with each other. The typical characteristics of each family are summarized in Fig. 2.3.1.

2.4. Interconnecting TTL Logic

The majority of logic circuit interconnections consist of gate outputs connected to the inputs of similar gates. To accomplish such connections, the gate inputs and outputs must be electrically compatible.

The notation for TTL voltage levels are as follows:

V_{IH} = Logical 1 input voltage level

V_{OH} = Logical 1 output voltage level

V_{IL} = Logical 0 input voltage level

V_{OL} = Logical 0 output voltage level

The relationship between these levels is illustrated by the diagram in Fig. 2.4.1.

Family	Gate Delay (nsec)	Power Dissipation (mW)	Max Clock (Mhz)
74XX	10	10	35
74LSXX	9.5	2	45
74SXX	3	19	125

Fig. 2.3.1 TTL Family Characteristics

From Fig. 2.4.1, it can be seen that the worst-case input and output levels differ by some fixed amount in each state. In other words, V_{OL} is less than V_{IL} and V_{OH} is greater than V_{IH}. This fixed difference is called the guaranteed noise immunity. This means that any gate output signal can be corrupted by a noise voltage equal to the fixed difference and still be used as a reliable gate input signal. The values for each of the TTL families is given in Fig. 2.4.2.

An input of a TTL gate requires current that the gate output which is driving it must supply, either as sink current (current toward the output) or source current (current from the output). Thus, the number of gate inputs that can be connected to a single gate output is dependent on the maximum amount of current the output can sink or source. The maximum number of inputs that can be connected to the output of a gate is called the maximum fanout. To determine the maximum fanout, the current characteristics of the gate inputs and outputs must be known. The notation for TTL gate current parameters are as follows.

I_{IL} is the current a driver must sink for a logical 0 input.

I_{IH} is the current a driver must source for a logical 1 input.

I_{OL} is the current a driver can sink and still maintain a logical 0 output.

I_{OH} is the current a driver can source and still maintain a logical 1 output.

These currents are illustrated in Figs. 2.4.3 and 2.4.4.

Thus, given these parameters, the maximum gate fanout can be found.

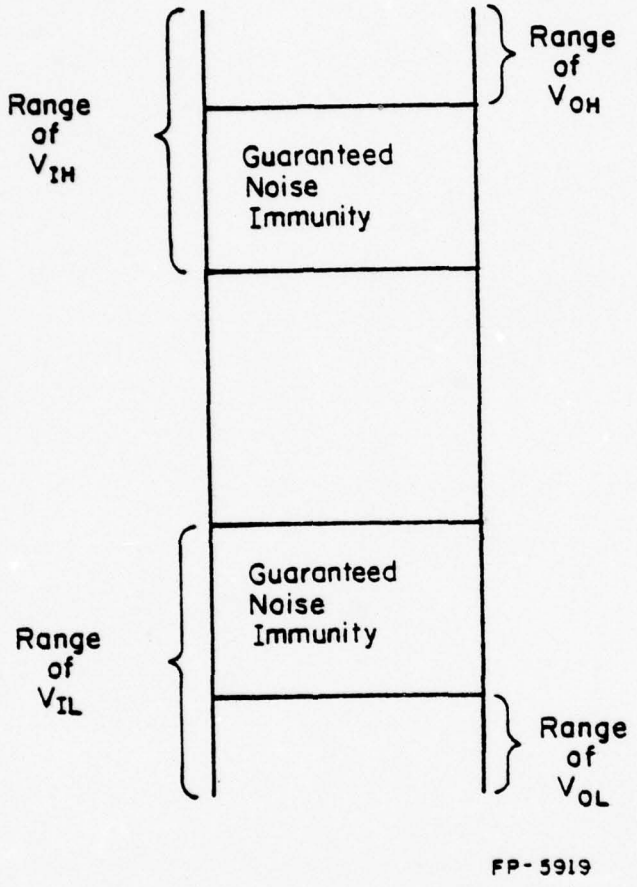


Fig. 2.4.1 TTL Voltage levels

	74XX	74LSXX	74SXX	Units
V_{OL}(max)	.4	.5	.5	volts
V_{IL}(max)	.8	.8	.8	volts
V_{OH}(min)	2.4	2.7	2.7	volts
V_{IH}(min)	2	2	2	volts
Logical 0 noise immunity	.4	.3	.3	volts
Logical 1 noise immunity	.4	.7	.7	volts

Fig. 2.4.2 TTL Family logic levels

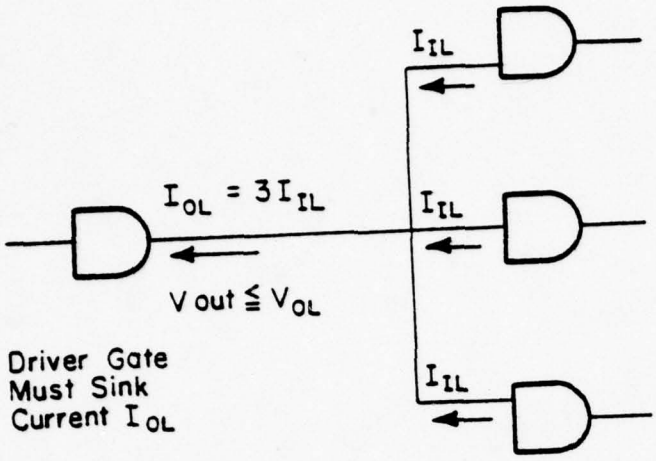


Fig. 2.4.3 Driver with logical Ø output

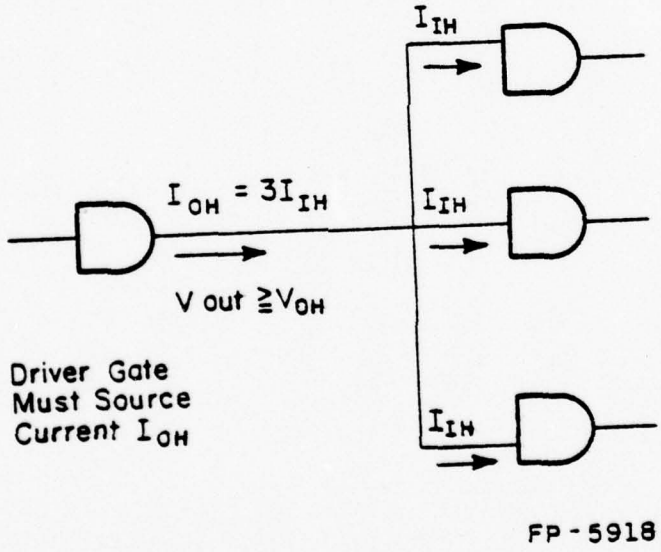


Fig. 2.4.4 Driver with logical 1 output

Maximum fanout = MIN $\{(I_{OL}/I_{IL}), (I_{OH}/I_{IH})\}$

Shown in Fig. 2.4.5 are the standard current characteristics of several TTL families. Shown in Fig. 2.4.6 is a fanout matrix which indicates how many standard inputs of a given family can be driven by an output of another family.

Circuit specifications for any particular IC will give the current capability needed to check fanout. Several TTL IC's have input pins which represent more than one load. This usually occurs in some MSI IC's because the input signal is fanned out directly to several gates within the chip.

In cases where the output is not driving similar gates, any type of load may be used as long as it does not exceed the current capability of the gate output as specified by I_{OL} and I_{OH}.

It is very important to check fanout on any given design and to eliminate violations since the consequences of exceeding the fanout are degradation of V_{OL} and V_{OH} voltage levels causing at least loss of noise immunity. If the violation is severe enough, it may result in improper operation and damage to the gate.

A gate input with no connection is referred to as a "floating" input. In TTL circuitry, a floating input is interpreted as a logical 1 input level. A floating input can be detected by measuring the voltage at the gate input. A floating input will read from 1.4 to 1.6 volts.

Floating inputs are a potential source of problems and should generally be avoided. They can cause degradation in gate performance

	74XX	74LSXX	74SXX	Units
I_{IH}	40	20	50	μA
I_{OH}	400	400	20	μA
I_{IL}	1.6	.4	2	mA
I_{OL}	16	8	20	mA

Fig. 2.4.5 TTL current parameters

	Number of Inputs		
Output	74XX	74LSXX	74SXX
74XX	10	20	8
74LSXX	5	20	4
54SXX	12	50	10

Fig. 2.4.6 TTL Fanout matrix

and in some cases can result in a false signal due often to coupling inside the IC package. Floating inputs are particularly sensitive to coupling.

Thus all unused inputs should be connected to an appropriate voltage. For a logical 0, system ground is used. For a logical 1, any voltage between 2.5 and 5.5 volts can be used. This can be done by either connecting the unused input to V_{cc} through a 1000 ohm pull-up resistor or tieing the inputs to the output of an unused gate in the system which is wired to produce a logical 1 output. Alternatively, when the logic function of a gate is not affected, some logic input to the gate can be repeated on a floating input. This does not normally increase the loading of that signal in the zero state which often has a more restrictive fanout limit.

2.5. Tri-State Logic

Tri-State logic is a variation of TTL in which the totem-pole output circuit has been modified such that it can assume one of three states. Standard TTL can assure two output states, low-impedance logical 0 output and low-impedance logical 1 output. In Tri-State logic there is a third state, the high-impedance or disabled state. Shown in Fig. 2.5.1 is a Tri-State inverter.

Circuits with Tri-State outputs have an additional input which is used to control the state of the output. This $\overline{\text{Enable}}$ control input overrides the normal gate operation. When $\overline{\text{Enable}}$ input is low, the disable circuit output (input to the multiemitter transistor is high and the gate functions as a normal TTL inverter. When the $\overline{\text{Enable}}$ input is in

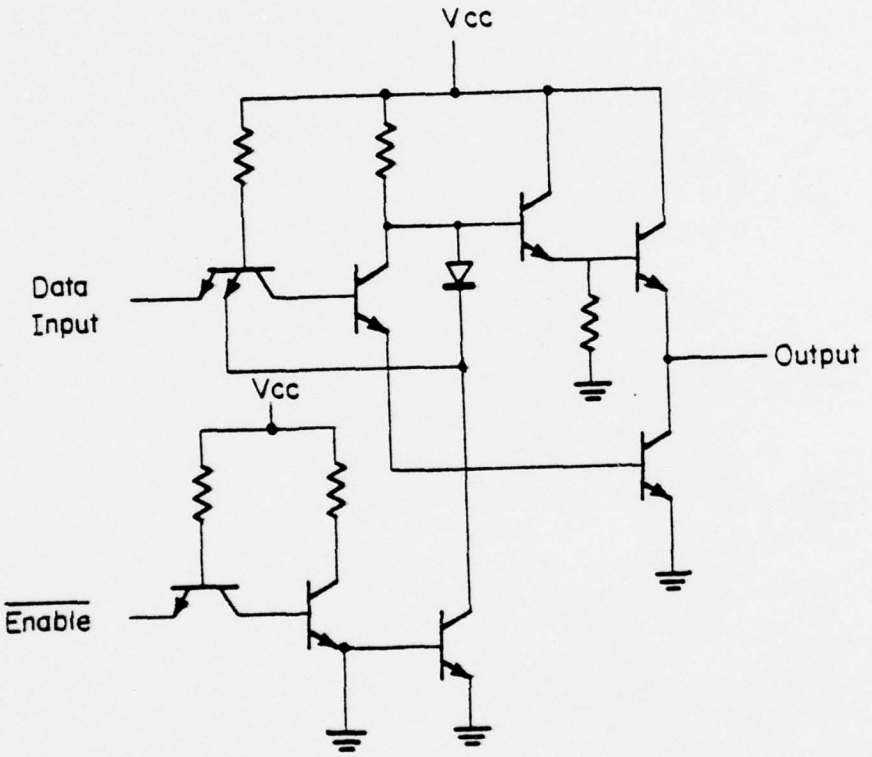


Fig. 2.5.1 Tri-state inverter circuit

the disable state (logical 1), the multiemitter transistor is turned on regardless of the data inputs thereby turning the bottom output transistor off. The diode conducts thereby turning the top output transistor off. The output is thus forced to the high-impedance state. In this state, both output transistors are off and the output resembles an open circuit, sinking or sourcing a maximum of 40 μa. of leakage current.

The development of Tri-State logic was motivated by a major shortcoming of the totem-pole output. The totem-pole output prohibits the use of wired logic. The capability to implement wired logic is very useful in a bus organized system. Thus, before Tri-State logic, open-collector TTL gates which permit wired logic were used as bus drivers. These gates are two state devices whose output is either a logical 0 or floating. The use of open collector gates requires a passive pullup resistor on each bus line which reduces system speed. The pullup risistor pulls the bus high when all open-collector gates connected to the bus present floating outputs. The propagation delay of the bus is increased due to the RC time constant of the pull-up resistor and the bus capacitance. Also the number of devices that could be attached to the bus is decreased because of the limited drive capability of the open collector output.

Tri-State logic eliminated these problems by allowing gates with totem-pole outputs to be used in a wired logic configuration. The good drive capability of the totem-pole output minimizes bus delays so that buses with Tri-State drivers are capable of TTL speeds.

Outputs of Tri-State gates are generally tied together. Only one of these gates should be enabled at a time. However, some protection

is needed should more than one be enabled at the same time. If two Tri-State outputs on a common line ever become enabled simultaneously, there is typically a built-in current limiting process to prevent destruction of the gate. In this situation however, the output level is unpredictable.

In a bus-organized system, when Tri-State devices are used as bus drivers, many devices can be hard-wired to a single bus and made to time-share that bus. Normally, all but one output on a common bus should be disabled at any given time. One line of a typical Tri-State bus is shown in Fig. 2.5.2.

This circuit is somewhat equivalent to multiplexing, an essential function in bus-organized systems. This equivalence is illustrated in Fig. 2.5.3.

This circuit equivalence is valid if exactly one enable signal is logical 1 at a time. When all enable signals are logical 0, the multiplexer bus is a logical 0 while the Tri-State bus "floats". When more than one enable signal is a logical 1, the multiplexer bus carries the "OR" of the corresponding data inputs, while the Tri-State bus is indeterminate.

The Tri-State drivers form a bus which functions as a "distributed" multiplexer in the sense that the Tri-State devices may be located in different modules of the system. The bus wire serves as a distributed OR gate.

By using Tri-State gates as bus drivers and low input current gates as bus receivers, it is possible to connect a large number of

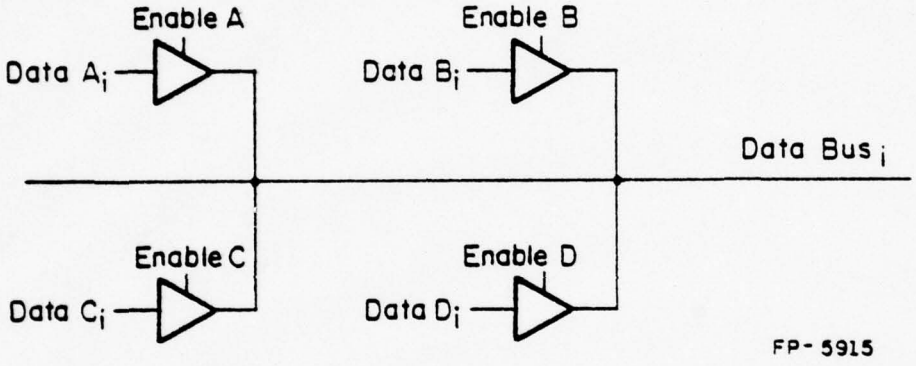


Fig. 2.5.2 Bus line with Tri-state drivers

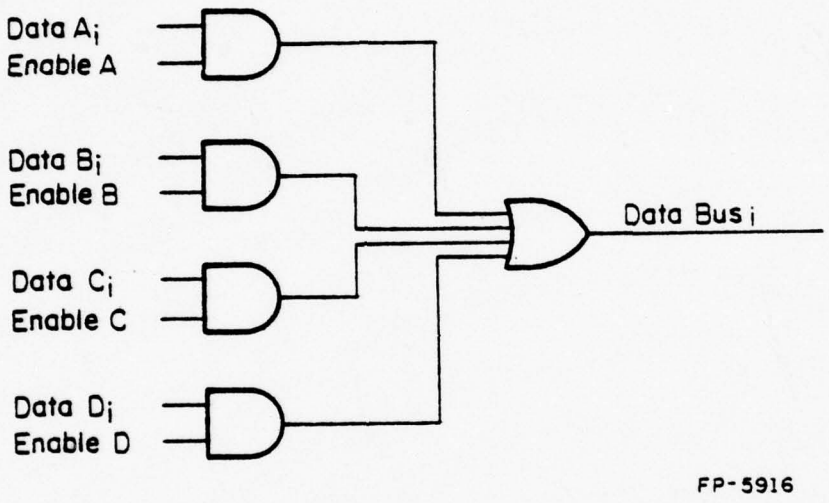


Fig. 2.5.3 Functional equivalent of Fig. 2.5.2

devices to a single bus. The chart below gives typical characteristics of a Signetics 8T95, which is a High Speed Hex Tri-State buffer package. Since the input current requirements for these gates are small, this chip can function effectively as a bus driver or a bus receiver.

Characteristics of Signetics 8T95

Bus driver characteristics	Logical 1 output current sourcing capability (I_{OH})	5.2 ma. max
	Logical 0 output current sinking capability (I_{OL})	48 ma. max
	Third state output leakage current (I_{03})	40 μa. max
Bus receiver characteristics	Logical 1 input current (I_{IH})	40 μa. max
	Logical 0 input current (I_{IL})	400 μa. max
	Third state input current (I_{13})	40 μa. max

2.6. TTL MSI Functions

TTL integrated circuits are categorized into two classes, SSI and MSI. SSI, or small-scale-integration, refers to packages which contain a small number of gates. The gates in SSI packages are typically not interconnected. Each individual gate performs a simple logic function such as NAND or NOR. MSI, or medium-scale-integration, refers to packages that typically contain 20-100 gates and perform a specific complex function. Designing systems using MSI packages reduces design time and effort because it avoids the repetitious design of commonly-used functions. Examples of available TTL MSI functions are given in Fig. 2.6.1.

Bus drivers	Bus transceivers
Registers	Decoders
Arithmetic Logic Units (ALU's)	Adders
Shift Registers	Counters
Comparators	Multiplexers

Fig. 2.6.1 TTL MSI functions

The operation of these functions is not discussed here, since the function name is self-explanatory.

The detailed description of MSI packages can be obtained from manufacturer's data sheets. Wiring constraints and behavioral properties are similar to thos discussed for SSI circuits.

2.7. Power Supplies and Power Distribution Wiring

In logic systems with a large package count there are several design guidelines normally followed to assure the proper distribution of power to each package.

The voltage output of all power supplied is specified along with a maximum current rating. No power supply should be operated with a load that draws more than the rated current. Each IC package in a system draws a specified current, denoted I_{CC}. The total power supply load, which is the sum of all package currents, must not exceed the rated supply current. The result of overloading is a drop in power supply output voltage. While most power supplies are "short circuit protected", severe overloading of an unprotected supply can cause permanent damage to the supply. Use of "overload protected" power supplies is recommended to protect circuits from line surges and power supply failures.

One technique used to implement power supplies is a microprocessor system is the use of three-terminal IC voltage regulators. The only additional parts required to implement the supply are a transformer, rectifier, and several capacitors. This approach is attractive because of the low cost involved. Regulators are available for most standard voltages and can supply currents up to five amps, thus they are sufficient for most microprocessor systems. So, one approach is to have the power supply generate all voltages necessary and then bus the regulated voltages around the system. Another approach is to bus around unregulated voltage rails and place the regulators on each card. This is an attempt to improve regulation by placing the regulators closer to where the voltage is used.

It should also be noted that the use of voltage regulators solves the current overloading problem since most regulators have a built in overload detection mechanism which causes them to shut down when the specified current load is exceeded.

The power supply in digital systems sees a varying load since when a logic device changes state, the package current, I_{CC}, displays significant variations. These variations can be attributed to two distinct causes.

1. The D.C. current level, I_{CC}, is different for each output state. The current level for a logical 0 output is denoted I_{CCL}. The current level for a logical 1 output is denoted I_{CCH}.

2. A transient current variation occurs each time an output changes state. This is due to the current spike generated by the totem-pole output of TTL circuits and the current needed to charge (or discharge) the load capacitance. TTL power requirements thus increase, greatly during high frequency operation.

While some spikes on the supply voltages are inevitable, it is essential that the supply voltage remain as stable as possible throughout the system during current variations.

Most voltage variation can be minimized by a technique called decoupling. This involves inserting capacitors between power and ground lines throughout the system. The capacitors tend to keep the power supply current load constant by storing charge to be converted to current when a momentary high current demand must be met.

The decoupling network, shown in Fig. 2.7.1, consists of a single 30 to 50 uf. capacitor and a variable number of .01 to .1 uf. capacitors. The large capacitor is placed directly across the power supply terminals and/or the power supply connection of each circuit card. The small capacitors are distributed throughout the system with approximately one capacitor for every one to six IC packages. The small capacitors should be the high-frequency (non-inductive) type, usually ceramic disc. The small capacitors suppress high frequency transients; the large capacitors, low frequency transients.

2.8. Noise in Logic Interconnections

In high-speed logic systems such as TTL, the interconnection wiring used to carry signals between packages must be done with care. When this wiring is done improperly, it increases the presence of noise in the system and is a potential cause of signal errors. The noise appearing on interconnection wiring can be attributed to two independent phenomena, crosstalk and reflections.

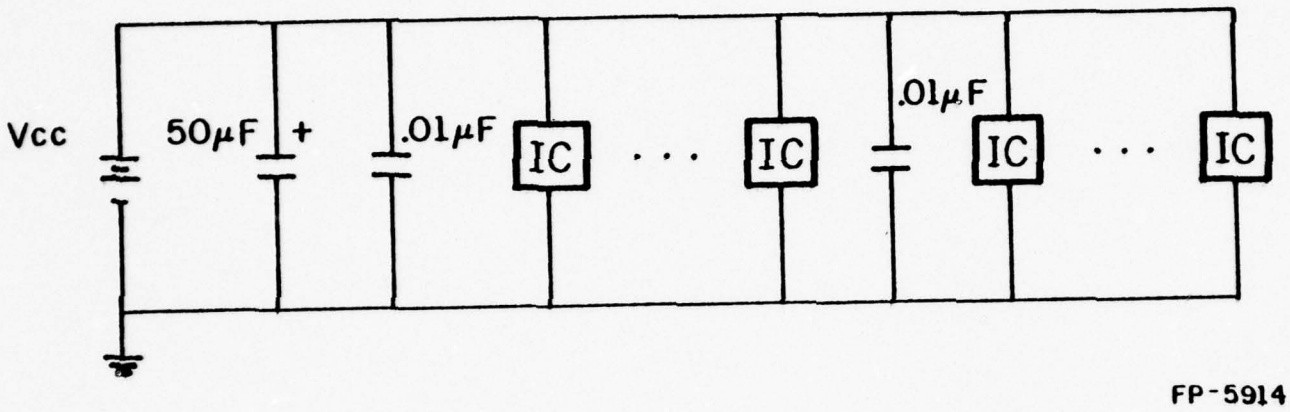


Fig. 2.7.1 Power supply decoupling network

Crosstalk refers to electrical coupling that often occurs between two adjacent conductors. Thus coupling is caused by mutual capacitance and mutual inductance that exist between the conductors. When this coupling is strong enough, a signal level change on one conductor will also appear falsely on the adjacent conductor.

