Ch. 6 - 12-bits

Photos to get:

1) Fig. 2 LINC 62
from [CLARK AND MOLNAR, 1964]

2) Fig. 3 The production version of the LINC.
from (DEC brochure 5049 20-7/65)

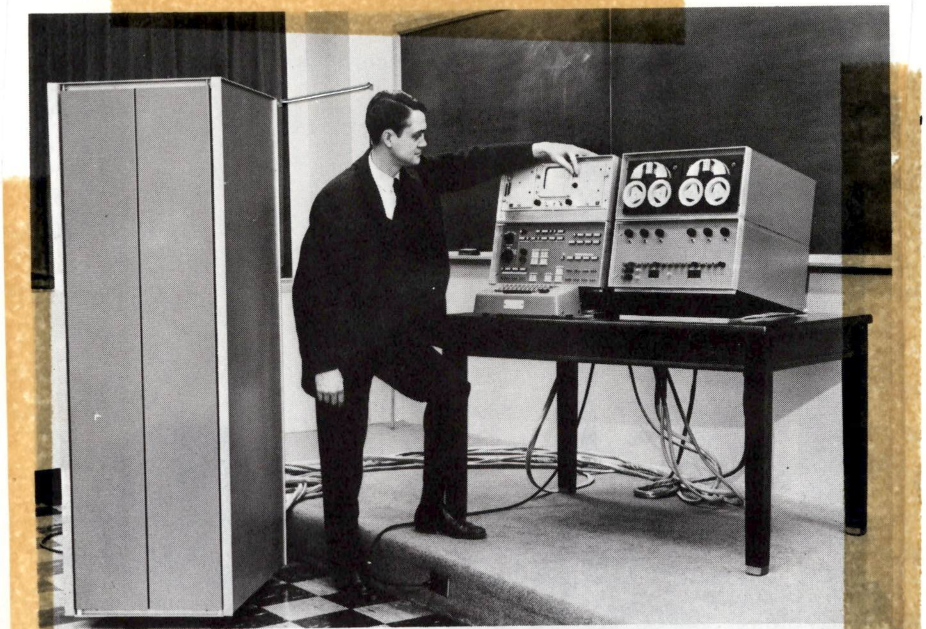
3) VT 14 screen display w/ ladder diagram

Fig. 18 - Courtesy of Intersil?
(block diag CMOS-8 from HM-6100 broch)

Fig. 19 "

Fig. 30 - wheel of Rein (p/r from Sutherland?)

Fig. 31 - NO caption. Is this DEC's?



~~FIGURE 1~~. The LINC (Laboratory Instrument Computer) is a small stored-program digital computer designed to accept analog as well as digital inputs directly from experiments, to process data immediately, and to provide signals for the control of experimental equipment. The LINC system comprises five physically distinct subassemblies which include four console modules connected by separate cables to a remote cabinet containing the electronics and power supplies. The control module contains indicator lights, push buttons, and switches used in operating the LINC. A second module provides for display oscilloscopes, while a third module holds two magnetic tape transports of special design. The last module is provided with sockets, jacks, and terminals for interconnecting the LINC and other laboratory equipment. This photograph and those following show the prototye version demonstrated on March 27, 1962, at the MIT Lincoln Laboratory.

Figure 2 Linc62

[taken from [Clark and Molnar, 1964]

CH.6 12-bits

Wes Clark

864-5229 : re: photo from Clark + Molnar '68
LINC

NA 5-12

NA 5-10

GIVE JOHN FIGS DELETED / XEROXES?

CHAPTER 6

1. XEROX FIGS. TO BE DELIV. TO T.D. ✓

*2. XEROX FIG. LIST (GIVE TO JOHN TO GIVE TO LOUISE?) ✓ FILE

3. REASSEMBLE ART IN RED COPY ✓

4. TRACK PHOTOS (2 FROM DEC): XEROX/GIVE GLOSSIES TO TD

5. ANSWER PERMISSION ?'s

* FIG. #'S NEED TO BE CHANGED IN TEXT, TOO.

Reprinted from Annals of The New York Academy of Sciences
Volume 115, Article 2, Pages 653-668
July 31, 1964

THE LINC:
A DESCRIPTION OF THE LABORATORY INSTRUMENT COMPUTER

W. A. Clark
Massachusetts Institute of Technology
Cambridge, Mass.

C. E. Molnar
Air Force Cambridge Research Laboratories
Bedford, Mass.

Introduction

There has been a growing recognition in recent years of the value of digital techniques for controlling apparatus and for handling experimental data within the laboratory. Several kinds of special-purpose digital devices, some of them capable of simple fixed calculations, have been developed by various workers and applied to problems of averaging, correlation, signal generation, event recording, etc. For the most part, each of these specially-designed devices is capable of only a narrow range of tasks and lacks the versatility desirable in a general laboratory tool.

The so-called general-purpose digital computer is potentially the most versatile digital device by virtue of its ability to execute complex programs of internally stored instructions. However, it has been either too large and inaccessible or too slow for useful work in a laboratory environment. Furthermore, the technical development of the general-purpose machine has tended to emphasize those design features which exploit its ability to solve well-defined mathematical problems in symbolic terms, with a consequent deemphasis of features useful in processing data in less highly stylized and rigid forms.

This deficiency of traditional computer design is particularly apparent in the context of the biological or medical research program. Here, the problems of achieving adequate operational flexibility, handling large quantities of data (often in the form of electrical signals), and having to deal with incompletely developed mathematical procedures all underline the need for more appropriate computer systems.

The LINC (Laboratory Instrument Computer) is an attempt to satisfy this need. It reflects some of the experience gained in several years of collaboration by members of the Digital Computer Group of the MIT Lincoln Laboratory* and the Communications Biophysics Group of MIT's Research Laboratory of Electronics.† The photographs show the LINC prototype which

*Operated with the support of the U. S. Army, Navy, and Air Force.

†Supported in part by the U. S. Army Signal Corps, the Air Force Office of Scientific Research, and the Office of Naval Research; in part by the National Science Foundation; and in part by the National Institutes of Health.

was built in early 1962 at the Lincoln Laboratory. Since that time the instrument has been redesigned under the general supervision of the authors in association with N. T. Kinch, S. M. Ornstein, D. Malpass, Jr., W. Simon, M. J. Stucki, and Miss M. A. Wilkes. H. H. Loomis, Jr., D. F. O'Brien, and T. C. Stockebrand participated in the development of the prototype.

Design Objectives

In designing the LINC, the principal underlying objective has been to maximize the degree of control over the instrument by the individual researcher. Only in this way, it is felt, can the power of the computer be usefully employed without compromising scientific objectives. In particular, the goal of the development has been a machine which: (1) is small enough in scale so that the individual research worker or small laboratory group can assume complete responsibility for all aspects of administration, operation, programming, and maintenance; (2) provides direct, simple, effective means whereby the experimenter can control the machine from its console, with immediate displays of data and results for viewing or photographing; (3) is fast enough for simple data processing "on-line" while the experiment is in progress, and logically powerful enough to permit more complex calculations later if required; (4) is flexible enough in physical arrangement and electrical characteristics to permit convenient interconnection with a variety of other laboratory apparatus—both analog and digital—such as amplifiers, timers, transducers, plotters, special digital equipment, etc., while minimizing the need for complex intermediating devices; and (5) includes features of design which facilitate the training of persons unfamiliar with the use of digital computers.

The present LINC design represents a reasonable balance among the conflicting requirements set by these objectives; the success of the design can be evaluated only by using the computer in a wide variety of laboratory situations. Responsibility for the further evolution of the LINC now resides under NIH sponsorship,* in the MIT Center Development Office for Computer Technology in the Biomedical Sciences. The program is now in the final development phase, during which a number of instruments are being made available for evaluation and use in biomedical laboratories. A list of the laboratories in which LINC's will be in operation is available on request.

Description

The LINC is a small stored-program digital computer which uses transistor circuitry and a random-access ferrite-core memory. The speed of the computer is fixed by the length of time required to read information from or store

*Under contract PH43-63-540 with the Division of Research Facilities and Resources of the National Institutes of Health, in cooperation with the Bio-Sciences Office of the National Aeronautics and Space Administration.

information into one of the 1,024 (expandable to 2,048) 12-bit memory locations. Most of the LINC's instructions require from one to four memory-cycle times of eight microseconds each for execution. The instructions available may be grouped as follows: (1) arithmetic instructions which perform addition, multiplication, counting, etc.; (2) logic instructions which perform simple logical manipulations; (3) data transfer instructions which move information from one computer location to another; (4) indexing instructions which provide a convenient means of referring to tables; (5) input-output instructions which transfer information to and from external equipment; (6) magnetic tape instructions which control various digital magnetic tape operations; and (7) control instructions that determine which of alternative sets of instructions are to be executed according to various criteria.

use this caption

The LINC consists of four independent console modules connected (through easily detachable 30-foot cables) to a cabinet containing the electronics and power supply (FIGURE 1). One console module houses most of the controls used in operating the computer as well as indicator lights. A second module contains terminals used to connect the LINC to other laboratory equipment. The remaining two modules house a display oscilloscope and a pair of specially designed magnetic tape transports which form an integral part of the LINC. The four console modules are mechanically separate and may be stacked and rearranged in any desired configuration. The front panel of each console module may be removed from its box and mounted in a standard rack with other laboratory equipment (FIGURE 2).

FIGURE 2. Two of the console panels mounted in a standard rack with other laboratory equipment in an arrangement convenient for simple electrophysiological experiments. The oscilloscope on the table can substitute for the console module scopes shown in other figures. The remaining panels are operating in another part of the room.

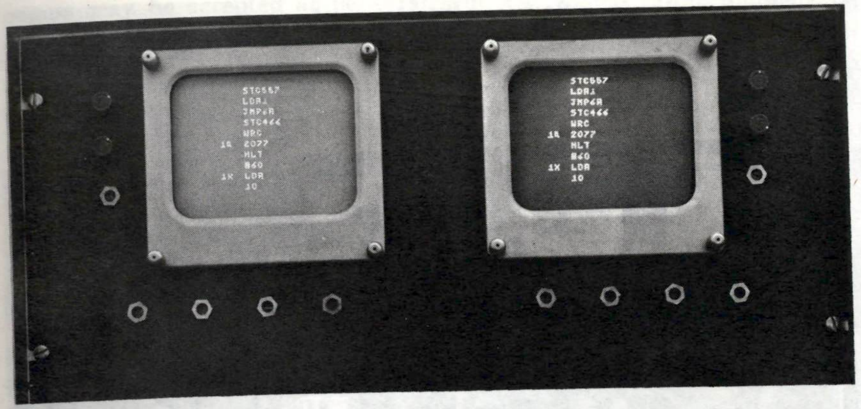


FIGURE 3. The oscilloscope is a primary means of communication between LINC and user. A typical instruction manuscript which has just been typed on the keyboard is displayed on both scopes.

Programs are initially typed on a simple keyboard, stored directly in the LINC's core memory, and simultaneously displayed on the oscilloscope (FIGURES 3 and 4). They may be typed either as octal numbers or, aided by a simple assembly program, in a symbolic form which uses three letter mnemonics to represent instructions. Several features which aid in the process of "debugging" programs are incorporated in the LINC. The computer can be set to stop whenever a chosen memory location is referred to, or a program may be executed slowly, one instruction at a time, while its actions are observed by means of the indicator lights on the control console. Once a program has been written and corrected, it may be stored on magnetic tape and conveniently retrieved for use at a later time.

A wide variety of both digital and analog data and control paths for connecting the LINC to other laboratory equipment has been provided. There are 16 analog input channels connected to an internal analog-to-digital conversion device which translates an input voltage on any of these channels into an eight-bit binary number. Up to 31,000 conversions, under control of the computer program, can be made per second. A simple program will display data converted in this way on the oscilloscope (FIGURE 5). Eight of these analog input channels are normally connected to potentiometers whose knobs can be used as manual controls or parameter inputs (FIGURE

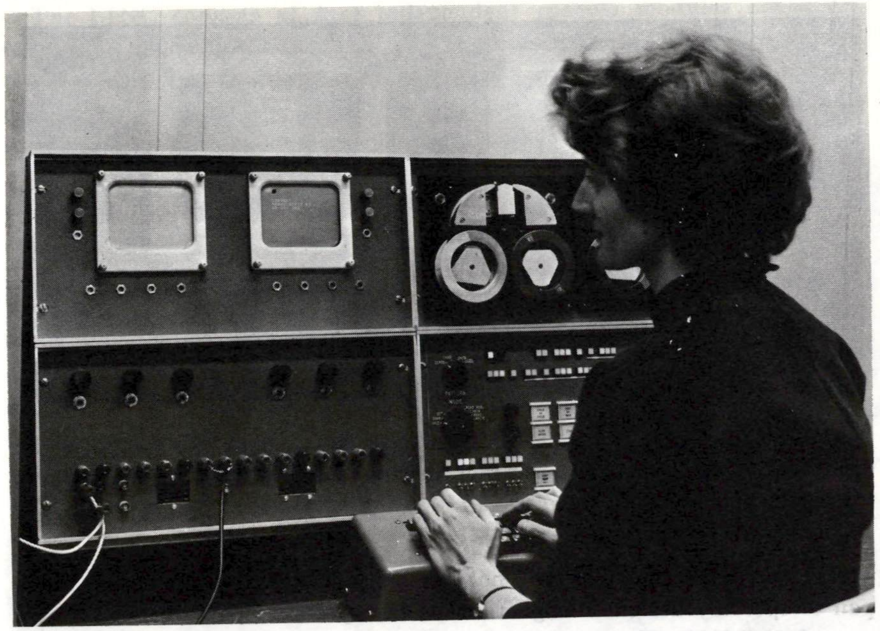


FIGURE 4. Information typed on the keyboard and displayed on the scope is simultaneously stored in the central memory. From here it may be transferred to magnetic tape and recovered whenever desired.

6). Four sets of 12-bit digital input terminals may be connected to other digital equipment, such as counters, timers, special encoders, or other devices which produce signals representable in parallel binary form. These digital

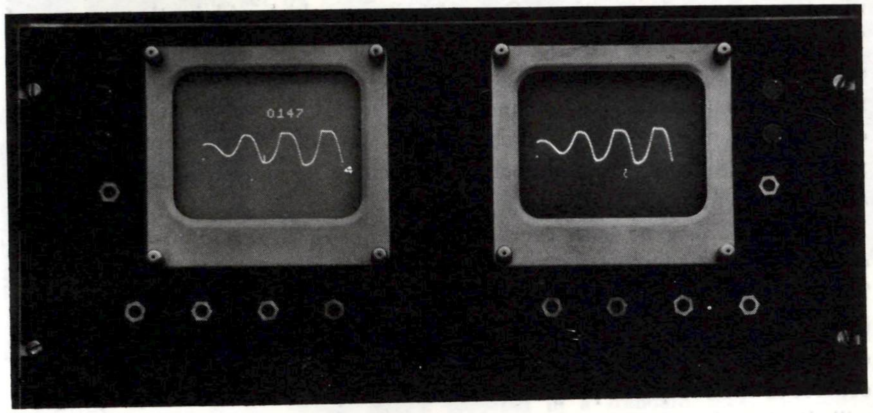


FIGURE 5. The oscilloscope is operated point-by-point and can display experimental or calculated curves as well as characters. Note that the displays on the two scopes need not be identical.

inputs may be accepted at peak rates up to 125,000 12-bit numbers per second. Another set of inputs to the LINC provides a convenient means for the computer to sense external events and synchronize itself with external equipment.

The principal output of the LINC is the oscilloscope display which may be viewed directly or photographed (FIGURES 7*a-f*). Displays of programs, data, results of calculations, etc. can be generated point-by-point in graphical or symbolic form at rates of 10,000 to 20,000 points per second in typical programs. Letters or digits are generated and displayed by a special instruction at a rate of 4,000 characters per second. There are two distinct display channels which may be connected either to the display scope mounted in the console module or else to remote standard oscilloscopes with special plug-in units. LINC provides each display scope with a pair of deflection signals which position the spot with a precision of one part in 512 along each coordinate and an intensification signal to brighten the selected location on the scope face.

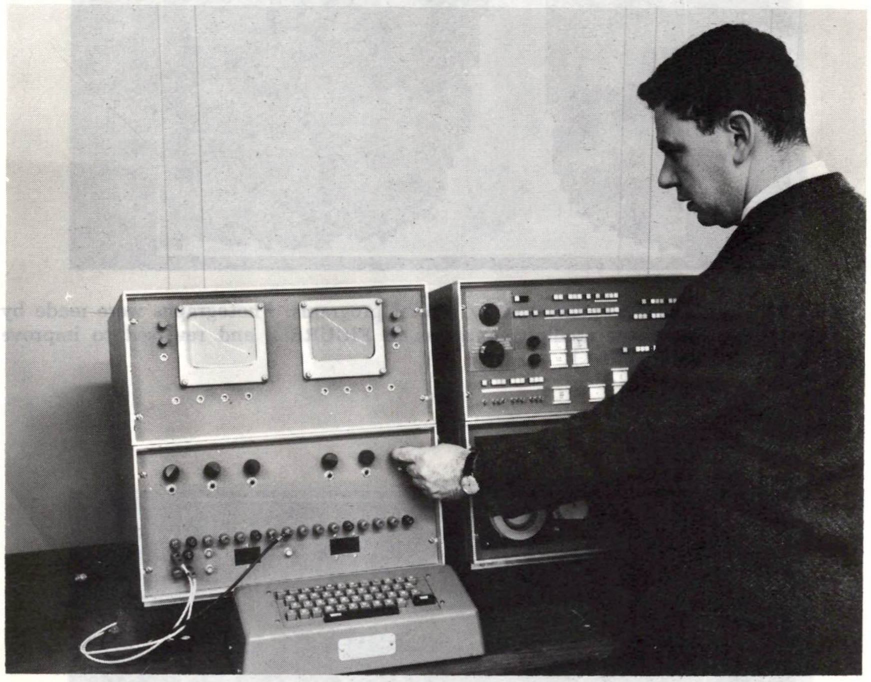
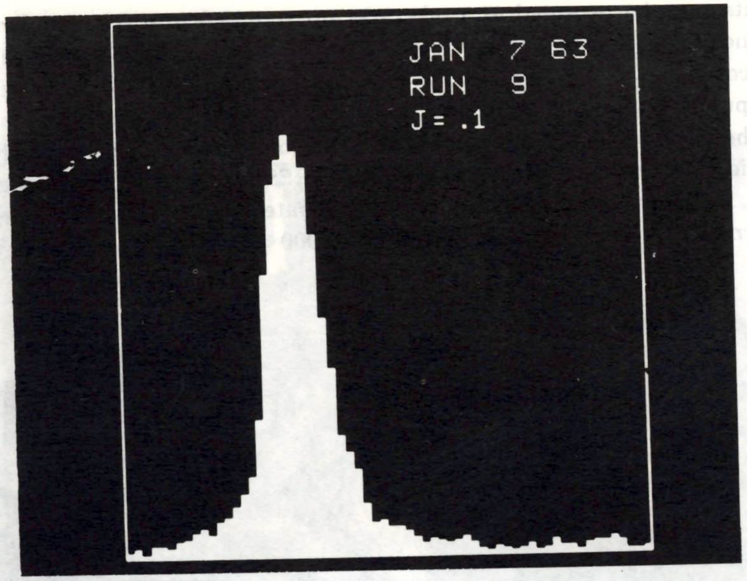


FIGURE 6. The knob is being used to adjust the fit of a straight line to a curve derived from a signal connected to one of the LINC prototype's eight analog input channels.



FIGURES 7*a-f*. Displays generated by typical programs. Photographs were made by a Polaroid camera on the oscilloscope shown in FIGURE 2 and reversed to improve legibility. (FIGURE 1*a, above*.)

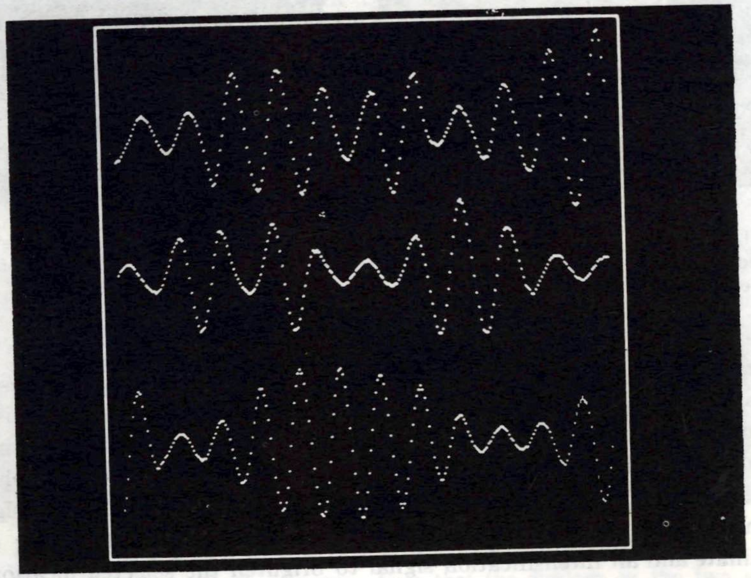


FIGURE 7*b*.

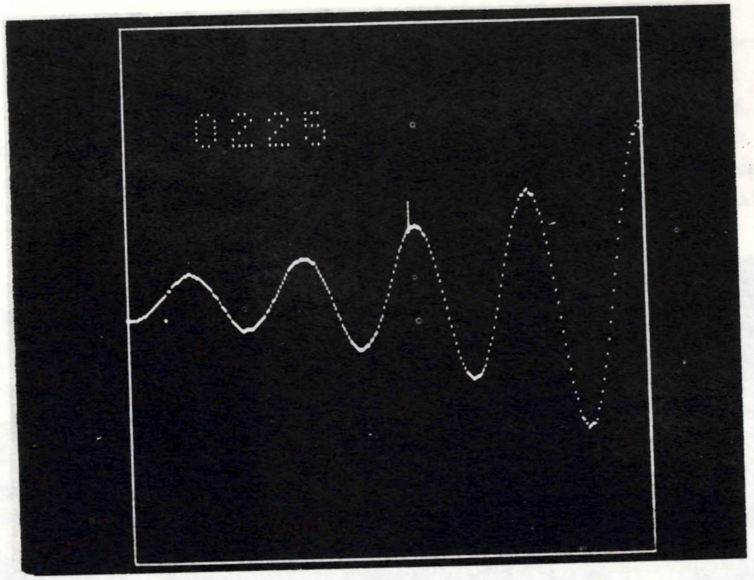
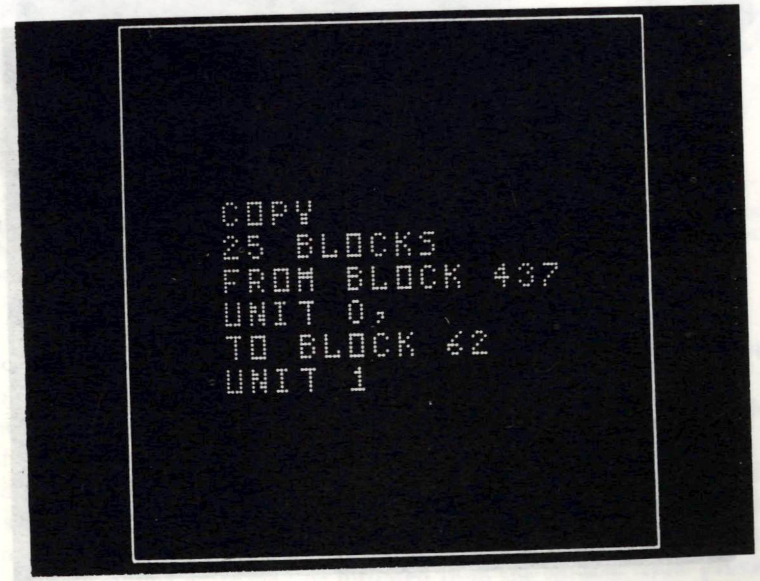
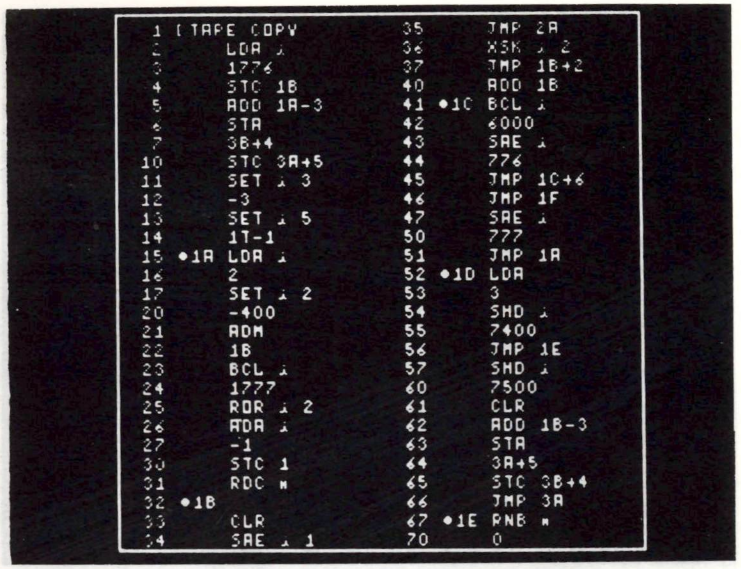


FIGURE 7*c*.