TTL logic is a low impedance system, therefore most of the signal crosstalk noise occurs by mutual inductance, rather than mutual capacitance. The reason for the relatively large noise immunity to capacitively coupled noise can be shown with a simplified model. Consider Fig. 2.8.1, where R_{out} is the output impedance of a TTL gate and C_{coup} is the capacitive coupling between the signal line and some noise source.

When transient noise is coupled onto the signal line, it has a duration related directly to the time constant $R_{out}C_{coup}$. In the case of TTL logic, which has a low output impedance in both states, R_{out} is usually small and thus the duration of the crosstalk noise is short. If the duration of the noise is short enough, it may be ignored by the input of gate B. Thus in this way, TTL gates are relatively immune to capacitively coupled noise. This model is a simplification in that a distributed coupling is being modeled as a lumped capacitor, however it does illustrate the way in which the low output impedance improves capacitive crosstalk noise immunity.

Since most of the crosstalk noise in TTL circuits occurs by mutual inductance, the wiring should be done such that the mutual inductance, the wiring should be done such that the mutual inductance between lines is minimized. Mutual inductance is a function of the spacing

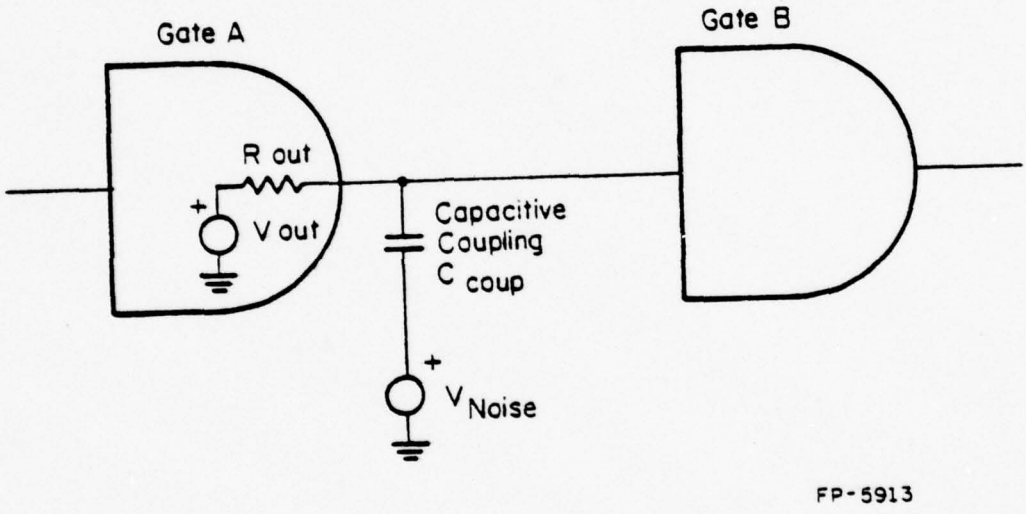


Fig. 2.8.1 TTL Noise immunity circuit model

between two conductors. Thus one method of reducing it is to avoid placing leads adjacent to each other. In situations where leads must be adjacent, such as bussing, another technique is commonly used. The mutual inductance is reduced by running the signal line in close proximity to a ground line. This reduces the amount of flux in the field around the noise sourcing wire and therefore reduces the coupling as well. This technique is used for coaxial cable, flat cable and twisted pair leads.

Reflections can occur on a TTL interconnection when the gate driving that line changes state. These reflections produce transient voltage variations originating at both ends of the line. The condition for reflections to occur is that the length of the wire (and therefore the propagation delay of the wire) be long compared to the signal rise time. If the wire is short, the reflections still occur but they occur during the rise time and therefore are not a problem. For TTL wiring, the maximum wire length for which reflections may be ignored is about 12". Longer wires behave not as simple wires, but as transmission lines.

Every transmission line is described by a parameter called its characteristic impedance, denoted Z_0. Z_0 is defined as $\sqrt{L/C}$ where L equals the inductance per length and C equals the capacitance per length. Reflections occur on a transmission line whenever it is terminated in an impedance not equal to its characteristic impedance. In TTL wiring, reflections occur because the line is terminated by the gate input impedance, which is typically 10^5 ohms. This impedance creates a mismatch since the characteristic impedance of the wire is typically 150 ohms.

In addition to reflections at the receiving end of the line, the low output impedance of TTL devices creates a mismatch which causes reflections to occur at the driving end also. Reflections from both ends of the line can result in a voltage variation called ringing. Shown in Fig. 2.8.2 are the decaying oscillations about the final steady-state value that are characteristic of a ringing signal.

Since ringing is a transient effect, one obvious solution to the problem of ringing is to slow the system down, therefore not looking at the signal until the ringing has sufficiently died down.

Another method of reducing reflections is a procedure called termination. Termination involves placing an impedance equal to the characteristic impedance at one or both ends of the line, since no reflections occur at a matched end. Termination impedances are usually implemented with discrete resistors. To use termination techniques effectively, the line must have a known and uniform characteristic impedance. There are two approaches to termination, series and parallel. Series termination places the matching impedance at the driving end of the line. Parallel termination places the matching impedance at the receiving end of the line.

Fig. 2.8.3 illustrates a configuration commonly used in parallel termination.

Resistors R1 and R2 are located at the receiving end of the line. The value of the terminating resistance is the parallel combination of R1 and R2. Thus R1 and R2 are selected such that their parallel combination is equal to Z_0, the characteristic impedance of the line.

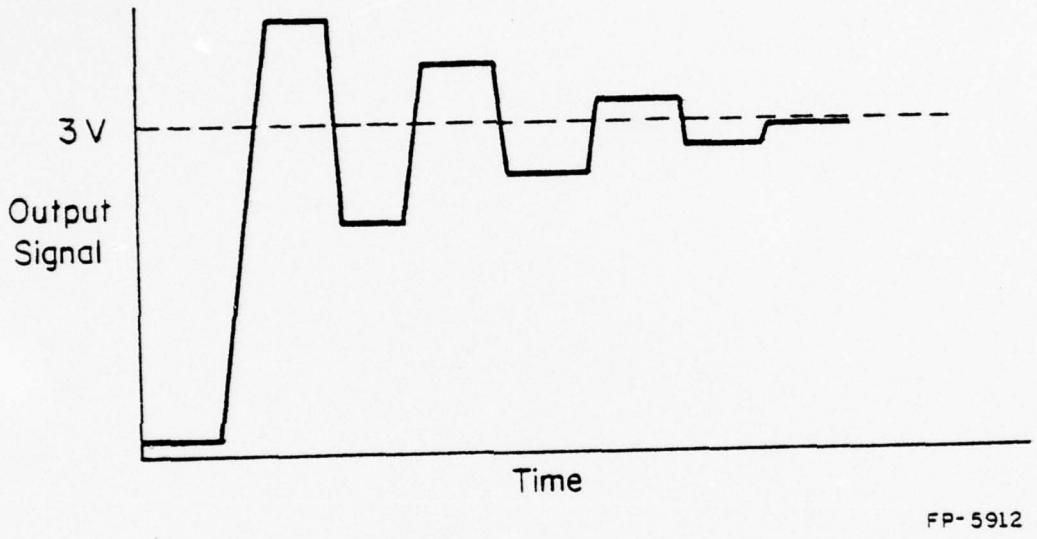


Fig. 2.8.2 Ringing waveform

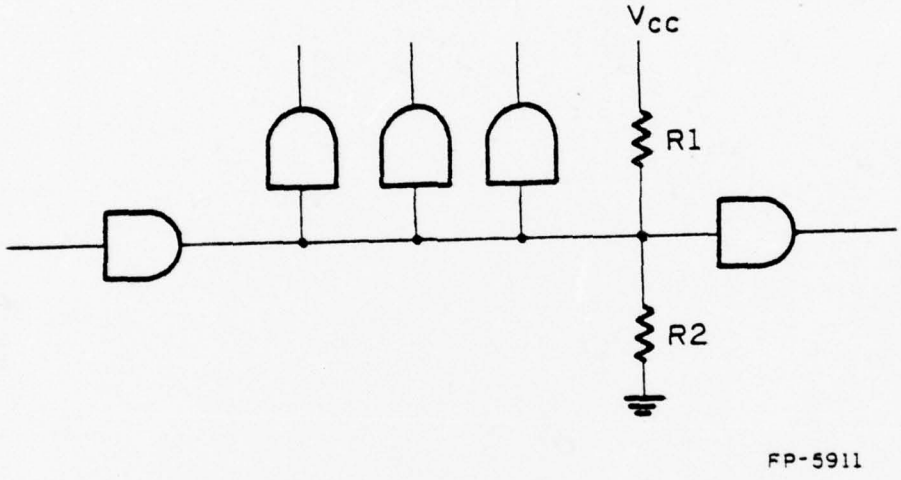


Fig. 2.8.3 Parallel termination

Two resistors are used to produce the termination impedance so that the termination can be accomplished without using an additional power supply voltage. Shown in Fig. 2.8.4 is the parallel termination circuit along with the Thevenin equivalent.

The Thevenin equivalent circuit illustrates that the termination could be accomplished with a single resistor R_{TH} and a single power supply voltage V_{TH}. However, in most small systems, the cost of obtaining the additional power supply voltage is not justified.

The next consideration is determining the values of resistors R1 and R2. There are two constraints needed. One constraint can be set by specifying V_{TH}. For TTL circuitry, V_{TH} should be equal to the logical 1 output voltage, which is typically 3 volts. The other constraint can be set by specifying R_{TH}. From the transmission line considerations given in this section, R_{TH} should be equal to Z_0.

Thus the two constraints for determining R1 and R2 are:

$$\frac{R1 \cdot R2}{R1 + R2} = Z_0$$

$$\frac{R2}{R1 + R2} \cdot V_{cc} = 3$$

Therefore, given Z_0, R1 and R2 are determined.

An advantage of parallel termination is that it allows distributed loading, thus receivers can be distributed the entire length of the line. A disadvantage of parallel termination is the large current that flows through R1 when the driver output is a logical 0. The power supply must be capable of supplying this additional current

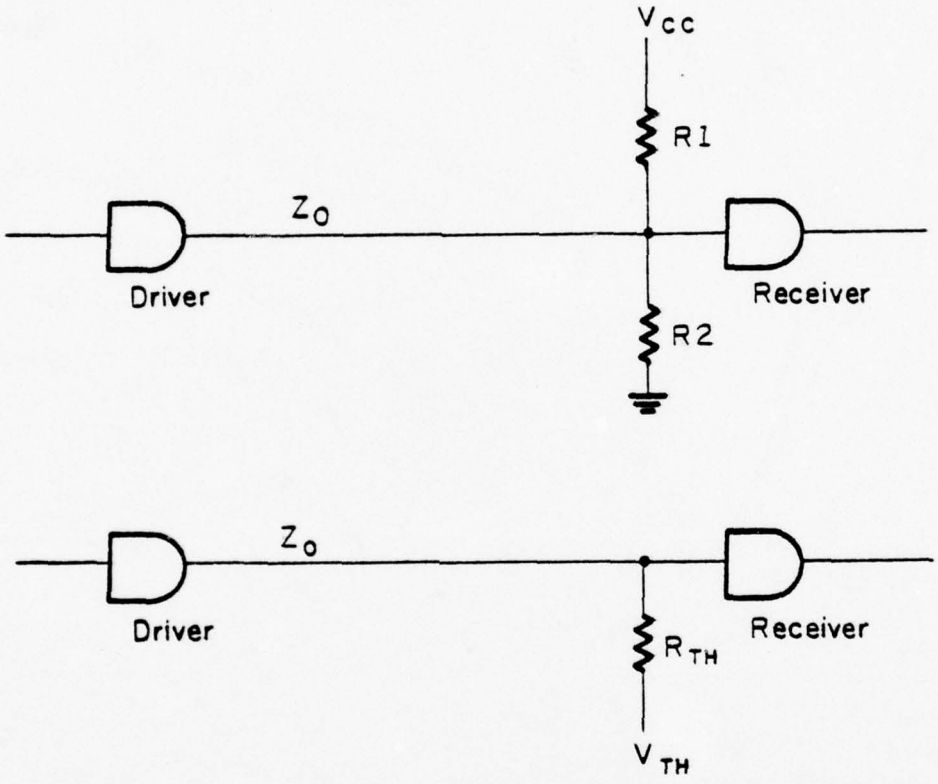


Fig. 2.8.4 Thevenin equivalent of parallel termination

and the gate driving the line must be capable of sinking this additional current.

Parallel termination is accomplished by selecting R1 and R2 to match the characteristic impedance Z_0. Shown in Fig. 2.8.5 is a chart which gives the values of R1 and R2 for values of Z_0 to to 360 ohms. The graph is constructed by simultaneously solving the two constraints given above.

After selecting R1 and R2, the bussing circuit should be checked to verify that the drivers and receivers are not operating under conditions that exceed their specifications.

The bus drivers must be capable of driving a load that consists of all the receiver inputs in addition to the current through the termination resistors. The worst loading occurs when the driver output is a logical 0. As the value of R1 decreases, this loading increases. The value of R1 and R2 which give a perfect impedance match may result in currents that exceed the capability of the bus driver. In that case, values of R1 and R2 are used which are as close to proper termination as possible without exceeding the capability of the bus driver. When terminating a line, a partial match is better than no match at all. A partial match may reduce the magnitude of the reflections to an acceptable level. The goal is to obtain as close a match as the circuit conditions allow.

Series termination requires a resistor between the driver and the transmission line, as shown in Fig. 2.8.6. The receiving end has no termination resistance. The series resistor value should be

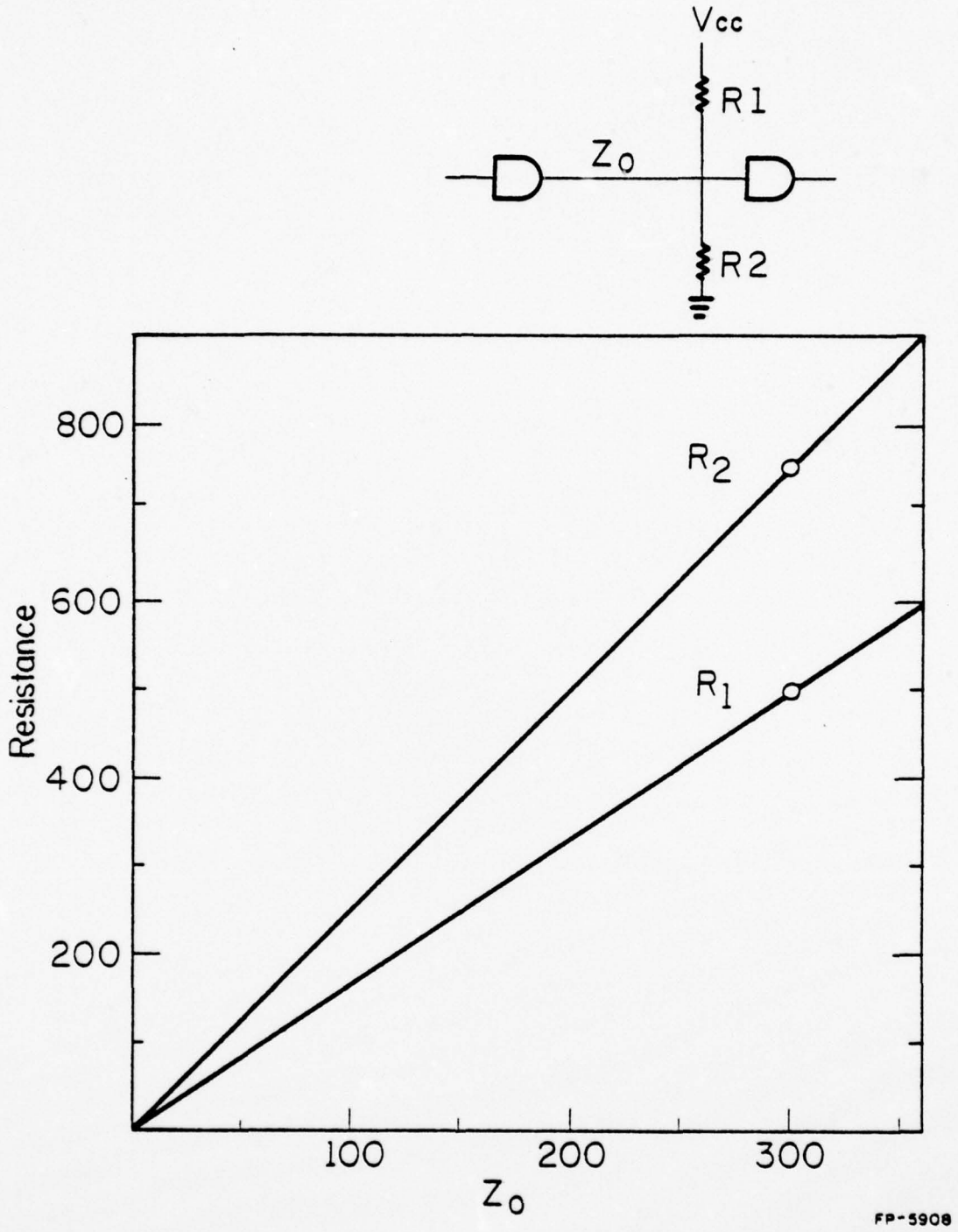
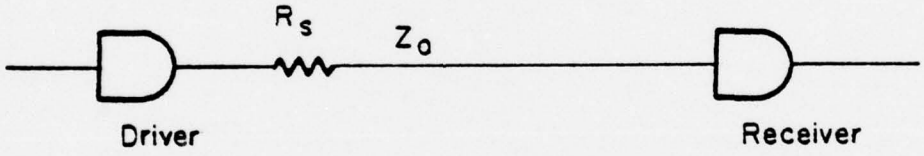


Fig. 2.8.5 Graph to select termination resistors



$R_s = Z_o - R_{out}$

R_s = Series Termination Resistor

R_{out} = Driver Gate Output Impedance

FP-5907

Fig. 2.8.6 Series termination

selected so that when added to the driver output impedance, the total resistance equals the characteristic impedance of the line.

Any reflections along the line that return to the source are absorbed. Series termination is also useful for controlling overshoot and undershoot that may damage receiving devices. An advantage of series termination is the low power dissipated in the termination. A disadvantage is that all the receiving gates must be located at the end of the line, thus receivers cannot be distributed along the bus. Another problem with series termination is that when the value of the resistor becomes large, the resultant voltage across that resistor degrades the noise immunity.

Series damping is similar to series termination except that in selecting the resistor value, no attempt is made to match the characteristic impedance of the line. Instead, a small (27-47 ohm) resistor is inserted between the driver and the transmission line, as shown in Fig. 2.8.7. This gives a partial match and reduces overshoot and undershoot. This technique is useful when the characteristic impedance of the line is unknown or the voltage drop across a proper R_s cannot be tolerated.

When building logic circuits in a breadboarding situation, the single most important wiring practice to follow is the use of short leads. Always make leads as short as possible and try not to exceed 12" in high speed circuitry. Another practice to follow is to avoid running wires in neat, parallel bundles. Instead, use the more random point-to-point wiring. These practices will minimize both crosstalk and reflections.

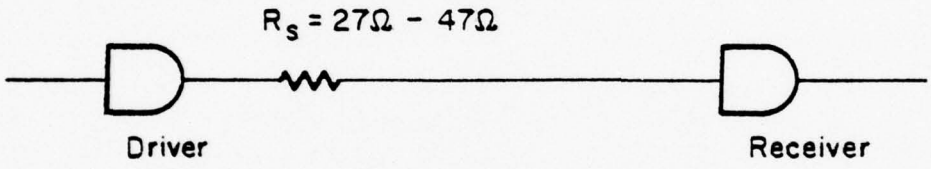


Fig. 2.8.7 Series damping

In situations where longer connections are needed, such as bussing between circuit modules, termination is needed to assure reliable operation. Parallel termination is necessary when distributed receivers along the bus are required. For a unidirectional bus, termination is done at the receiving end, while for a bidirectional bus, both ends are terminated. The use of termination implies the bus cable has a uniform characteristic impedance. Flat cable is currently the most widely used interconnection in bussing applications because it has a convenient variety of available connectors and has a uniform characteristic impedance when used in the GROUND-SIGNAL-GROUND configuration, which means every other conductor is grounded for signal isolation. Terminations are applied to the SIGNAL lines only. Shown in Fig. 2.8.8 is a table of the properties of Scotch-Flex flat cable.

In practice, the most common termination is the parallel termination with $R_2 = 330\Omega$ and $R_1 = 220\Omega$. These values are proportionately scaled upwards when the driver cannot sink adequate current. If bidirectional bus drivers have adequate current sinking capability, terminations are used at both ends of the line rather than one. Note further that ground isolation of flat cable signals may be omitted between adjacent signals such that neither one changes state near the time that the other is sampled by any receiver, i.e. if it is assured that any cross-coupling effects will have died out whenever a signal value is required to be stable.

TYPICAL PROPERTIES OF Scotchflex BRAND FLAT CABLES

3M Part Number	3350/__	3366/__	3401/28	3306/__	3451/24	3405/28 3405/36	3380/__	3469/__	3476/__
Insulation Material	Polyvinyl Chloride (PVC) U.L. Rated FR-1						Polyvinyl Chloride (PVC) U.L. Rated FR-1		
Color	Gray						Gray		
Edge Marking	Red		None	Red			Green	Red	Black
Center Spacing	.050" (1.27 mm)				.0625" (1.59 mm)	.075" (1.91 mm)	.050" (1.27 mm)		
Conductor	Solid Copper	Stranded Copper		Solid Copper	Stranded Copper		Solid Copper	Stranded Copper	
Conductor Size	30 AWG	28 AWG (7x36)		26 AWG	24 AWG (7x32)		30 AWG	28 AWG (7x36)	
Conductor Quantity	10 14 16 20 24 26 34 40 50	10 14 16 20 24 26 34 40 50	28	10 14 16 20 24 26 34 40 50	24	28 36	40 46 50	14 16 20 26 34 40 50	20 26 34 40 50
Impedance	125 ohms	105 ohms		95 ohms	85 ohms	100 ohms	75 ohms	65 ohms	
Capacitance	12 pF/ft. (39 pF/m)	15 pF/ft. (49 pF/m)		14 pF/ft. (46 pF/m)	16 pF/ft. (52 pF/m)	13 pF/ft. (43 pF/m)	24.0 pF/ft. (78.7 pF/m)	29.0 pF/ft. (95.1 pF/m)	
Inductance	0.23 μH/ft. (0.75 μH/m)	0.20 μH/ft. (0.66 μH/m)		0.16 μH/ft. (0.52 μH/m)	0.17 μH/ft. (0.56 μH/m)	0.17 μH/ft. (0.56 μH/m)	0.20 μH/ft. (0.66 μH/m)	0.14 μH/ft. (0.46 μH/m)	
Propagation Delay	1.42 ns/ft. (4.66 ns/m)	1.40 ns/ft. (4.59 ns/m)		1.42 ns/ft. (4.66 ns/m)	1.29 ns/ft. (4.23 ns/m)	1.33 ns/ft. (4.36 ns/m)	1.68 ns/ft. (5.51 ns/m)	1.65 ns/ft. (5.41 ns/m)	
Insulation Resistance	$>10^{10}$ ohms/ 10 Ft. (3 m)						$>10^{9}$ ohms/ 10 Ft. (3 m)		
Voltage Rating	300 VRMS						300 VRMS		
Temperature Rating	-4°F to +221°F (-20°C to +105°C)						-4°F to +221°F (-20°C to +105°C)		
U.L. Style Number	2651						2682		

Fig. 2.8.8 Properties of Scotch-Flex flat cable

2.9. TTL/MOS Interfacing

Many systems contain a combination of TTL and MOS logic. These systems require that some TTL outputs drive MOS inputs and some MOS outputs drive TTL inputs. Most MOS integrated circuits are now built with inputs and outputs that are compatible (or nearly compatible) with TTL voltage levels, thereby allowing direct interconnection and eliminating the need for level conversion circuits. In situations where the signals are not compatible, an interface circuit must be used.

Whenever using MOS integrated circuits, it is good practice to check the V_{OL}, V_{OH}, V_{IL}, and V_{IH} specifications. Often MOS inputs and outputs that are specified as "TTL compatible" are not truly so. True TTL compatibility means that these specifications will be the same as the TTL counterparts. An example of this is the Intel 8080 which has MOS inputs which have a V_{IH} specification of 3.3. Since TTL only guarantees a V_{OH} of 2.4, the 8080 may not receive a logical 1 when the TTL gate is driving in a valid logical 1 level.

The input impedance of an MOS gate is mainly capacitive. Thus, the input current is small and the only effect on the TTL gate driving in is increased load capacitance which can increase the rise time.

Next consider an MOS output driving a TTL gate. MOS outputs are not well suited for supplying the drive current (I_{IH}, I_{IL}) required by TTL inputs. The current drive capability of an MOS output is proportional to the size of the MOS devices in the output stage.

Therefore, the current drive is designed to be very small and even most TTL compatible MOS outputs have a fan-out of 1 standard TTL load. Often, MOS outputs are interfaced to Low-power Schottky TTL since it requires much less drive current.

In many systems MOS inputs and outputs are connected to a bus. This requires that the MOS outputs drive the bus, i.e., both the AC and the DC load. The DC load consists of the input currents of the gates connected to the bus and the leakage currents of the three-state outputs connected to the bus. The AC load consists of capacitance created by the devices on the bus, along with the capacitance of the interconnection. The AC load can be calculated using the following:

TTL input 5=pf.

PNP TTL input=10 pf.

Three-state output=5 pf.

PC trace 3=pf/inch

This gives the total AC load seen by the output. If this is greater than the specified load, the effect is an increase in propagation delay. The amount of delay is found by derating according to the specifications of the MOS output. When either the DC loading or the decrease in speed is unacceptable, the problem can be solved by using TTL bus drivers.

Wiring Guidelines Summary

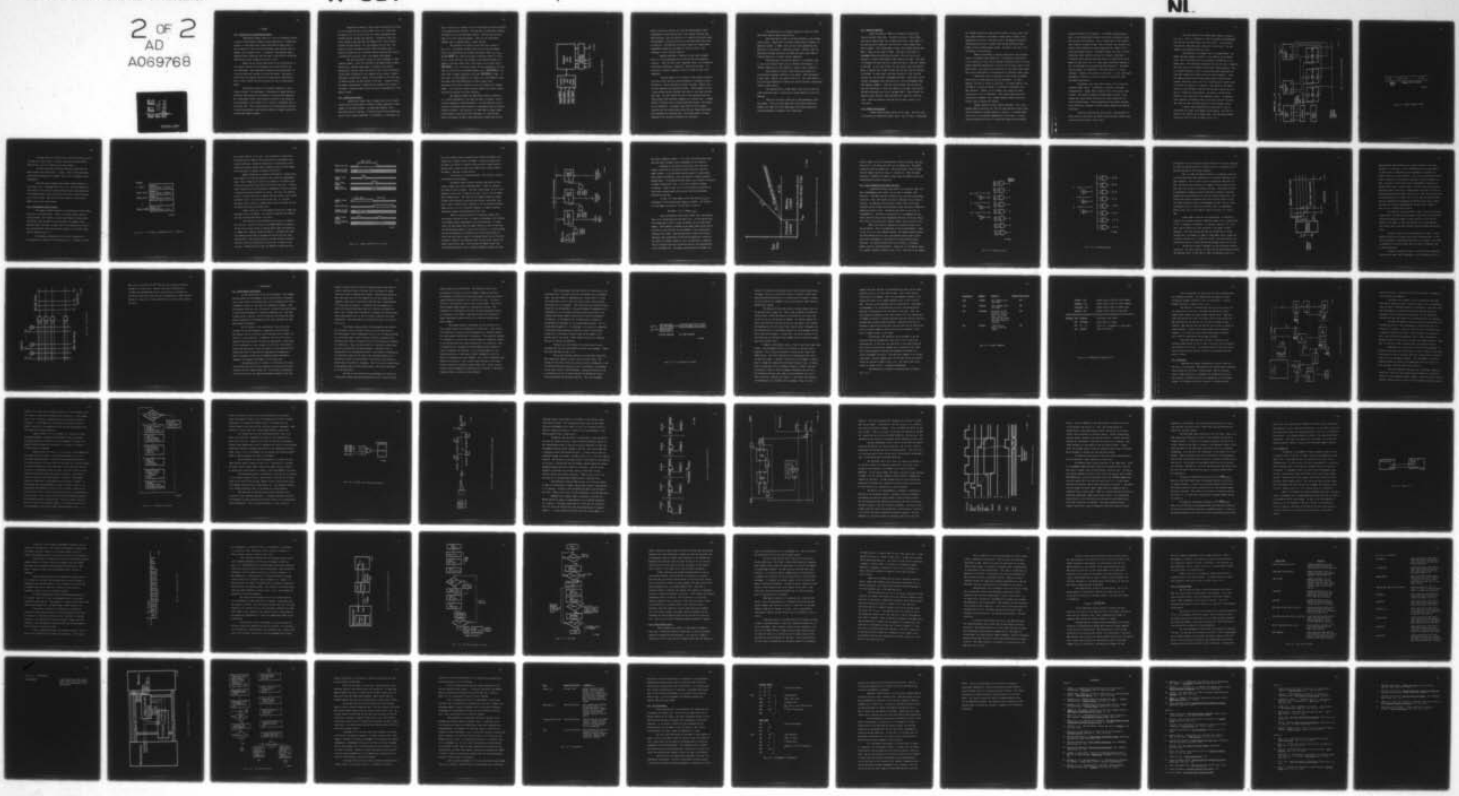
1. Follow TTL Loading Rules to avoid problems caused by exceeding the fanout capability of a gate. (Section 2.4)
2. Avoid floating inputs by tieing unused inputs to the appropriate levels. (Section 2.4)
3. When using MSI functions, check the loading for each input pin, since it often may represent more than 1 standard TTL load. (Section 2.6)
4. Flip-flops should not be used to drive long lines because reflections may cause unwanted transitions to occur in the flip-flop.
5. Check that the power supply current capability can match that needed for the system being powered. Also, make sure there is a low impedance path between the power and ground pins on each package and the corresponding power supply terminal. (Section 2.7)
6. Decouple the circuit by placing one 30 to 50 μf capacitor across the power supply connection to the circuit board. In addition, place one .01 μf ceramic disk to other suitable by-pass capacitor across the power and ground lines for every one to six IC's in the system. (Section 2.7)
7. When wiring signal lines, use short leads. Do not run leads in parallel bundles, use point-to-point wiring. For long lines where reflections are a problem, use some form of termination. (Section 2.8)
8. When using flat cable use GROUND signals to isolate signals for which cross-coupling cannot be tolerated. (Section 2.8)

AD-A069 768 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
THE DESIGN OF MICROCOMPUTER SYSTEMS.(U)
JUL 78 T A LANE

UNCLASSIFIED R-817 DAAB07-72-C-0259

NL

2 OF 2
AD
A069768



END
DATE FILMED
7-79
DDC

3. MEMORY

3.1. Introduction to Semiconductor Memory

Semiconductor memory refers to a class of integrated circuits in which a large-capacity memory has been fabricated on a single chip. Advances in technology have produced semiconductor memory which is more attractive in both price and performance than previous types of memory, such as magnetic cores. In addition, semiconductor memory possesses unique properties that have made it applicable in situations where previous types of memory could not be used.

Memory was one obvious application of LSI technology due to its regular structure and was therefore one of the first LSI circuits developed. After semiconductor memory was developed there was a need for additional systems to utilize the memory. Some people believe that the original reason for introducing the microprocessor was to create a market that would support the sale of semiconductor memory.

Semiconductor memory is an essential component in microcomputer systems. The development of semiconductor memory and microprocessors made possible the realization of low-cost microcomputers. The system memory should receive special attention in the design of any microcomputer, since memory cost is usually a significant part of the system cost. This is easily seen since a system usually contains a single microprocessor whose operation requires a memory system that contains many memory packages.

Semiconductor memory is usually made by fabricating an array of 1-bit storage circuits, called memory cells, on a single chip. The memory cells are circuits which are usually implemented with standard bipolar and MOS logic families. In addition, other circuitry needed to read and write the memory cells, such as address decoders and data buffers, are also designed into the chip. The ability to place the address decoder within the memory chip is essential because the use of external address decoding would require a prohibitively large pin count on the memory chip package.

The above discussion reveals two basic advantages of semiconductor memory. Since the circuitry on the memory chips is implemented with a standard logic family, the memory chips possess compatibility with the logic that forms the rest of the system which allows simple interfacing to the remainder of the system. Another advantage of semiconductor memory is the placement of circuitry such as address decoders and data buffers on the chip. Because of this, the memory system design is simplified and package count is kept to a minimum. Individual memory chips are easily interconnected to form memory systems.

3.2. Random Access Memory

Random access memory (RAM) is memory which has the characteristic that the time needed to execute a memory operation is independent of the memory address. An additional property, often associated with the term RAM, is that both read and write operations must be done in times independent of the address. In operation, the

data is written into a memory cell at a fixed address location specified by the memory address provided. The same data is retrieved by reading the memory cell at the same memory address. The data does not move from cell to cell inside the memory. There are two technologies commonly used to implement RAM's: bipolar and MOS.

The structure of a typical static RAM chip is shown in Fig. 3.2.1. The address decoders select one cell from the memory array to be operated upon by decoding the address inputs A_0 to A_{N-1}. The READ/$\overline{\text{WRITE}}$ line controls the mode of operation (read if 1, write if 0). In the read mode, the contents of the addressed memory cell appear at the data output after a delay equal to the access time if $\overline{\text{CHIP SELECT}}$ is low. In the write mode, the data appearing at the data input is written into the addressed memory cell if $\overline{\text{CHIP SELECT}}$ is low. Notice that no memory operations occur when $\overline{\text{CHIP SELECT}}$ is high. In this state, the DATA OUTPUT pin is normally at a high impedance state. Such three-state DATA OUTPUT pins of several memory chips associated with distinct addresses can be tied together for expanded memory. Also, there are timing relations between the various signals that must be observed for proper operation.

One inherent property of semiconductor RAM is volatility, i.e., stored information is lost when the power is removed. This is a serious drawback in some applications. In cases where the loss of data due to accidental power removal is unacceptable there are several possible solutions. One solution is to sense when the power supply voltage begins to drop and at that time branch to a routine then places the contents of RAM in some nonvolatile storage such as disk.

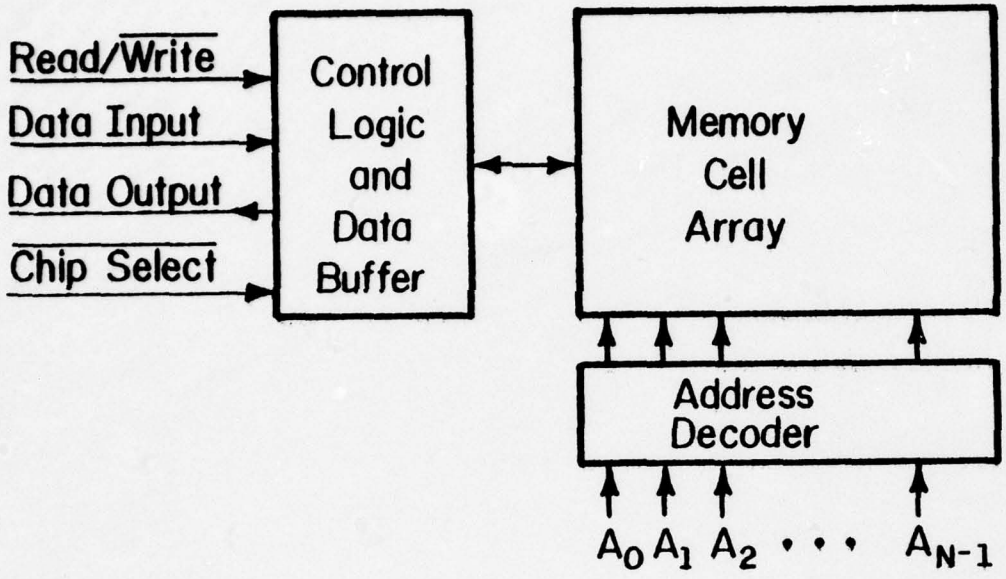


Fig. 3.2.1 Static RAM chip

Another solution is based on the fact that MOS RAM memory power requirements are much smaller when in the standby condition than when operating. Thus the solution is to sense when the power supply voltage begins to drop and switch to an alternate power source such as batteries. The batteries can usually supply the standby power requirements needed for the memory to retain its data. This technique is called "battery back-up".

Bipolar technology is familiar to most logic designers since it is the technology used to build TTL and ECL integrated circuits. There are three basic components which can be integrated with bipolar technology: transistors, diodes and resistors. All circuitry on bipolar integrated circuits is based on these three components.

A bipolar memory cell is simply a cross-coupled transistor flip-flop circuit which has been redesigned for integrated circuit fabrication. Bipolar memories are faster than MOS. However, they are more expensive and consume more power. Bipolar memory also has a lower packing density than MOS, thus there is less logic per unit chip area. One major reasons for this is the isolation that must exist between each individual component on a bipolar chip. A common method of isolation uses a reverse-biased PN junction. This method has two drawbacks. It requires large areas for the isolation junctions and the reverse-biased junctions have large capacitances which degrades the switching time. The packing density is being improved as new isolation techniques are discovered.

The characteristics of bipolar RAM make it ideal for high-speed-small capacity memory applications.

MOS technology is based on the MOS transistor as the primary circuit device. There exists two types of MOS devices, P-channel, or PMOS and N-channel, or NMOS. Early circuits were implemented with PMOS since it was easiest to fabricate reliably. However, since the fabrication problems associated with NMOS have been solved, NMOS is preferred because it is faster and TTL compatible.

MOS technology is attractive because it is relatively easy to lay out and requires fewer steps to fabricate than bipolar. In addition, MOS gates are more powerful in terms of logic capability.

When compared to bipolar, MOS provides much denser circuitry because component isolation is not needed. Power dissipation is much lower. However, the speed is much less. The slower speed is due to the high impedances of MOS devices and the parasitic capacitances present in MOS circuits.

The characteristics of MOS, namely dense circuitry and low power dissipation make it attractive for large capacity, low cost LSI memories.

There are two types of RAM cells in MOS technology, static and dynamic. Static cells are made with cross-coupled flip-flops. Dynamic cells make use of the capacitance of the MOS transistor by storing information as charge on this capacitance.

3.3. Read-only Memories

Read-only memories (ROM's) are memories in which each location contains fixed data. The ROM is actually a form of RAM, since every location must have a fixed access time. Since ROM's do not have the write capability, the circuitry is much simpler than that for RAM's. This allows ROM's to have a much higher density than RAM's. There are two types of ROM's. One type is that which is programmed during the fabrication process. The last step in processing is to form a metalization layer defined by a mask. This mask is customized for each user and is made such that it fixes the contents of the ROM. In this type of ROM, the user cannot alter the contents. There is another type of ROM in which the user may alter the contents of the ROM. In these cases, the write operation is very slow and requires the use of special programming hardware. ROM's which can be altered (programmed) after fabrication are called Programmable Read-Only Memories (PROM's). These are useful because they provide the user the flexibility to alter the contents of the ROM. Both bipolar and MOS technologies are used to implement ROM's. Bipolar ROM's are typically small and fast, while MOS ROM's are large and relatively slow. ROM's are typically more than one bit wide, usually 4 or 8 bits wide.

3.4. Memory System Design

Memory system design consists of two steps. The first step is selecting the appropriate memory chips. This is done by considering

the intended application along with such factors as access time, cycle time, power dissipation, the number of power supply voltages, logic compatibility, and cost. The second step is designing the system by interconnecting an array of memory chips and any additional logic needed to form a working memory system. Included in this step is the interfacing to the processor.

Memory system design is greatly simplified by the characteristics of the memory chips themselves. One important characteristic is the inclusion of logic such as decoders and buffers within the chips. Another important characteristic is the sue of Tri-state outputs on the memory chips. Such outputs are enabled by the chip enable input signal. This greatly simplified the interconnection of individual memory chips.

Memory chips can be classified as either static or dynamic, which refers to their method of storing information. In static memories, the data is stored in flip-flops. In dynamic memories, the information is stored as charge on the parasitic capacitance of an MOS transistor. However, due to leakage, this charge can escape resulting in a loss of information. Thus dynamic memories must be periodically refreshed. Thus, dynamic memories require additional control logic to provide the refresh.

Dynamic memories do have several advantages. One is cost: dynamic memories typically cost 33% less than equivalent static chips. Another advantage of dynamic memories is density, i.e. dynamic RAM's have a four to one capacity advantage over static chips. In small systems requiring less than 4 K of RAM static memories are probably

preferred because of the simplicity. In systems requiring greater than 4 K, the dynamic RAM's become an attractive alternative because they are more cost-effective. This is because most of the overhead logic required in memory systems, such as latches, bus receivers, bus drivers, parity checkers and parity generators are common to both static and dynamic systems. In dynamic systems, the logic to control refreshing, which typically consists of a counter, multiplexer, oscillator, shift register and clock driver are shared by the entire memory and add little cost to a large memory system when compared to the cost advantage of the dynamic RAM chips themselves. The memory itself, however, is unavailable to the system while it is being refreshed. Normally, this results in very little, if any, performance degradation. However, it can be a problem in real-time or high-performance applications.

Furthermore, most static RAM's require only a single TTL compatible power supply. In addition, the design is relatively straightforward. Dynamic RAM's usually require several power supply voltages. The design and debug of dynamic memory systems is more complex than static memory systems and can pose problems for the novice system designer. Some microprocessors have memory interface chips specifically designed to control dynamic memories and simplify the design.

Typically, RAM chips are one bit wide. When designed for words that are K bits wide, the memory system contains K memory chips in parallel with suitable control logic.

The block diagram of an example memory module is shown in Fig. 3.4.1. This is a diagram of a 1024 by 8 memory card implemented with 1024 by 1 bit RAMs (such as the Intel 2102). In other words, each card has 1024 words where each word is 8 bits wide. The card assumes 16 bit addresses and 8 bit data.

The operation of the memory card is straightforward. The memory operation begins by storing a 16 bit address in the address latch. The 10 low-order bits are connected by a common bus to all memory chips. The remaining 6 high-order bits are fed into a comparator. The other input to the comparator comes from a switch register. This 6 bit switch register is set to a bit pattern which established the high order address bits pattern which this card recognizes. When the compare between the high order memory address and the switch register indicates a match, the card is selected and the memory chips on the card are enabled. To complete a read operation, the memory chips are cycled through a read and then the data is gated onto the data bus through Tri-state buffers. The Enable on the Tri-state buffers is also controlled by the comparator output. To complete a write operation, the memory stores a 8 bit data word into the Write Data Latch and then the memory chips are cycled through a write.

A memory system implemented with this type of card is partitioned into equal size segments called modules. The size of a module is the capacity of one memory card. Thus the memory address consists of two bit fields as shown in Fig. 3.4.2.

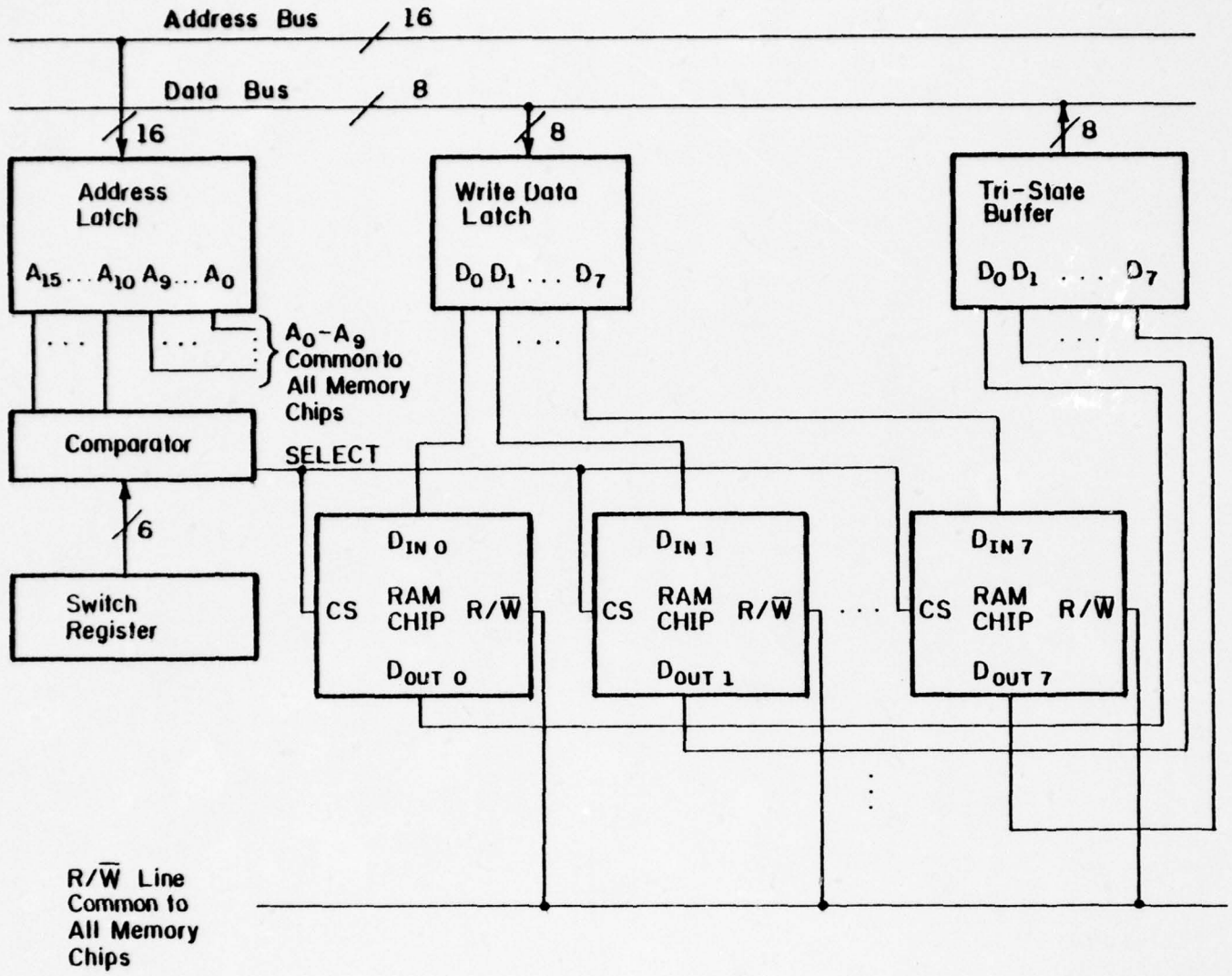


Fig. 3.4.1 A typical memory module

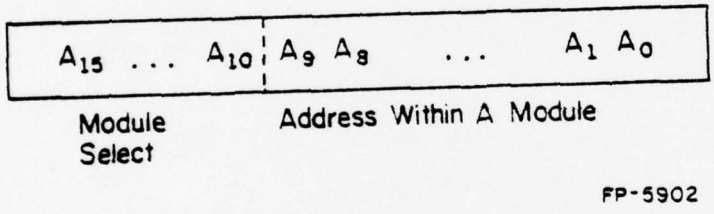


Fig. 3.4.2 Memory address format

The high order bit field selects a particular module and the low order bit field selects a location within the selected module. Shown in Fig. 3.4.3 is a memory map of this system.