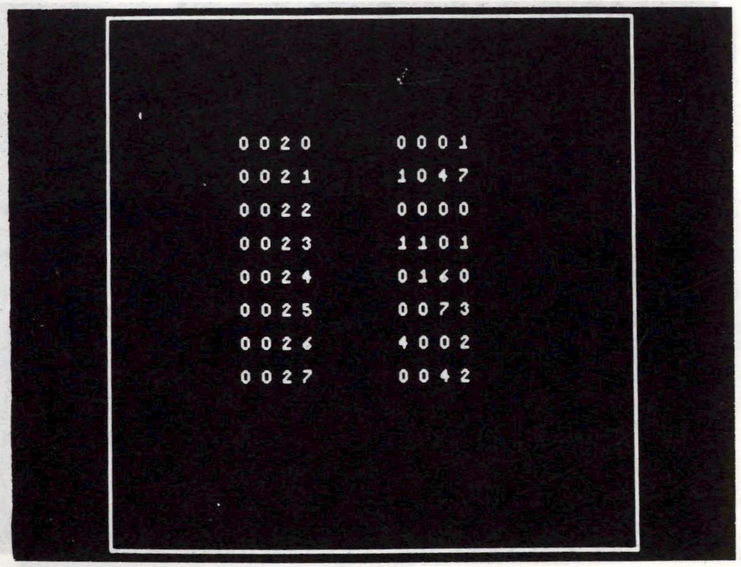
```
COPY
25 BLOCKS
FROM BLOCK 437
UNIT 0,
TO BLOCK 62
UNIT 1
```

FIGURE 7*d*.



```
 1 [TAPE COPY          35        JMP 2A
 2       LDA i         36        XSK i 2
 3       1776          37        JMP 1B+2
 4       STC 1B        40        ADD 1B
 5       ADD 1A-3      41 •1C    BCL i
 6       STA           42        6000
 7       3B+4          43        SAE i
10       STC 3A+5      44        776
11       SET i 3       45        JMP 1C+6
12       -3            46        JMP 1F
13       SET i 5       47        SAE i
14       1T-1          50        777
15 •1A   LDA i         51        JMP 1A
16       2             52 •1D    LDA
17       SET i 2       53        3
20       -400          54        SHD i
21       ADM           55        7400
22       1B            56        JMP 1E
23       BCL i         57        SHD i
24       1777          60        7500
25       RDR i 2       61        CLR
26       ADA i         62        ADD 1B-3
27       -1            63        STA
30       STC 1         64        3A+5
31       RDC m         65        STC 3B+4
32 •1B                 66        JMP 3A
33       CLR           67 •1E    RNB m
34       SAE i 1       70        0
```

FIGURE 7*e*.



```
0020        0001
0021        1047
0022        0000
0023        1101
0024        0160
0025        0073
0026        4002
0027        0042
```

FIGURE 7*f*.

The magnetic tape system was specially developed to form an integral part of the LINC system (FIGURE 8). It can be used to store programs, numerical data and results on small reels of 3/4 inch-wide tape which contain 512 consecutively numbered blocks, each capable of storing 256 12-bit numbers. The tape moves at approximately 70 inches per second in either direction and information is recorded at a track density of about 400 bits per inch. A block occupies 2.5 inches of tape and can, therefore, be scanned in about 1/25 of a second, although the time required to start or reverse a transport is approximately 1/10 of a second.

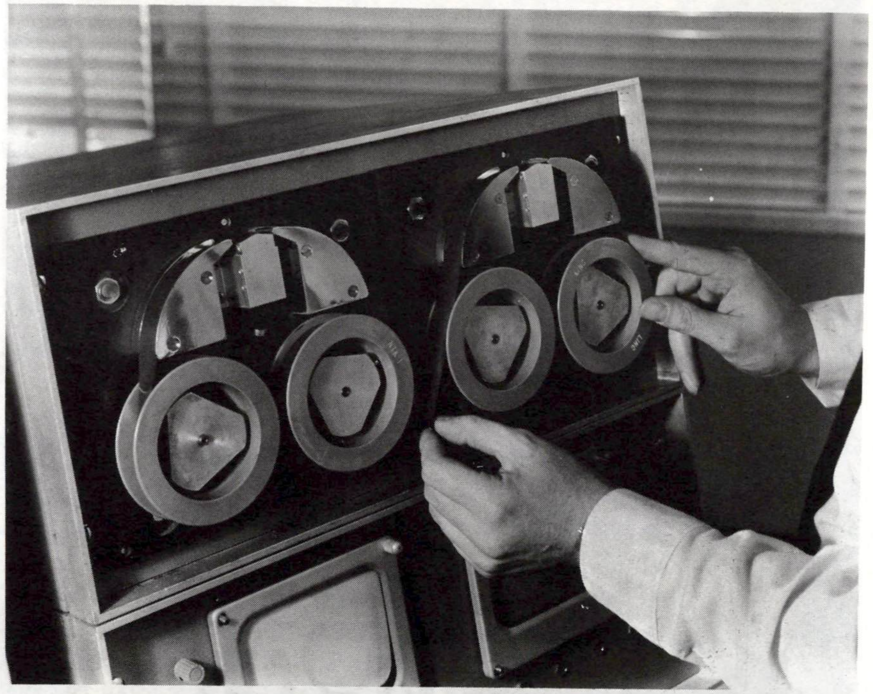


FIGURE 8. To use the magnetic tape unit, a loaded reel is snapped onto the transport and the tape is drawn over a simple open guide to a take-up reel. Push buttons facilitate loading and unloading.

The entire process of searching the selected tape for a desired block, transferring information in either direction between the block on tape and the core memory of the computer, and checking the transfer is controlled by a single computer instruction. A variant of this instruction permits transfers involving as many as eight consecutive blocks to be made. All operations of the magnetic tape units may be initiated from the console, thus providing a convenient way of changing programs (FIGURE 9).

Output paths from the LINC to other equipment include two channels of analog output signals which are usually used to operate the oscilloscope display but which can drive an x-y plotter or be used for other purposes. These analog output signals have a precision of one part in 512 and may be changed at peak rates up to about 60,000 times per second. Parallel digital outputs are available over one channel which provides up to 125,000 12-bit numbers per second. Two forms of control outputs are available, one of which consists of six sets of relay contacts operated by the computer, and the other of 16 digital output lines which can be pulsed individually by a special computer instruction. These features make it relatively simple, for example, to install a typewriter as an auxiliary device.

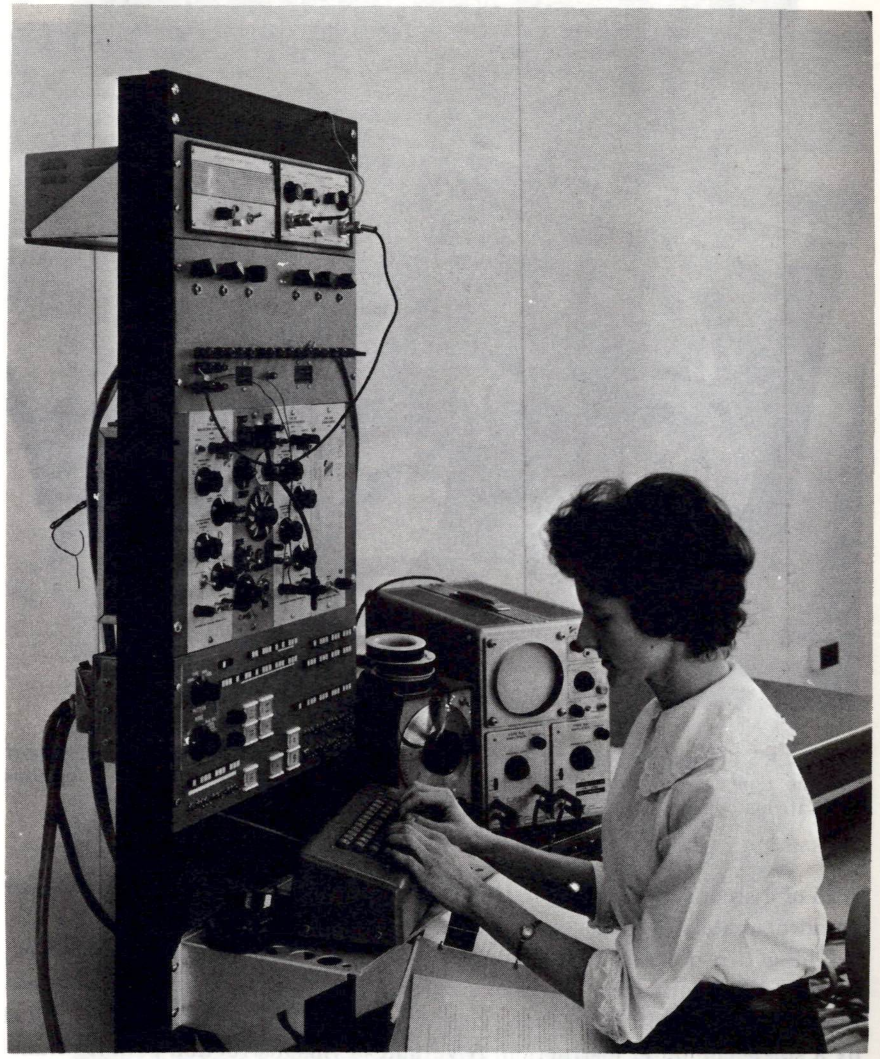


FIGURE 9. The keyboard can be used to change the values of experimental parameters or to modify operating programs. Records of such changes can be stored on magnetic tape.

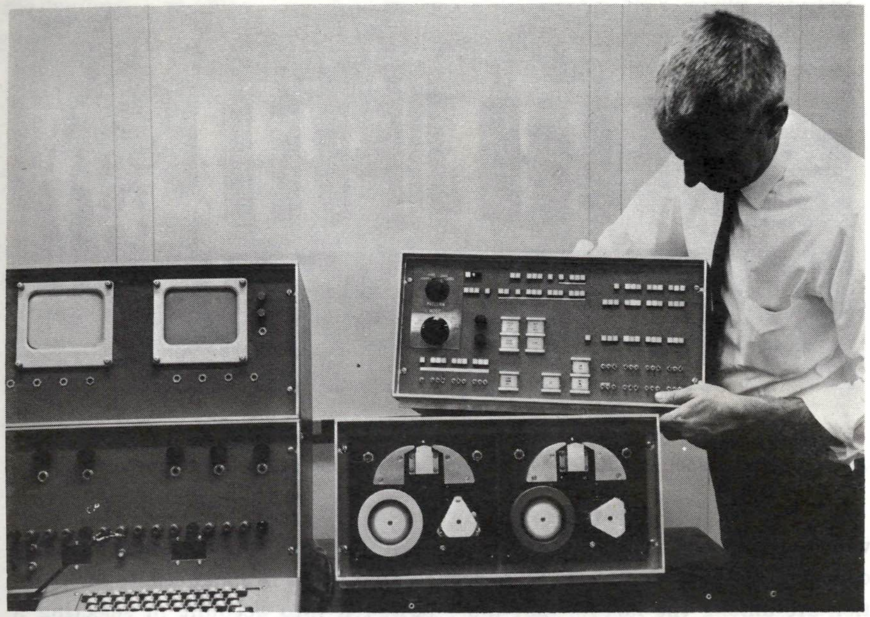


FIGURE 10. The control module being stacked on top of the magnetic tape module. Cables connecting console panels to the electronics cabinet are plugged in from the rear.

A box-like design has been used in the console modules to simplify mounting and physical arrangement (FIGURE 10). Cables are installed from the rear to connectors on the removable console panels contained within the boxes (FIGURE 11).

For the most part, the electronic circuits used in the LINC are standard commercially manufactured units (FIGURE 12). These are in the form of cards which plug into a mounting frame held in the electronics cabinet along with the power supplies (FIGURE 13). The LINC system requires about one kilowatt of standard 115 volt A.C. power.

Examples of Applications

It is difficult to give a comprehensive summary of the capability of the LINC in various applications. The flexibility of a stored program machine allows one to choose, from among many alternatives, a programming scheme

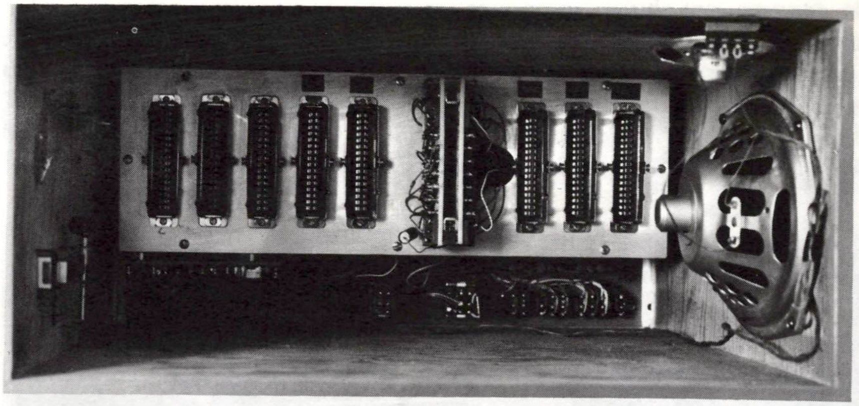


FIGURE 11. Rear view of the control module showing cable connectors on the back of the control panel. Loudspeakers provide audible indication of computer activity.

which efficiently utilizes the resources of the computer. Central storage capacity, speed of operation, word length, and speed of magnetic tape operation are among the factors that must be taken into account in selecting a programming scheme. For any particular problem there is usually no unique "best solution." A description of a few typical problems which have been considered and tried by various investigators may give some insight into the range of the LINC's usefulness.

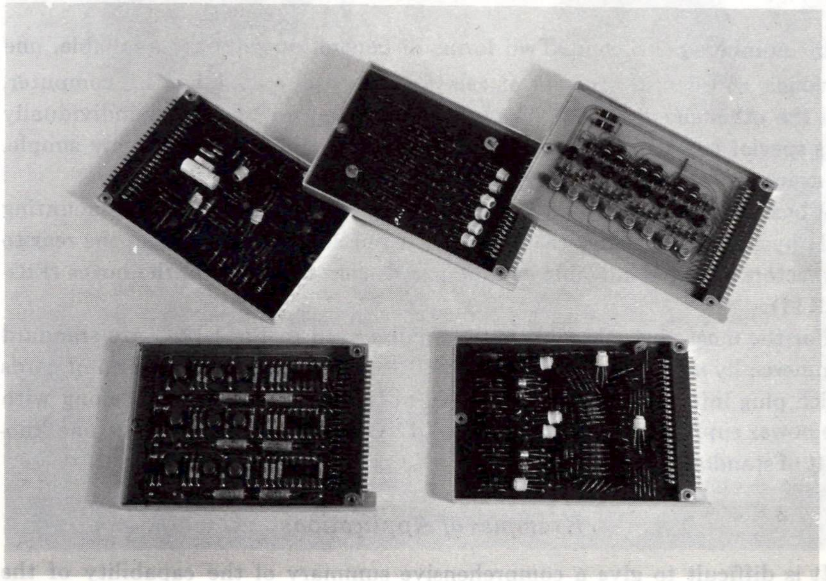


FIGURE 12. Typical transistor circuit plug-in units. About 300 plug-in units are used in the LINC.

Averaging of evoked electrophysiological responses. Presentation of acoustic stimuli to a cat with implanted electrodes and averaging of cortical and thalamic responses were performed by the LINC. Averaged responses to series of stimuli as well as information relating to the variability of the responses were immediately displayed and also automatically stored on the LINC magnetic tape for more detailed examination at a later time.

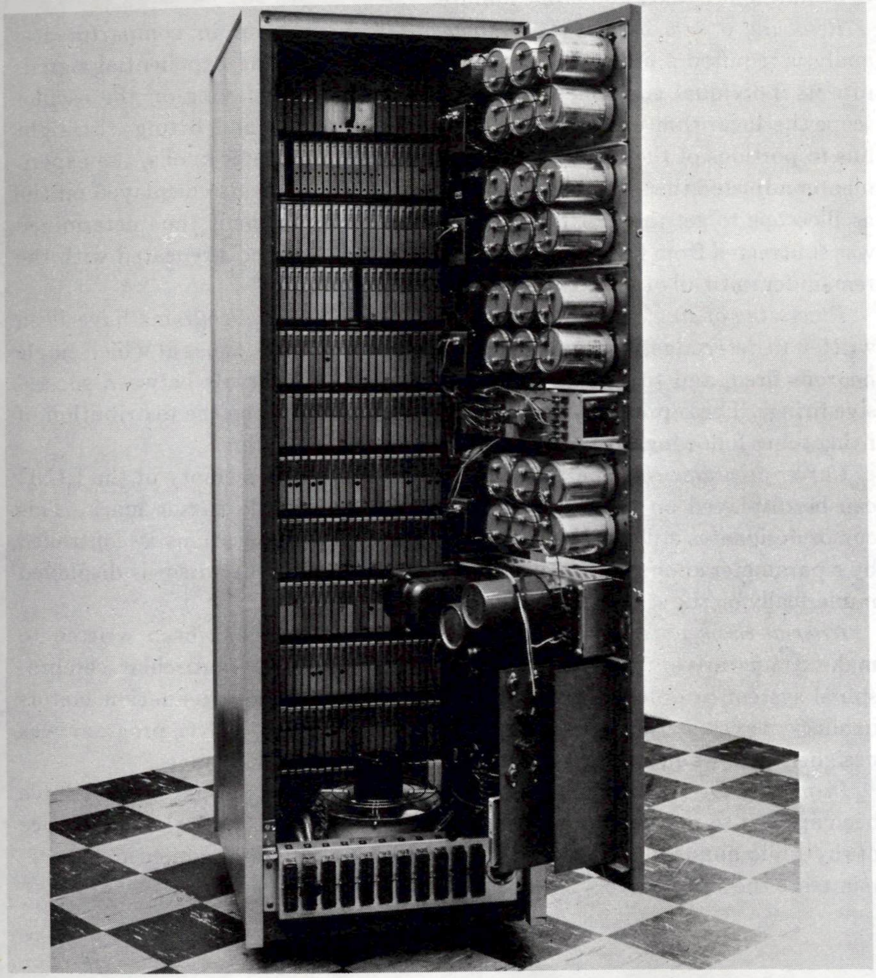


FIGURE 13. The electronics cabinet of the LINC prototype opened to show plug-in units and the power supply. The LINC system uses about one kilowatt of standard 115 volt A.C. power.

Fourier analysis. A program has been written to perform a Fourier analysis of electron diffraction data from thin metal films and to resynthesize the original data from its Fourier components for verification of the analysis. The Fourier components and the resynthesized data are displayed on the oscilloscope along with the original data. The analysis and resynthesis carried to 50 harmonics takes about one minute.

Resolving a sum of decaying exponentials. A problem in compartmental analysis required a program to resolve a sum of decaying exponential signals into its individual components. This was done by displaying on the oscilloscope the logarithm of the wave form being analyzed and fitting a straight line to portions of the resulting curve. With the parameter knobs, the experimenter adjusted the slope and position of a straight line also displayed on the oscilloscope to get the best fit to the data. The component thus determined was subtracted from the original wave form and the process repeated with the remainder until all of the components were resolved.

Processing of single-unit data from the nervous system. Programs have been written to determine, from microelectrode recordings, the times at which single neurons fired, and to calculate the distribution of intervals between successive firings. These programs can also be used to determine the distribution of firing times following the presentation of a discrete stimulus.

Cursor program. An experimental curve stored in the memory of the LINC can be displayed on the scope along with an adjustable cursor mark. This cursor designates a desired point on the curve and its locations is controlled by a parameter knob. The amplitude of the point under the cursor is displayed numerically on the scope.

Arterial shock wave measurements. A LINC program has been written to make comparative hydrodynamic measurements in the ventricular cerebrospinal system in order to determine the dissipation and attenuation factors in shock waves attributable to the arterial pulse. The LINC program was designed to work directly with amplified signals from straingauges.

In-phase triggering of stimuli from EEG alpha wave. Simple criteria have been applied to portions of EEG signals to identify and mark the occurrence of rhythmic bursts of alpha activity, and to trigger stimuli which are phase-related to the alpha wave.

5

Levy, "Buses, the Skeleton of Computer Structures"
1/27/78

Chapter ~~17~~ 10

Buses, The Skeleton Of Computer Structures

Contents

PRIOR
EDIT

Ø. Introduction

What is a Bus?

A bus is a communication pathway connecting two or more electrical devices. In the context of minicomputer design, buses are the physcial and electrical structures which determine how the building-blocks are interconnected.

In every computer system, there are many buses: Internal pathways connect the registers and arithmetic logic of a CPU; Input/Output pathways connect CPUs, memories, and peripheral devices; and external communication buses attach computer systems to the wide world of telephone and other data communication pathways. In this chapter, we restrict our discussion to buses which interconnect *computer system components which are to be designed by different engineering groups*.

This people-oriented criterion for a discussion of computer design may sound out of place, but in fact one of the most important functions of buses is to provide a well-specified interface between subsystems which are in themselves complex. On the basis of this criterion, we exclude from discussion many of the internal buses which exist inside a CPU, such as tri-state buses, and buses whose specifications are determined by engineers not involved in the minicomputer design process such as telecommunication lines. We also have not included any multiprocessor interconnection pathways here, simply because none has yet been made into a standard PDP-11 family product.

What Does a Bus Do?

A bus is a communication medium. Each one exists in order to transfer information from place to place within a computer system. Often, in studying the design of computer systems, the difficulty of establishing communications pathways is ignored by the student.

In this chapter, we attempt to illustrate the complexities of bus design by drawing on the real history of some PDP-11 family designs. Keep in mind that the success of bus designs is measured by the following criteria in computer systems being manufactured and sold:

1. Does the bus successfully establish the communication pathway required?

2. Is the bus well specified (and well documented), so that a series of interfaces to it designed concurrently & over time by different engineers will in fact be compatible?

3. Does the bus avoid imposing performance constraints on the system?

4. Is the <u>cost</u> of the bus commensurate with the computer system and the bus' role in it?

5. Does the bus design <u>anticipate expansion</u> of the system in the future (without excessive cost)?

6. Can the bus be <u>manufactured</u> and tested in high volume production without excessive hand-crafting or tuning?

Beyond the scope of this chapter are some additional functions of buses such as providing a means to diagnose and repair the system components connected to it, and to allow measurement of system loads and performance.

Why are Buses Important

As the list of success criteria above suggests, there are many ways in which poor bus design can spoil the performance or cost/performance ratio of an otherwise well-designed computer system. Failure to anticipate future expansion of a computer system is a common problem in bus designs. As we will see in Section V, the PDP-11 Unibus, a very successful bus, first became inadequate as the main interconnection pathway when processor and memory speeds surpassed the bandwidth capability of the Unibus. Later, the Unibus 18 bit memory address width became a limitation to be overcome.

Computer design is driven by advances in semiconductor technology. While the performance/cost (or storage capacity/cost) ratio for logic and memory is increasing at a rate of 70% to 100% per year, the bandwidth/cost and other performance ratios of interconnections is steady or *decreasing* slightly. As a result, *bus designs tend to persist in time across several redesigns of the other computer system components*. This provides all the more justification for extensive engineering effort in the initial design of a bus. Each bus is going to be with us for a relatively long time.

How are Buses Designed?

To design a bus, the engineer must first find out what system components are to be interconnected. Then, studying the requirements of communications between these components, the engineer chooses a structure. Finally, the cost constraints and available technologies lead to a choice of the implementation.

The 5-function model given in Section I below is not a set of bus designs. It is a functional model which results from taking the commonly used minicomputer building-blocks (CPUs, memories, and I/O controllers) and asking a simple question about each of them: What communications need to occur between this component and each other component? The model shows the five types of communications which were the answers to that question. The five functional pathways are the maximum number of interconnections that would be useful in a conventional single-processor minicomputer. Real bus designs combine these functions into cost-effective implementations.