It can be seen that the switch register setting maps the memory module into a particular 1 K space. Thus by using the appropriate switch settings on the memory cards, a 64 K x 8 memory can be implemented.

With this type of memory card, memory system expansion is very simple, up to a maximum size of 64 K x 8. The easy expansion is facilitated by the use of Tri-state buffers which provide modularity. To add memory cards, simply daisy-chain the control signals and data bus to the new card. Then set the switch register to map the new memory card into an unused module.

3.5. Microcomputer Memory Systems

The amount of memory required in microcomputers varies widely according to the application. Almost all systems require some non-volatile memory, such as ROM, to store initialization routines which are executed when the power is first applied. The amount of RAM required varies from none, in which case the ALU register file and on-chip microprocessor stack serve as data storage, to relatively large amounts (greater than 4 K).

One important characteristic of the use of memories in microcomputers is typically low utilization, i.e., the memory is used

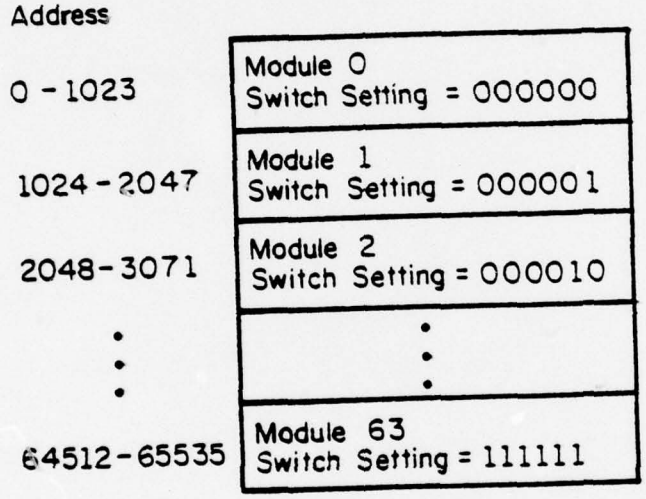


Fig. 3.4.3 64 K memory implemented with 1 K modules

only a small portion of the time. Low utilization is particularly pronounced when the memory and microprocessor are implemented with similar technology. During the execution of an instruction, microprocessors typically execute three nonmemory cycles for every memory cycle resulting in a memory utilization of only 25%.

Memory operations are typically performed in a single cycle, since memory cycle times and processor cycle times are approximately equal. Thus a "Read cycle" sends out an address to the memory early in the cycle and reads in the data at the end of the cycle. A "Write cycle" requires one or two microcycles, depending on the bus structure. A single bus structure requires two cycles. One cycle, the "Address cycle", is used to send out the address and then another cycle, called the "Write cycle", is used to send out the data. In a two-bus structure, this is done in a single "Write cycle" by sending out both the address and data in the same cycle.

Shown in Fig. 3.5.1 is the machine cycling for a single bus processor such as the IMP-16. The single bus requires that address and data be multiplexed within a bus cycle.

The read cycle, which requires one bus cycle, is shown in Fig. 3.5.1(a). Early in the cycle, the memory address is placed on the bus and an output strobe is supplied which causes the address to to loaded into a register within the memory. The memory then reads this address and the data is enabled onto the bus with the input timing strobe. The processor reads in the data during this period. The write cycle, which requires two bus cycles, is shown in Fig. 3.5.1(b). During the first cycle, the address is placed onto the

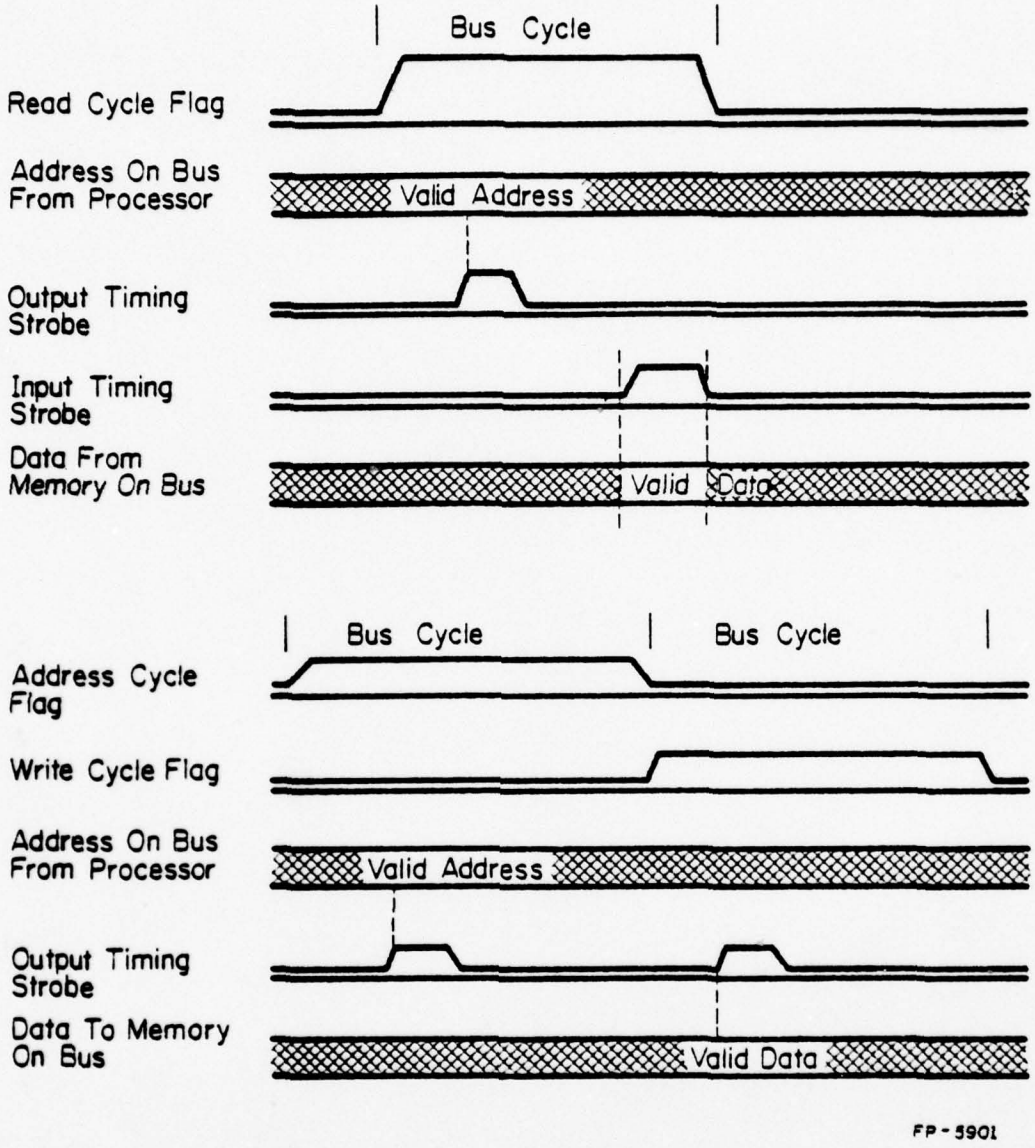


Fig. 3.5.1 Memory operations bus cycling

bus and an output strobe is supplied which causes the address to be loaded into a register within the memory. During the second cycle, the data to be written is placed on the bus and an output strobe is supplied which causes the data to be loaded into a register within the memory. The data is then written.

In the case of a two bus processor, the cycling is similar, but without bus multiplexing.

The memory system contains interface logic which utilizes control signals sent from the microprocessor. There are typically two types of control signals. One type, called flags, are set at the beginning of the cycle to indicate the type of cycle, such as Memory Read or Memory Write. The other type of signals needed are timing strobes, which indicate the actual period within a particular cycle that a latch or buffer driver connected to the bus should be enabled. Shown in Fig. 3.5.2 is a functional diagram of a memory interface, showing how signals from microprocessor are used.

There are several criteria for selecting a memory chip. One obvious criterion is cost. In terms of performance, the criterion is the memory access requirements of the microprocessor. In other words, the time between when the memory address is sent out and when the microprocessor expects to receive the data back. To avoid slowing down the microprocessor, the memory chip selected should have an access time less than or equal to the memory access time of the microprocessor. However, more important than the chip access time is the memory system access time. This includes the memory access time, plus additional delays, such as bussing propagation delays and inter-

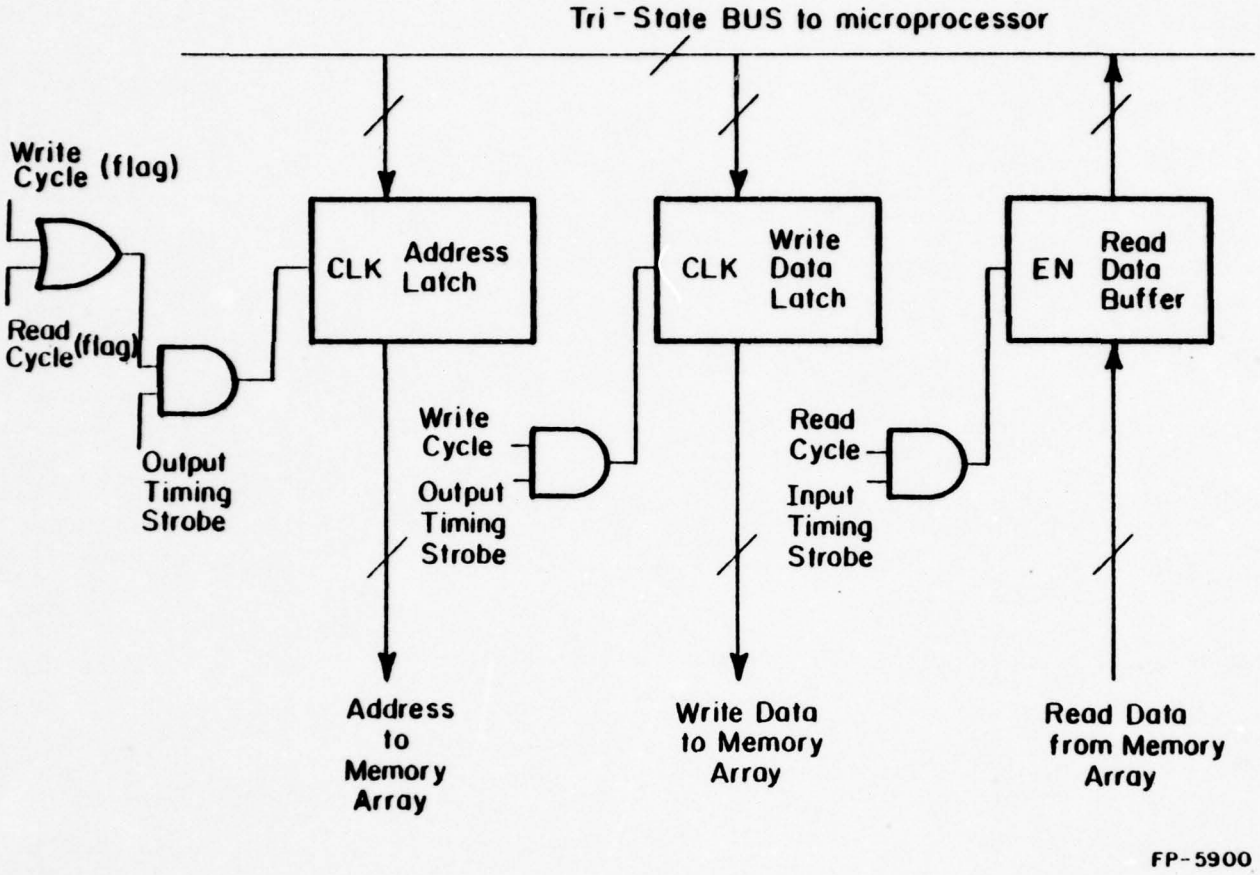


Fig. 3.5.2 Typical memory interface

face logic propagation delays. It is really the system access time that must meet the memory access requirement of the processor.

Depending on the memory system access time, the microcomputer system can be operating as either processor limited or memory limited. In the processor-limited region, a faster memory cannot improve performance, since the system is already running at maximum speed. In the memory-limited region, the run-time is sensitive to memory system access time. It is also a function of memory utilization. Shown in Fig. 3.5.3 is a chart which illustrates the relationship between memory system access time and performance, expressed as run-time.

In Fig. 3.5.3 when memory access time exceeds t_{sys} the performance of the microprocessor begins to degrade. The variable X is the memory utilization. The Run Time is given by

$$\text{RUN TIME} = (1-X) + X(\text{MAX}(1,t_{sys}))$$

Many microcomputer applications permit very slow execution while still accomplishing the specified task. In such applications, the designer may be able to lower the memory cost by using slower memory. Slower memory is defined as any memory whose access time is such that is does not meet the requirements of the microprocessor running at full speed. Many microprocessors support this type of operation, which essentially allows the microprocessor to pause in the middle of a memory operation, until the operation is complete. This is typically done with an external input to the microprocessor called the READY line. The memory system controls the READY line.

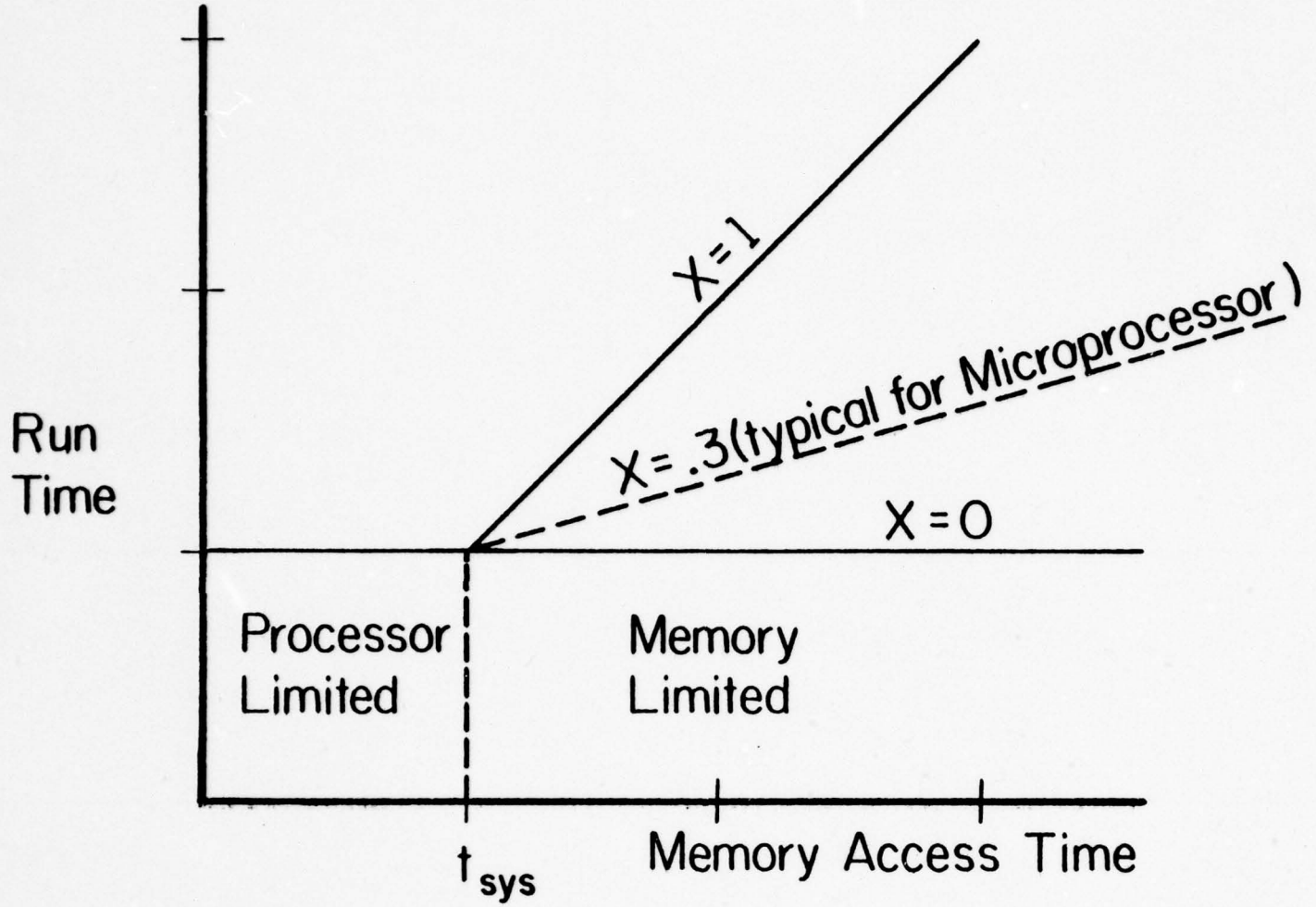


Fig. 3.5.3 Relationship between memory access time and run time

During a memory cycle, the microprocessor outputs an address (and data for writes) to the busses and then tests the READY line. The memory system pulls down the READY line. The microprocessor tests the READY line and remains paused as long as it remains low. When the memory operation is complete, the memory system raises the READY line and the microprocessor continues execution.

3.6. Logic Implemented with ROM's and PLA's

In applications where high speed is not essential, ROM's and PLA's (Programmable Logic Arrays) can be used to implement logic functions. Both ROM's and PLA's generate logic functions using AND-OR structures. Thus, both possess an array of AND gates and an array of OR gates. The AND gate array generates product terms of the input variables and the OR gate array generates output functions by ORing the appropriate product terms. The ROM and PLA differ in the level of programmability. The ROM is characterized by a programmable OR gate array, while the PLA has both programmable AND gate and OR gate arrays. This leads to the result that a ROM is simply a special case of a PLA.

ROM's can be used to implement any function directly as a sum of minterms. This is accomplished in the following manner. Shown in Fig. 3.6.1 is a 3-bit address decoder. The address decoder accepts the three address bits as inputs and generates eight outputs, where each output corresponds to selecting one of eight possible memory addresses. The address decoder can also be viewed in a different manner, that of a minterm generator. Assume each of the address inputs is a logical variable, as shown in Fig. 3.6.2. Then each of the outputs

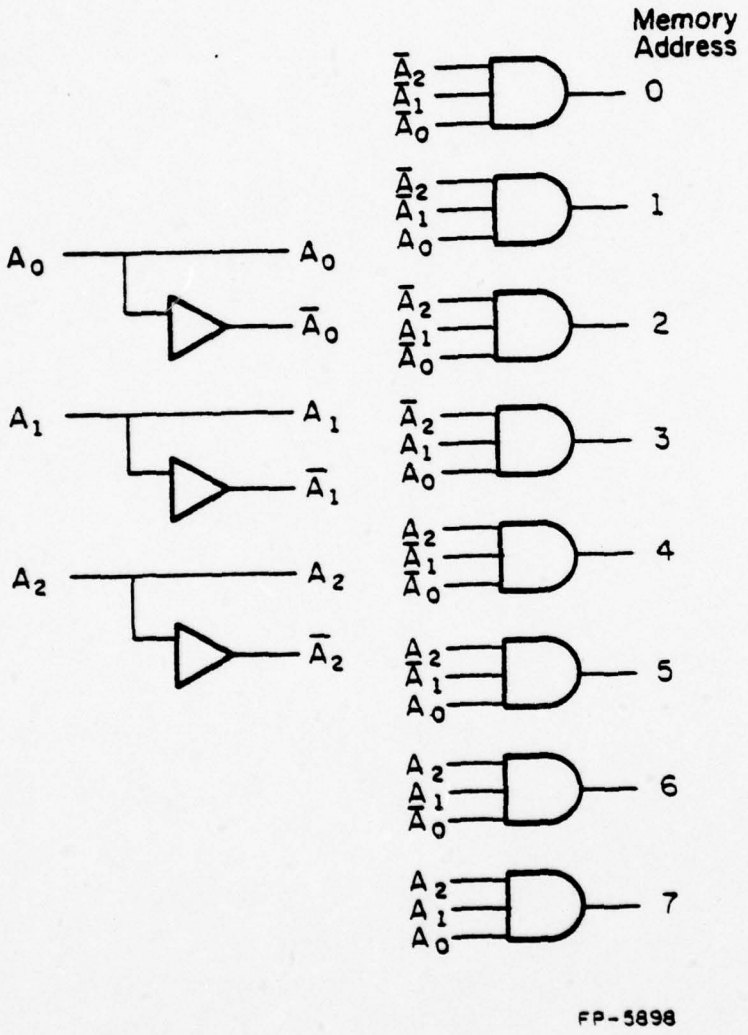


Fig. 3.6.1 Address decoder

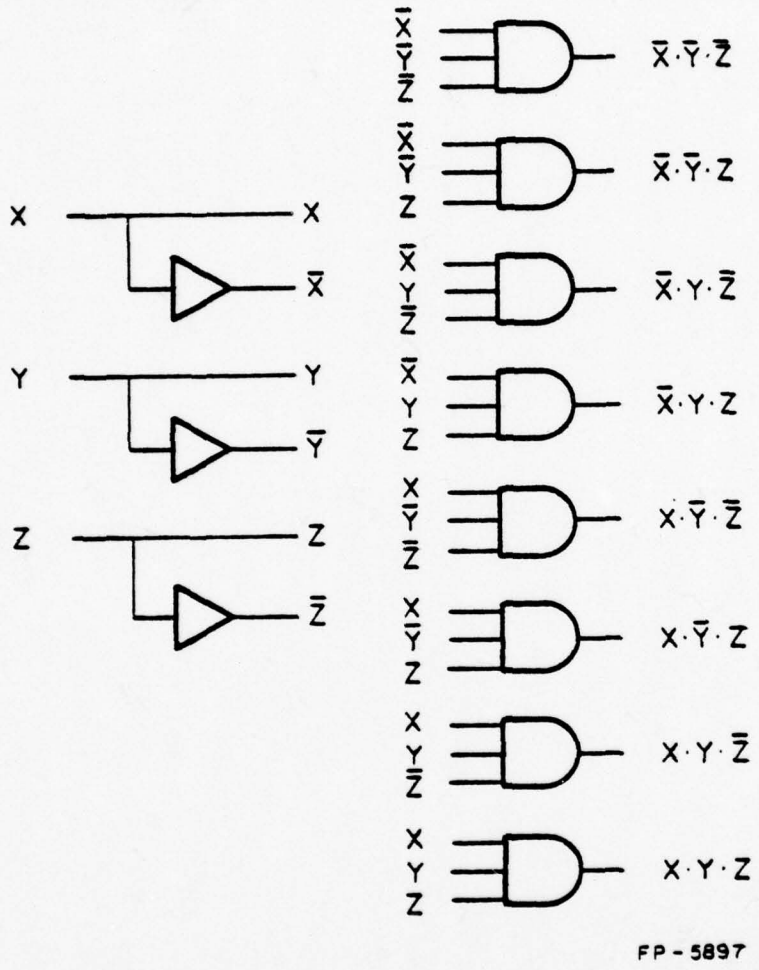


Fig. 3.6.2 Minterm generator

corresponds to one of the eight possible minterms of the input variables. In a ROM, the address decoder is complete, that is it generates all 2^N possible minterms of the N input variables.

Thus in a ROM, the address decoder is the AND gate array and the resulting product terms generated are all possible minterms of the input variables. There is thus no need to program the AND array in a ROM. The OR gate array consists of a number of output function lines, where each line serves as a distributed OR gate. Each output function is implemented by ORing the desired minterms by forming a connection between each selected minterm line and the output function line. The connections between the minterm lines and the output function lines are formed during the process of programming the ROM. Thus in a ROM, the AND gate array is fixed and the OR gate array is programmable. Shown in Fig. 3.6.3 is the symbolic internal structure of a typical ROM.

Using a ROM as logic has one disadvantage. To implement a function of N variables requires a ROM whose memory size is proportional to 2^N. To implement K functions of N variables requires a 2^N x K bit ROM. Thus the ROM size is very sensitive to the number of input variables. This fact can make the ROM very inefficient in certain applications. For example, to make a 4 stage adder (with 9 inputs and 5 outputs) requires 29 x 5 = 2560 bits of ROM. This same function can presently be made in a single dedicted MSI package without use of ROM.

The PLA can be used to implement any sum-of-products function expression. The PLA is similar to a ROM, with the major difference being the AND gate array. In the case of a ROM, and AND gate array is an

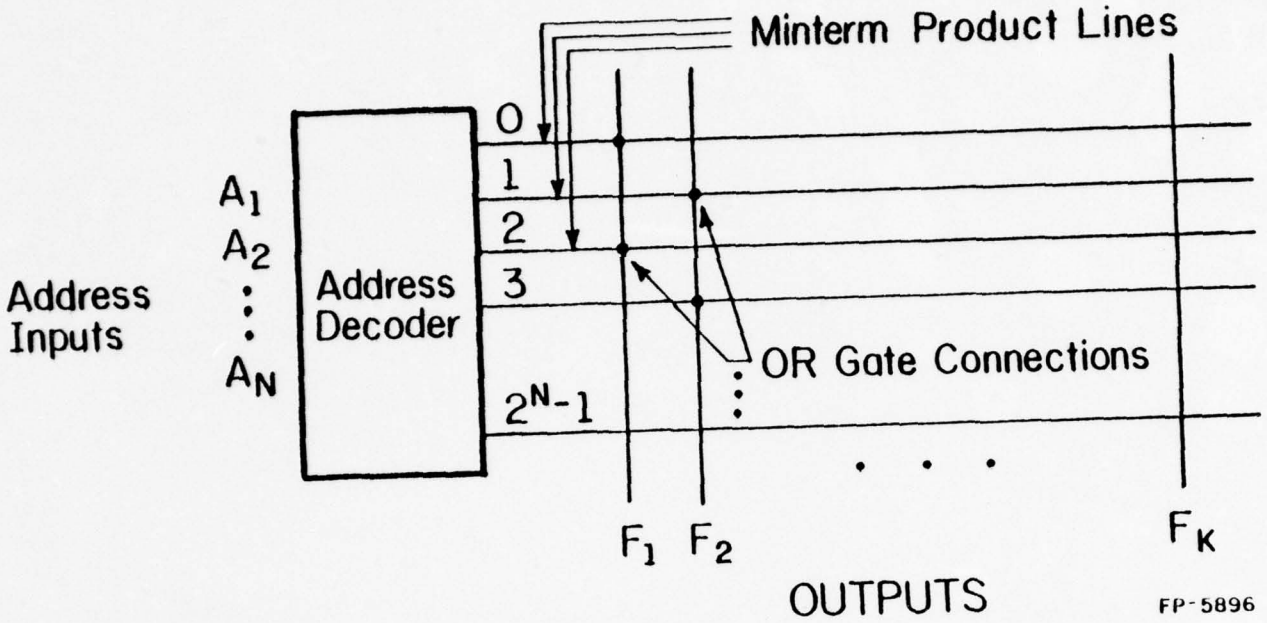


Fig. 3.6.3 ROM structure

address decoder which generates all possible minterms of the input variables. In the case of a PLA, the AND gate array is programmable. In other words, the AND gates can be programmed to recognize any product term of the input variables. There are two reasons why the PLA allows a reduction in the number of AND gates needed as compared to a ROM. One reason is that the PLA can be programmed to generate only needed product terms. Another reason is the fact that the PLA allows arbitrary product terms, thus a single product term gate in a PLA can replace several minterm product term gates in a ROM. The OR gate array in a PLA operates as in a ROM. Thus to use a PLA, the needed product terms are generated by programming the AND gate array and the needed functions are formed by programming the OR gate array. Shown in Fig. 3.6.4 is a typical PLA structure.

It can now be seen that a ROM is simply a PLA with 2^N unique product terms. The PLA has an advantage in that there is no necessary correlation between the number of inputs and the size of the PLA, except for the number of input pins and the size of each AND gate. This allows a PLA to use input variables without becoming excessively large.

The PLA is ideal for LSI implementation of logic. It has a regular structure which permits easy layout and fabrication. It also has the advantage of reducing the gate count to a minterm. This tends to minimize the chip area needed, which is always an important goal in chip fabrication.

Standard N variable PLA chips, available as MSI components, contain vastly fewer than 2^N AND gates, e.g. 48 AND gates with N = 14.

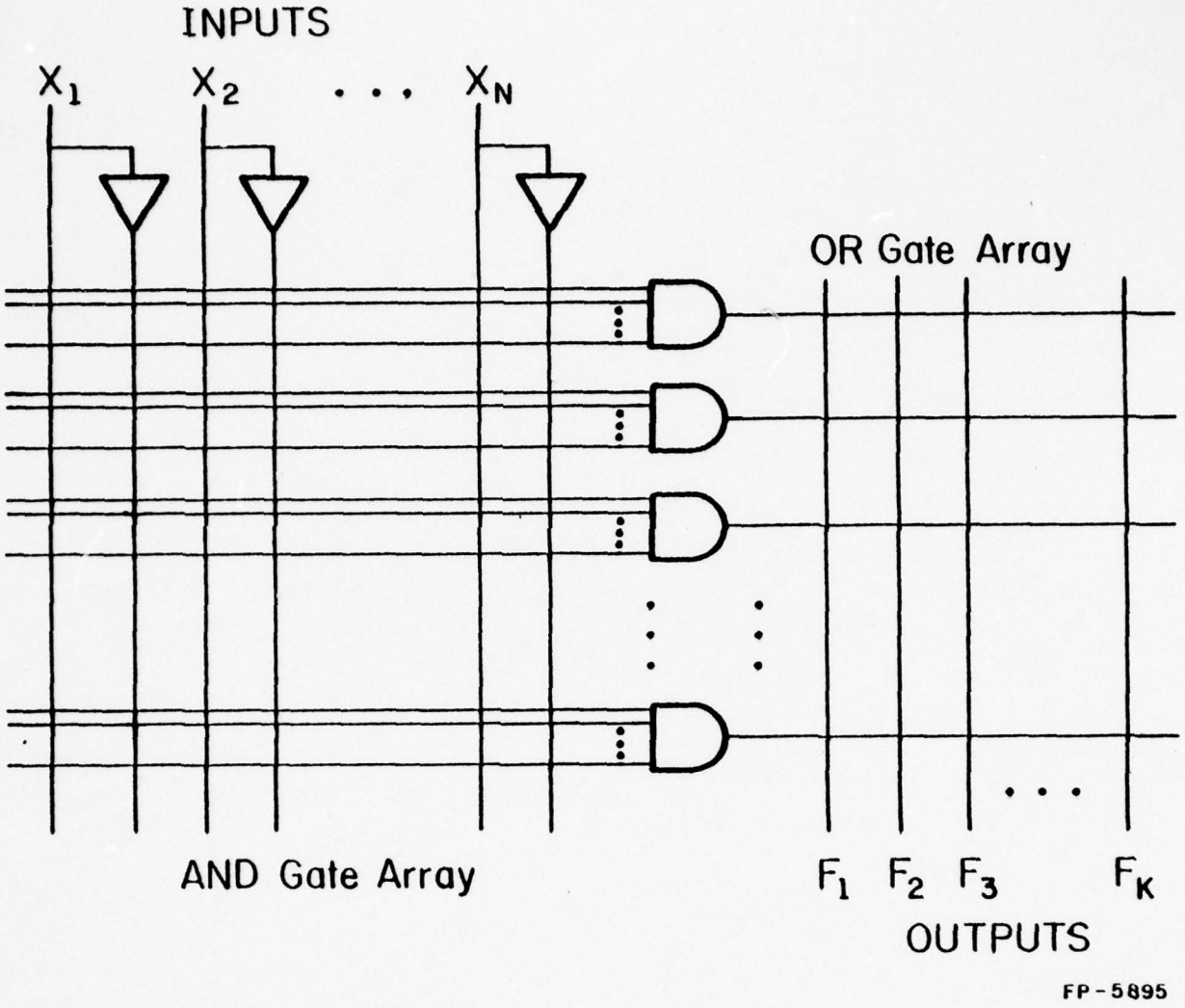


Fig. 3.6.4 PLA structure

Thus only a very few of the 2^{2^N} functions of N variables are implementable in a single chip. However, many useful functions of N variables are implementable and the larger value of N allowed in a PLA chip is often more useful than the completeness of a ROM chip with smaller N. Typically 8 or more functions can be output from a single PLA chip.

4. INPUT/OUTPUT

4.1. Microcomputer Input/Output

In most applications utilizing microcomputers, some communications between the microcomputer and the outside world is required, since in the course of performing tasks, the microcomputer must input information, process it, and output the appropriate response. Most microcomputer tasks are very I/O-oriented, controller-type applications in which the microcomputer is constantly performing I/O. This type of application is natural since the relatively slow processing speed and pin limitations of microcomputers make them unsuitable for high speed numeric processing.

The I/O section of the microcomputer contain the logic necessary for the microcomputer to communicate with I/O devices. The structure of the I/O section and the methods used to connect I/O devices to it are important. Furthermore, the low cost of microcomputers make them suitable for many new applications in which they will be interfaced to a wide variety of I/O devices. The variety of devices and applications creates the need for a wide spectrum of I/O capabilities, each of which must be implemented at minimum cost. Here the designer can make use of hardware-software tradeoffs to implement the necessary performance for the lowest cost.

One important point to realize in I/O design is the large speed differential that can exist between I/O devices and the microprocessor they are communicating with. Even though microcomputers are at the low end of the computer performance spectrum, they still

operate at speeds which are orders of magnitude faster than some I/O devices, especially those I/O devices that are designed for human interaction such as keyboards and displays. These devices operate at data rates which are very slow compared to even the slowest microcomputers, which have an instruction execution rate of approximately 10^5 instructions per second. In addition, the microcomputer can service I/O devices much faster than the I/O device can handle the data. Thus, although the microcomputer can appear to be doing several tasks simultaneously this is only an illusion created by the speed differential. The microcomputer can perform only part of one task in each cycle.

The primary characteristics of microcomputers that affects the structure of the I/O system is the use of a bus architecture. The microcomputer usually communicates with all its I/O devices through one or two main buses. Each I/O device controller contains device recognizer logic which is necessary since all I/O devices are connected to a common bus. Thus, the device which is to perform the data transfer must be specified by the microprocessor. Typically, this is accomplished by the microprocessor which in the process of executing an I/O instruction sends out the select code, specified within an I/O instruction, before the data transfer. Each device on the bus has a unique select code which it recognizes. Thus, to talk to a given device, the programmer simply specifies that device in the select code field of an I/O instruction.

The use of a bus structure for microcomputer I/O results in a system with several desireable characteristics of a low-cost system,

namely simplicity and flexibility. The simplicity results from interfacing all devices to the bus through a common bus protocol. The designer will define the timing relationships of data and control signals which are available to all devices on the bus. In effect, the user is defining the protocols for a system unibus. Thus, interfacing becomes simple because all interfaces are similar. The flexibility is illustrated by the ease with which additional devices are connected to the system. Additional devices are connected by simply connecting the data control bus to them and assigning them device numbers.

The primary method of performing I/O data transfers in a microcomputer system is by executing I/O instructions. Every time an I/O instruction is executed, the microcomputer will sequence through the steps needed to perform the I/O operation. This operation consists of a transfer of a word or byte of data between an accumulator internal to the microprocessor and a device connected to the system bus. Thus, I/O operations are initiated by the microcomputer which remains in control throughout the operation. Since the microcomputer is the controlling device, it must supply control information to the I/O devices to accomplish the data transfer. This control information must perform two functions. First, it must steer the data to the proper device on the bus, by issuing a device select code. Second, it must provide timing information to synchronize the I/O device to the microcomputer timing to perform the data transfer.

Every microcomputer has some form of I/O instruction in its instruction set. Currently, there are two methods used to implement them. The first method of implementing I/O instructions is to have two instructions, one for input and one for output, which are used exclusively for I/O transfers. Such instructions are typically called IN and OUT (or RIN and ROUT). The second method of implementing I/O instructions is to have memory-reference instructions used in conjunction with special memory locations which have been assigned to I/O devices. In this method, the I/O device must be capable of recognizing and responding to its assigned memory address. This is called memory mapped I/O. Its advantage is the ability to manipulate I/O data with memory-reference inatructions and to provide great flexibility in the amount of memory space and the number of I/O devices used in the system. Normal memory functions are inhibited when an I/O location is addressed.

Typically, the only explicit I/O instructions in microprocessors are the ones that perform the actual data transfers. These have the form shown in Fig. 4.1.1.

The op code specifies either an IN or OUT type instruction. Each instruction contains a field, called a control field. The contents of the control field are output as control information to the I/O devices during the execution of an I/O instruction. The meaning of the control field is user defineable. During the execution of an I/O instruction, each I/O device receives and decodes the control field and performs the specified function. Thus, the programmer

(a) 8 Bit Machines (b) 16 Bit Machines

FP-5894

Fig. 4.1.1 I/O instruction formats

controls I/O devices by specifying control field contents when writing a program. One use of the control field is to specify a select code, which identifies which device is to perform the I/O transfer. Another use is to send out I/O commands, such as start device, input status or perform data transfer.

The function of the select code is clearly needed, since all I/O devices share a common bus. Thus, a way is needed to specify the correct device. While executing an I/O instruction, the select code is sent out for all I/O devices to examine. All devices compare this select code with their own and only the one that matches will respond. The remaining bits of the control field contain control information for the selected device in some format agreed to by the programmer and the device controller designer. In some cases, the entire function of the instruction may be contained in the command control field and no useful data word transfer occurs.

Suppose a microcomputer wants to read in data from a paper tape reader. The microcomputer must tell the reader to read and input a character. This is done by executing a routine called a paper tape reader driver. The driver contains the I/O instructions that interact with the reader. A separate "start device" I/O command is usually given to engage the reader motor and begin moving the tape. No actual data is transferred with this command although a transfer cycle must be executed in order to send the command information since an IN or OUT instruction must be used. Next, the microprocessor will try to read in the data. However, the reader is a slow device and executing an IN immediately will probably read in garbage. Thus, the micro-

computer must have come way of interrogating the status of the reader controller to see if it has a word of data. This is done with an "input status" I/O command. Thus, the microcomputer executes a loop testing the status of the reader repeatedly until it has a word of data. The data is then obtained with an IN instruction. This method of programmed I/O is called busy-wait I/O since the microprocessor executes a loop waiting for the I/O device to be ready. Thus, the user defines bit patterns in the control field of the I/O instructions to command the devices to perform necessary functions and the controller hardware must interpret them accordingly. Successive words may be obtained by reentering the busy-wait loop and executing IN instructions until the desired number of words has been read, whereupon a "stop device" command is sent to the reader.

Alternatively, the controller can be designed to get the next word from tape automatically each time a word is read by the microprocessor. In this case, the controller is less complicated and start device and stop device commands are not needed for the reader. The following example illustrates the hardware and software of a typical programmed I/O device. This particular example is for a paper tape reader. The microcomputer has a 16 bit word and the I/O control fields are defined as shown in Fig. 4.1.2. The select code of the reader is assumed to be 8. H signifies hexadecimal.

The beginning of a program to read paper tape is shown in Fig. 4.1.3.

Instruction	Command	Function	Control Field Value
ROUT	STOPRDR	This command stops the reader	08H
ROUT	STARTRDR	This command starts the reader	18H
RIN	STATUSRDR	The status of the reader is input in bit Ø of the data word. Bit Ø = Ø indicates the reader has a word ready to transfer	28H
RIN	DATARDR	A data word is read in from the reader	38H

Fig. 4.1.2 Reader Commands

```
          STOPRDR = 08H          Define control field for stop command
          STARTRDR = 18H         Define control field for start command
          STATUSRDR = 28H        Define control field to input status
          DATARDR = 38H          Define control field to input data
*****************************************************************************
READCHAR: ROUT  STARTRDR         Start paper tape reader
          RIN   STATUSRDR        Check status of reader
          BOC   C3 , *-1         If Bit Ø = 1 branchback to check again
          RIN   DATARDR          Read in data word
          .
          .
          .
          .
```

Fig. 4.1.3 Beginning of reader driver

The I/O operations are done using this driver program along with compatible hardware. The hardware must be capable of interpreting the commands contained in the I/O instructions. A typical hardware interface is shown in Fig. 4.1.4.

The IMP-16 is a microprocessor with two 16-bit busses, one for address and one for data. The ADEN, RDP and WRP are control signals which indicate bus usage during a machine cycle. ADEN indicates the presence of an 8-bit I/O control field on the address bus. RDP indicates that the data bus will be used to perform an input transfer. WRP indicates that the data bus will be used to perform an output transfer. T4 and T78 are timing signals used during data transfers. T4 is used as a data strobe during output transfers. T78 is used as a data enable during input transfers.

Note that under busy-wait I/O with a slow device, the processor spends almost all of its time in the busy wait loop waiting for the device to be ready. In interrupt driven I/O this time can be used for other execution and the processor is interrupted when the device is ready.

4.2. Interrupts

Interrupts are signals originating in a device controller and sent to the processor. The processor can perform normal processing while waiting for the device to become ready. When an interrupt occurs, normal processing is suspended, the program which services the interrupt is executed and then the program that was interrupted resumes. An interrupt can thus be viewed as a "hardware forced"

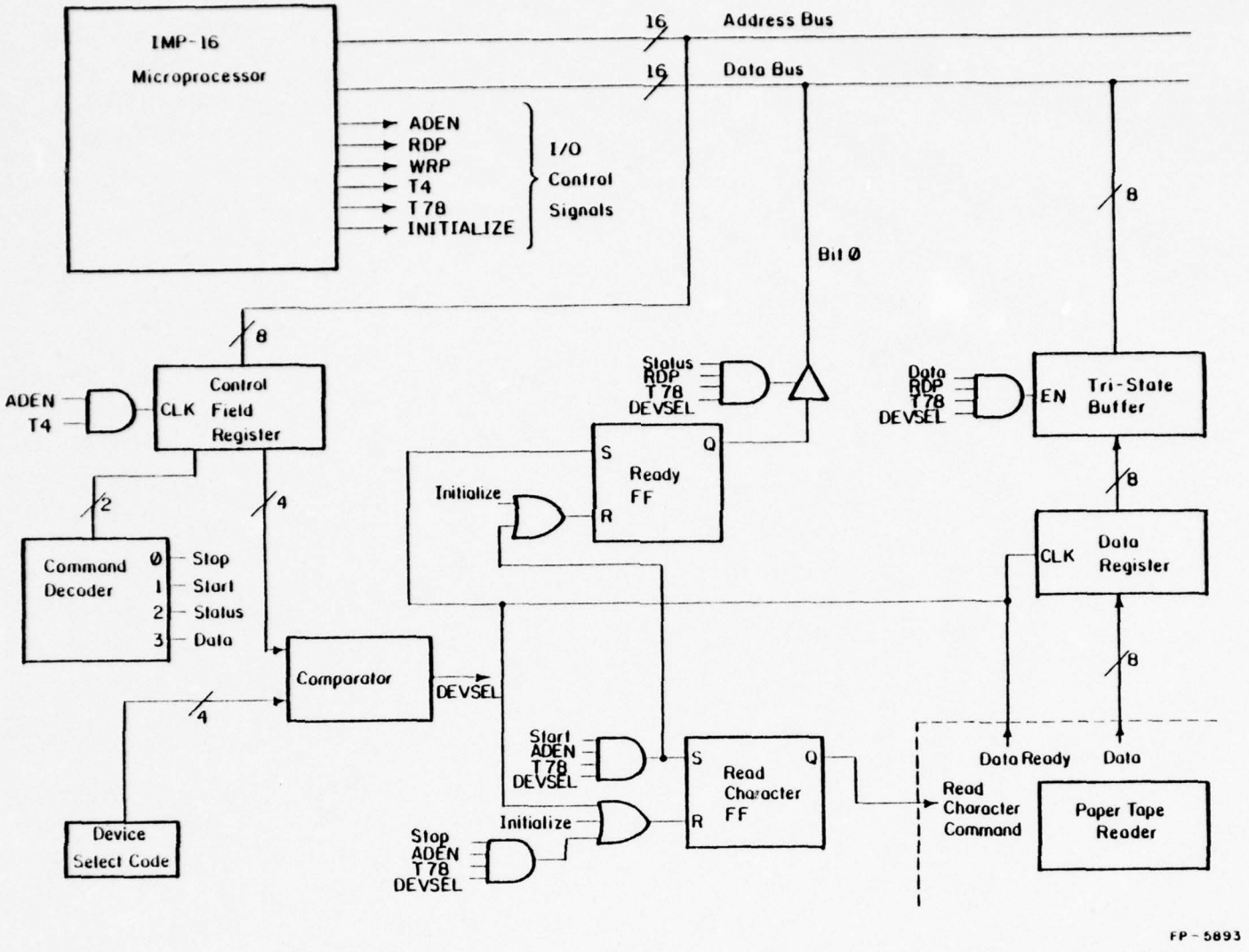


Fig. 4.1.4 Reader I/O interface

jump to subroutine. Interrupts allow external devices to demand service from the microcomputer.