After choosing the structure and functions of buses, the engineer must write a specification. This is crucial to the success of bus design if it is indeed to be interfaced by a number of different engineers. The level of detail in a specification can never be too great. An example of a Digital bus specification is shown in Figure Ø below. This fragment is taken from the Massbus specification, and it specifies the timing of a data read operation, the communication taking place between a disk and the disk controller. The timing is specified in three forms. Figure Øa shows the verbal description, which is the final authority, in case of discrepancy between the three. Figure Øb shows the same specification in timing-diagram form, including all of them minimum and maximum delay constraints. Figure Øc repeats the specification in flowchart form. These three parts together make up only one-fourth of the timing specification of the Massbus.

After writing a specification, the engineer builds a prototype and tests it. It is very helpful to have other engineers build interfaces to the bus concurrently, because misunderstandings of the specification will be uncovered sooner. Finally, it is important that the specification be *maintained*, updating it to conform to the latest known design constraints. A very useful appendix to a bus specification is a list of the design problems that came up during the engineering of interfaces to it, and the details of how they were resolved. This was done for the Massbus, in a section of the specification called "Design Notes."

What is the plan of this chapter?

Section I presents the 5-function model of computer interconnections. This focusses our attention on organization of computer systems. In the hierarchy (Chapter Ø) of architecture, organization, and realization, we intend to avoid questions of realization, except as aspects of realization affect choices in organization. Therefore, there will be no discussion of packaging, logic families, or electronic design parameters in this chapter. Architectural considerations in the PDP-11 family are covered in Chapters 7, 8, and 13.

Sections II, III, and IV develop classifications of techniques for arbitration, data transfer synchronization, and error control of buses. These are the chief design parameters at the organizational level.

Section V gives a structural history of the high-performance PDP-11 Systems, beginning with the PDP-11/2Ø. The sequence of designs for the PDP-11/45, the PDP-11/7Ø, and the VAX-11/78Ø illustrate how evolving technologies led to structural evolution of the buses interconnecting these systems' components.

Finally, Appendix A contains a summary chart of the performance and historical parameters of the buses used in examples throughout this chapter.

Levy, "Buses, The Skeleton of Computer Structures"
1/27/78

I. Functions of buses in computer systems: a 5-function model

The functional building-blocks of computers are: central processing units (Pc), random-access (primary) memory (Mp), input/output controllers (Kio), and peripheral units. Peripherals tend to be classed as either secondary memory (Ms), or transducers or terminals (T).

The *architecture* of a computer system is the set of concepts, states, and actions that can be seen and manipulated by the programmer. Often, the components of an architecture are *virtual*, that is, manipulable by the programmer, but not necessarily implemented in physical or real components.

We are concerned in this chapter not with architectures, but with the real *organization* of computer systems. In particular, we want to examine the interconnections between real components.

Figure 1 shows the components of a traditional single-processor minicomputer system. Five different paths are shown interconnecting these components. These paths are not intended to represent actual buses designed for such a system. Instead, we have considered each pair of components in the system, and asked whether they need to communicate with each other. If so, we have put in a pathway between this pair. This leads to a model which has the maximum number of interconnection pathways. We then pose two questions about each pathway:

1. What are the transactions that occur between the components connected to this pathway?

2. What distinguishes the engineering requirements of this pathway from others?

We will later see how in real computer systems, we combine the functions of these pathways into multi-function buses in order to get economical designs.

There are five types of interconnections shown in Figure 1, labelled A,B,C,D, and E. These labels have mnemonic value:

The pathway A, connecting the central processor (Pc), with the primary memory (Mp), is the path by which instructions and data are fetched from and data are stored into memory. Let us call a "word" the unit of memory accessed over this pathway. This pathway is distinguished by requiring *one address per word*. Random-access memory, by its nature, must be told on each access what address to use.

Pathway B connects one or more mass storage (secondary memory) controllers (Kio), to the primary memory (Mp). It represents the path by which blocks of data are transferred between primary memory and, for example, disk storage. It is distinguished by being a *block-transfer* medium. In particular, we do not require more than one primary memory address per block transfer, because the data is (almost always) stored in consecutive memory locations. (In paged virtual memory systems, the pages may be scattered in the physical address space, but the block would usually be stored in consecutive *virtual* memory locations.)

Pathway C is the *control* pathway. It connects the central processor to all of the I/O controllers. Over this path, I/O commands are sent from the central processor to the I/O controller, and status information from the controllers is returned. Also, it represents the path by which I/O controllers can cause an interruption to the central processor program. (Occasionally, we may find that the control path is also used to transmit small amounts of data to or from an I/O device, under program control, such as sending a character to be printed to a console terminal.)

Connecting I/O controllers (Kio) with their controlled peripheral units (Ms or T) are pathways labelled D, for device interconnections. In Figure 1, pathway D_1 represents a disk connection and D_2 a multiple terminal connection path. These two differ principally in that the terminal interconnections do not normally transfer blocks of data. Both D_1 and D_2 carry control information, however.

Finally, pathway E represents a connection to external communication lines. Usually, the computer designer does not have control over the specification of such external pathways. (Note that a communications controller, K_{comm}, may be connected via a B-type pathway to main memory, in addition to its C-type pathway connection.)

Table II. Requirements for the Five Types of Pathways

	Pathway Types				
	A	B	C	D	E
	CPU-Memory	Controller-Memory	CPU-Controller	Controller-Peripheral	Controller-External
Memory Addressing Requirements	large:2^{22} (one address per word)	large:2^{22} (one address per block)	none	none	none
Maximum Number of Connections	small:2^{4}	small:2^{4}	medium: 2^{6}	small-large:2^{8}	small-large:2^{8}
Latency Tolerance	low	high	medium	medium-high	medium-high
Bandwidth Requirements	high	high	low	low-high	low-high
Length Requirements	short	medium	long	medium-long	medium-long

Five key parameters or requirements for these pathways affect cost and performance, and are often traded off against each other. These are: memory addressing, number of connections, latency tolerance, bandwidth, and length. Table 2 summarizes these requirements for the five types of pathways.

1. Memory addressing applies only to the A and B type pathways by which memory is connected to other functional components. "Memory addressing" means selecting a word or block of words within the address space of the memory subsystem. Memory address bits are no different from data bits, from the standpoint of the bus designer. Both must be transmitted from one bus connection to another. However, the handling of memory addresses is so important to computer system performance and expandability that we cite it separately, in order to call attention to it.

On both A and B type pathways, we need to be able to address all of the memory words. In PDP-11 systems through the PDP-11/7Ø, the largest address required is 22 bits (to address 2^{22} bytes of memory). The Unibus was originally designed to implement the A, B, and C type paths for PDP-11s; it carries 18 bits of memory address. Expansion of PDP-11 memory beyond 2^{18} bytes caused major structural revisions to be made, as we will show in Section VI.

2. The maximum number of connections to a bus tells us how many signals must be used to select a destination for a data transfer on the bus. Typically, a bus will carry some number, n, of "select" signals, and therefore be able to deliver data to as many as 2^n connections. On a type A pathway, a CPU accesses connections which contain memory. We do not typically need more than four select signals, allowing up to 16 memory connections. Often, some of the memory address bits are used to make the memory bank selection. If a memory connection implements contiguous addresses, then higher-order memory address bits are used to select the connection. In "interleaved" memory systems, some lower-order address bits are also used to select the connection, so that apparently-contiguous memory addresses are accessed from different connections (memory banks). Interleaving is used to gain performance when a memory has a longer cycle time than access time. When sequential addresses are accessed in an interleaved memory system, the completion of a cycle in one part of the memory can be overlapped with the accessing of the next address in another part. In the case of multiprocessor shared-memory systems, it may be necessary for some "select" signals to be present when data is delivered from memory to processor, in order to identify which processor is the destination for the data.

However, if the memory-reading operation is performed without relinquishing the bus between delivery of an address to memory and return of data from memory, then no processor-identification is needed. The "owner" of the bus during the transaction is clearly the correct receiver of the data.

On a type B pathway, selection and memory addressing have the same properties as on type A pathways. What distinguishes type B from type A is that blocks of data (say 512 bytes) may be transmitted to and from memory without attaching a memory address to each byte or "word". This could lead to cost-saving in some bus designs. Note that typical type B pathways have multiple non-memory connections, such as disk controllers.

A type C pathway has no memory addressing to do. Selection of a destination connection may typically be done with six signals, allowing up to 64 controllers to be connected to a CPU.

The Unibus carries no select signals at all. Transfers on the Unibus are not directed to physical connections. Instead, there is the single concept of memory addresses. Each data transfer (type A or type B) on the Unibus is directed to or from a one- or two-byte section of memory. The memory address is broadcast to all connections. If one of the connections recognizes the address as being one of its "own", then it participates in the data transfer. This anonymity allows an essentially unlimited number of connections to be made to the Unibus, with each connection implementing a locally-determined number of memory bytes.

For control transfers (type C), the Unibus has a concept called "the I/O page". A block of memory addresses (the I/O page) is reserved for use in accessing control and status registers which are located in peripheral controllers. The uppermost 8192 bytes of memory addresses are never implemented in real memory. Instead, small segments are assigned (by administrative procedures) to each I/O controller type. When a controller is attached to Unibus, it responds to data transfers to and from addresses within its assigned segment.

In this manner, there is no restriction on the number of controllers, and no fixed amount of address space need be allocated to a given controller. If two controllers of the same type are connected to a Unibus, one of them is assigned to a "floating" address segment, which is an area reserved for such conflict resolution.

I/O controllers which perform direct memory access (DMA) do so by making data transfers to memory addresses below the "I/O page". Block transfers are performed a word at a time to and from successive memory addresses, with the incrementing address being maintained by the I/O controller.

An I/O controller causes an interruption by a special control transfer to the unique interrupt-fielding CPU. In this transfer, the destination always the CPU, and the interrupting controller identifies itself by transmitting an "interrupt vector" as the data. Therefore the address lines of the Unibus are not used in this transfer.

Type D and E pathways may need as many as eight select signals, allowing up to 256 devices or external connections to be selected as a destination.

3. Latency tolerance means: how long a delay (latency) after a connection "decides" to make a transfer can be tolerated by that connection.

4. Bandwidth means: how many transfers per second can be made.

Latency is different from bandwidth: latency refers to the delay, for any one transfer, from the time it is initiated until it is completed; bandwidth is the repetition rate at which the initiation and completion of requests can be sustained, over some period time. In particular, peak bandwidth -- the maximum possible repetition rate -- is the parameter which affects the cost of a bus, and is the one we refer to here.

Type A pathways require both low latency and high bandwidth. The performance of a CPU-memory system depends critically on the rate (bandwidth) at which words can be delivered to the CPU. Furthermore, the nature of program interpretation prevents the CPU from knowing very far in advance which memory words will be required next. (If it did know, then it could request them far in advance, and not be sensitive to the latency.) Therefore, the CPU-memory pathway is also very dependent on low latency. The sooner after a memory address is generated the memory can deliver the memory contents, the faster the CPU will run. In this type of pathway, effective bandwidth and latency are directly (inversely) related to each other.

On a type B pathway, high bandwidth is also typically required. Usually, this is the path on which disk and other mass storage data is moved to and from memory. In most cases, the rate at which data is transferred is determined by the disk subsystem. In minicomputer systems developed through 1977, the bandwidth required has not exceeded 1 Megabyte/second for an individual disk-controller-to-memory pathway.

Type B pathways, on the other hand, tolerate relatively long latencies. If there is sufficient buffering of data at the controller, system performance is relatively insensitive to delays of as much as 1ØØ to 1ØØØ microseconds in starting up a block transfer. The insensitivity is due to the relatively long delays already present in disk data accessing. Mechanical positioning delays, both rotational and radial, may take tens of milliseconds in a typical disk data access operation.

Type C pathways -- the control and interruption links -- do not require high bandwidth compared with CPU instruction and DMA data activity. I/O control commands are issued relatively infrequently compared with the instruction execution rate in the CPU. Interruptions occur even less frequently. However, latency tolerance is not very high on the control pathway. It is important for interruptions to be delivered promptly; and CPU instructions which access I/O control and status registers usually are held from completion until the access has been completed. Therefore, Table 2 shows latency tolerance as "medium" (1 to 1Ø microseconds) for type C pathways: it is permissible to take a little longer to complete an I/O control instruction than other instructions, but not so long as initiating a block transfer from a disk.

Type D and E pathways tend to handle interactions which are a mixture of the communications on type B and type C pathways. Therefore, their requirements for latency and bandwidth vary over the range shown for types B and C.

Length refers too the maximum possible distance, along the pathway, from one connection to another. Maximum length is important because it affects both performance and cost of a bus. The CPU to memory pathway (type A) has been shrinking in length in recent computer designs because of the critical dependence of latency on length. The speed of light (or, more properly, of signals in a wire) limits the minimum delay between request and response. As a result, we see memories and CPUs more frequently packaged together or in very close proximity. Fortunately, the continual size reduction of a given amount of CPU logic or memory has encouraged this trend. The current length range of a type A pathway for minicomputers is approximately Ø.1 to 3 meters.

Type B pathways must interconnect memory with all high-speed I/O controllers. These controllers are also tending to be packaged closer to the memory in recent system designs. But since there may be many controllers, the length of the pathway may have to be two to ten times longer than the CPU-memory pathway (Ø.2 to 3Ø meters).

Control pathways, connecting the CPU to all I/O controllers often have to be extended out of the CPU-memory package to reach peripheral control subsystem packages. These tend to be the longest pathways which the computer system designer has to deal with. Frequently, the design choice in connecting a peripheral to a minicomputer system is between (a) extending the main type C bus out to reach the farthest peripherals and (b) designing type D buses which extend from a locally-packaged controller to a remote peripheral. Alternative (b) gives maximum flexibility and performance, but it costs more than (a) and may lead to a proliferation of buses in the computer system. (See Figure 2.5)

Cost is not shown as an explicit parameter in Table 2, because all parameters shown contribute to cost in ways we will discuss next.

Cost

The cost of a computer system could be allocated in a simple way to power, logic, and package. As applied to the cost of buses, these become power, logic complexity, and cable/connector costs. Let's examine how the requirements of interconnection pathways are reflected in these costs.

Increasing memory addressing requirements lead to more signals in the pathway with each signal adding to power and cable costs. Or we could trade lower bandwidth for wider memory addresses by time-multiplexing the addresses with data. Increasing the maximum number of connections leads to increased power in the bus drivers in order to maintain a given signal level, or to lower bandwidth as it takes longer for signals to settle due to the number of electrical loads. Also, more signals are required (logarithmically increasing with the number of connections) to select the destination of a transfer. Increasing maximum length also requires more bus drive power for a given signal level. Obviously, long cables also cost more than short ones. Since longer buses have greater propagation delays, we can trade both bandwidth and latency for increased length. Both length and connections contribute to signal decay, and therefore theses two are often traded against each other. For example, each section of a Unibus is rated for a maximum length of 50 feet or a maximum of 2Ø bus "loads". Reaching either limit requires insertion of a "bus repeater" circuit. In fact, a Unibus with fewer loads could be operated at longer lengths, but the configuration rules would be too difficult to explain.

Decreased latency and increased bandwidth can be achieved by using higher power drivers and receivers such as ECL circuits, which have lower propagation delays through their circuits.

We can also increase bandwidth by providing more buffering logic (complexity) at each connection. For a given level of reliability, either faster logic (with higher power) or more parallelism (complexity) is required to increase the data rate. More parallelism would mean more signals and, therefore, higher cable and connector costs.

Lower latency can sometimes be achieved by distributing the task of arbitration among the connections. More logic is then required at each connection. (We will discuss arbitration in Section II.)

Figure 2.5 A design Tradeoff for Type C Pathways

Although we haven't mentioned them before, there are also considerations of physical and electrical environment which affect costs. For noisy electrical environments, we may choose balanced transmission lines or low-voltage high-current signals, both of which lead to higher power consumption compared to conventional single-ended 3- to 5-volt logic signals. To compensate for noisy enviornments we may add error detection and correction circuits at each connection, adding to their complexity. Or we may use shielded or twisted-pair cable, adding to their cost. For physically stressful environments, cable costs may become dominant as the cables are armored, strengthened or given non-corrosive wrapping. In general, we can trade reduced bandwidth for increased immunity to electrical noise, since most noise-induced errors can be overcome by repetition and redundant signalling. (At this tradeoff, bus design becomes one with applied communication theory.)

II. Arbitration Methods

Since buses are intended for use by more than one source of data transfer requests, there must be a means of deciding which source is to use the bus next. This is what we mean by arbitration.

If we imagine a connection following a procedure to use a bus for a data transmission, it would consist of the following two steps:

1. (Arbitration) Obtain the use of the bus
2. (Data Transfer) Transfer data on the bus

Classifying the methods of arbitration

To assist our examination of arbitration methods, we will classify some of the methods into 12 categories using three discriminating criteria. The criteria are:

(Where?) (1) Location of the arbitration logic (*Centralized* or *Distributed*)

(How?) (2) Allocation rules (*Priority*, *Democratic*, or *Sequential*)

(When?) (3) Timing relationship of arbitration to data transfer (*Synchronous* or *Asynchronous*)

Centralized arbitration means that there is a single common arbitration point. A signal must pass from a requesting connection to the common arbitration point, and a response signal must return to the requesting connection before it may transfer data.

Distributed arbitration means there is no single common arbitration point.

The Unibus, for example, has centralized arbitration method (with the exception noted below). A contention-arbitrated serial bus, such as used in Ethernet [Metcalfe, 1975], has a distributed arbitration method.

The resolution of conflicting requests is accomplished in all arbitration methods by allocation rules.

Priority arbitration means that in case of an apparent tie in the race to request use of the data transfer facilities, the rules always let one connection (or a certain group of connections) go ahead of another connection (or group of connections).

Democratic allocation means that there are no priority rules. An apparent tie is resolved arbitrarily or by some "fairness" rule which attempts to keep any one connection from monopolizing use of the data transfer facilities.

Sequential allocation insures that there are never any apparent ties by giving only one connection at a time the opportunity to request use of the data transfer facilities. (The sequence is not necessarily round-robin).

The Unibus has priority allocation. (Each priority "BR" level is wired sequentially through the devices which share that level and, therefore, represents a group of connections). Most contention-arbitrated serial buses have `democratic allocation (the "p-persistent" retry algorithm is intended to cause random allocation among the contending connections). Centralized, polled, type D buses are frequently used to connect character terminals to a concentrator, and are examples of sequential allocation.

Finally, there is the question of timing. This is not a question of the timing relationship between communicating connections (which we call data synchronization and clocking). It is the timing relationship between the arbitration of a request and the data transfer which is going to occur as a result of the request. Arbitration synchronous with respect to data transfer means that the choice of the next connection to use the data transfer facilities occurs at a fixed time relative to the data transfer. This category includes buses in which the signal lines used in the data transfer are also used for the arbitration process, and therefore arbitration and data transfer cannot be overlapped in time.

Arbitration *asynchronous with respect to data transfer* means that a connection may request use of the data transfer facilities at any time, independent of the current state of the data transfer facilities.

The Unibus has asynchronous arbitration. Polled buses have synchronous arbitration because data transfer always occurs in the time slot immediately after the arbitrator has polled a requesting connection. Contention-arbitrated serial buses are usually synchronous, too, in that the data transfer *is* the request for use of the bus.

Table 3 summarizes the categories of arbitration methods. Below, we illustrate some of the categories with examples from Digital-designed buses.

Examples of Arbitration Methods

Example 1: Unibus

Table 3. The twelve categories of arbitration methods

Arbitration Methods	Synchronous with respect to Data Transfer	Asynchronous with respect to Data Transfer
Central, Priority	CPS	CPA Unibus (+serial wire-through at each BR level)
Central, Democratic	CDS	CDA
Central, Sequential	CSS (typical multidrop teletype polling-shared data bus)	CSA
Distributed, Priority	DPS SBI	DPA
Distributed, Democratic	DDS (Contention-arbitrated serial bus)	DDA
Distributed, Sequential	DDS (Rings with token passing + data)	DSA (cf Unibus for each $\overline{\text{BR}}$ level + NPR --serial wire-through)

Note: The Massbus has no arbitration at all, because all control transfers originate from one point (and data transfers are initiated by a communication over the control section).

Connections to the Unibus other than the CPU can use the bus for two different types of "data transfer" transaction. One is a DMA data transfer (using the bus as a type B pathway, to transfer data between a controller and memory). The other is an interruption transaction which transmits an "interrupt vector" across the data lines to the CPU. Arbitration of requests to use the bus for these two types of transactions is similar, as we explain below. A third type of transaction is a CPU-memory (type A) or CPU-controller (type C) data transfer, for which a unique arbitration method applies. (The CPU contains the arbitration logic, and takes advantage of its "ownership" of the logic).

Figure 4 shows a diagram of a simplified Unibus with two controllers, C_1 and C_2, sharing a request line, BR ("Bus Request"). When C_1, the controller closest to the left end of the Unibus, wants to use the bus for an interruption transaction, it asserts (i.e., puts a logic "1" on) the shared BR signal line. The Arbitrator logic, shown at the far left end, receives the assertion of the BR signal; it is not able to distinguish which of the two controllers asserted the signal. At some later time (when the CPU is in a state capable of receiving an interruption), the arbitrator asserts the BG ("Bus Grant") signal. When controller C_1 receives the assertion of BG, it knows it may my fault

use the Unibus data transfer section next, as soon as the ongoing data transfer is complete. C_1 acknowledges its selection as the next data transfer "master" by asserting the SACK ("Selection ACKnowledge") signal. Since the BG signal is wired serially through the two controllers, it is up to controller C_1 to pass the assertion of BG if it does not want to use the next data transfer cycle.

If controller C_2 requests the bus by asserting BR, the BG signal would pass through controller C_1 to controller C_2, which then asserts SACK. No matter which controller initiates a request by asserting BR, it is clear that C_1 can use any BG assertion (that is, can block the BG assertion from being passed) that arrives after is (C_1) that arrives after it (C_1) has asserted BR. We could call this serial wiring a kind of priority arbitration, but we prefer to think of it as a sequential (or polling) type of allocation, in which the sequence beginnings on demand and always starts at the leftmost controller (the one closest to the CPU and arbitrator).

The Unibus actually has four groups of controllers, each group connected to a Bus Request line (called BR4, BR5, BR6, and BR7) wired as we have shown in Figure 4 above. In addition, every controller which is capable of doing DMA data transactions is connected into a fifth group (called NPR, for "non-processor request"[for data]). All five groups share a common SACK line.

In the most general case, a single controller can participate in three types of transactions.

(1) As the target of a control data transfer (type C), the controller behaves as if it were a memory. It receives commands (as data writes) into control registers and transmits status (as data reads) from status registers this way. The controller does not request the bus for these transactions: it is the "slave" of the CPU which obtained the bus for this purpose.

(2) As the originator of a data transfer (DMA, type B), the controller moves data to or from memory. To obtain the bus for this purpose, it asserts the NPR line, and waits for the NPG signal to be passed to it from the left. All DMA controllers share the NPR line.

(3) As an interruption source, the controller sends an interrupt vector to the CPU. To obtain the bus for this purpose, the controller asserts one of the four BR lines (BR4, BR5, BR6, or BR7), and waits for the corresponding BG signal (BG4, BG5, BG6, or BG7) to be passed to it from the left. Each controller is assigned a single BR level at the time of its installation in the system. Thereafter, it never blocks any of the other three BG signals.

Some controllers, such as simple terminal interfaces, do no DMA transfers. These controllers perform an interruption transaction for each character of input or output. When a character is received, for example, a terminal controller interrupts the CPU, and then the CPU retrieves the character by doing a control (actually memory) read from the character buffer register contained in the controller.

The priority arbitration of the Unibus is affected directly by the priority state of the CPU. The CPU program priority execution (PRI) varies from Ø to 7. The Unibus arbitrator grants use of the bus to non-CPU connections by the following rules:

1. At any time, when assertion of NPR is received, assert NPG. (A controller may do DMA data transfers at any time.)

2. Whenever the CPU is between instructions (i.e., is interruptable), then

   (a) if PRI <7 and BR7 is asserted, then assert BG7, else

   (b) if PRI <6 and BR6 is asserted, then assert BG6, else

(c) if PRI <5 and BR5 is asserted, then assert BG5, else

(d) if PRI <4 and BR4 is asserted, then assert BR4.

(When the CPU is interruptable, it will accept interruptions from a controller which is in a group whose priority is greater than the current program execution priority of the CPU.)

The priority arbitration rules of the Unibus involve both the CPU priority (which causes blocking of BG signals to groups lower than or equal to the CPU priority) and the relative priorities of the BR signals, among themselves. Assertion of a BR7, for example, blocks BG6, BG5, and BG4 until all controllers asserting BR7 have accomplished their interruption transactions. Therefore, we classify the Unibus, with its complex arbitration method, as centralized and asynchronous, with a mixture of priority and sequential allocation rules.

Example 2: Q-bus (LSI-11 bus)

The Q-bus serves the same functions for the LSI-11 system that the Unibus serves for most of the larger PDP-11 family processor systems. The Q-bus was constrained to use fewer conductors and, therefore, less power and logic than the Unibus. It achieves the reduction from 56 signals to 36 signals primarily by time-multiplexing memory addresses and data on the same conductors (accepting lower bandwidth in order to achieve lower cost).

Arbitration for DMA transfers on the Q-bus is essentially identical to that of the Unibus.

Figure 4 suffices to show the connectivity of DMA arbitration logic for the Q-bus. The corresponding signal names on the Q-bus are SACK (for SACK), DMR (for NPR), and DMG (for NPG).

Arbitration for the interruption transaction on the Q-bus is different from arbitration for interruptions on the Unibus. There is only one priority-group for all interrupting controllers on the Q-bus. When a controller wants to interrupt the CPU, it asserts the IRQ (Interrupt Request) signal on the Q-bus. This is similar to BR signals on the Unibus. However, the description of the interruption transaction following the assertion of IRQ properly belongs in the section on data transfer synchronization, and will be covered there (see Section __.) Arbitration on the Q-bus, like the Unibus, is classed as centralized, asynchronous, with allocation rules which are mixed priority and sequential.

Example 3: Synchrounous Backplane Interconnect (SBI), the VAX-11/780 Memory Bus

This memory bus is distinguished by its limited length (three meters: it does not extend beyond the etched backplane of the computer mainframe), and its master clock which synchronizes all transaction on the bus. The functions of the SBI are the same as those of the Unibus. However, the SBI differs in physical configuration because every controller must be directly connected to the mainframe backplane. Another difference between Unibus and SBI is that all transactions on the SBI are of fixed time duration, which gives much higher bandwidth for data transfer. (The SBI is rated at 13.3 Megabytes/sec, while the Unibus is capable of approximately 1.7 Megabytes/sec when operating with equivalent-speed memory). To achieve the SBI bandwidth, it is necessary to split the memory read operation into two bus transactions -- one to transmit an address to the memory, another to transmit data back to the requesting connection. In this way the SBI can accomodate memories of various cycle times, as can the Unibus, but the SBI does not occupy the bus facilities for the duraction of the cycle. [This concept occurs also in the Honeywell Series 6].

Arbitration on the SBI is distributed, priority-ordered and synchronous with respect to data transfer. Figure 5 shows a simplified diagram of the signals involved in SBI arbitration.

A master clock, represented here by a single signal, defines a sequence of time slots on the bus. Each slot is of long enough duration to complete a transfer of data from one connection to any other connection. However, there is not enough time in a slot for a reply signal to be sent back to the transmitting connection from the receiving connection. (A time slot in the VAX-11/780 SBI is 200 nanoseconds long.)

There are four TR ("Transfer Request") signals in this simplified example: TR0, TR1, TR2, and TR3. Each TR signal "belongs" to one connection; that is, only one connection is permitted to assert the signal.

Each TR signal has a priority associated with it: TRØ has highest priority, TR1 has next highest, and so on. A connection requests the use of the SBI data transfer facilities by the following procedure:

(1) At the beginning of the next time slot (after deciding to transfer data) assert the TR signal which belongs to this connection.

(2) At the end of the time slot, sense the state of ["strobe"] all of the higher-priority TR lines.

(3a) If none of the higher priority TR lines is asserted, then at the beginning of the next slot negate "my own" TR signal and begin transmitting data.

(3b) If any of the higher priority TR lines is asserted, then do not negate "my own" TR signal, and go back to step (1).

Figure 6 shows a timing diagram for a sample set of data transfers on the simplified SBI of Figure 5. In this example, connection number 3 (corresponding to TR3) requests the bus during slot 1, and connection numbers 1 and 2 (corresponding to TR1 and TR2) request the bus during slot 2.

At the end of slot 1, connection 3 detects no higher-priority TR signals, so it negates TR3 and transmits data during slot 2.

At the end of slot 2, connection 2 senses that TR1 is asserted, and therefore waits, leaving TR2 asserted. At the same time, connection 1 senses no higher-priority TR signals, so it negates TR1 and transmits data during slot 3.

Some transactions on the SBI require that a connection be allowed to transmit data on two or more consecutive slots. In order to accomplish this, the highest priority TR signal, TRØ, is not assigned to any connection. Instead, each connection which requires an additional slot (beyond its first one) asserts TRØ at the beginning of its first data transfer slot.

The example in Figure 6 shows connection 2 doing a 2-slot data transfer. Having first obtained use of the bus by asserting TR2 and waiting for connection 1 to go ahead, connection 2 "holds" the bus for its record slot by asserting TRØ (also known as the "HOLD" signal) at the beginning of slot 4. This guarantees that slot 5 will be available for use by connection 2.

In the VAX-11/78Ø, connections are limited to transmitting in no more than three consecutive slots.

We have shown four connections in this example, although only three TR signals are assigned. The lowest priority connection, number 4, does not have a TR signal assigned to it --- because there is no other connection which needs to sense a TR signal from this lowest-priority connection! Connection 4 senses all of the TR signals. It transmits only when no other connection is requesting the next slot. Note that connection 4 gains one advantage by being lowest-priority: it may transmit in any slot not used by the other SBI connections, without taking the time to assert a TR signal of its own. This gives it a shorter memory-access latency. For this reason the CPU is usually given lowest priority on the SBI in single-processor systems.

The master clock is crucial to the operation of the SBI. Not only must every connection sense the TR signals late enough to be sure that they have arrived from the farthest connection, but they must also be assured that they do not sense the early assertion of a next-slot TR signal from a nearby connection. In the VAX-11/78Ø, the master clock is distributed on six signal lines, consisting of three clocks on balanced-pair lines; the slots are defined by combining the three clocks into four equal-interval phase markers. All transmitted TR signals are asserted at the beginning of phase 1, and all received TR signals are sensed at the beginning of phase 4 (3/4 of the way through the nominal slot period.)

Example 4: A Polled Character-Input Bus (Type D)

Figure 7 shows a diagram of a simple character-input bus. The controller at the left end accepts all input from the keyboards. It "asks" each keyboard in turn whether it has a character to send, and if so, the controller takes the character during the next time slot. This arbitration scheme is centralized, sequential, and synchronous with respect to data transfer. The function of this bus is to transfer data from peripherals to a controller: the bus is of Type D.

Three signals are broadcast from the controller to all terminals. One is the clock, which defines the time slots, as did the clock in our previous example, the SBI. The other two signals, called UNITØ and UNIT1, send out a two-bit code which selects one of the four keyboards during each slot. The coding is binary. Allowing an asserted signal to represent a 1, and a negated signal a Ø, then the UNIT signals together select keyboards as follows.

UNIT1	UNITØ	Keyboard Selected
Ø	Ø	Ø
Ø	1	1
1	Ø	2
1	1	3

The controller changes the UNIT select signals at the beginning of each slot. The keyboard which is selected by the unit code senses its selection at the end of the slot. If the keyboard contains a character to be transmitted, then at the beginning of the next slot it asserts the SEND signal and transmits the character on the data lines.

In the example timing diagram shown in Figure 8, keyboard 1 transmits two characters and keyboard 2 transmits one character.

In this type of arbitration scheme, the polling (sequential sampling) of potential sources of data (the keyboards) eliminates the need for contention or priority rules. This makes the logic of each connection simpler, but it makes poorer use of the data transfer lines in typical applications. Note especially that this scheme limits each connection (keyboard) to using a maximum of 25% of the data transfer bandwidth.

Example 5: Massbus

The Massbus is a peripheral-to-controller (type D) bus which has no arbitration at all. Like the previous example, a single controller at one end of the bus receives or sends on each data transfer. Control information is transferred in a way similar to the Unibus, but the "master" of the transfer is always the controller. Data blocks are transferred using a peripheral-generated clock. These block transfers are always initiated by a Control word write into a register in the peripheral. Therefore, there is never any arbitration required for data or control transfers.

Interruptions to the CPU are generated by the controller on demand from any peripheral. For this purpose an Attention signal exists in the control section of the Massbus. Each peripheral is capable of asserting this signal. Status relating to an interruption is read by the controller (on command of the CPU), so there is no need for arbitration in this transaction, either.

III Synchronization of Data Transfers

In the section on arbitration methods, we showed bus usage to be a two-stage process, arbitration followed by data transfer. In this section, we describe how the timing of a data transfer occurs. We classify and explain data transfer timing for the five buses shown as examples in the Arbitration Methods section. This does not purport to be a complete compendium of data transfer synchronization methods, but it gives an introduction to a variety of approaches.

Synchronization of a data transfer means coordinating the timing between two bus connections which are involved in a data transfer. Note that the method by which data transfer is coordinated can be different from the arbitration method.

To classify the methods of data transfer synchronization, we use two criteria:

(From where?) (1) Location of the source of the synchronizing signals (*Centralized*, *Sender*, or *Receiver*)

(Periodicity) (2) Type of synchronizing signals (*Periodic*, or *Aperiodic*)

Table 9 shows the six resulting categories and how the examples fit into them.

The location of the synchronizing signal or signals may be at the connection which is sending the data (*Sender*), at the connection which will receive the data (*Receiver*), or at neither (Centralized). The Unibus has synchronization which does not fit neatly in this scheme, because every data transfer is synchronized by signals from both connections, the sender and the receiver.

The synchronizing signal may be a clock (*Periodic*), or it may be something else (*Aperiodic*). The Unibus uses an aperiodic "handshake" or exchange of signals whose timing varies with the speed of the memory involved.

Table 9. Data Transfer Synchronization Methods

	Periodicity	
Location of signal source	Periodic	Aperiodic
	CP	CA
Centralized	SBI Polled Character-Input	(no examples given)
	SP	SA
From Sender	Massbus data-read contention-arbitrated serial	Unibus data-out Q-bus data-out Massbus Control write
	RP	RA
From Receiver	Massbus data-write	Unibus data-in Q-bus data-in Massbus Control read

Example 1: Unibus

The two types of data transfer on the Unibus, DMA (type B) and CPU-memory (type A) are accomplished with the same data timing. We omit discussion of the interrupt-vector transaction timing because it would add nothing of significance to our explanation.

Figure 10 shows the data transfer section of a Unibus with two connections: a controller or CPU which will be the "master" in a data transfer, and a memory which will be the "slave". (For type C control and status information transfers, a controller plays the role of memory or "slave").

The timing of transfers on a Unibus is shown in Figure 11. BBSY ("Bus Busy") is the signal which, when asserted, indicates that the data transfer facilities are in use. Control and Address signals are a group which tell the "slave" connection what kind of transfer to do and from or to what memory address.

MSYN ("Master Sync") is asserted by the master (or requesting connection) to indicate that Control and Address signals are present.

Figure 1Ø. Unibus Data Transfer Synchronization

BBSY

MSYN

Address and Control

Data

Controller
or
CPU

Memory

BBSY

MSYN

SSYN

Address and Control	from sender	from receiver
Data	from sender	from sender

d set

s set

Figure 11. Timing Diagram of Unibus
Data-Out and Data-In Transactions

SSYN ("Slave Sync") is asserted by the slave connection to indicate that signals carry the data from sender (either the master or the slave) to the receiver.

A. Unibus Data-Out

This transfer causes data from the requesting connection to be written into memory.

The requesting connection must of course first receive permission from the arbitrator to use the data transfer facilities. Having received permission and acknowledged it by asserting Select Acknowledge (SACK), the connection waits for Bus Busy (BBSY) to be negated. It then asserts BBSY and negates SACK. This connection now "owns" the data transfer section of the Unibus.

Next, it must wait for SSYN to be negated, to prevent its own logic from mistakenly sensing SSYN in the asserted state too early.

Next, the master connection asserts the Address and Control signals and the Data. It then waits for fixed interval. This interval is known as the "deskew time," because it compensates for the variable delay in transmission of a signal from one connection to another.

This deskew interval is necessary here because in the worst case, the delay on the MSYN signal could be very small while the delay on the Address and Control signals could be large. Without the deskew interval, the slave could sense Address and Control signals which are indeterminate. An additional "set-up" time is inserted to allow all slave connections time to sense and match against the Address and Control signals.

The slave connection senses the Address and Control signals at all times. In this case, the address being transmitted by the master matches one of the memory addresses "owned" by this slave connection. Therefore, this slave responds to the transition of the MSYN signal from negated to asserted. In particular, since the Control signals indicate a "Data-out" transaction, the slave senses and stores the signals on the Data lines as soon as it receives the MSYN assertion.

Having captured the data, the slave asserts the SSYN signal. When the master receives the transition of SSYN from negated to asserted, it knows that the data transfer has been completed.

The master then negates (or disables) the Address and Control and Data signals, and it negates MSYN. It also negates BBSY at this time.

B. Unibus Data-In

This transfer causes a data word from memory to be read and transmitted to the requesting connection.

Having obtained permission to use the data transfer section, the master (which will be the receiver of data in this case) waits for BBSY to be negated. It then asserts BBSY and negates SACK as it did for a Data Out transaction. It is now the "owner" of the data transfer section.

The master connection next waits for SSYN to be negated, as before.

Next, the master asserts the Address and Control signals, but not the Data signals. Data will come from the slave connection later. The master waits the deskew interval, plus the extra "set-up" delay time and then asserts MSYN.

The slave connection has detected a match of its "owned" memory addresses with the address on the Address and Control lines, as before. When it receives the assertion of MSYN, the slave connection begins a read cycle to access the required word of data. When the data word is ready, the slave asserts the data on the Data lines and also asserts the SSYN signal.

When the master receives the transition of SSYN from negated to asserted, it begins waiting the deskew interval. This deskew wait is required to compensate for the variable delay in transmission of the data signals and the SSYN signals from the slave connection.

After the deskew interval, the master senses and stores the data on the Data lines. It then negates (disables) the Address and Control signals and negates the MSYN signal. It also negates BBSY at this time.

When the slave senses the transition of MSYN from asserted to negated, it negates (disables) the Data lines and negates the SSYN signal. This completes the Data-In transaction.

Data transfer on the Unibus is aperiodic --- there is no clock. Synchronization occurs by a "handshake" interaction between the MSYN and SSYN signals. In fact, two round-trips of signalling occur. We could look at this signalling in tabular form as follows:

	Data-Out	Data-In
MSYN assertion	Address & Control asserted and Data asserted	Address & Control asserted
SSYN assertion	Data captured (by slave)	Data asserted
MSYN negation	Negate Data and and BBSY	Data captured (by master) negate BBSY
SSYN negation	----	Negate Data

The sequence of four events insures a fully "interlocked" data transfer. The timing of a transfer is variable, depending on the speed of the slave's memory (for Data-In) and on the speed of the logic at both connections. On the Unibus, 75 ns is allowed for deskew and an additional 75 ns for set-up, where noted.

Figure 11.2 Q-bus Data Transfer Synchronization

Figure 11.4 Q-bus Data-In and Data-Out Synchronization

Figure 11.6 Q-bus Data-In-Out Synchronization

Figure 11.8 Q-bus Interruption Transaction Synchronization

Example 2: Q-bus

Data transfers on the Q-bus are also of types A and B. The synchronization of the two types of transfer are almost exactly the same. Below we describe the CPU-memory (type A) transfers.

Figure 11.2 shows the signals involved in Q-bus data transfers between CPU and memory. The CPU initiates all data transfers of this type. Type C (control and status) transfers are also made using the synchronization described next, with a controller playing the part of "memory" in the transfer.

Figures 11.4 and 11.6 show the timing of data transfers on the Q-bus. The 16 DAL signals are used to transmit address and then data, sequentially. SYNC is the signal which tells all memory devices on the Q-bus to examine the DAL lines and to test for a matching address. DIN and DOUT initiate the memory read and memory write cycles, for Q-bus Data-In and Data-Out transfers, respectively. RPLY, which is similar to the Unibus SSYN signal, indicates the presence of a response from the memory.

A. Q-bus Data-In

This transfer causes data to be read from the memory and sent to the CPU. The CPU must first wait until both SYNC and RPLY have been negated, to be sure that no other transfer is in progress on the Q-bus.

The CPU asserts the memory address on the DAL lines. After waiting for a fixed interval, to allow for deskew and set-up at the memory, the CPU asserts SYNC.

The memory senses the DAL lines when it receives the assertion of SYNC. The memory matches the address received and finds that it contains the data word being addressed.

After another fixed delay, to guarantee that the SYNC assertion always arrives at the memory first, the CPU asserts DIN and negates the address from the DAL lines.

As soon as the memory receives the DIN assertion, it knows that a read cycle is denied. It retrieves the data word and asserts it on the DAL lines. Meanwhile, it may assert the RPLY signal as much as 125 ns before asserting the data.

When the CPU receives the RPLY assertion, it waits at least 2ØØ ns to be sure that the data is present on the DAL lines, and then senses and stores the data. Then the CPU negates DIN.

As soon as the memory receives the DIN negation, it negates RPLY. Not more than 1ØØ ns later, the memory negates the data from the DAL lines.

When the CPU receives the negation of RPLY, it negates SYNC. The bus is now available for the next data transfer.

B. Q-bus Data-Out

This transfer causes data from the CPU to be sent into memory.

The CPU waits until both SYNC and RPLY have been negated. Then it asserts the memory address on the DAL lines. After a fixed delay, the CPU asserts SYNC.

The memory receives the assertion of SYNC, senses the DAL lines, and matches the address.

The CPU waits at least 100 ns after it asserts SYNC. Then it negates the address and asserts the data word on the DAL lines. After another delay of at least 100 ns, the CPU asserts DOUT.

When the memory receives the assertion of DOUT, it assert RPLY to indicate that it will accept the data. The memory now begins to perform a write cycle.

When the CPU receives the assertion of RPLY, it waits 150 ns and then negates DOUT.

When the memory receives the negation of DOUT, it senses and stores the data, and negates RPLY.

The CPU keeps the data asserted on the DAL lines for at least 100 ns after it has negated DOUT. Then, after negating the data and waiting at least another 75 ns, the CPU negates SYNC. This completes the Data-Out transfer.

C. Q-bus Data-In-Out

Figure 11.6 shows the timing of another type of Q-bus data transfer, the "Data-In-Out" operation. In this transfer, a data word is read from an address in memory, sent to the CPU, and then a word is sent back to memory, to be stored at the same address. This operation is useful for certain PDP-11 CPU instructions such as INC (Increment Memory), which modify a single word in memory, and two-operand instructions much as ADD, which store a result at the address of the second operand.

Note that the Data-In-Out operation saves bus transmission time by not requiring the address to be sent a second time for the Data-Out portion of the cycle.

The Data-In-Out operation proceeds exactly the same as the Data-In transfer until the memory negates the RPLY signal. Not more than 100 ns later (as for Data-In), the memory negates the data from the DAL lines.

When the CPU receives the negation of RPLY, instead of negating SYNC, it keeps SYNC asserted and begins using the data. When the CPU has generated the data word which is to be stored into memory, it asserts it on the DAL lines. The CPU waits at least 1ØØ ns, and then asserts DOUT.

The remainder of the Data-In-Out operation proceeds the same as the Data-Out transfer.

This operation and the Data-In and Data-Out transfers all may all be performed by DMA controllers on the Q-bus. When performed by a controller, the timing differs only in that the SACK signal, asserted by the controller during arbitration, is negated at the time shown above for SYNC negation. The SYNC negation is then delayed an additional 3ØØ ns, to guarantee that the CPU does not interfere with the DMA data transfer.

D. Q-bus Interruption Transaction

Figure 11.8 shows the timing of the interruption transaction on the Q-bus. This transaction includes both arbitration and the transfer of a data word (an interrupt vector) from a controller to the CPU.

The IRQ (Interruption Request) signal is asserted by a controller when it wants to interrupt the CPU. This signal is similar to the Unibus BR signals. But the Q-bus has only one priority group for interruptions, and the whole group shares the IRQ line.

The IAK (Interruption Acknowledge) signal is similar to the Unibus BG signals. IAK is wired from the CPU (arbitrator) serially through all controllers, just as each BR signal is wired serially through the corresponding priority group on the Unibus.

A controller may assert IRQ at any time. When the CPU is ready to receive an interrupt vector, it begins a sequence which roughly resembles a Data-In transfer. However, the SYNC signal is not used and no address is sent out on the DAL lines.

The CPU first asserts DIN. After a delay of at least 150 ns, it asserts IAK.

The IAK assertion cascades through the controllers which are not requesting an interruption until it arrives at a requesting controller, which blocks the passage of the IAK assertion. (See Fig. for the equivalent BG circuit in a Unibus controller.) When the requesting controller receives the DIN assertion followed by the IAK assertion, it negates IRQ and asserts RPLY. Not more than 125 ns later, the controller asserts the interrupt vector on the DAL lines.

When the CPU receives the RPLY assertion, it negates DIN and IAK.

When the controller receives the DIN negation, it negates RPLY. The controller continues to assert the interrupt vector on the DAL lines for at least 100 ns.

When the CPU receives the RPLY negation, it senses and stores the interrupt vector. The CPU then uses the vector to enter an interrupt service routine.

Example 3: Synchronous Backplane Interconnect (SBI)

There is only one sequence of events which causes information transfers on the SBI, and that sequence is quite simple. However, the meaning of the information transferred from one connection to another has two interpretations: Command and Address, or Data. A memory read or write operation always consists of two sequences, one to transfer a command to the memory connection, the other to transfer data.

Figure 12 shows a simplified SBI data transfer section. The ID signals are used to identify the destination of the transfer when the information transferred is data. The other use of the ID signals is explained below.

The DATA lines carry 32 bits of information. This information is either (a) 32 bits of data, or (b) 28 bits of address and 4 bits of command code. The FLAG signal is asserted to indicate case (b), or negated to indicate case (a). In case (b), the destination of the transfer is determined by the 28 address bits, in a way similar to the Unibus addressing method. For these transfers, the ID lines carry the identity of the source of the transfer. The connection receiving a Read command saves this identity, to know where to send the data later.

Figure 12. Simplified SBI Data Transfer Section

ID (2)

DATA (32)

FLAG

Clock

	Controller 1	Memory 2

TRØ

TR1

TR2

ID	ID=1				ID=1			
DATA	address from 1				data from 2			
FLAG	"control and address"				"data"			
Clock								
Time Slots	1	2	3	4	5	6	7	8

Figure 13

Two SBI Transactions Which Make Up a Memory Read Operation

A. SBI Read

Figure 13 shows the timing of the two SBI information transfers which make up a read operation from memory. Remember that there is a master clock which defines a series of time slots. The TR (Transfer Request) lines are shown again to recall the fixed time sequence of arbitration, which always immediately preceeds a transfer.

In Figure 13, the controller, (connection 1) decides at the beginning of slot 1 to initiate a memory read. It obtains permission to use the data section at the end of slot 1, because no higher priority TR lines are asserted. In slot 2 it transmits the following bits:

ID = 1, the identity of the source connection

DATA = "Read" command code, plus 28 bits of memory address

FLAG = asserted, indicating the format of DATA above (command, address)

At the end of slot 2, the memory connection senses all of these bits, and captures them in a buffer register.

In fact, every connection on the SBI captures all of these bits on every slot. Subsequently, each connection matches the ID bits (if FLAG is negated) or the address (if FLAG is asserted) against its own identity or "owned" memory addresses. If there is a match, the connection accepts the data or begins performing the operation indicated by the command code.

In this case, the memory connection detects a match of the address with a section of memory contained in itself, and it therefore begins a read cycle.

Memory subsystems know what their own cycle times are. Therefore, the memory connection in this read operation example can assert its TR signal (TR2) one slot before it is ready to transmit data. If there is no higher priority TR line asserted, the memory transmits its data to the requesting controller in the next slot. In this example, the memory transmits the following bits in slot 7:

ID = 1, the identity of the destination connection

DATA = 32 bits of data from memory

FLAG = negated, indicating the format of DATA above

Levy, "Buses, the Skeleton of Computer Structures"
1/27/78

Figure 14. SBI Transactions
Which Make Up a Memory Write Operation

TR2	asserted by 1				
TR1					
TRØ					
ID	ID = 1				
DATA	"Write" & Address			DATA	
FLAG	"Command & Address"			"DATA"	
Clock					
Time Slots	slot 9	1Ø	11	12	13

At the end of slot 7, all connections to the SBI capture this information, and controller 1 recognizes the match between the ID bits and its own identity. A memory read has now been finished.