Interrupts are essential in real time applications where data must be processed within a specified time or lost. Furthermore, they are useful to perform efficient I/O operations, since they allow the microcomputer to continue processing between the times that relatively slow I/O devices need servicing. The I/O devices are serviced only when they request it by transmitting an I/O interrupt.

An idealized example illustrates the efficiency gained with interrupt driven I/O. The microcomputer is sending data out to a printer at 10 char/sec. Assume it takes 100 μsec. of processor time to execute a routine which sends a single character to the printer. Thus, only 0.1% of the microcomputer time is spent servicing the printer and 99.9% of the time is available for other processing. Busy-wait I/O can do no other processing while servicing the printer.

Interrupts can also be used to notify the microcomputer of external conditions such as power failure, parity errors and real-time clock events as well as internal conditions arising from the processor itself, such as arithmetic overflow. If, for example, no overflow interrupt capability exists, the program must check the validity of each arithmetic operation which might overflow.

Most microcomputers have some type of interrupt capability. Those that do have an input through which interrupts are requested, typically called Interrupt Request (INTREQ). They also usually have an internal interrupt enable flip-flop which determines when the interrupt

system is ON. This input is tested periodically by microcomputer hardware, usually during each instruction fetch operation. If the INTREQ line is low, no device is requesting service and normal processing continues. If the INTREQ line is high and the interrupt system is enabled, some device is requesting service and the microcomputer hardware begins the interrupt handling.

Shown in Fig. 4.2.1 is a flowchart of a typical interrupt processing sequence. The functions performed during an interrupt processing sequence are common to all microcomputers. However, the methods differ and there are variations in the amount of interrupt processing done in hardware.

Decision block A of Fig. 4.2.1 is the test of the INTREQ line. This simply decides if there is an interrupt pending. If so, the interrupt processing begins. Since most microcomputers can process only one interrupt at a time, usually at this point the interrupt system is disabled automatically, thus ignoring all other interrupt requests. In Block B, the processor status is saved. This must be done so that processing can resume after servicing the interrupt. The Program Counter is saved at some fixed location. In Block C, the Program Counter is loaded with a fixed address and execution is started. At this address the Interrupt Handler routine is located. Block D is a program which determines the source of the interrupt if there can be more than one. This is done by polling devices until the interrupting device is found. Block E sends out an Interrupt Acknowledge signal to the interrupting device being serviced. This will usually cause the device to turn off its interrupt request. This must be done or else the microcomputer could see the same interrupt request twice. In

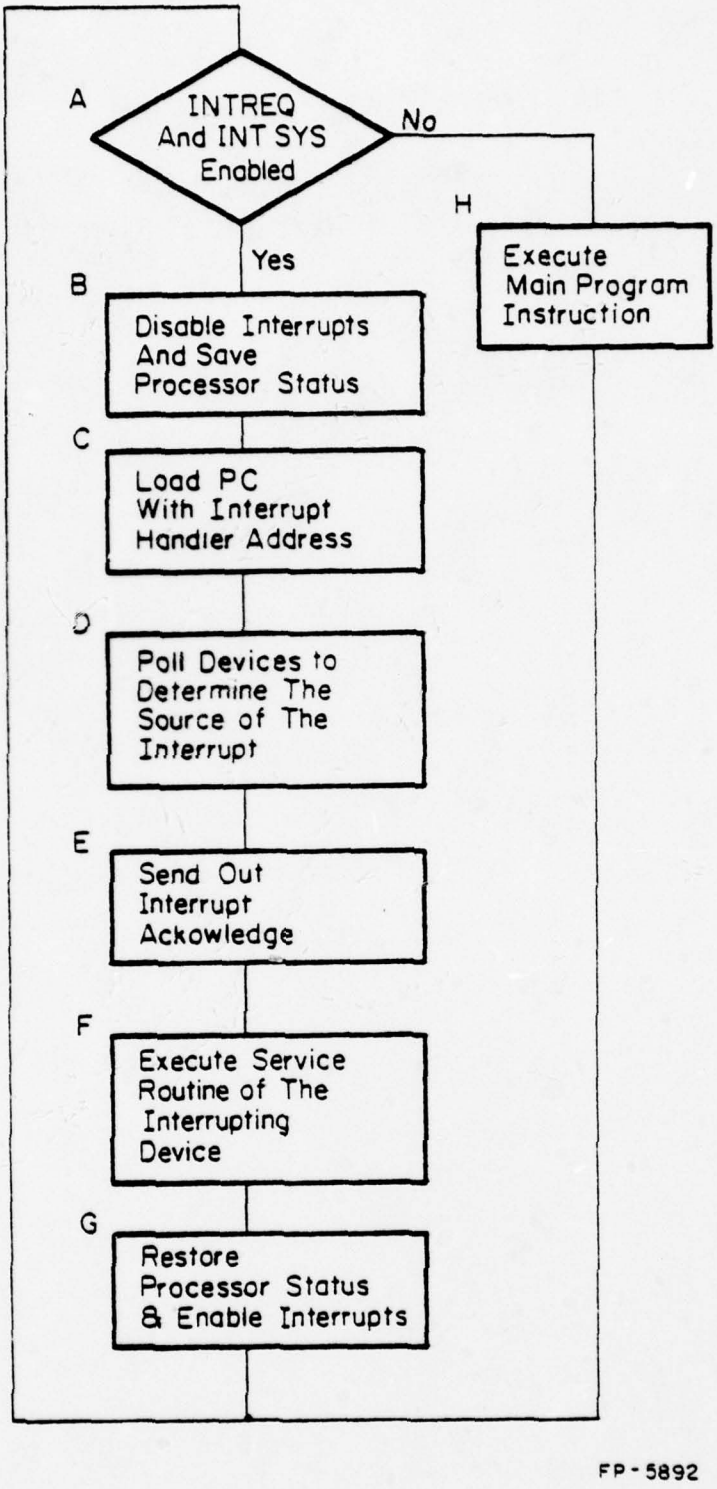


Fig. 4.2.1 Interrupt processing

Block F the service routine of the interrupting device is executed. Block G restores the status of the microcomputer and returns to normal processing. This means the Program Counter is restored from the location where it was saved and the Interrupt system is reenabled. This function is usually done with a Return-from-interrupt instruction.

In a system with only one interrupting device, there is no need to poll devices to determine the source of the interrupt since there can be only one. However, when there are multiple interrupting devices, some scheme must be used to determine the source of the interrupt along with a method of assigning priorities to the interrupting devices. Shown in Fig. 4.2.2 is the diagram for the single-line interrupt method, where all device requests are input to an OR gate.

The OR function is usually implemented on a single wire with open-collector gates as shown in Fig. 4.2.3. Alternatively, inverting Tri-state circuits can be used to behave like open-collector circuits by making their inputs equal to a logical 1 and using DEVREQ as an enable.

The interrupting device is determined by polling all the devices in the interrupt handler routine. The priority of the devices is determined by the order of the polling, highest priority device polled first, next-highest priority device polled second, etc. The potential problem with software polling is that it can become very time-consuming.

The time used for device polling can be eliminated with a technique called "vectored interrupt". A vectored interrupt allows the microcomputer to begin executing the service routine of the interrupting device immediately. This is done by reading in a word, called an

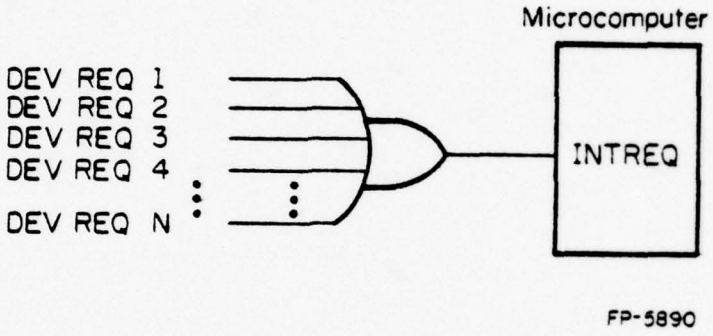


Fig. 4.2.2 Single line interrupt connection

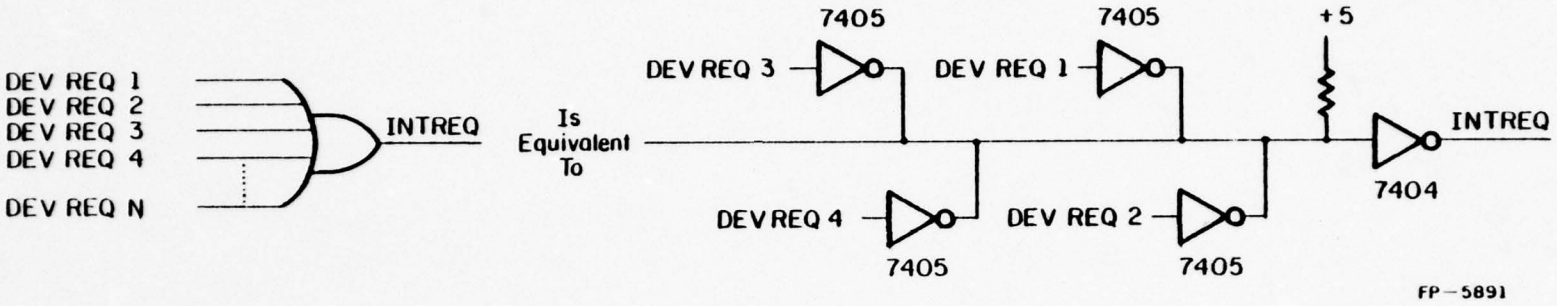


Fig. 4.2.3 OR gate implemented with open collector gates

interrupt vector, which points to the address of the correct interrupt service routine. The interrupting devices uses the IAK signal from the microcomputer as an enable to place its interrupt vector on the bus. During this time it is read in by the microcomputer. Each device typically has a unique vector.

During the time the vector is being read in, only one device can have its vector on the bus. Thus, vectored interrupts require all the interrupting devices to resolve priorities externally so that only the highest priority device respond to the IAK. This can be done with a technique called daisy-chained priority. A single line is serially connected through all devices, as shown in Fig. 4.2.4. This line carries priority. The priority of any given device is determined by the position on the line. A device must have the highest priority among the devices presently needing service to request an interrupt. Once any device requests an interrupt the priority chain is broken and no lower priority devices can request interrupts. Shown in Fig. 4.2.5 is the circuitry for a vectored/daisy-chained priority interrupt card.

The interrupt circuitry shown in Fig. 4.2.5 would be present on every I/O card within a system. The priority chain is daisy-chained to all devices. Initially, both the DEVICE INT REQ FF and INT REQ FF are clear. There is one clock, I/O CLK, generated by the microprocessor.

Whenever an I/O device wants to interrupt, it pulses the DEVICE INT REQ set input to asynchronously set the DEVICE INT REQ FF to a logical 1. The next rising edge of I/O CLK sets the INT REQ FF. This will cause the INT REQ line into the microprocessor to become a logical 1. It also clocks the priority chain and clears PROUT to a

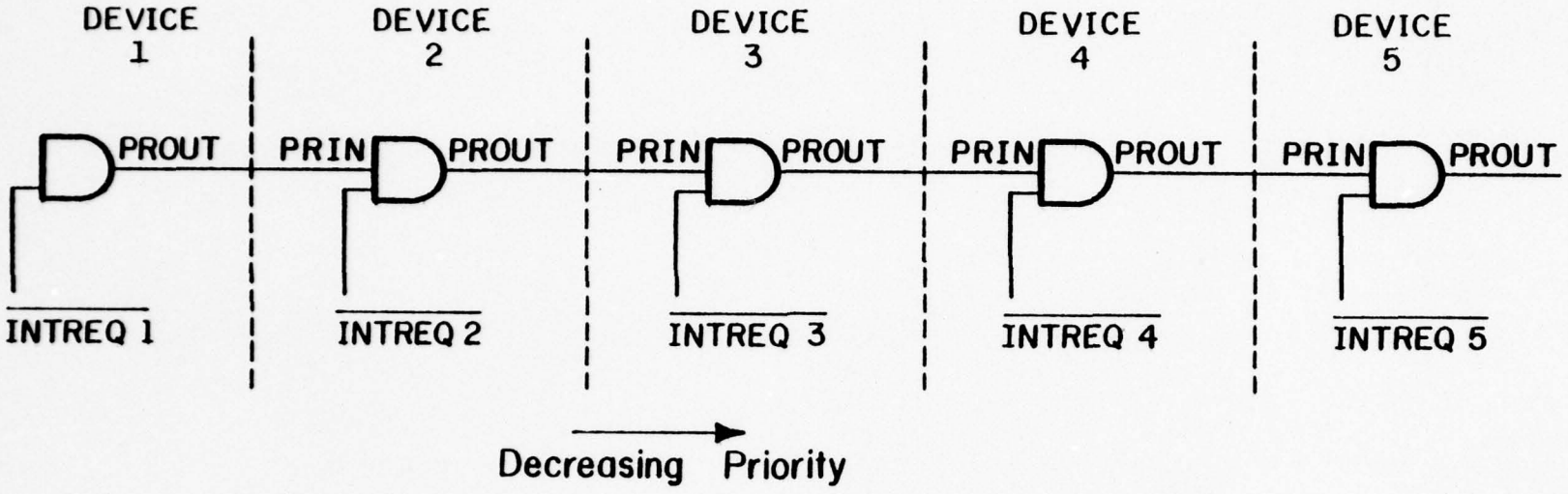


Fig. 4.2.4 Daisy-chained priority

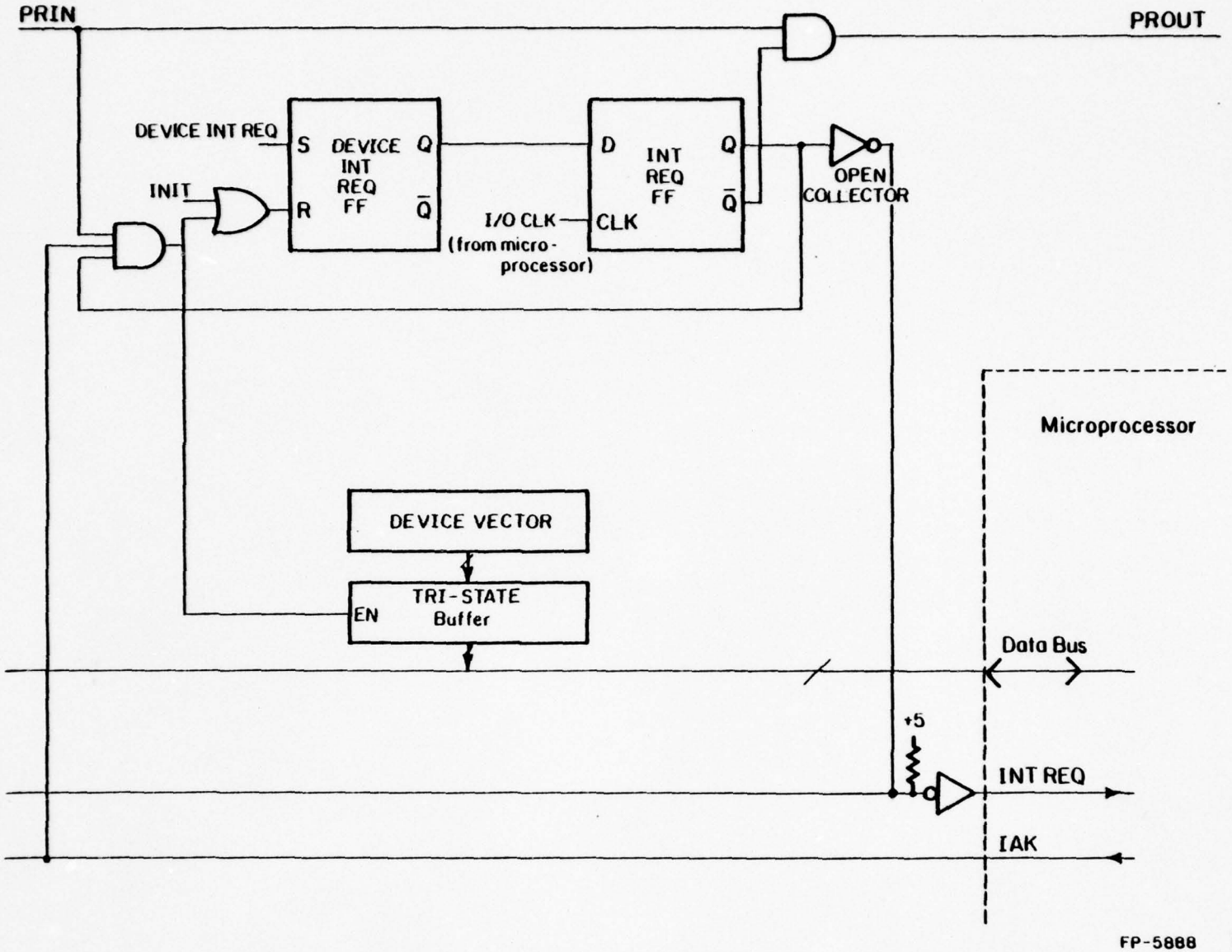


Fig. 4.2.5 Vectored/Daisy-chained interrupt circuit

logical 0. Once the microprocessor recognizes the interrupt it generates the IAK signal. Assuming this card has priority (i.e., PRIN = I), this causes two things to happen. First, the DEVICE INT REQ FF on the card is reset. Then, on the next I/O CLK the card removes its INT REQ FF. Second, the card drives its DEVICE VECTOR onto the Data Bus. The operation of the circuit is illustrated by the waveforms of Fig. 4.2.6.

The key requirement of this circuit is that both the priority chain and INT REQ FF's be stable during IAK. The I/O CLK is used to synchronize the INT REQ FF's with the microprocessor. Note that there is a settling time for the priority chain (following the rising edge of each I/O CLK) which must be allowed before IAK.

The interrupt vector for a device is usually the address of its service routine or an arbitrary instruction (e.g. jump to subroutine at that address) or an address into a table of routine addresses. In the latter scheme, the table is stored in fixed locations of memory, but the routines themselves may be moved by changing the contents of the table. On some systems there are both vectored and nonvectored interrupts and a mixture of devices of each type as well as those which do not interrupt at all can be present.

The ability of a microcomputer to respond to interrupts is measured by the interrupt latency. Interrupt latency is defined as the maximum time that can occur between a device interrupt request and the beginning of the interrupt routine that services that device. Interrupt latency is the sum of several components. The first is the longest execution time of any instruction of the processor, since this is the worst case time for sensing an interrupt request. The next component is the time to sense the interrupt, swap the PC and save

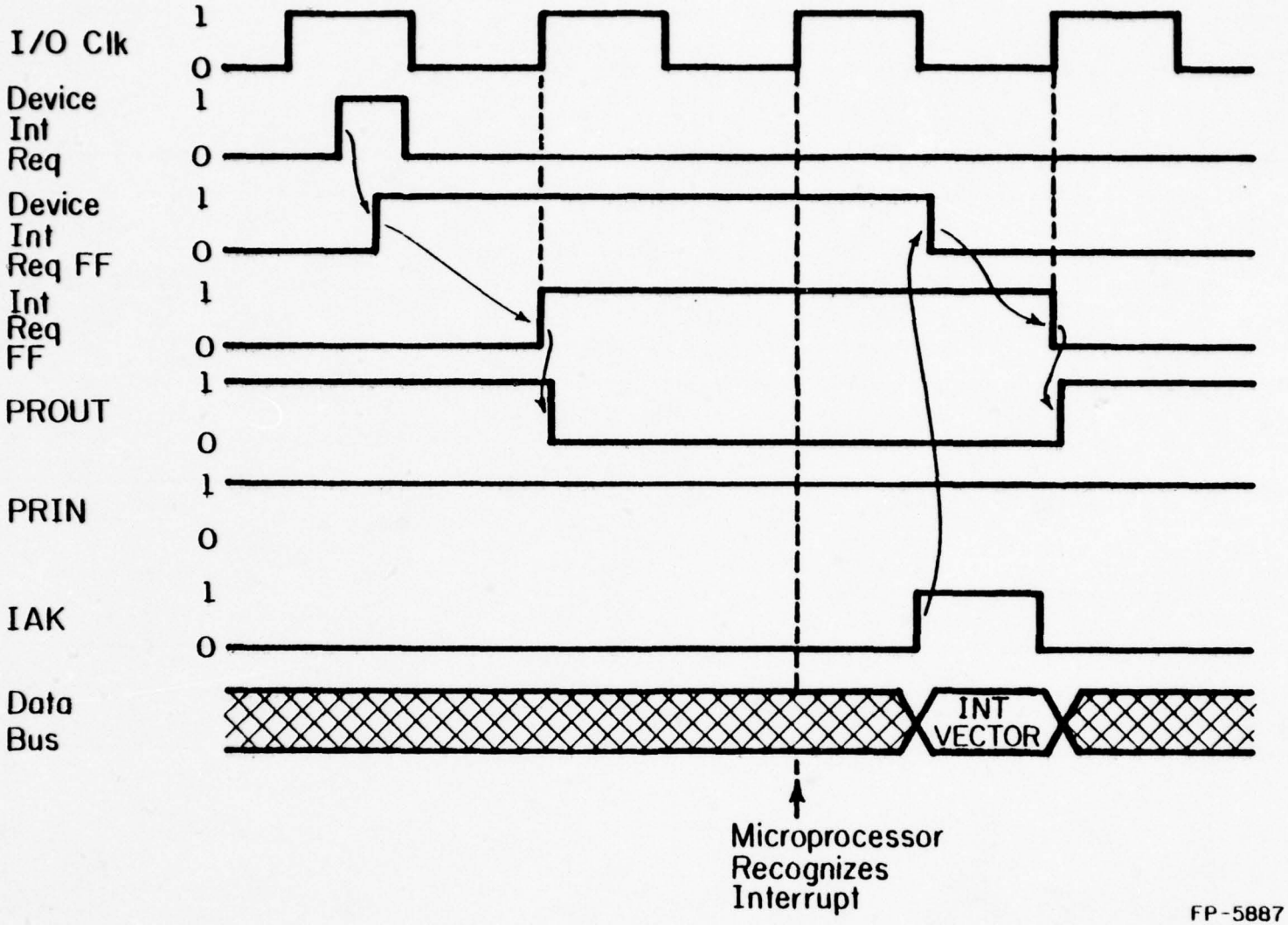


Fig. 4.2.6 Waveforms of interrupt circuit

status. The last component is the time needed to determine the source of the interrupt and service it. Thus, each microprocessor has characteristics, including execution speed, instruction set and architecture, which determine its interrupt latency. Certain architectural features enhance (minimize) the interrupt latency. Vectored interrupts minimize the time needed to determine the source of an interrupt. Data stacks minimize the time needed to save and restore status. In many cases, the designer needs to know the interrupt latency since I/O devices may be designed to operate with some specified latency.

Following are descriptions of how four current microprocessors process interrupts. Basic similarities can be seen.