On the SBI, a memory may take a variable number of slots before replying to a Read command. Clearly there is a performance penalty for memories which require slightly more than an integral number of slot-times to access a word. They must wait until the next slot to transfer their data. As a result, the SBI clock is "tuned" to be an integral submultiple of the access time of the memory sub-system we intend to use. However, we could attach a variety of memory subsystems with different access times to one SBI, without serious performance degradation, as long as the memory access times are sufficiently large multiples of the slot-time.

The VAX-11/780 system uses a slot-time of 200 ns and has, at first delivery, a memory subsystem access time of just under 800 ns (including error detection). Therefore, the four-slot access time shown in Figure 13 is typical of this system.

B. SBI Write

Figure 14 shows the timing of a memory write operation. The controller, connection 1, obtains use of the data section by asserting TR1 during slot 9. It then transmits on the next two consecutive slots. In the first slot (slot 1Ø), FLAG is asserted to indicate that command and address information is present. The command is "Write". In slot 11, the data to be written is transmitted. The memory connection must be prepared to accept and capture the sequence of two transmissions. It begins writing the memory at the end of slot 11.

The ID lines contain the identification of the controller during slot 1Ø. In slot 11, the ID lines are not significant. (On VAX-11/78Ø, the FLAG signal is replaced by a multiple-bit field which indicates the nature of the data transmission. The ID lines are used during "write data" slots to send the identity of the source of data. The memory subsystem can then verify that the data in fact comes from the same source that sent the command in the previous slot. Without this check, some failure modes could cause the wrong data to be written.)

In summary, the SBI synchronizes data transfers by using a master clock to define time-slots. Transmission of command/address information and data is accomplished in exactly the same way, using the same lines. This requires two slots to be used for each memory read and write. The read operation is split, allowing other transactions to take place while a memory connection is accessing data. The two-slot write operation is kept contiguous by using the highest-priority TRØ "hold" signal to obtain use of the second slot. The SBI minimizes the slot time by eliminating all round-trip delays (i.e., the delay of transmitting a signal from one connection to another and then transmitting a response back, such as the MSYN-SSYN sequence on the Unibus). This maximizes the data transmission bandwidth of the SBI.

Example 4: Polled Character-Input Bus

Data transfer on this bus was described in the section on arbritration methods. Refer to Figure 8.

Data transfer occurs in time-slots just as on the SBI. The time-slots are defined by a master clock, and the receiver (always the controller) must accept the data at the end of the time slot. In contrast to the SBI, this polled character-input bus pre-allocates one of every four slots to each keyboard connection. The controller must keep internally an indication of which character was received from which keyboard.

Example 5 (a): Massbus Control Section

The Massbus actually consists of two sections, a Control Section for reading and writing the contents of registers in the peripherals, and a Data Section for moving blocks of data. All transfers are between the controller and one of the (up to eight) peripherals. Figure 15 shows a simple schematic diagram of the Massbus configuration. The two sections operate independently, except that a Control Section write into a control register of a peripheral is required to initiate a block transfer on the Data Section.

The Control Section of the Massbus is a miniature Unibus. However, the controller is always the master, and one of the peripherals is always the slave in the transfer. Figure 16 shows the Control Section signals involved in data (i.e., control) transfers. The Demand (DEM) signal takes the place of MSYN, and Transfer (TRA) takes the place of SSYN. Instead of Address and Control lines, there is an eight-bit address on the Massbus Control Section: three bits of Drive Select (DS), and five bits of Register Select (RS). Thus, each of eight peripherals (drives) may contain up to 32 one-word registers. There is also a "Controller to Drive" (CTOD) signal, which, when asserted, indicates that the transfer is a write into a peripheral register. Otherwise (CTOD negated), the transfer is a read from the peripheral.

Figure 15

Massbus

Controller

Control Section

Data Section

Peripheral
Ø

Peripheral
1

Figure 16

Massbus Control Section (simplified)

Controller

DEM

TRA

DS(3)

RS(5)

C(16)

CTOD

ATTN

Peripheral Ø

Peripheral 1

Control information is transferred 16 bits at a time on the C lines. Timing of these transfers is equivalent to that shown for the Unibus in Figure 11. The correspondence between Unibus signals and Massbus Control Section signals is summarized below:

Unibus Signal	Massbus Control Section Signal
BBSY	none (there is no arbitration)
MSYN	DEM
SSYN	TRA
Address & Control	DS, RS, and CTOD
Data	C

There is also an Attention (ATTN) signal in the Control Section. It is a shared line which may be asserted by any peripheral which requires CPU intervention. It may be asserted at any time. The controller normally creates an interruption to the CPU soon after ATTN is asserted.

Timing of normal Read transfers is shown in Figure 16b. It is equivalent to a Unibus Data-In transfer, (cf, Fig. 11, second part).

There is one special case which uses different timing on the Massbus Control Section. In order to determine which of the peripherals has caused an Attention interruption, the CPU reads the Attention Summary pseudo-register via the controller. This is a special "register" which is composed of one bit stored in each peripheral. When the RS lines carry a code of 04 and the direction of transfer is drive to controller (CTOD negated), each peripheral (drive) asserts its ATA (Attention Active) bit onto one signal of the Control (C) lines. Peripheral number 0 asserts its ATA onto C0, peripheral 1 onto C1, and so on. Control lines 8 through 15 always remain negated, since there are never more than 8 peripherals on the Massbus. The word received at the controller therefore looks like this:

C15		C8	C7	C0
0	0	0	ATA7	ATA0

The timing of this transfer is different because the TRA signal is asserted by more than one peripheral. Since there is no way of knowing when all peripherals have asserted their ATA bits, the controller must wait the maximum possible access time, and then complete the control read. It ignores the TRA signal entirely.

Figure 16b. Timing of Massbus

Control Section Control Read (normal case)

Unibus Equivalent	Massbus Signal
[control]	CTOD
[address]	DS and RS
[MSYN]	DEM
[SSYN]	TRA
[DATA]	C

This maximum delay "time-out" is present in the controller logic anyway, to guard against possible non-response from an addressed peripheral or register. (The next section will discuss such error control methods in more detail). The Attention Summary read operation makes use of this time-out interval to terminate its wait for the ATA bits. The timing of this transfer is shown in Figure 17. In order to make this transfer effective, each drive must be designed to beat the time-out interval when it gates out its ATA bit. (It must also beat this interval in normal reads. See the next section).

Example 5 (b): Massbus Data Section

The Massbus Data Section contains 18 Data (D) lines, which carry data in both directions. Two clock signal lines SCLK (Synchronizing Clock) and WCLK (Write Clock) carry a clock from and back to the peripheral, respectively. The RUN and EBL (End-of-Block) signals control the termination of a block data transfer. The EXC (Exception) signal is used to indicate error conditions.

Massbus Data Read

Data in the Massbus Data Section is always transferred in multiple-word blocks. The data read from or written to a mass storage device, such as a disk drive, must be synchronized with the mechanical positioning of the recording medium. Therefore, the clock used to synchronize these data transfers (SCLK) originates in the peripheral.

A Massbus Data Read begins when a control register in the selected peripheral is written with a "Read" command code. The peripheral then knows it is to begin examining the Data Section signals.

Figure 19 shows the timing of a Massbus Data Read. The controller asserts the RUN signal as soon as it is ready to receive data.

When the peripheral has received the RUN assertion, it begins reading data from its storage medium. When the first data word is ready, the peripheral asserts the word on the D lines, and it asserts SCLK. The assertion of SCLK indicates that a new data word is present on the D lines.

Mass storage peripherals have a characteristic interval time between data words. The peripheral doing a Massbus Data Read will assert and negate the SCLK signal at a cycle rate equal to the characteristic data rate.

Approximately half an interval time after asserting SCLK, the peripheral negates it. When the controller receives the negation of SCLK, it samples and stores the data word from the D lines.

At the next and succeeding data word intervals, the peripheral asserts the new data on the D lines and asserts SCLK. The controller samples and stores each successive data word at the negation of SCLK. Note that the peripheral does not receive any positive indication that the data word was received by the controller: the data transfer is "open loop".

At the end of the block of data words, the peripheral stops asserting SCLK and asserts EBL to indicate that it has reached the end of the data block. At this point, the peripheral is prepared to continue by transmitting the next data block or to shut down the data transmission.

When the controller receives the EBL assertion, it decides whether to continue (usually by inspecting a word count register). Within slightly over one microsecond, the controller must negate RUN or decide to accept another block of data.

The peripheral negates EBL not less than 1.5 microseconds after asserting it. As it negates EBL, it senses the RUN signal. If it is negated (as shown in Fig. 19), the peripheral disconnects itself from the Massbus Data Section. Otherwise, the peripheral would repeat the sequence for another block of data.

It is possible that the number of words desired by the controller is less than an integral number of data blocks. In this case, the controller may negate RUN long before EBL is asserted. The controller then simply ignores the data words being transmitted. However, the controller must wait until the assertion and negation of EBL is complete before it may initiate another data transfer.

Massbus Data Write

Figure 20 shows the timing of a Massbus Data Write. As for a data read, the peripheral controls the rate at which data is transmitted. However, this time the data is coming from the controller. The WCLK signal is used to return the SCLK signal back to the peripheral at the same time that data is made available on the D lines.

The transfer begins when a control register in the peripheral is written with a "Write" command code. The controller then asserts the fist data word on the D lines and asserts the RUN signal.

When the peripheral receives the RUN assertion, it prepares to write data onto the storage medium. The peripheral asserts SCLK.

When the SCLK assertion arrives at the controller WCLK is asserted.

When the peripheral receives the WCLK assertion, it senses the data on the D lines and begins writing.

Thereafter, the peripheral negates and asserts SCLK at the characteristic word rate. Each time the controller receives the SCLK negation, it asserts the next data word on the D lines and negates WCLK. The peripheral senses the D lines each time it receives the assertion of WCLK.

The controller must have a data word ready each time SCLK is negated. If it does not have one ready, the peripheral will sense whatever data value is on the D lines at the next WCLK assertion, and this erroneous data will be written onto the storage medium. This is a "data overrun" condition which should be detected by the controller.

Figure 17. Timing of Control Read From Attention Summary Pseudo-Register (Massbus Control Section, Register No. = 04

C0

C1

C2

C7

C8

.

.

.

C15

DEM

TRA

CTOD

C0

C1

C2

C7

C8

Levy, "Buses, the Skeleton of Computer Structures"
1/27/78

.

.

.

C15

DEM

TRA

CTOD

RS(5)

DS(3)

Figure 18. Massbus Data Section (simplified)

Controller

SCLK

WCLK

D(18)

EBL

RUN

EXC

Peripheral

Figure 19. Timing of Massbus Data Read

RUN

EBL

EXC

SCLK

WCLK

D word 1 word 2 word 3 word 4

Figure 20. Timing of Massbus Data Write

RUN

EBL

EXC

SCLK

WCLK

D word 1 word 2 word 3 word 4

IV. Error Control Strategies

We have so far assumed that data transfers always succeed in delivering to the receiving connection what was transmitted from the sending connection. Unfortunately, this is not always the case. There are many ways in which data transfers can fail to deliver the bits.

Many other kinds of failures are possible, too. We restrict our discussion in this section to techniques which apply to errors in data transfers. Causes of errors are not examined here, only countermeasures. Causes of errors include logic failures, electromagnetic interference, broken conductors, shorted conductors, and power failures.

The countermeasures used against data transmission errors are in the following five categories:

1. Check-bits -- Extra information is sent which allows the receiver to detect and sometimes to correct errors in the data.

2. Acknowledgement -- A reply from the receiver to the sender tells whether the data (presumably with check bits) appeared "good".

3. Time-out -- Failure of an expected acknowledgement to be received by the sender within a time limit indicates unsuccessful data transmission.

4. Retry -- A transfer which was unsuccessful is attempted one or more additional times.

5. Error Reporting and Logging -- Failures of all categories above are recorded and reported to higher level (usually software) logic, which may attempt additional retries or other countermeasures. "Logging" means recording the history of errors in a file which can later be read by a repair engineer.

We should now revise our procedure (given in Section II) for a connection to do data transfer. The more general procedure is the following:

1. (Arbitration) Obtain the use of the bus.
2. (Data Transfer) Transfer data on the bus.
3. (Check) Check for error-free transfer, and receive or send an acknowledgement.

4. (Retry) If check or acknowledgement failed, repeat steps 1 through 3.

5. (Log) If all retires fail, enter a failure report in the log file, or send a message to higher level software routines.

The example buses described in the previous sections use a variety of error control methods. Table 21 summarizes their use.

Example 1: Unibus

Data transfer on the Unibus is not checked. However, there are two lines reserved for use by memory connections, to signal whether a parity error has been detected on reading a word from memory. Memory parity is generated by the memory connection as each word is written into memory.

The Unibus does have a time-out, for protection against non-existent connections. This means that if a CPU or controller attempts a data transfer to or from a memory address which no connection responds to, the CPU or controller gives up after a fixed time. On the Unibus, this time is set at 5 microseconds after MSYN has been asserted by the master. Assertion of SSYN must be received before this time-out or the master will assume there was no response.

The "non-existent memory" time-out error always causes serious consequences. It usually indicates an erroneous memory mapping (from the CPU) or an invalid DMA address (from a controller). In either case the current stream of activity is abandoned.

Table 21. Error Control Methods Used By Example Buses

Bus	Check Bits			Ack	Time-out	Retry	Log
	parity	CRC	ECC				
1. Unibus				X(SSYN)	X	Note 1	Note 2
2. Q-bus				X(RPLY)	X	Note 2	Note 2
3. SBI	X			X(CNF)	X	X	Note 2
4. Polled character input	(X)						
5a. Massbus Control	X			X(TRA)	X	Note 1	Note 2
5b. Massbus Data	X			X($\overline{\text{EXC}}$)	X	Note 1	Note 2

Note 1: Retry is implemented by software in some PDP-11 operating systems.

Note 2: Logging is implemented at various levels by operating system software.

Example 2: Q-bus

The Q-bus does not have check bits for data transfers. However, it has two lines (DAL 17 and 16) which can be used for transmitting the results of memory parity error checking.

The Q-bus also has time-outs specified for responses to the assertion of DIN and DOUT. If a memory does not respond with the assertion of RPLY within 10 microseconds, the CPU or controller will asume that no memory matched the address. The consequences are the same as on the Unibus.

Example 3: SBI

Data transfers on the SBI carry several parity-check bits. Parity is generated at the sending connection and is checked at the receiving connection.

The SBI also does acknowledgement on every data transfer. A code is returned to the sending connection two time-slots after the data was sent. Separate Confirm (CNF) lines are used to carry this code. The code indicates one of four possible events:

1. No response --- there is no connection responding to this address or ID value.

2. Parity error --- the parity check shows an error in transmission; transfer is rejected.

3. Busy --- (for commands only) the memory connection addressed cannot accept another command now; transfer is rejected.

4. Accepted --- parity checks "good" and the command or data is accepted.

The Confirm code itself is error-protected by using a distance-2 coding scheme. Of course, the "No Response" code is with all CNF signals negated, because there is no responding connection to assert them. The other codes have CNF values which differ from each other and from the No Response code by having different values in at least two bit positions. Therefore, if one CNF bit is inverted the resulting CNF code will not appear to be valid.

Figure 22 shows the timing of SBI data transfer acknowledgements. The example in this figure is a data word transfer from memory to a connection which previously requested a read (the second half of a read operation).

In time-slot 1, the memory (connection 1) asserts TR1 to obtain use of the data transfer section. Since there is no higher priority TR signal asserted, it transmits a data word, with parity check bits, during slot 2. The ID field carries the identification of the destination connection (controller 2, in this case). At the end of slot 2, all connections have captured the data and parity check bits. Controller 2 matches its identity with the ID field and prepares to use the data. During slot 3, it checks to see that the parity is correct. At the beginning of slot 4, the controller asserts the Confirm code on the CNF lines. The CNF lines are always reserved for a reply from a receiving connection exactly two slots after a data transfer. At the end of slot 4, the sending connection (memory 1) receives the confirm code.

Figure 22. Timing Of Parity Check And Acknowledgement On SBI

Clock								
Time slot	1	2	3	4	5	6	7	8
TR1	(1)							
ID		ID = 2						
Data		from 1						
FLAG								
Confirm				from 2				
parity checking			by 2					
			data received by 2		confirmation received by 1			

Actually, all connections check the parity of every data transfer on the SBI. Since each connection must capture the data and ID bits to match against, it is simple for each one to put the data through a parity check network. The error control philosophy on the SBI says that if any connection detects bad parity on a data transfer, then the validity of the data transfer is suspect. Therefore, any connection may assert a "parity error" code at the beginning of slot 4 in Figure 22.

In order to prevent the occurrence of "bad parity" indications after unused data transfer slots, "even" parity (sum of bits = 0 modulo 2) is used on the SBI.

As implemented in the VAX-11/780, the SBI also uses time-outs and retries. Time-out applies only to memory read requests. If the memory does not respond within a fixed number of slots, the CPU or controller causes an interruption, possibly leading to retry or logging of the event. The VAX-11/780 CPU also does microprogram-controlled retry of transfer requests which received the Busy confirm code.

Example 4: Polled Character-Input Bus

Since we made up this example, rather than describing an existing polled bus, we cannot claim to explain its actual error-control methods. It would be reasonable, however, to add one data signal which would carry a parity check bit for each character. A time-out would not be relevant here, but an acknowledgement with retry could be implemented by having a confirm signal transmitted back to the originator of a character during the slot following the data transfer. (Refer to Figure 23.) If the confirm signal did not indicate "good transfer", the keyboard could hold the last character and attempt to send the character again 4 slots later (when its turn comes around again). The parity is checked during the latter part of the data transmission, by gating the incoming character through a parity-check network at the controller.

Example 5a: Massbus Control Section

The Massbus Control Section closely resembles the Unibus in timing, but it does carry one data parity check signal. On reading a control register, the controller will pass on the "bad parity" indication. (In the case of passing the information to the Unibus, the controller passes

Figure 23. Timing of a Plausible Error-Checking Scheme With Acknowledgement And Retry For Polled Character-Input Bus

0 1 2 3 0 1 2 3 0 1 2 3

UNIT 0

UNIT 1

clock

slot no.

Data &
Parity

send

parity
checking

confirm

a "memory parity-error" indication). On writing a control register, a parity error will first fail to modify the contents of the register and second, the peripheral will assert the Attention signal, with an internal "Control Bus Parity Error" indication recorded in an error status register.

The Massbus Control Section also has the same acknowledgement (using TRA) and time-out properties as the Unibus, with the exception of the reading Attention Summary pseudo-register (see Section III). Reading this register always uses the time-out to terminate the read cycle, so the controller will not detect which if any peripherals are responding. However, the default state of signals on the Massbus is "negated", so there should be no false indications of attention status.

Example 5b: Massbus Data Section

The Massbus Data Section also carries a parity check bit with each 18-bit word of data. Acknowledgement of "good" data received is done by not indicating "bad" data. A signal called Exception (EXC) can be asserted from either end to indicate a bad data transfer, or other exceptional conditions which will prevent further successful data transfers. Figure 24 shows an example Massbus Data Write operation which suffers a parity error during the transmission of the second word. The peripheral asserts the EXC signal as soon as the error is detected. Although this is too late to stop the next word from being transmitted, the peripheral does cease to accept data words, and it terminates the block transfer early by asserting the end-of-block (EBL) signal immediately. The controller acknowledges the termination by negating the RUN signal.

Block transfers to disk or tape are not useful if even one word is erroneous, so it is not important to finish the block transfer or to recover the words already written. The entire block will have to be retransmitted. In this example, the controller will display a "Transfer Error" when it interrupts the CPU for "end-of-transfer" service. The word count in the controller will show the approximate number of words written (off by one from the actual number, in this example).

Figure 24. Massbus Data Section
Timing of Exception Signal While Writing

RUN

SCLK

WCLK

EBL

EXC

Data word 1 word 2 word 3

parity error
detected on
word 2

Two time-outs are used on the Massbus Data Section, both in the controller. One starts at the transition of RUN from negated to asserted. If the SCLK signal does not make a transition within 7 seconds, the controller shuts down the attempted transfer. (This lengthy time is required to permit magnetic tapes to transport a maximum of 25 feet of inter-record gap before encountering a data record, following ANSI standards for inter-record gaps.)

A shorter time-out, approximately 100 microseconds, is used to detect a failure in a peripheral after at least one signal transition has been received on the SCLK line. If this limit is reached, the <u>controller</u> asserts EXC to tell the peripheral to disconnect. The controller then shuts down the data transfer operation.

V. Evolution of the High-Performance PDP-11 Systems, A Bus Design History of PDP-11/20, -11/45, -11/70, and VAX-11/780.

The PDP-11 was first introduced in 1970. It was a radical departure from prior minicomputer designs, having features such as automatic stack-indexing instructions and the "I/O page" concept which eliminated the need for special I/O control instructions. [See Chapter 7] In terms of bus structures, the Unibus introduced with the PDP-11/20 was also novel, because it was a single bus to which all system components were attached, it could be extended indefinitely, and it did not require memory modules to operate synchronously with the rest of the system.

In this section we trace the evolution of the high performance descendents of the PDP-11/20, with emphasis on the development of buses in response to design goals for each model.

Table 30 summarizes the design goals and related bus developments. Figures 25, 26, 27, and 28 show the bus configurations of the four systems discussed here.

PDP-11/20 (See Figure 25)

The Unibus design is integral with the PDP-11 architecture in the handling of interrupts (the priority level of the CPU affects arbitration) and in the I/O page concept (control registers appear as memory locations). But the important aspect of Unibus design, as a bus, is its support of modularity.

Digital had long been in the business of supplying standardized modules for interfacing to a variety of electronic equipment. When the PDP-11/20 was designed, it was natural to offer a bus which could be interfaced to many types of equipment, including custom laboratory electronic devices. With the introduction of the PDP-11/20, Digital offered Unibus interfacing modules (such as the DR11 series) which users of the PDP-11 could easily adapt to their own equipment.

The standardization of interfacing was also a deliberate attempt to allow long service lives of Digital's peripheral equipment. As new members of the PDP-11 family were introduced, older peripherals could still be attached to the Unibus without electrical modifications.

The asynchrouous data transfer of the Unibus allowed Digital to introduce a series of memory subsystems with successively increasing speeds without changing the Unibus timing or data transfer protocol. In a single system, various memory technologies could be intermixed.

The Unibus, as a standardized boundary between CPU, memory, and peripherals, also facilitated the division of engineering design responsibilities within the company's development department. A memory subsystem designer, for example, could make any modification which would add to memory performance, as long as the subsystem interface conformed to the Unibus data transfer rules.

PDP-11/45 (See Figure 26)

Driven by the availability of a higher-speed logic family (Schottky TTL), the first goal of the PDP-11/45 design was a faster CPU. The success of the result is attested by the fact that the logic design of the PDP-11/45 remained competitive for over five years (and it was used virtually intact in the PDP-11/70).

In addition, semi-conductor memory became available at a cost which justified its inclusion, in small quantities, in minicomputer systems. In an effort to match the speed of the new PDP-11/45 CPU, an optional semiconductor memory subsystem could be included in the CPU cabinet. This subsystem had its own private pathway to the CPU, called the "Fastbus". Placing the semiconductor memory in close proximity to the CPU allowed elimination of many of the access delays present when a Unibus was between the CPU and memory. For compatibility, however, it was necessary for the semiconductor memory to be accessible to DMA transfers from outside the CPU. For this reason another Unibus was brought out of the CPU cabinet, as we shall see below.

With higher CPU speed came the need for larger memory sizes. While the PDP-11/20 could have up to 64K bytes of memory (less 8K bytes reserved for the "I/O page"), the PDP-11/45 introduced a memory segmentation module (the KT11) which allowed addressing of up to 256K bytes. The Unibus, with foresight, had been implemented with two spare address lines, allowing immediate use of the 18 bits of physical memory address from the PDP-11/45.