The National IMP-16 senses interrupts at the INTRA input. There is an INTERRUPT ENABLE FLAG which turns the interrupt system on and off. This flag is program controlled and hardware controlled also. If there is an interrupt pending at the end of the current instruction, and the INTERRUPT ENABLE FLAG is sent, the processor resets the INTERRUPT ENABLE FLAG, saves the current PC on the stack and sets the PC to 1. It then starts executing the interrupt handler which must begin at location 1. The IMP-16 also has another interrupt, CPINT, named under the assumption that it originates from a control panel. CPINT is a vectored interrupt and has an associated acknowledge, CPINP (control panel input). The interrupting device drives a 16 bit instruction onto the data bus when CPINP occurs. This single instruction is executed. Then main program processing resumes, except that a jump to subroutine instruction causes the entire

subroutine to be executed. The return from that subroutine, a return from interrupt instruction (RTI), resumes main program processing and enables the interrupt system.

The Intel 8080 senses interrupts at the INT input. There is an INTE signal which indicates the status of the interrupt system and is program alterable. If there is an interrupt pending at the end of the current instruction and INTE is a logical 1, the processor starts a fetch cycle and sends out the INTA status bits which serves as an interrupt acknowledge. Also, the PC is not incremented, so the current PC is still present. When the processor asks for an instruction, the interrupting device drives an instruction on the data bus. Note that the 8080 did not save the PC. The user must save and modify the PC with an instruction as a JSB type. The 8080 has a one byte JSB instruction, RST, which causes the PC to be saved on the stack and jump to a location encoded within the instruction.

The Intersil IM6100 senses interrupts at the $\overline{\text{INTREQ}}$ input. There is an INT EN FF which turns the interrupt system on and off and is program alterable. If there is an interrupt pending at the end of the current instruction and the INT EN FF is set, the processor grants the device interrupt. The current PC is written at location Ø, and the PC is set to 1. It then starts executing the interrupt handler routine at location 1.

The Signetics 2650 senses interrupts at the $\overline{\text{INTREQ}}$ input. There is an INT EN flag in the program status work (PSW) which indicates the state of the interrupt system and is program alterable. If there is an interrupt pending at the end of the current instruction and the INT EN

flag is set, the processor first disables the INT EN flag in the PSW and then executes a jump to subroutine instruction. It also sends out an INT ACK signal which causes the interrupting device to put an 8 bit address on the data bus. This address becomes the target of the jump-to-subroutine instruction. Since the 8 bit address can be indirect, this allows the interrupt handler routine to begin at any addressable memory location.

4.3. Serial I/O

In addition to programmed I/O which transfers words or bytes in parallel there is another I/O method used with microcomputers called serial I/O. As the name implies, this method is for serial or bit-stream devices. Of course serial I/O can be implemented by a controller which transfers words to and from the processor and bits to and from the device. To implement serial I/O directly the microcomputer needs an input, called sense, whose logic state can be tested under program control. There must also be an output, called flag, which can be set and reset under program control. With the sense input and flag output the microcomputer can read and write bit streams, thus forming a serial I/O port. Shown in Fig. 4.3.1 is a typical serial I/O connection.

Serial I/O transfers are totally program controlled. Typically, serial I/O operates with a predetermined data format and data rate which are implemented within the program. In effect, the microcomputer is serving as its own interface. The main function of the program is serial to parallel conversion on the data at the sense input parallel to serial conversion on the data at the flag output, and accurate timing.

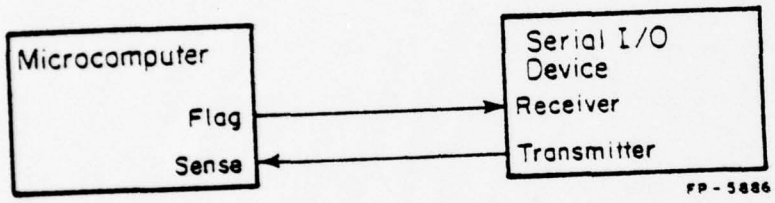


Fig. 4.3.1 Serial I/O

Serial I/O is the ultimate programmable interface, since it consists of a program only. This allows for flexibility in data rate, data format and error checking. For the same reason, serial I/O requires microcomputer control for the entire duration of the transfer.

Since serial I/O operates with a direct link between the microcomputer and the I/O device, there is no hardware interface needed. Thus there are no hardware costs associated with serial I/O. This makes it especially useful with microcomputers where low-cost is essential.

Almost every application of microcomputers requires the use of a terminal or teletype. The teletype needs a serial interface to communicate with the microcomputer. This interface can be built into a hardware controller at some cost or serial I/O can be used to implement a flexible, low-cost teletype interface.

Teletype communication involves sending ASCII characters as bit streams. Data bit 1, which is transmitted first, is the least significant ASCII code bit. The character is framed with one start (low) and two stop (high) bits. The data format is shown in Fig. 4.3.2.

The start of a character is indicated by a high-to-low transition. Each character requires 11 bits including one start and two stop bits. Typically the data rate is 10 char/sec. or 110 baud (bits/sec). The resulting bit time is 9.09 msec. Higher baud rate terminals (up to 9600 baud) are available which use the same protocol except that only 1 stop bit is used.

Data is represented within the teletype by current. This is known as the Teletype 20 ma. current loop convention. A TTL logical

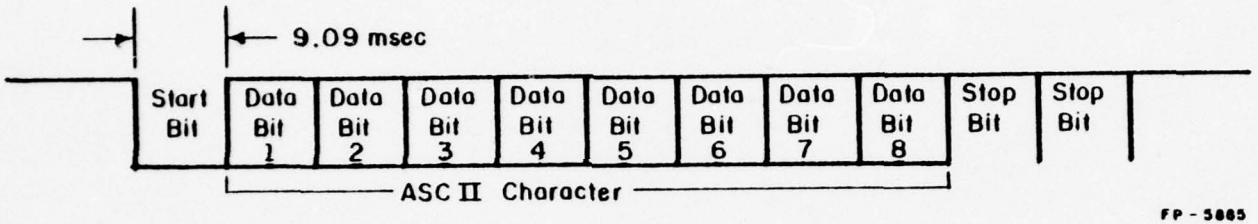


Fig. 4.3.2 TTY character format

one corresponds to a current of 20 ma. A TTL logical 0 corresponds to a current of 0 ma. Therefore, a level converter is needed. A level converter circuit is given in Fig. 1.9.8.

Thus, the serial interface for the teletype consists of the level converter hardware and the serial I/O program, as shown in Fig. 4.3.3. Many terminals are available which use the EIA convention rather than the 20 ma. current loop for interfacing. These accept any voltage $\geq$ 3 volts as logical 0 and any $\leq$ -3 volts as logical 1. A TTL compatible or "forgiving EIA" will accept and drive TTL voltage levels directly with a negative logic convention, i.e. logical 1 is LOW and logical 0 is HIGH. The program must be capable of reading and writing data in the teletype format. The flowchart for routines which input and output characters is shown in Fig. 4.3.4. The programs are generalized for any microcomputer.

The output routine assumes that the 8 bit ASCII character to be transmitted is right justified in a register, called REG, when the routine is entered. The input routine assumes that the 8 bit ASCII character to be received is to be left justified in REG with the other bits of REG, if any, shifted to the right by 8 bits. Timed delays in the program are implemented by a loop of an appropriate number of NOP instructions.

Note that the serial I/O programs tie up the processor for the entire time that characters are sent or received. If a controller is used which has a word interface to the processor (and a serial interface to the Teletype), busy-wait I/O can be programmed which allows

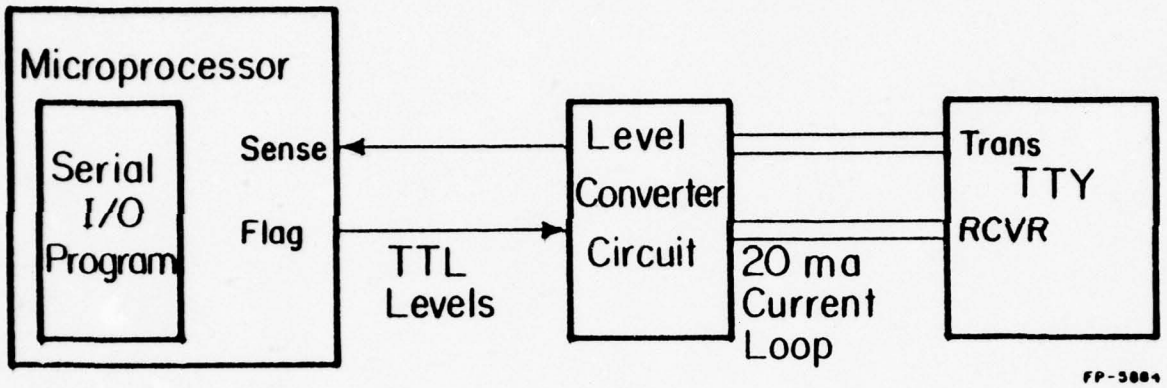


Fig. 4.3.3 TTY connection to microprocessor

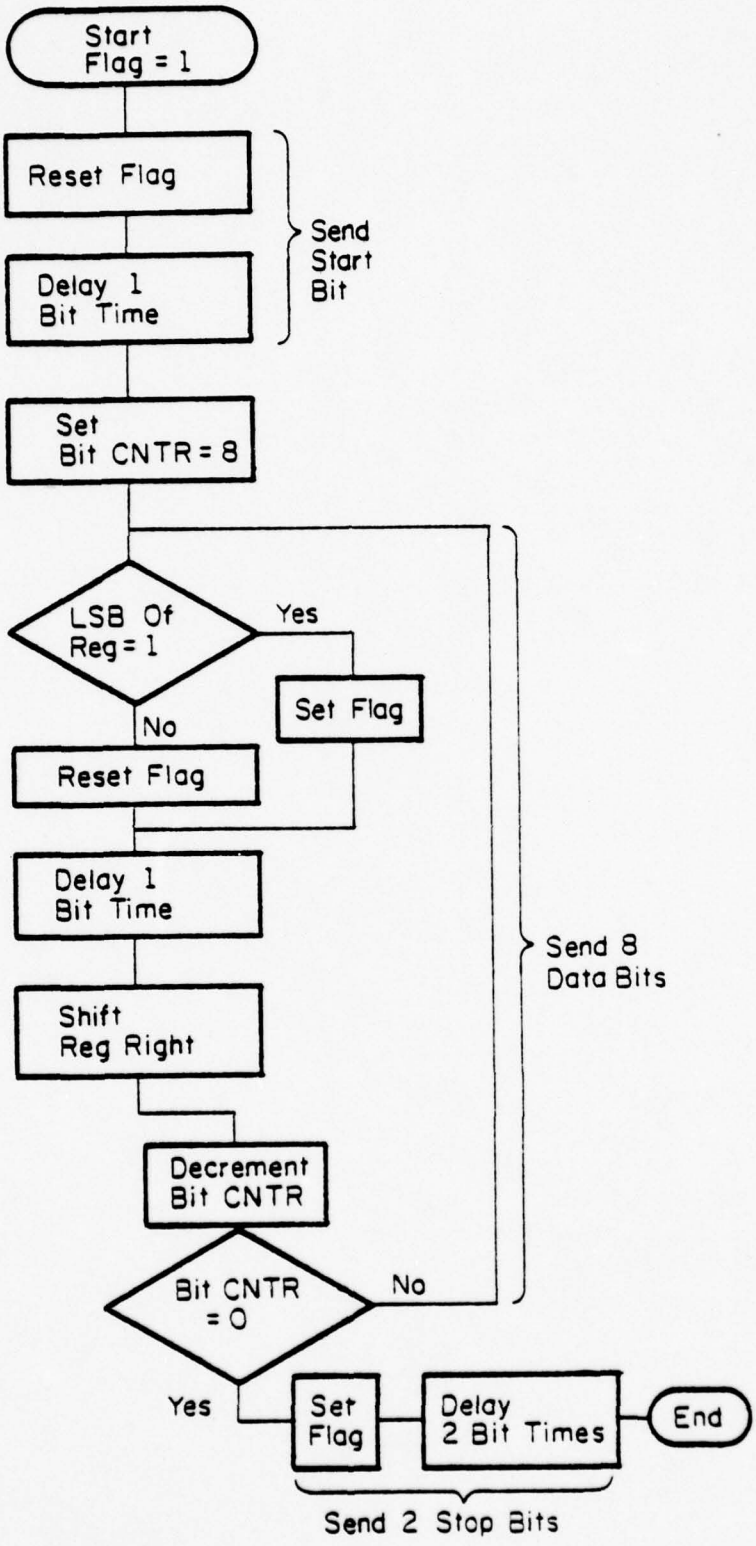


Fig. 4.3.4 TTY data transfer routines

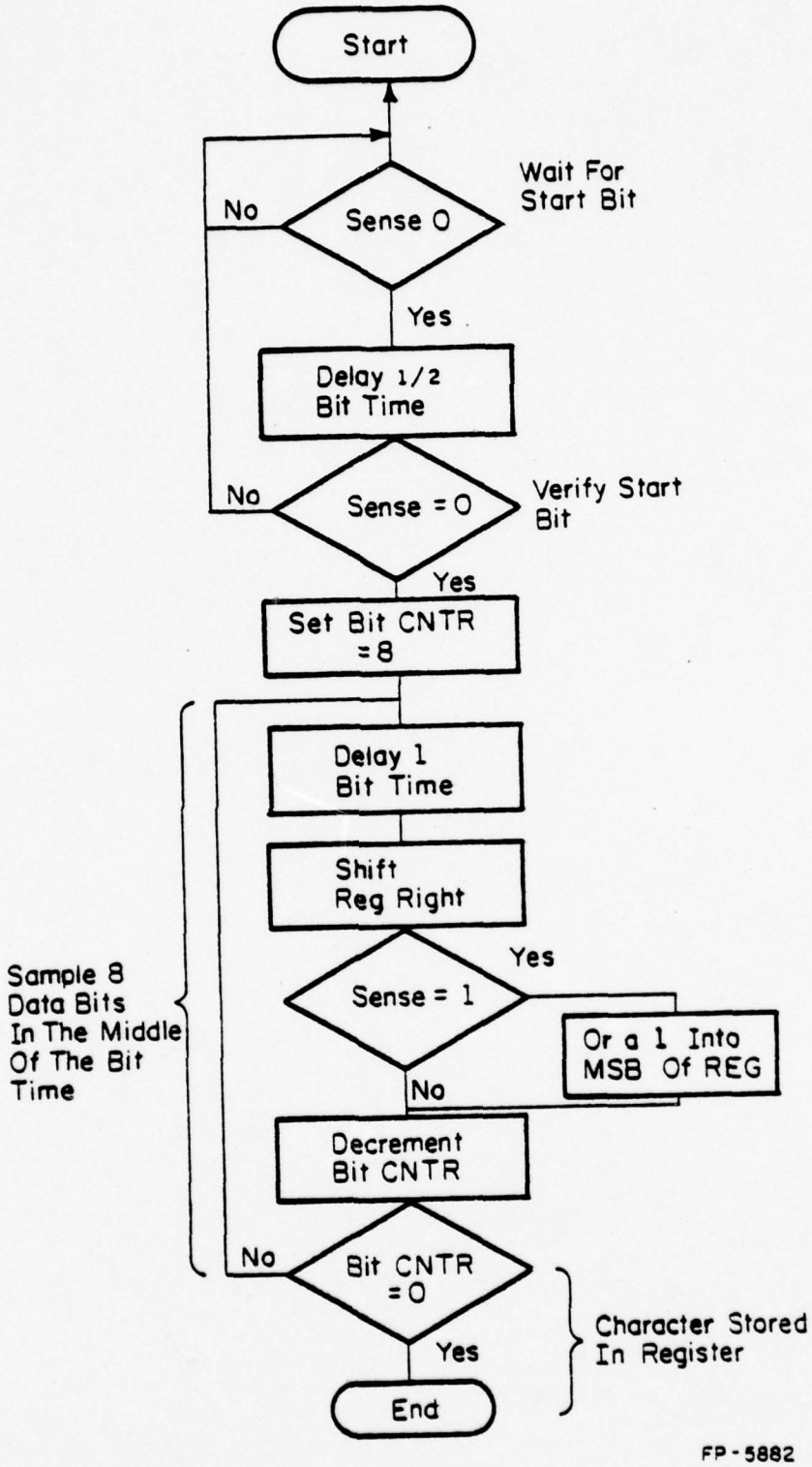


Fig. 4.3.4 continued

simpler software for input-output routines and allows some time between characters for other processing, provided only that the busy-wait loop is reentered in time to receive input characters at the maximum rate. Reentering the busy-wait loop for output routines is not so critical since the stop bit level may be held for any amount of time greater than or equal to 2 bit times (1 bit time for faster terminals).

Either controller can be made interrupt driven to free up the processor further. For the serial I/O controller, a received start bit edge can interrupt the processor to enter the input routine. This is particularly useful for input messages arriving at much slower than the maximum rate, e.g. human keyboard entry. Note that if interrupt latency is high for this system, then sampling of subsequent data bits is after the midpoint of the bits. If the latency is excessive, transmission errors can result. For the parallel I/O controller, interrupt capability can be applied to both input and output to free up the processor to a greater extent. Each successive step in increased controller capability involves added controller hardware cost, less system software, and higher system performance (if the processor has other processing which can be done while it waits for I/O). This is the so-called "hardware-software tradeoff" in design.

4.4. Direct Memory Access

Direct memory access (DMA) is a high speed I/O method in which data is transferred directly between the I/O device and memory, completely bypassing the microprocessor. The data rate of DMA is limited only by the cycle time of the memory rather than the execution

rate of the microprocessor (as in programmed I/O). DMA is therefore the fastest means of I/O for a given memory system.

The use of DMA requires extra logic which initiates and controls DMA cycles. This logic, called a DMA controller, interacts with elements of the system (microprocessor, I/O device, and memory) to perform DMA transfers. The DMA controller performs three basic functions; 1) Accept and execute commands from the microprocessor. 2) Request and be granted the use of system resources such as busses and memory. 3) Generate the control signals to the I/O device and memory to perform the transfer. The DMA controller temporarily takes control of the system for each of its memory access cycles. DMA controllers allow the processor maximum time for other processing while I/O operations are in progress.

When DMA is enabled it is operating with a specified I/O device. Thus, the DMA controller is constantly monitoring the DMA Service Request line from the I/O device. A DMA cycle is initiated whenever a DMA Service Request is issued. Thus, although DMA is continuously enabled, the DMA transfer rate is determined by the I/O device.

Most applications of the DMA involve block transfers of data in which a specified number of words are transferred between the I/O device and memory. Block transfers require several parameters to be set up in the DMA controller, namely the word count which specifies the number of words to be transferred and the memory address which specifies the starting address for DMA data. During each DMA cycle the word count is decremented and the memory address is incremented.

The DMA operation is complete when the word count equals zero. A DMA operation consists of a series of DMA cycles. On data word is transferred during each DMA cycle. Of course the block can be transferred to memory in reverse order, if required, by decrementing memory addresses. Sometimes a capability is built in for a processor to abort a DMA block transfer in progress, either under program or device control.

DMA as an I/O method thus has several desirable characteristics, namely high transfer rates and minimal interference to concurrent processing. The trade-off for obtaining these advantages is the extra hardware costs of the DMA controller.

The high data rates are due to the nature of DMA which allows transfers into memory every memory cycle and therefore transfer at the data rate of the memory. Processing interference arises when both DMA and the microprocessor request a memory cycle at the same time. Typically, DMA will be granted a memory cycle and the microprocessor will pause for one cycle. The DMA controller will initiate a cycle (by requesting a memory cycle) only when the I/O device using DMA has data ready to transfer. Thus, processing interference is minimal since the microprocessor is paused only one memory cycle per DMA cycle and a DMA cycle is initiated only when the I/O device requests it. Note that even in interrupt driven I/O, typically a 100 μsec. routine may be required per word transferred. Assuming a 1 μsec. memory cycle, the processor load per word transferred is reduced to about 1% of the load for interrupt driven I/O.

DMA is essential for servicing high-speed I/O devices where other I/O methods are insufficient. This has been the traditional application of DMA. DMA is also useful for servicing low to medium speed devices because it performs I/O transfers with a minimum amount of interference to the microprocessor. Thus, DMA is an efficient I/O method for all devices. Historically, the high cost associated with DMA restricted its use to high-speed devices. However, in microcomputer systems, the simple and economical implementation of DMA has made DMA a realistic alternative to be considered for any device.

Microprocessors can be designed to facilitate the use of DMA. This is done with a Pause input which when asserted by the DMA controller can cause the microprocessor to enter a pause state. In the pause state, processing is suspended and no memory cycle requests are generated by the microprocessor. This prohibits the microprocessor from interfering with a DMA cycle. Also, while in the pause state the microprocessor will let the memory and data busses "float" (high-impedance three-state output), thus making them available to the DMA controller.

If both the microprocessor and one or more DMA controllers are requesting memory cycles, there must be some method of priority resolution. The simplest method is to allow a DMA controller to pause the microprocessor when DMA wants a memory cycle. This makes the microprocessor the lowest priority memory requester. Priority resolution between multiple DMA controllers can be costly and difficult to design correctly. One simple solution, if possible, is to activate only one DMA controller at a time.

DMA can even be used for processors with no pause facility. For some processors, the clocks can be shut off for one or more cycles, thereby achieving a forced pause. If the processor does not use memory intensively, the processor can be given high priority and a DMA request can wait for the next unused memory cycle (many microprocessors have at least one idle memory cycle per instruction). Of course, some mechanism must be used to substitute DMA information for processor information on the memory busses during DMA cycles and suitable control signalling must be designed.

One characteristic of DMA is the DMA latency. This is the maximum amount of time between a DMA service request from an I/O device and the completion of the DMA transfer. Note that for minimal I/O buffering,

$$t_{latency} \leq \frac{1}{\text{I/O data rate}}$$

For the DMA memory priority method of pausing the microprocessor, the latency is the execution time of the longest instruction, which could be relatively long. Other implementations of DMA, as suggested above can provide shorter latencies if needed.

Although DMA data transfers operate independent of the microprocessor, the microprocessor manages DMA operations through commands in the form of I/O instructions. The microprocessor treats the DMA controller as an I/O device and utilizes I/O instructions to initialize DMA registers such as word count, memory address and direction of transfer. After loading these registers the microprocessor sends a Start DMA command via an I/O instruction. Following this command, the DMA

contoller operates independently until command completion. When a DMA command is completed, the controller notifies the microprocessor of a completion, typically through an interrupt, so the processor can take appropriate action. Listed in Fig. 4.4.1 are the control signals typically present in a DMA controller.

Shown in Fig. 4.4.2 is example of a DMA controller interconnected with all the system elements. A flowchart detailing a DMA input operation using such a system is shown in Fig. 4.4.3.

4.5. I/O System Design

The I/O-oriented nature of most microcomputer applications make I/O a major factor in microcomputer system design. I/O is the least structured section of any microcomputer and the most flexible since it must accommodate a wide variety of devices. Thus, the definition and design of the I/O section is a large part of microcomputer system design.

The design of the I/O section begins with a complete definition of all I/O requirements within the system. The I/O section can then be defined utilizing the particular mix of I/O methods which accomplish the system requirements in the most economical way. The design should also consider the possibility of future expansion.

The I/O requirements of any system can be defined by examining the task to be done and the I/O devices to be used. The I/O requirements can be characterized by the data rates and service latency of each I/O device. Service latency is defined as the maximum time from a device requesting service to the time that service is complete. An additional

Signal Name	Function
LOAD WC (Load Word Count)	Command generated by an I/O instruction which loads Word Count Register from the Data bus.
LOAD ADDR (Load address)	Command generated by an I/O instruction which loads Address Register from the Data bus.
LOAD IN/OUT	Command generated by an I/O instruction which loads the transfer direction flip-flop to indicate input or output (and possibly other control information).
START DMA	Command generated by an I/O instruction which starts DMA monitoring the specified I/O device DMA Service Request.
STOP DMA	Command generated by an I/O instruction which terminates any DMA operation in progress.
DMA ADDR EN (DMA Address Enable)	Signal generated by DMA control logic which enables the DMA address register onto the Address Bus during a DMA cycle.
INC ADDR (Increment Address Register)	Signal generated by DMA control logic which increments the address register to point to the next DMA memory location.
DEC WC (Decrement Word Count)	Signal generated by DMA control logic which decrements the Word Count Register.
DMA COMPLETE	Signal generated when DMA Word Count Register equals zero used to notify I/O device and/or microprocessor that DMA is complete.

Fig. 4.4.1 DMA control signals

Fig. 4.4.1 (continued)

I/O DATA IN	Signal generated by DMA control logic which causes I/O device to place data word on the Data Bus during a DMA cycle.
I/O DATA OUT	Signal generated by DMA control logic which causes I/O device to load a data word from the Data Bus during a DMA cycle.
START DEVICE	Signal generated by DMA control logic which tells the I/O device to start at the end of a DMA cycle when the Word Count is not equal to zero.
DMA SER REQ (DMA Service Request)	Signal generated by the I/O device to notify the DMA controller that it is ready for a DMA cycle.
MEM READ	Signal generated by DMA control logic which tells the memory to perform one memory read cycle.
MEM WRITE	Signal generated by DMA control logic which tells the memory to perform one memory write cycle.
MEM DATA IN	Signal generated by DMA control logic which causes the data word on the Data Bus to be latched into the memory data register.
MEM DATA OUT	Signal generated by DMA control logic which causes the memory to place the data word in the memory data register onto the Data Bus.
PAUSE REQ	Signal generated by DMA control logic which causes the microprocessor to pause and relinquish control of the memory and busses.
PAUSE ACK	Signal generated by the microprocessor in response to a PAUSE REQ which tells the DMA controller that the microprocessor is paused.

Fig. 4.4.1 (continued)

END TRANSFER	Signal generated by DMA control logic during a DMA cycle which acts as an acknowledge to clear DMA SER REQ.

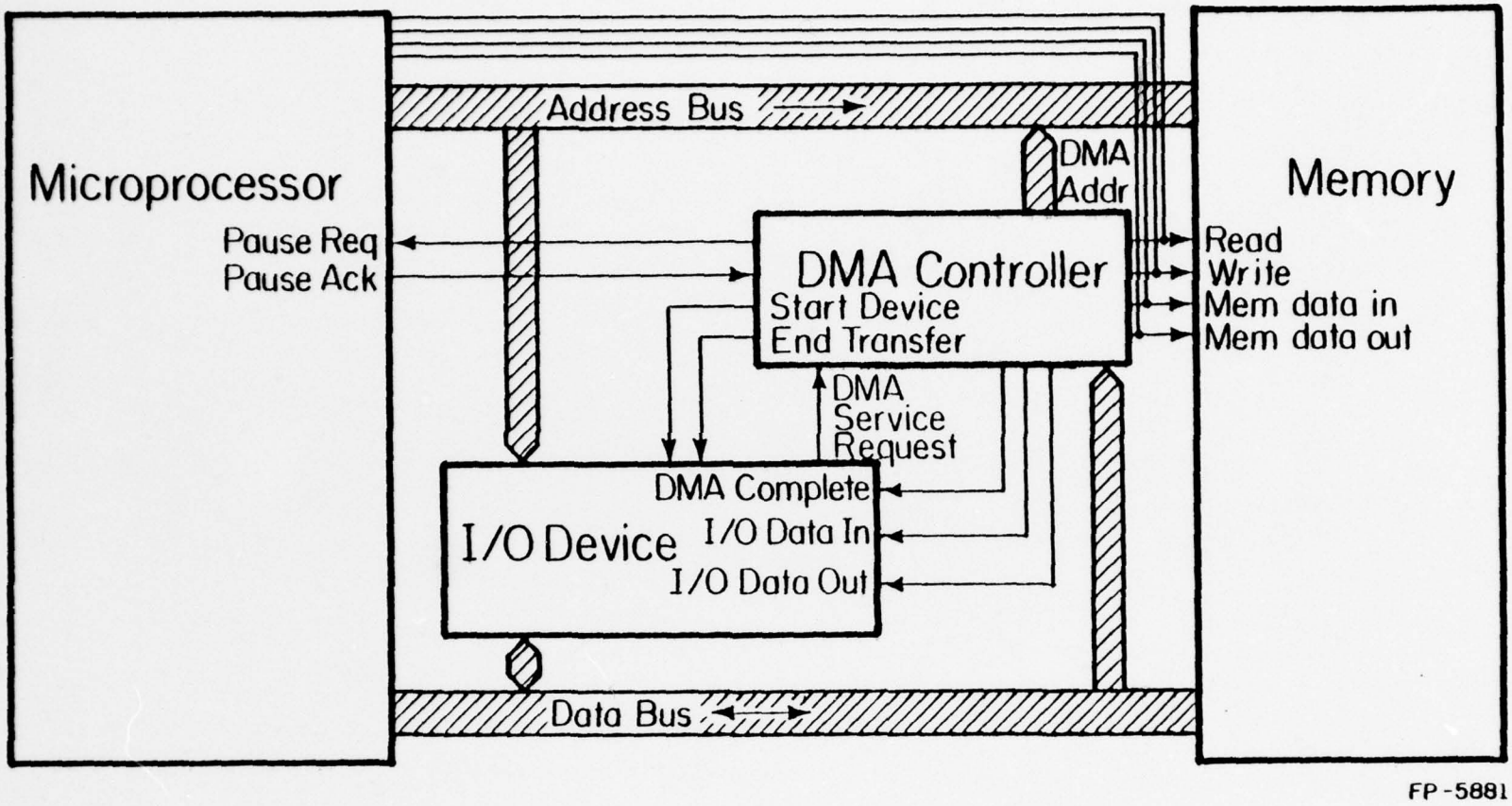


Fig. 4.4.2 DMA controller in a system

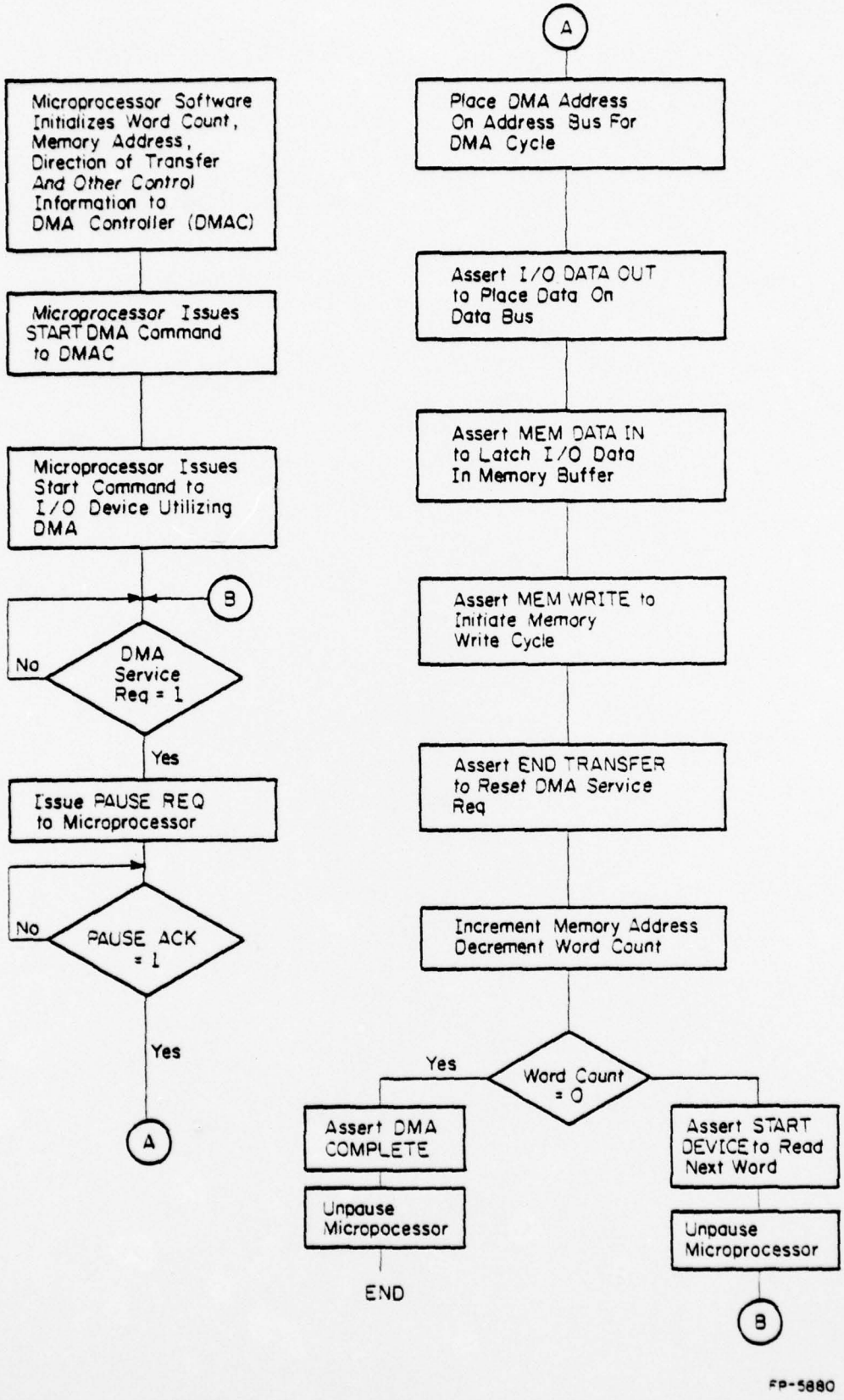


Fig. 4.4.3 DMA input operation

design consideration is the amount of concurrent processing that must be done during I/O operations.

There are two types of I/O devices. The first type is the word-mode devices which perform single word transfers. For each START DEVICE command, one word is transferred and another transfer does not begin until the next START DEVICE command. These devices have no minimum required data rate and allow arbitrarily large service latency.

The second type are the block mode devices which transfer words in blocks at some minimum data rate. This type requires that block mode devices demand service from the system or data will be lost. The service latency of block mode devices specifies the amount of time the system has to respond to requests before data is lost. These device controllers typically have an overrun error indication as part of their status word. An overrun error indicates that data has been lost due to excessive processor latency.

Programmed I/O is the most simple and economical I/O method and should be used whenever possible. In simple systems, with a small number of low-speed I/O devices, programmed I/O can usually meet system requirements. Programmed I/O does not allow the I/O device to demand service and therefore the I/O program must poll the I/O devices to see if service is needed. Normally, a single high-speed I/O device can be serviced using programmed I/O if the program can be dedicated to the device for the duration of the block transfer.

Interrupt driven I/O can be used to provide a maximum service latency equal to the interrupt latency. In addition, the use of

interrupts allows the microprocessor to do concurrent processing while servicing relatively slow I/O devices.

The use of DMA provides both the highest maximum data rate and the shortest service latency. In addition, DMA allows the maximum amount of concurrent processing for any given data rate. Shown in Table 4.5.1 is a description of each of the I/O methods.

Thus, by knowing the system I/O requirements, the designer can supply the I/O methods which accomplish the task best. However, the advantages gained in using I/O methods such as interrupts and DMA are accompanied by a considerable increase in system complexity and often cost. Thus, they should only be used when required.

The complexities of interrupts and DMA are inherent in the hardware and software needed to implement them. In addition, there are interactions between the methods which create more complexity and timing problems. For example, the interrupt latency is difficult to estimate in a DMA environment. Also, if more than one DMA or interrupting device in present, the specified service latency is only valid for the highest priority device. Any timing loops that exist in software, such as in programmed I/O, are not valid if interrupted or paused through I/O interrupts or DMA. When all these interactions are occurring, the system design can become very complex. Restrictions regarding the number of kinds of active I/O devices are normally made in most microprocessor systems to insure proper I/O operation.

When utilizing programmed I/O, in most applications some minimum data rate is required. When selecting a microprocessor for a particular

Type	Typical Data Rates	Comments
Serial I/O	Low data rates	Minimal external hardware needed. Microprocessor serves as I/O controller by executing a program which receives data by sensing a data input and transmits data by controlling a data output.
Busy-Wait I/O	20K-40K words/sec	Data transfer done through the microprocessor by program execution. Makes use of I/O instructions to transfer words between internal registers and I/O device.
Interrupt-driven I/O	20K-40K words/sec	Same as busy-wait I/O except service latency-interrupt latency. Allows some concurrent processing while I/O is in progress.
DMA	Up to 2M words/sec	Data transfers done directly between I/O device and memory bypassing the microprocessor. Requires DMA controller hardware.

Fig. 4.5.1 I/O methods

application, often microprocessors are compared by using benchmarks. Benchmarks are sample programs which perform the same function on each distinct microprocessor under consideration. To illustrate how well a given microprocessor will move data, a benchmark which transfers a block of words from memory to an I/O device can be written. Shown in Fig. 4.5.2 are programmed I/O benchmark programs for a National IMP-16 and Intel 8080.

4.6. I/O Developments

Future applications of microcomputers will demand more I/O performance for minimal cost. This will be provided by the use of general purpose LSI I/O chips. The chips are general purpose in that their function and mode of operation are programmable by the microprocessor. It is natural to extend the cost advantages of LSI microprocessors and LSI memory to the I/O section. Thus, future microprocessors will have a family of compatible I/O chips.

The use of LSI facilitates the placement of large amounts of logic on the I/O chip which allows the creation of an intelligent I/O chip. These chips will be capable of performing many I/O functions independent of the microprocessor. This eliminates the I/O bottleneck that arises when the microprocessor controls the I/O directly in detail and therefore must respond to every I/O event in the system.

The use of LSI I/O chips allows intelligent I/O chips to be implemented economically. The use of intelligent I/O chips creates a system with distributed processing capability in which the I/O chips

National IMP-16

```
        LI    R1, -100          Loop Initialization
        LI    R2,  20
Loop:   LD    R0,  (2)          Loop execution
        ROUT        10          time = 34.4 μsec
        AISZ        2,1         I/O data rate =
        AISZ        1,1         1/34.4 μsec = 2.91 x 10^4 words/sec
        JMP         Loop        = 5.82 x 10^4 bytes/sec.
        HALT
```

INTEL 8080

```
        LXI         H           Loop Initialization
        MOV         B,M
        LXI         H
Loop:   MOV         A,M         Loop execution
        OUT         10          time = 21 μsec
        INX         H           I/O data rate =
        INR         B           1/21 μsec = 4.76 x 10^4 bytes/sec.
        JNZ         Loop
        HLT
```

Fig. 4.5.2 Programmed I/O benchmarks

process I/O concurently with microprocessor execution. Thus the I/O handling responsibility is removed from the microprocessor and system I/O performance is enhanced.

Currently, several types of I/O chips exist, namely parallel interface chips and serial interface chips. DMA controllers are also being implemented as LSI chips in which several channels of DMA are designed into a single chip. In addition, interface/controller chips are being developed for common microcomputer peripherals such as CRT's, floppy disks and magnetic cartridges. These chips will lower system cost by reducing the chip count and minimizing the design time.

As microcomputer systems become standardized (within a given family) with respect to instruction set I/O commands, interrupt system protocols and DMA handling, general purpose I/O interface chips will be developed which can give each device programmed I/O, interrupt and DMA capability. In one type of I/O system, each I/O device has an interface, but all control exists within the microprocessor which issues commands to each interface.

In a distributed intelligence I/O system, each I/O device is connected to an intelligent I/O chip. In effect, the I/O control that existed in the microprocessor has been distributed to each I/O chip. The I/O chip functions as the device controller and is capable of controlling the I/O device independent of the microprocessor. The I/O chip has all the interface logic needed to communicate with the microprocessor through programmed I/O or interrupts. The I/O chips also has the logic needed to perform DMA operations with the

device. The use of intelligent I/O chips which are capable of functioning independently requires an additional system element to resolve requests from all the devices needing resources. This device arbitrates all request and grants system resources on a priority basis. This type of system will result in minimal hardware costs for the capability provided. The design of the I/O system will then consist mainly of writing the software to manage all the concurrent processing.

REFERENCES

Section 1

1. Altman, L., "Single-Chip Microprocessors Open Up a New World of Applications," Electronics, April 12, 1974, pp. 81-87.

2. Bursky, Dave, "Choosing a uP By Its Capabilities is a Growing 'Family Affair'," Electronic Design, July 5, 1977, pp. 26-38.

3. Davidow, W. H., "General-Purpose Microcontrollers Part I: Economic Considerations," Computer Design, July 1972, pp. 75-79.

4. Davidow, W. H., "General-Purpose Microcontrollers Part II: Design and Applications," Computer Design, August 1972, pp. 69-75.

5. Faggin, F., and others, "The MCS-4 - An LSI Micro Computer System," IEEE Region 6 Conference Record (1972), pp. 1-6.

6. Hoff, M. E., Jr., "Considerations For the Use of Micro Computers in Small Systems," Wescon Technical Papers (1972), Paper 26/3.

7. Hoff, M. E., Jr., "The New LSI Components," 6th IEEE Computer Society International Conference Digest (1972), pp. 141-143.

8. Laliotis, T. A., "Microprocessors Present and Future," Computer, July 1974, pp. 20-24.

9. Lewis, D. R., and Siena, W. R., "How to Build a Microcomputer," Electronic Design, September 13, 1973, pp. 60-65.

10. National Semiconductor, A Microprogram Development System, Application Note #AN-123, Santa Clara, Cal. 1974.

11. National Semiconductor, GCP/C Product Description, Pub. #4200005B, Santa Clara, Cal. 1973.

12. National Semiconductor, IMP-16C Application Manual, Pub. #4200021C, Santa Clara, Cal. 1974.

13. Rattner, J., and others, "Bipolar LSI Computing Elements Usher in New Era of Digital Design," Electronics, September 5, 1974, pp. 86-96.

14. Reyling, G., Jr., and Weissberger, A. J., "Microprocessor Components and Systems," (unpublished, written at National Semiconductor).

15. Reyling, G., Jr., "Considerations in Chosing a Microprogrammable Bit-Sliced Architecture," Computer, July 1974, pp. 26-29.

16. Reyling, G., Jr., "Single-Chip Microprocessor Employs Minicomputer Word Length," Electronics, December 26, 1974, pp. 87-93.

17. Schultz, G. W., and Holt, R. M., "MOS LSI Minicomputer Comes of Age," AFIPS Conference Proceedings, FJCC (1972), pp. 1069-1080.

18. Schultz, G. W., and others, "A Guide to Using LSI Microprocessors," Computer, June 1973, pp. 13-19.

19. Smith, H., "Impact of LSI on Micro Computer and Calculator Chips," NEREM Record (1972), pp. 143-146.

20. Texas Instruments Staff, Designing With TTL Integrated Circuits, McGraw-Hill, New York, 1971.

Section 2

1. Blood, William, Jr., MECL System Design Handbook, Motorola Semiconductor Products, Inc., Mesa, Arizone, 1972.

2. De Falco, John, "Comparison and Uses of TTL Circuits," Computer Design, Feb. 1972, pp. 63-68.

3. De Falco, John, "Reflection and Crosstalk in Logic Circuit Interconnections," IEEE Spectrum, July 1970, pp. 44-50.

4. Fairchild Semiconductor, The ECL Handbook, Mountain View, Cal., 1974.

5. Garrett, Lane S., "Integrated-circuit digital logic families, Part II-TTL devices," IEEE Spectrum, Nov. 1970, pp. 63-72.

6. National Semiconductor, Application of Tri-state IC's, Application Note #AN-45, Santa Clara, Cal., 1972.

7. Peatman, John, The Design of Digital Systems, McGraw-Hill, New York, 1972.

8. Priel, Ury, "Take a look inside the TTL IC," Electronic Design, April 15, 1971, pp. 68-81.

9. Signetics Corp., Signetics Data Book, 1974.

10. Texas Instruments Staff, Designing with TTL Integrated Circuits, McGraw-Hill, New York, 1971.

11. Texas Instruments Staff, The TTL Data Book, Dallas, Texas, 1973.

12. Texas Instruments, Low Power Schottky Users Guide, 1977.

13. The 3M Company, Scotchflex Cable/Connector System.

Section 3

1. Davis, Sydney, "Selection and Application of Semiconductor Memories," Computer Design, Jan., 1974, pp. 65-77.

2. Feeney, H., "Microcomputer Applications of Electrically Alterable ROM's," 1972 Wescon Technical Papers, Session 4.

3. Frankenberg, Robert J., "Designers Guide to: Semiconductor Memories - Part 1 through Part 8," EDN, August 5, 1975 - Nov. 20, 1975.

4. Gorman, Ken., "The programmable logic array: a new approach to microprogramming," EDN. Nov. 20, 1973, pp. 68-75.

5. Hoff, Marcian, "Designing with Semiconductor RAM's - Part I," EDN, August 5, 1973, pp. 30-35.

6. Intel Corp., The Intel Memory Design Handbook, Santa Clara, Cal., August 1973.

7. Strauss, Leonard, Wave Generation and Shaping, McGraw-Hill Book Co., New York, NY, 1970, pp. 547-612.

8. Thomas,A. Thampy, "Design Techniques for Microprocessor Memory Systems," Computer Design, August 1975, pp. 73-78.

Section 4

1. Bass, J. E., "A Peripheral-Oriented Microcomputer System," Proc. of the IEEE, June 1976, pp. 860-873.

2. Bond, J., "Interfacing peripheral devices with minicomputers," EDN, Dec. 5, 1973, pp. 48-54.

3. Falk, H., "Linking Microprocessors to the Real World," IEEE Spectrum, Sept. 1974, pp. 59-67.

4. Gladstone, B., "Designing with microprocessors instead of wired logic asks more of designers," Electronics, Oct. 11, 1973, pp. 91-104.

5. Intel Corp., 8080 Microcomputer System Manual, Santa Clara, Cal., 1975.

6. Moffa, R.,"Interfacing Peripherals in Mixed Systems," Computer Design, April 1975, pp. 77-84.

7. National Semiconductor, IMP-16 interrupts, Application Note #AN-107, Santa Clara, Cal., 1975.

8. National Semiconductor, IMP-16C peripheral interfacing simplified, Application Note #AN-124, Santa Clara, Cal., 1974.

9. National Semiconductor, IMP-16C Application Manual, Pub. #4200021C, Santa Clara, Cal., 1974.

10. Sawyer, G., "Tools and techniques of microprocessor data transfer," Proc. National Computer Conference, 1975, pp. 15-20.

11. Van Gelder, M. and others, "A Primer on Priority Interrupt Systems," Control Engineering, Mar. 1969, pp. 101-105.