The IBM 3330 disk technology (100 megabytes per spindle) had by 1973 become available at a cost attractive to minicomputer system users. The Massbus was developed specifically to interface this and other high-data-rate devices which were planned. The RH11 controller connected the Massbus to the two Unibuses of PDP-11/45 systems as shown in Figure 26. The upper Unibus, Unibus A, was to carry the control and interruption (type C) transactions, and the lower Unibus, Unibus B, was reserved exclusively for DMA (type B) data transfers. For this purpose, a special stand-alone Unibus arbitrator module was developed, because Unibus B had no CPU present to perform Unibus arbitration. (Note, however, that the BR signals are not used on Unibus B, because there is no CPU to be interrupted.)

Unfortunately, the configuration shown in Figure 26 could not be used, for two reasons:

1. DMA transfers from the RH11 controller could not reach memory modules attached to Unibus A if all block transfers were made on Unibus B. (The proposed solution of having the DMA path selected by program control was rejected because of the complexity of determining which memory was connected to which bus.)

2. DMA transfers from controllers on Unibus A could not reach the semiconductor memory unit.

The second problem was fatal. The CPU was capable of dealing with only one I/O page, and that was on Unibus A. Therefore, old DMA controllers had to be attached to Unibus A. In fact, all controllers had to attach to Unibus A, because that is the only interruption path. Since compatible use of old peripherals was essential to success of the family, the PDP-11/45 was configured only as shown in Figure 26b. Unibus B, when connected to Unibus A (and with its separate arbitrator module removed) becomes part of the single-Unibus system.

In the configuration shown in Figure 26b, the PDP-11/45 system was, for I/O programming, completely compatible with PDP-11/40 and other later Unibus-based PDP-11s (see Figure 29, Genealogy of PDP-11 family buses).

<u>PDP-11/70</u> (See Figure 27)

By 1974, semiconductor memory costs had become much lower. Therefore, a cache memory became a feasible performance enhancing addition to the PDP-11/45. (See Chapter 9.) Without great modification to the CPU logic, a cache memory was added, with a width of 32 bits, twice the word size of the PDP-11. The cache effectively interfaces to the PDP-11/70 CPU over the same "Fastbus" that was present in the PDP-11/45.

In order to gain memory bandwidth for increases in both CPU and DMA performance, a new "backing store" memory bus was added, with a 32-bit wide data path [Northrup, Levy, and Griggs patent, 1977)]. Closely related to the memory bus was a backplane interconnection to the RH70 controllers (up to four of them), which can carry 32 bits at a time. The RH70-to-memory path is shown as going through the cache because of a "look-aside" feature of the cache memory. (See Chapter 8.)

The Massbus had been designed to provide very high block transfer bandwidth, while keeping the control registers accessible to the CPU at all times. The successful splitting of control path (type C) and data path (type B) in the PDP-11/70 matched well with the Massbus design goals, and this match accounts in part for the relatively long life of the PDP-11/70 system in its marketplace.

The PDP-11/70 also required more memory addressing capacity to balance its increased speed. The KT11 segmentation unit was easily expanded to address four Megabytes of memory, and the RH70 controllers were designed to directly generate the required 22 bits of memory address. However, the Unibus was already at its limit of 18 bits of physical address.

Slower-speed peripherals were still to be interfaced to the Unibus, and in doing DMA transfers from them, it is necessary to transform the 18-bit address on the Unibus into a 22-bit main memory address. To do this, a Unibus Map module was inserted between the Unibus and the cache memory. This path required to carry 16 data bits at a time, and the bandwidth demands are relatively low.

VAX-11/780 (See Figure 28)

Late in 1977, Digital introduced the first of a new series of systems based on the PDP-11 architecture but with greatly enhanced performance capabilities. The VAX-11/780 returns to a single central bus concept, based on the SBI.

The SBI was originally conceived in 1974 for use on a PDP-11 family CPU. That CPU design was not completed, but the SBI was carried into the VAX-11/780 design and tailored for the 32-bit family environment.

High performance in the CPU is obtained by 32 bit data paths and use of a 64-bit wide cache memory. Overcoming the 16-bit virtual address limit of the PDP-11, the VAX-11 architecture is capable of 32 bits of virtual address. The VAX-11/780 system generates 28 bits of longword (4-byte) address, yielding a maximum memory size of one Gigabyte.

High DMA bandwidth is obtained by the SBI short time-slot and by the read-operation splitting which releases the bus during the memory read-access delay [patents by Levy, Rodgers]. To help overcome the delay associated with having to do a full bus transaction to start a memory read cycle, the memory control logic is capable of receiving and storing a queue of up to four memory read or write requests while it is working on one of the requests [VAX-11/780 patent by Durvasula].

Compatibility with existing PDP-11 peripherals is provided by controllers which adapt the SBI to a Unibus (the UBA in Figure 28) and to several Massbuses (MBA).

On the SBI, the one-Gigabyte address space is divided in half, with the Unibus I/O page concept being extended to cover the upper half. Within this rather large address space are contained control registers for all peripherals, the 18-bit memory address space of the Unibus, and a number of internal status and control registers, such as those that contain error reporting information.

PDP-11 Family Genealogy Based on Bus Structure

Figure 29 shows a genealogy of the PDP-11 family buses. Some of the PDP-11 models shown in Figure 29 were not discussed earlier in this chapter. Grouped by bus structure, the models fall into the following categories:

The Unibus PDP-11s: PDP-11/20, /40, /05, /34, /04

The Q-bus PDP-11s: PDP-11/03 and LSI-11

The PDP-11s with Unibus and an additional memory bus:
PDP-11/45, /55, /70, /60

The VAX-11 series (using SBI): VAX-11/780

The Massbus has been used on many models of PDP-11, DECsystem-10, and DECsystem-20 computers. Some of the Massbus controllers implemented for these computers are listed below.

Computer	Massbus Controller Name	Controller Connects From Massbus
Unibus PDP-11s and /45, /55, /60	RH11	Unibus
Q-bus PDP-11	not used	-----
PDP-11/70	RH70	Internal 32-bit memory bus
VAX-11/780	MBA	SBI
DECsystem-10	RH10	Internal I/O channel bus
DECsystem-20	RH20	Internal CBUS and EBUS

J. Levy - ch. ___

Bibliography

draft 18 Jan 78

1. Bartee, Digital Computer Fundamentals (Fourth Edition), chapter 10, "Interfacing - buses"

2. Blaauw, Gerrit A., Digital System Implementation, chapter 9., Communication, pp. 286-316, Prentice-Hall (1976).

3. Chen, Bus Communication Systems, Ph. D. Thesis (1974)

4. U.S. Patent (Delagi.., (Unibus)

5. U.S. Patent (Durvasula.(VAX memory command buffer)

6. Enslow, Jr., Philip H.(ed.), Multiprocessors and Parallel Processing, chapter 2., Systems Hardware, pp. 26-80, Wiley (1974).

7. Kaman, C, Computer buses - a survey of design factors, Digital Equipment Corp., unpublished technical note, (1974).

8. U.S. Patent (Levy,..., (SBI)

9. U.S. Patent (Levy...., (Massbus)

10. MacLaren, Don, Contention-arbitrated serial buses, (1977)

11. Metcalfe, Robert M., Packet Communication, Massachusetts Institute of Technology Project MAC report MAC TR-114 (Dec.,1973).

12. Metcalfe, Robert M. and David R. Boggs, Ethernet: Distributed packet switching for local computer networks, Xerox Palo Alto Research Center report CSL 75-7, (Nov.,1975).

13. U.S. Patent (Northrup..(11/70 memory bus)

14. Ornstein *et*. *al*., Pluribus, NCC '75.

15. Tanenbaum, Andrew S., Structured Computer Organization, chapter 4, pp. 196-204, Prentice-Hall (1976).

16. Thurber *et*. *al*., A Systematic approach..., FJCC '7x.

Chapter 17 - art (Levy)

124a

ART LIST

To rev. 1/24

Fig. No.	Figure Title	I.D. No.	In Proc	Rdy or Rev	Stat Ord/Av	Size
Øa, Øb, Øc						
1	Memory modules do not participate in					
	arbitration, since they never initiate					
	data transfers.					
2	No caption (Table)	Fig. 2.5 (a+b)				
3	Unibus 2 (Table)		✓			
4 3a	No caption		✓			
5 4a, 4b	Simplified diagram of SBI Arb. signals					
6	Timing diagram of arbitration for an					
	example set of data transfers on a					
	simplified SBI					
7	A Polled Character Input Bus		✓			
8	Timing diagram of a Polled Character		✓			
	Input Bus		—			
9	(Table) (no rough)					
10	Unibus Data Transfer Synchronization		✓			
11	Timing Diagram of Unibus Data-Out		✓			
	and Data-In Transaction	11.2, 11.4, 11.6 + 11.8				
12	Simplified SBI data transfer section					
13	Two SBI transactions which make up					
	a memory read operation					
14	SBI transactions which make up a					
	memory write operation					
15	Massbus					
16	Massbus Control Section (Simplified)					
16a	Timing of Massbus Control Section		✓			
	Control Read (normal case)		—			
17	Timing of Control Read from Attention					
	Summary Pseudo-register					
	(Massbus Control Sect, Reg. No. = Ø4)					
	(2 Figs)					
18	Massbus Data Section (simplified)		✓			

NOTE: Please indicate special or unusual layout considerations by way of a slip sheet.

Chapt 17 - art (Levy)

1246

ART LIST

To review 1/24

Fig. No.	Figure Title	I.D. No.	In Proc	Rdy or Rev	Stat Ord/Av	Size
19	Timing of Massbus Data Read			✓		
20	Timing of Massbus Data Write			✓		
21	~~No roug~~ (Table)					
22	Timing of Parity Check and Acknowledgement on SBI					
23	Plausible error-checking scheme w/ Acknowledgement + retry for polled character-input bus			✓ — —		
24	Massbus Data Section Timing of Exception Signal While Writing			✓ —		
25	PDP-11/20 Configuration			✓		
26	PDP-11/45 Configuration			✓		
26a	PDP-11/45 as actually used					
27	PDP-11/70 Configuration			✓		
28	VAX-11/78 Configuration					
29	Genealogy of PDP-11 Family Buses			✓		
30	Key Design Goals of High-Performance PDP-11 Family Systems (Table)					

NOTE: Please indicate special or unusual layout considerations by way of a slip sheet.

4.3.1.1 The data bus is used for transmission of data from and to the drive recording medium. Timing of transfers is controlled by a clock which is generated by the drive.

4.3.1.3 Transfers are oriented towards blocks of data which are transmitted as a group (e.g., sectors on a disk, records on mag tape). The drive will normally send and receive data only as whole blocks. If the number of data words desired by the CPU is not an integral times the number of words per block, it is up to the controller to stop the transfer to memory on reads or to provide filler words on writes.

4.3.1.2 The data bus is shared among all drives. Only one drive may be attached to it at a time. The controller should prevent a data transfer command from being loaded into a drive while OCC is asserted.

4.3.1.4 A drive attaches itself to the data bus and asserts OCC when a data transfer command is loaded into its Control register. After transferring one or more blocks of data (unless a class B error occurs), the drive disconnects from the data bus and negates OCC. Disconnect always occurs at the trailing edge (negation) of an EBL pulse.

4.3.1.5 For detailed description of error conditions and their effects on data bus signals, see section 7.

4.3.2 Data Bus Read Sequence

4.3.2.1 This section describes a typical data bus read sequence with no errors. 4.3.2.3 is a timing diagram of a read of a single sector with four words. 4.3.2.2 is a flowchart, with timing restrictions, of the read sequence. The following sequence occurs on a data bus read (refer to 4.3.2.3).

1. A read command is loaded into the Control register of the drive. If the command is valid, the drive enables its data bus receivers and drivers and asserts OCC.

2. Not more than 1ØØ microseconds after step 1, the controller asserts RUN.

3. After a cable delay, the drive receives the RUN assertion. Disk drives now begin searching for the desired sector. Tape drives begin tape motion.

4. When the drive has read the first data word, it generates parity for the word; the data and DPA are gated onto the data lines and SCLK is asserted.

5. After a cable delay, the controller receives the SCLK assertion.

6. The drive negates SCLK no less than T nanoseconds after asserting it, where T is either 225 nanoseconds or 3Ø

percent of the nominal burst data period of the drive, whichever is greater. The Data lines should be maintained valid for no less than one half of the SCLK interval after SCLK is negated.

7. After a cable delay, the controller receives the SCLK negation. The controller strobes the D lines and DPA, and checks the parity.

8. If there is more data to be read in this block, then not less than T nanoseconds after step 6, the drive gates out the next data word onto the D lines, generates DPA, and asserts SCLK. Steps 5, 6, and 7 then follow.

9. After the negation of SCLK (step 6) on the last word of data in the block, the drive asserts EBL.

10. After a cable delay, the controller receives the EBL assertion. At this time, the controller must decide whether or not to have the drive read the next block of data without disconnecting from the data bus (the controller may already have negated the RUN line).

11. If the controller decides not to read the next .block, it negates the RUN line not later than 500 nanoseconds after step 10.

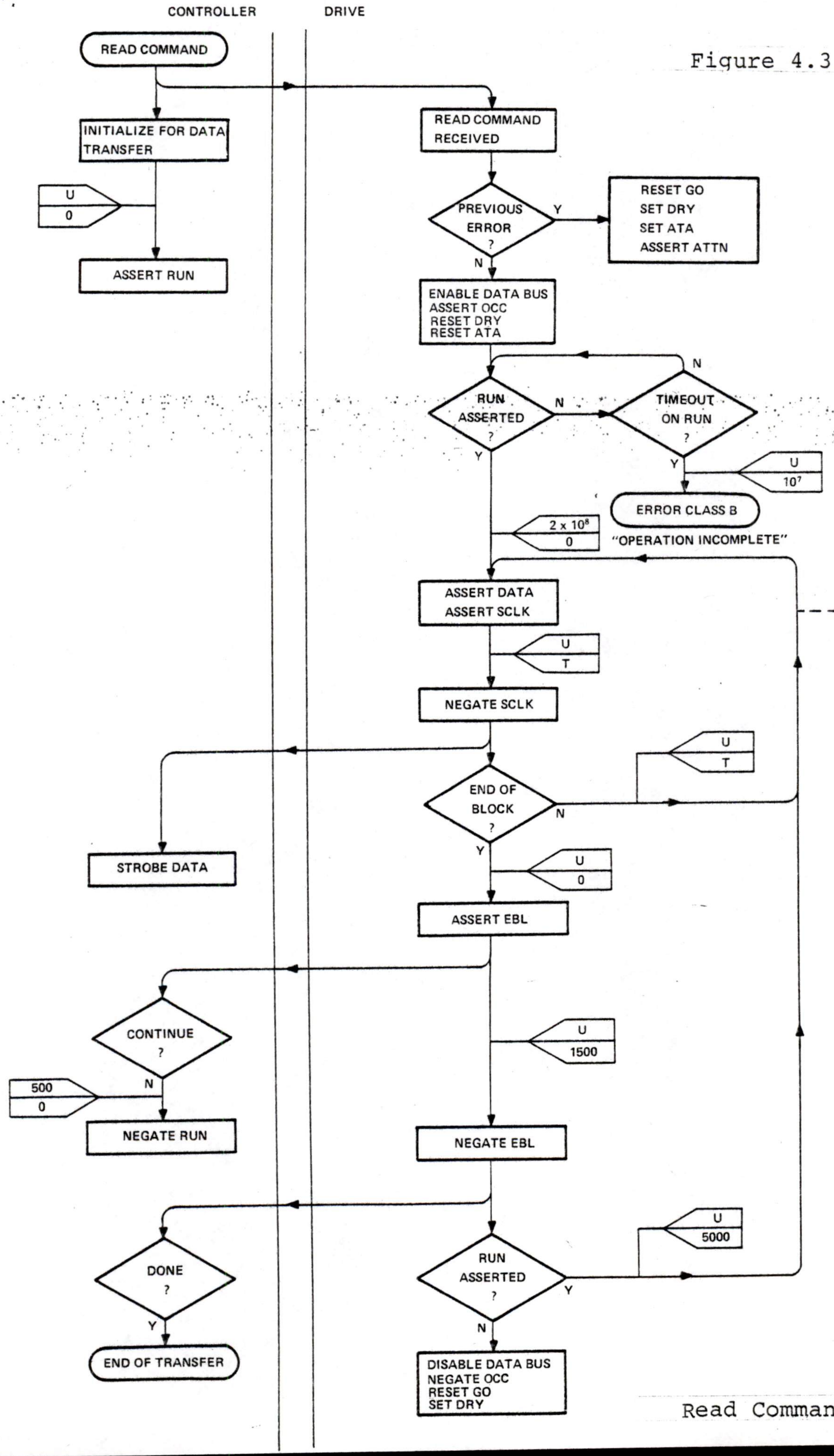
12. After a cable delay, the drive receives the RUN negation (the RUN line may already have been negated).

13. Not less than 1500 nanoseconds after step 9, the drive negates EBL. At this time the drive strobes the RUN line. If RUN has been negated, the drive disconnects from the data bus (the DRY bit should be set and OCC negated at this time).

14. After a cable delay, the controller receives the EBL negation (the controller may now generate an end-of-transfer interrupt, and start another data transfer).

Levy fig 6b

Figure 4.3.2.2



NOTE: MINIMUM TIME FROM ONE ASSERTION OF SCLK TO THE NEXT IS EITHER 500 ns OR P, WHICHEVER IS GREATER; MAXIMUM UNSPECIFIED.

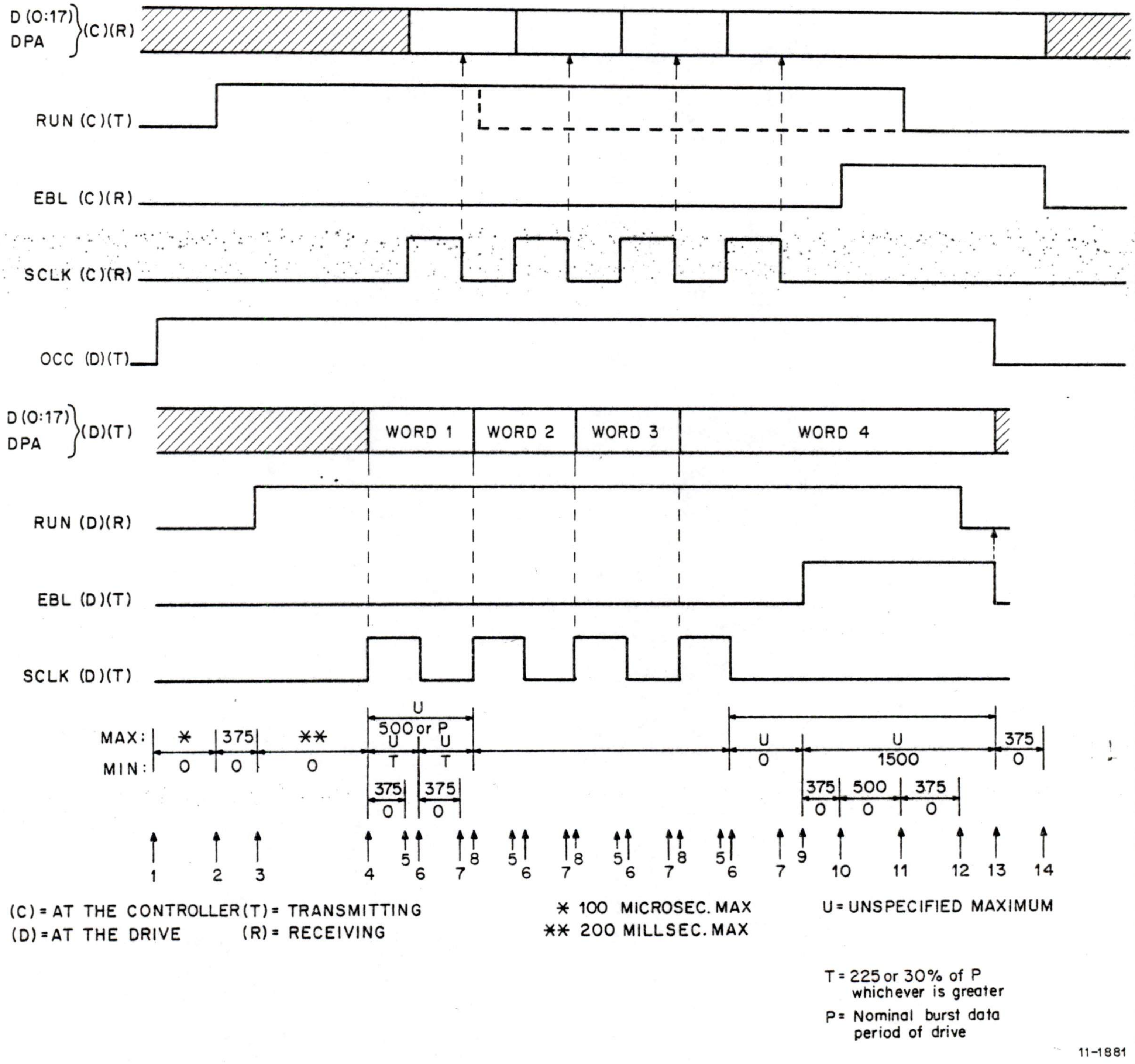
T = 225 nsec OR .3P, WHICHEVER IS GREATER

P = NOMINAL BURST DATA PERIOD OF DRIVE

Read Command Flowchart

11-1877

Figure 4.3.2.3



Data Bus Read Timing

2X

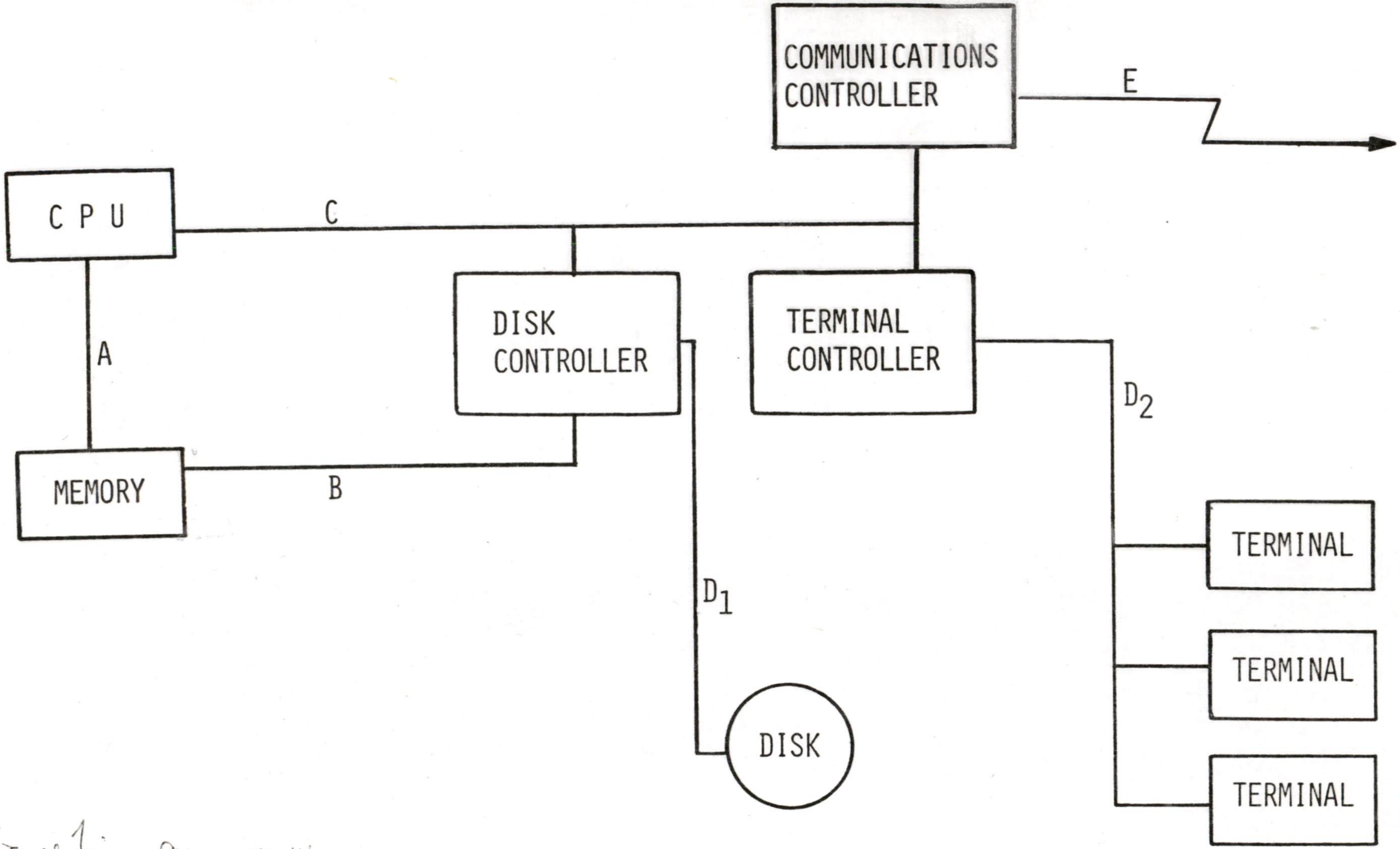
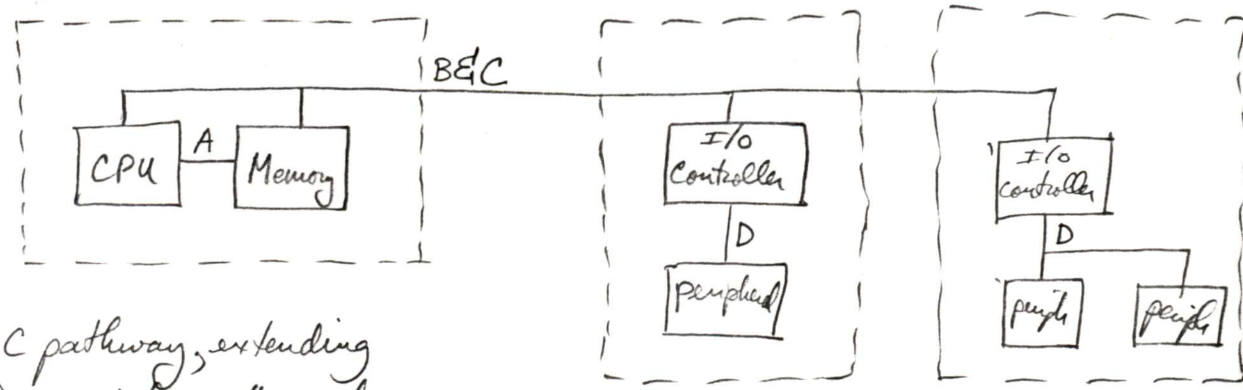


Figure 1: "~~5-BUS~~ 5-HIGHWAY MODEL OF FUNCTIONAL INTERCONNECTIONS

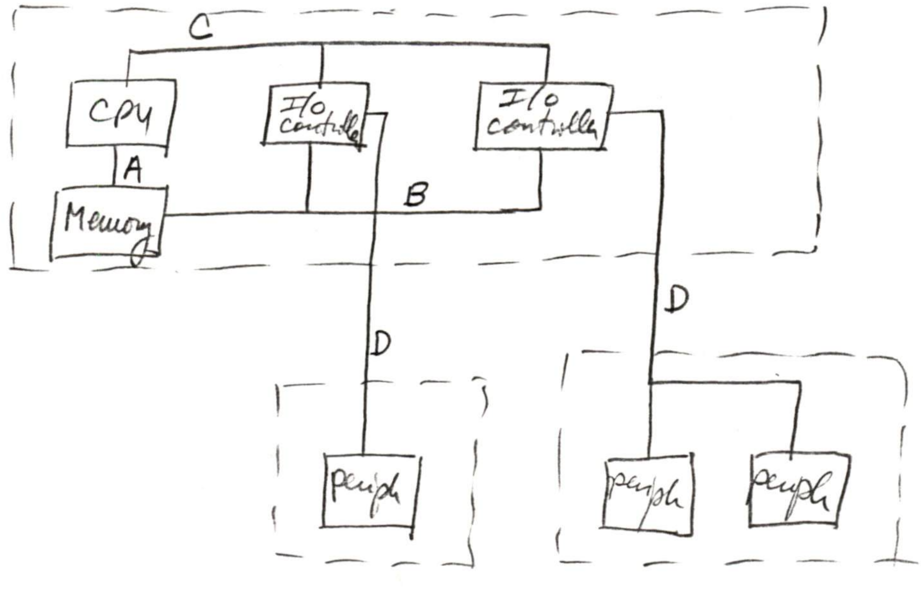
Levy Fig. 1

129

John Levy



(a) Single type C pathway, extending out of the "mainframe" package; short type D paths.

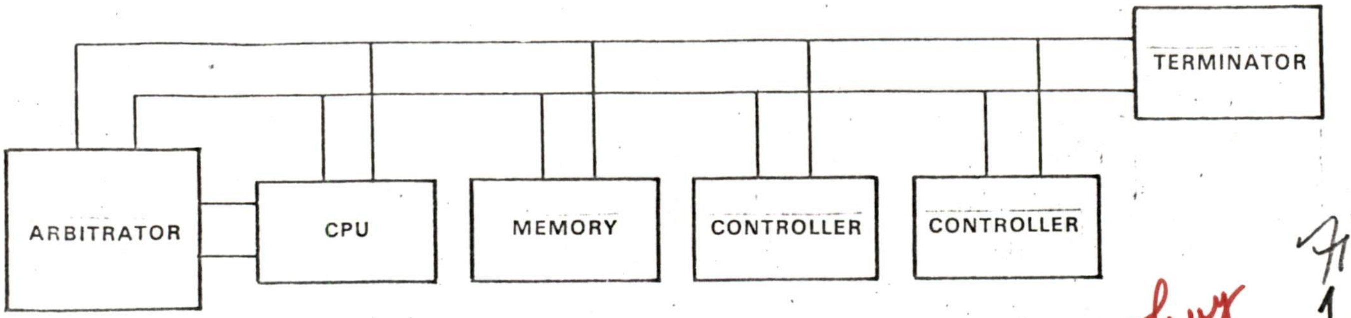


(b) Type C pathway contained within the mainframe package; longer type D paths.

Levy Figure 2.5 A design tradeoff for Type C pathways

19 (ref: 1st ed, p.18

180a



NOTE:

MEMORY MODULES DO NOT PARTICIPATE IN ARBITRATION, SINCE THEY NEVER INITIATE DATA TRANSFERS.

Levy Fig 3a ~~Figure 1~~

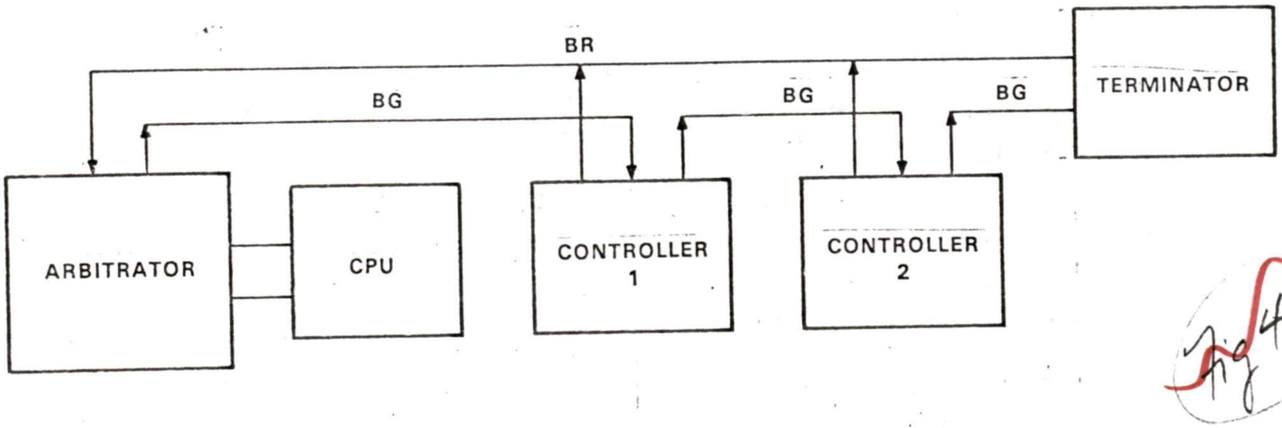
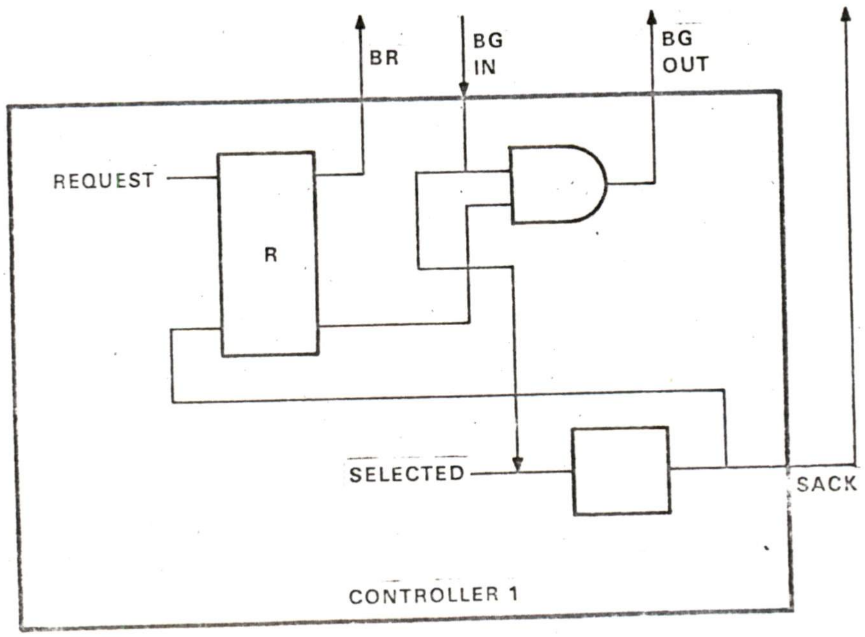
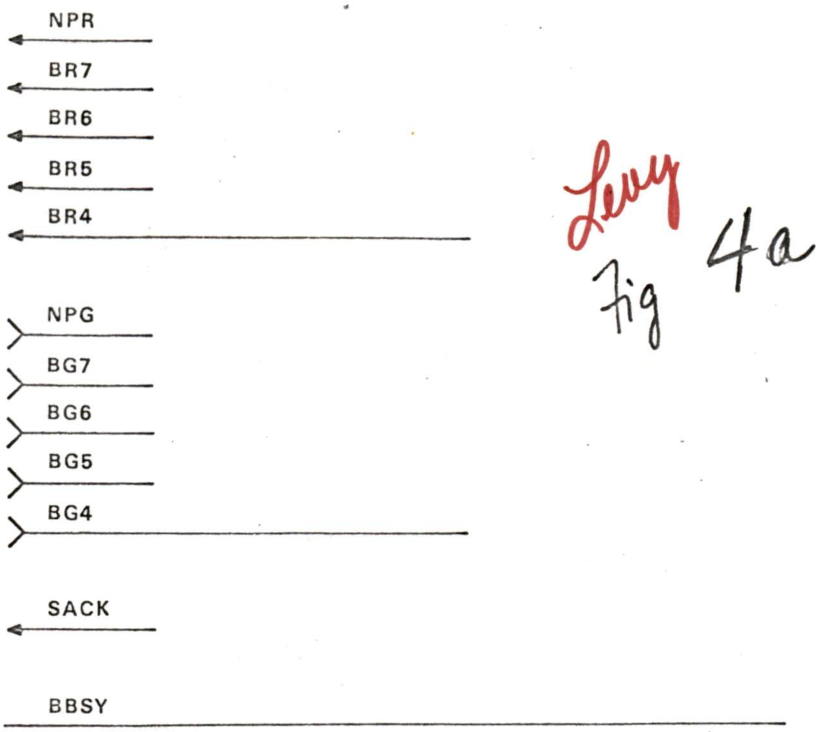


Fig 4!



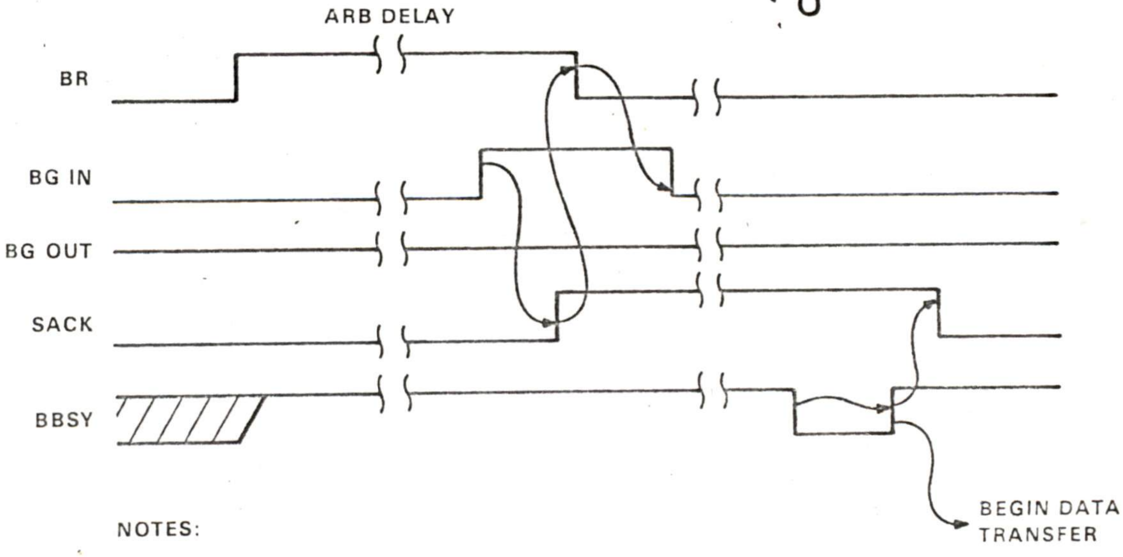
~~Fig. 3~~
131



Levy
Fig 4a

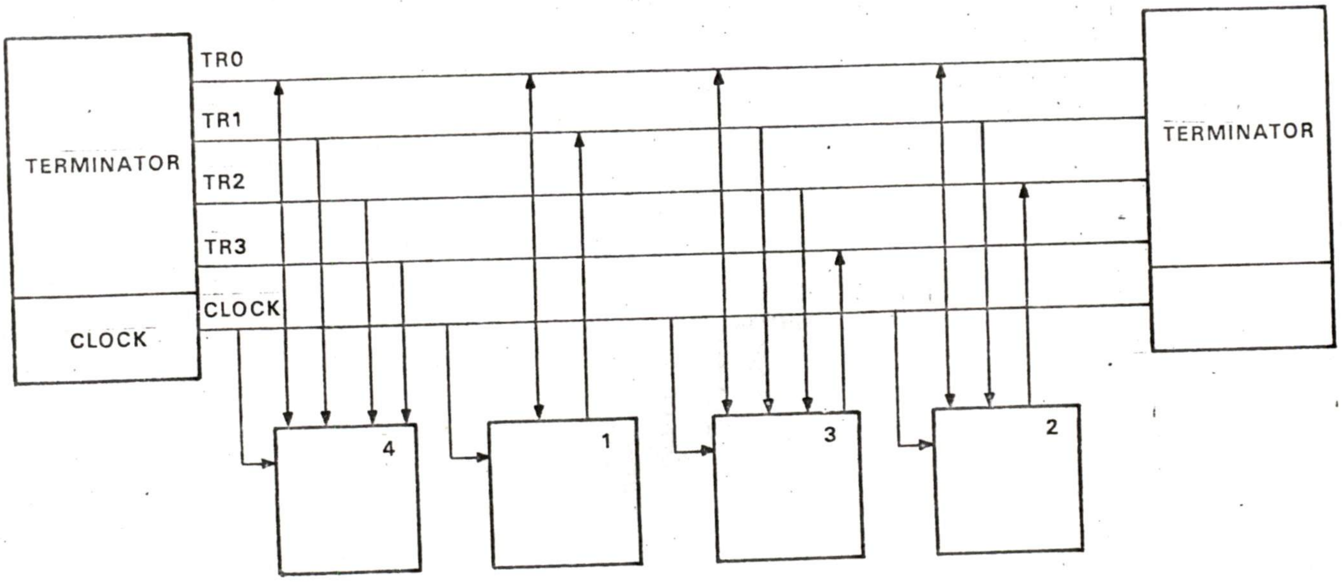
Fig. 4

Levy
Fig 4b

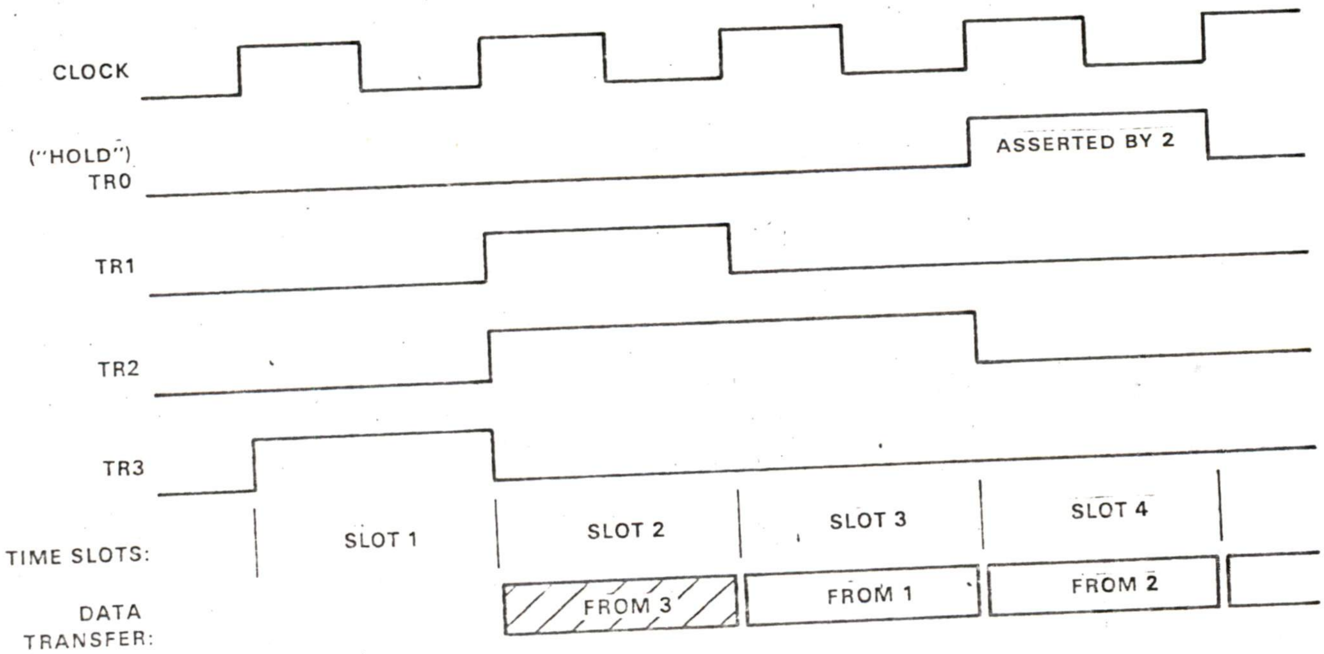


NOTES:

1. WAIT FOR CPU NOT USING BUS.
2. IF NPR, THEN ASSERT NPG.
3. WAIT FOR CPU INTER-INSTRUCTION.
4. IF NPR, THEN ASSERT NPG ELSE
5. IF PRI<7 AND BR7, THEN ASSERT BG7 ELSE
6. IF PRI<6 AND BR6, THEN ASSERT BG6 ELSE.
7. IF PRI<5 AND BR5, THEN ASSERT BG5 ELSE.
8. IF PRI<4 AND BR4, THEN ASSERT BG4.



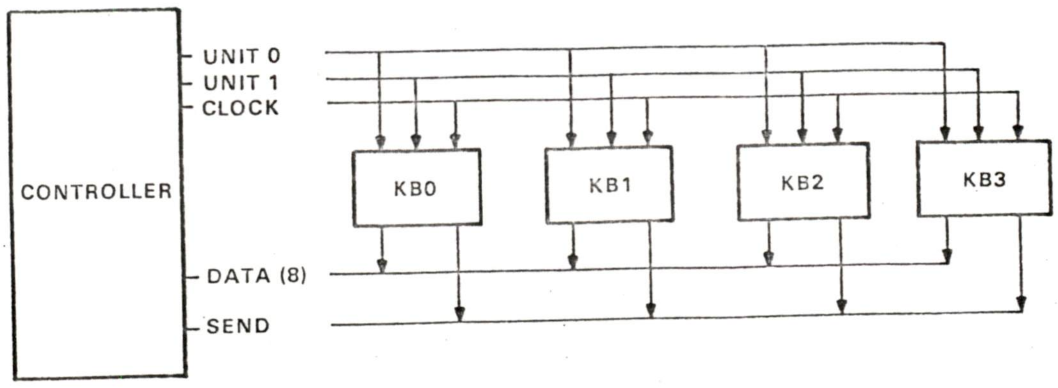
Levy Fig. 5



Levy Fig. 6

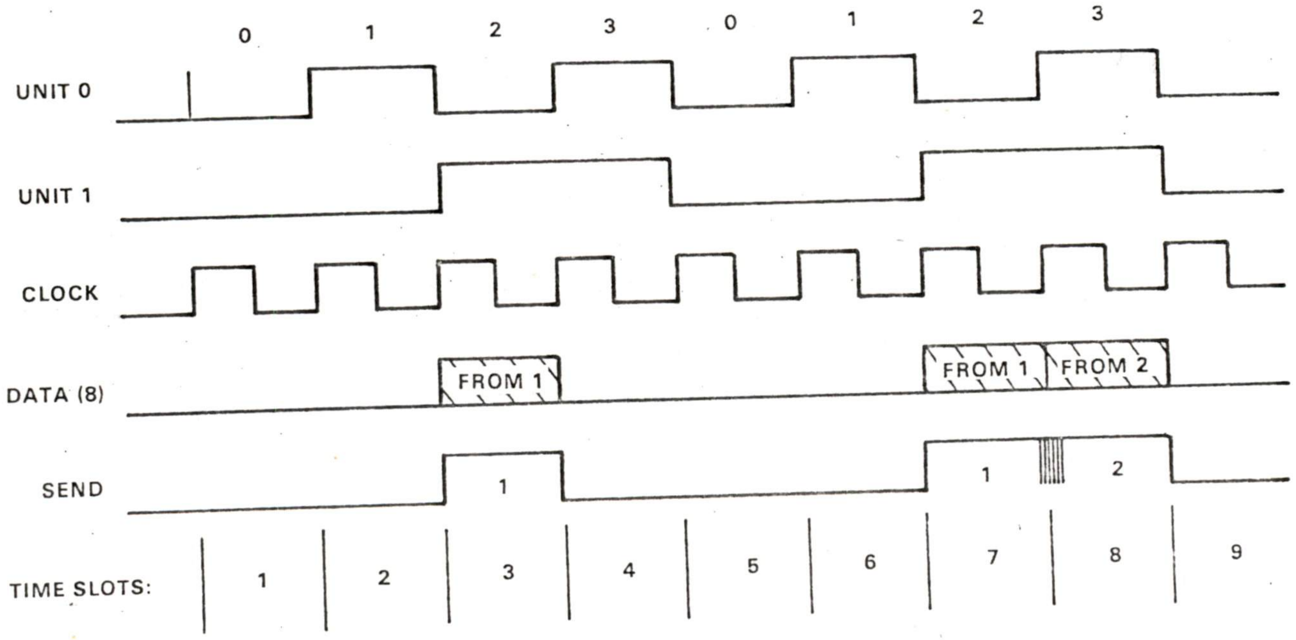
Fig 7

POLLED (CENTRAL, SYNCHRONOUS, SEQUENTIAL)



Levy
Fig. 7

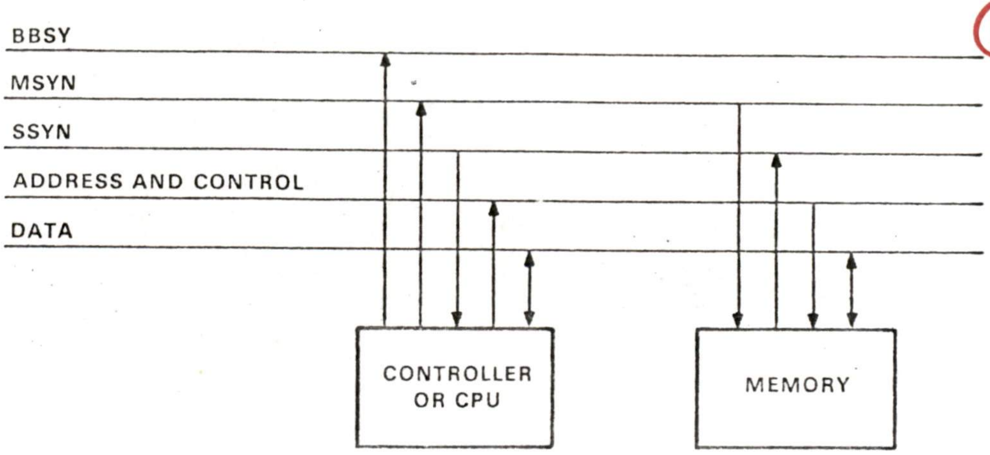
Fig 8



Levy
Fig. 8

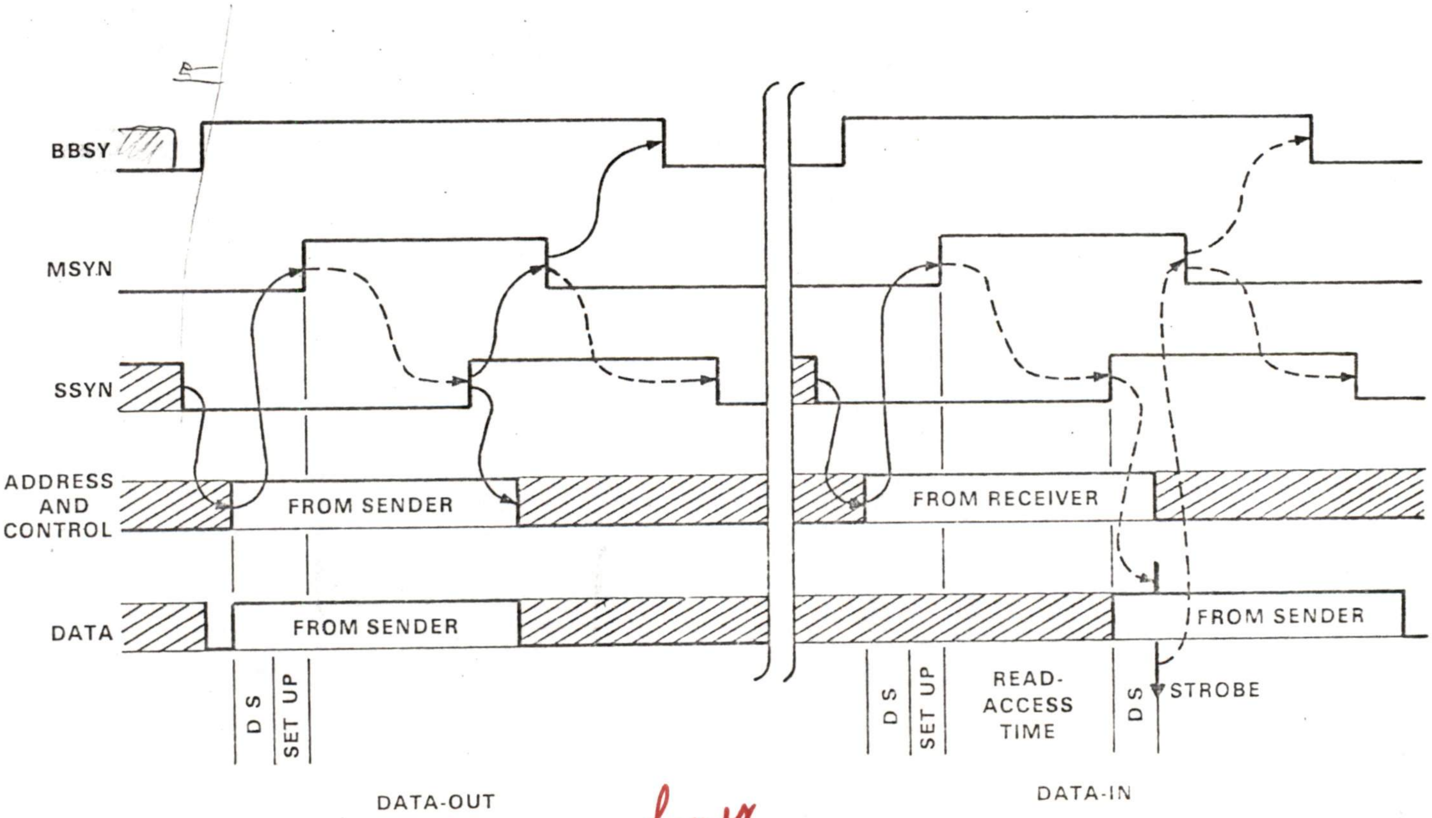
Fig 10

134



Levy Fig. 10

Fig 11

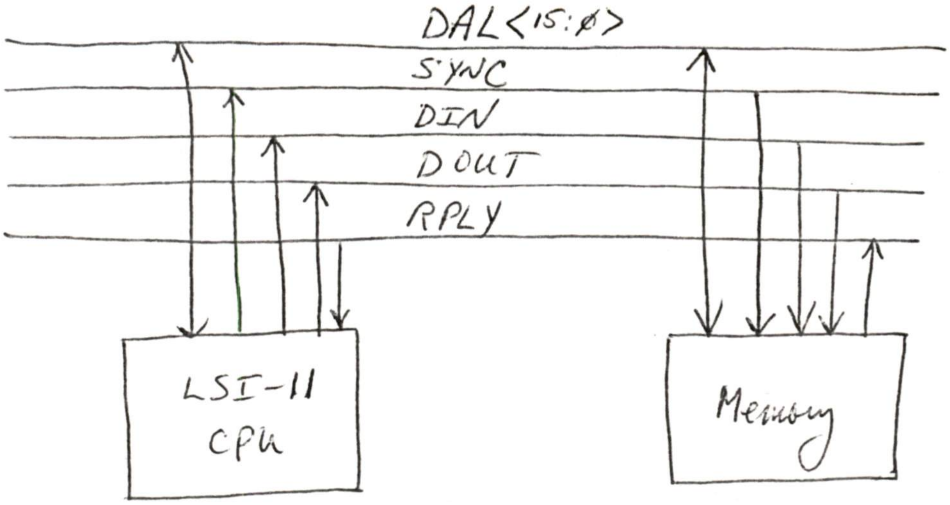


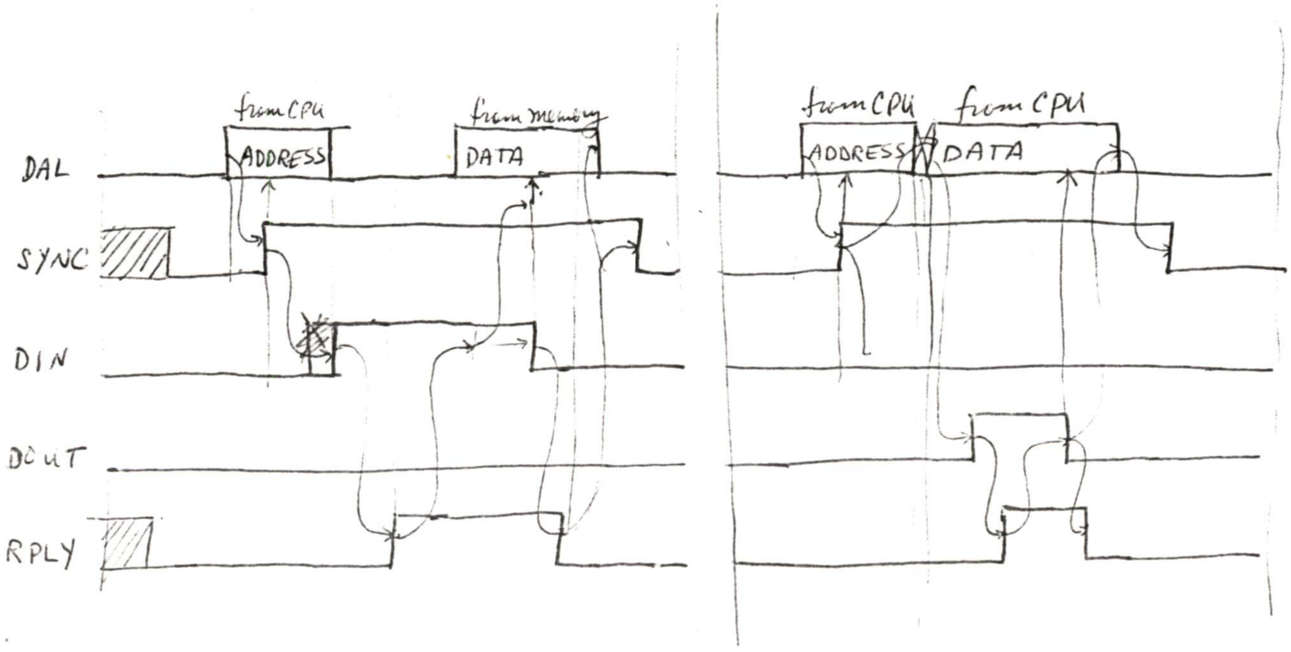
Levy Fig. 11

③

Levy Figure 11.2 Q-bus Data Transfer Synchronization

135





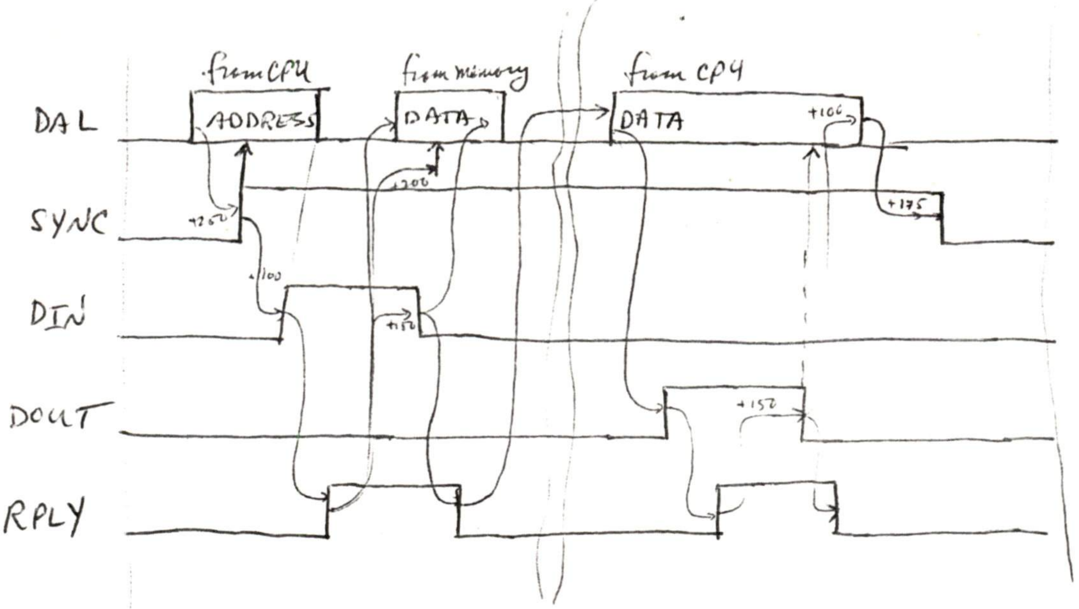
Data-In

Data-Out

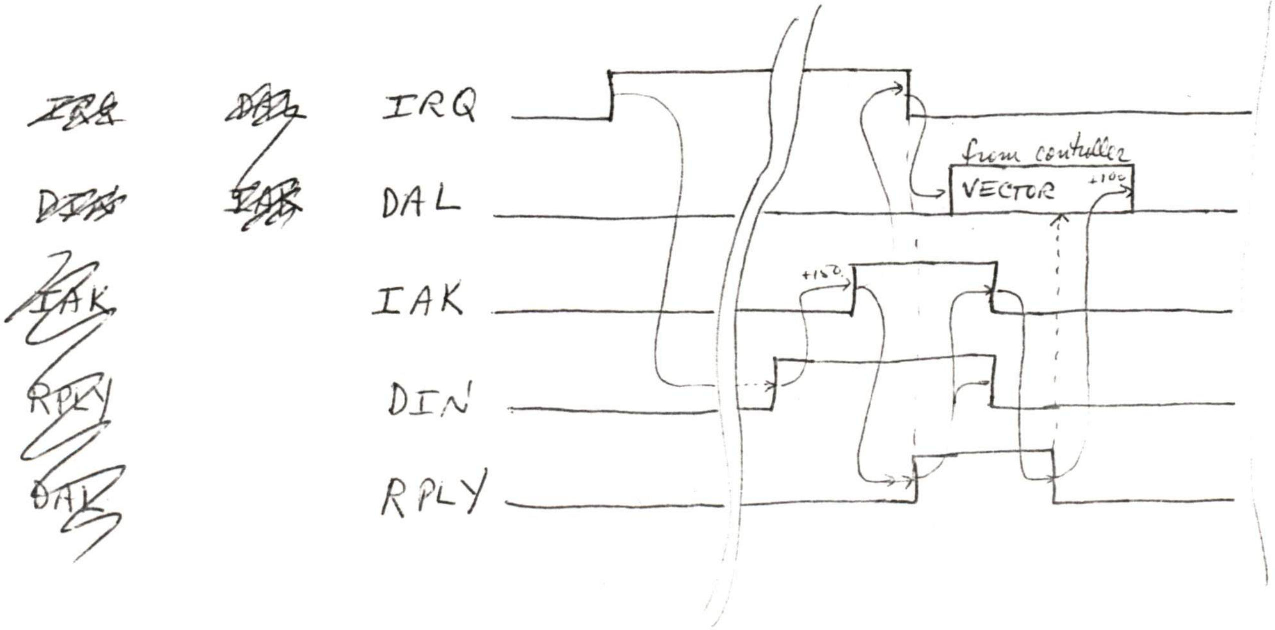
Levy Figure 11.4 Q-bus Data-In and Data-Out Synchronization

(ref? see pp. 45-49)

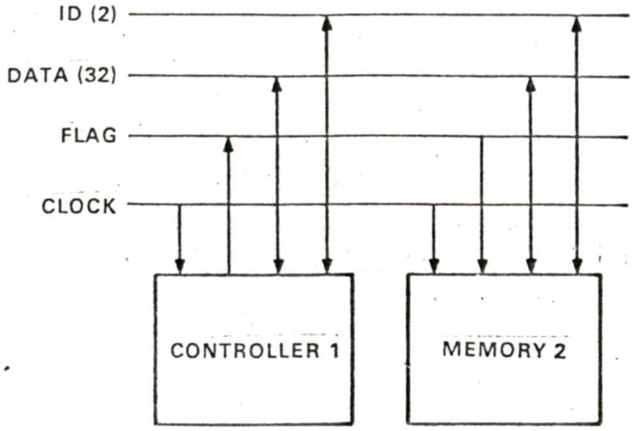
56



Levy Figure 11.6 Q-bus Data-In-Out Synchronization

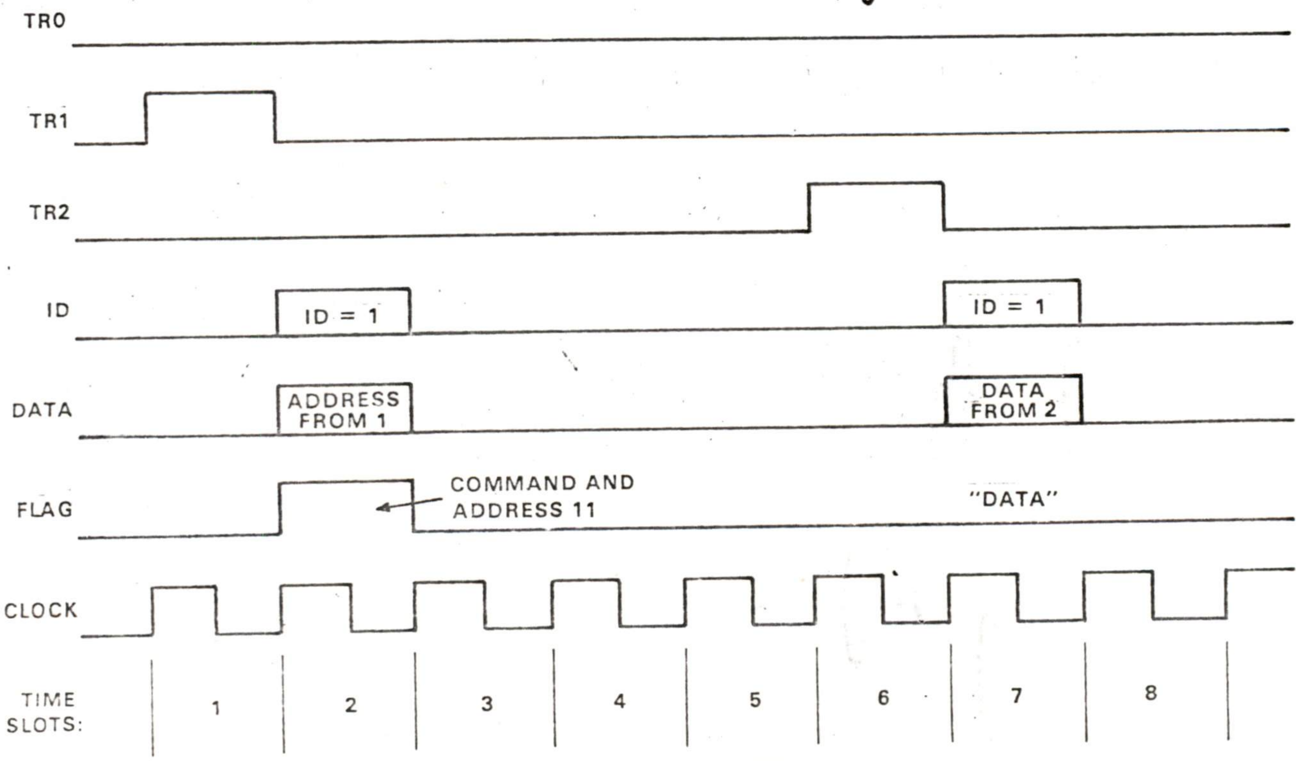


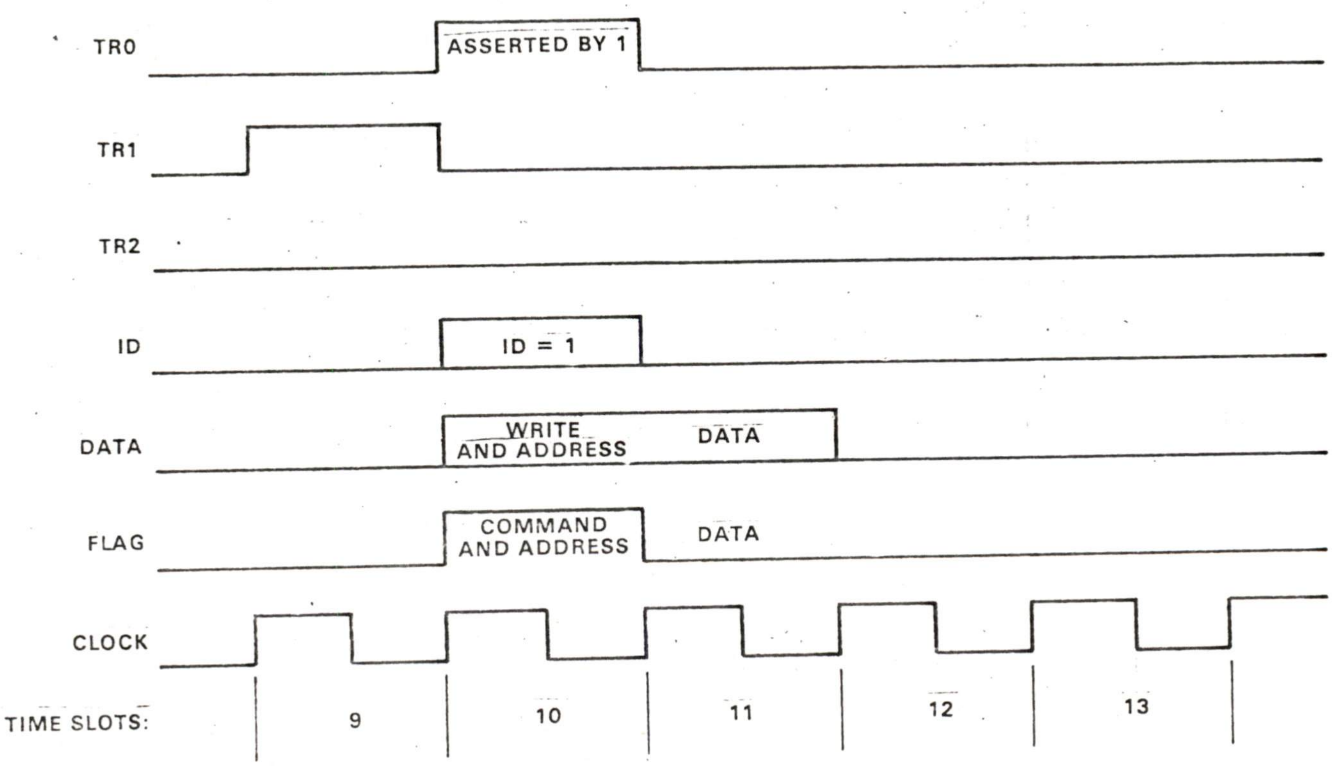
Levy Fig. 11.8 Q-bus Interruption Transaction Synchronization fig



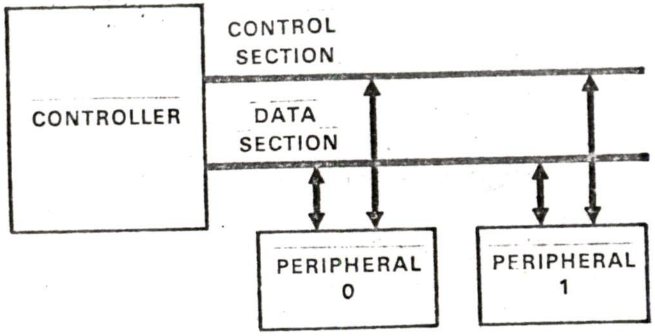
Levy Fig. 12

Levy Fig. 13



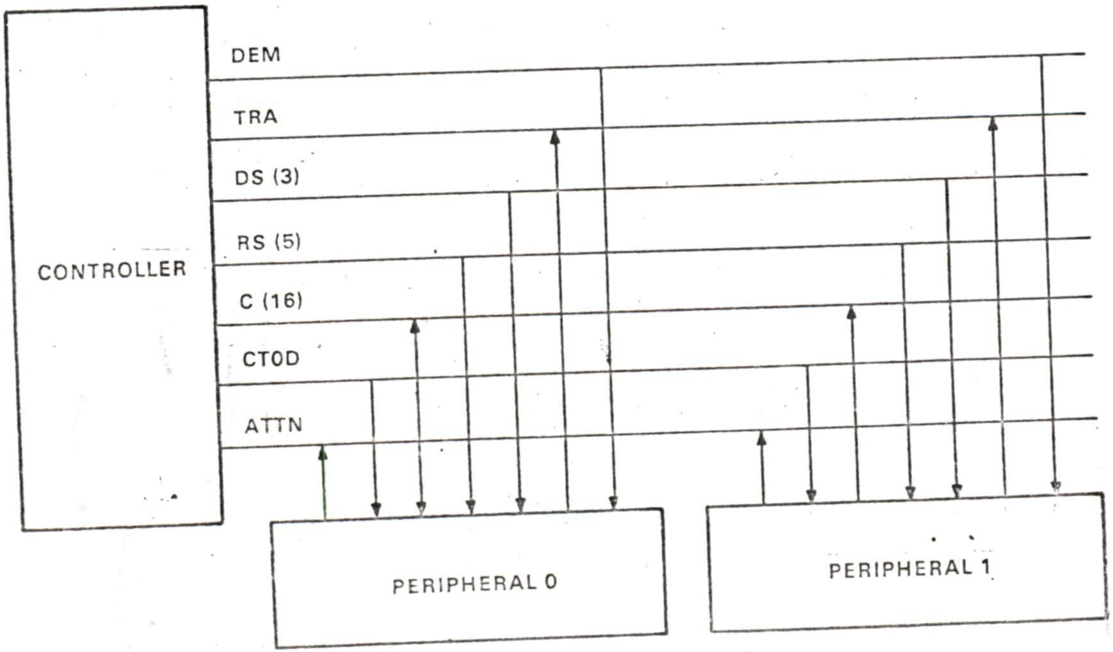

TR0
ASSERTED BY 1
TR1
TR2
ID
ID = 1
DATA
WRITE
AND ADDRESS
DATA
FLAG
COMMAND
AND ADDRESS
DATA
CLOCK
TIME SLOTS:
9
10
11
12
13

Levy Fig. 14



Levy
Fig. 15

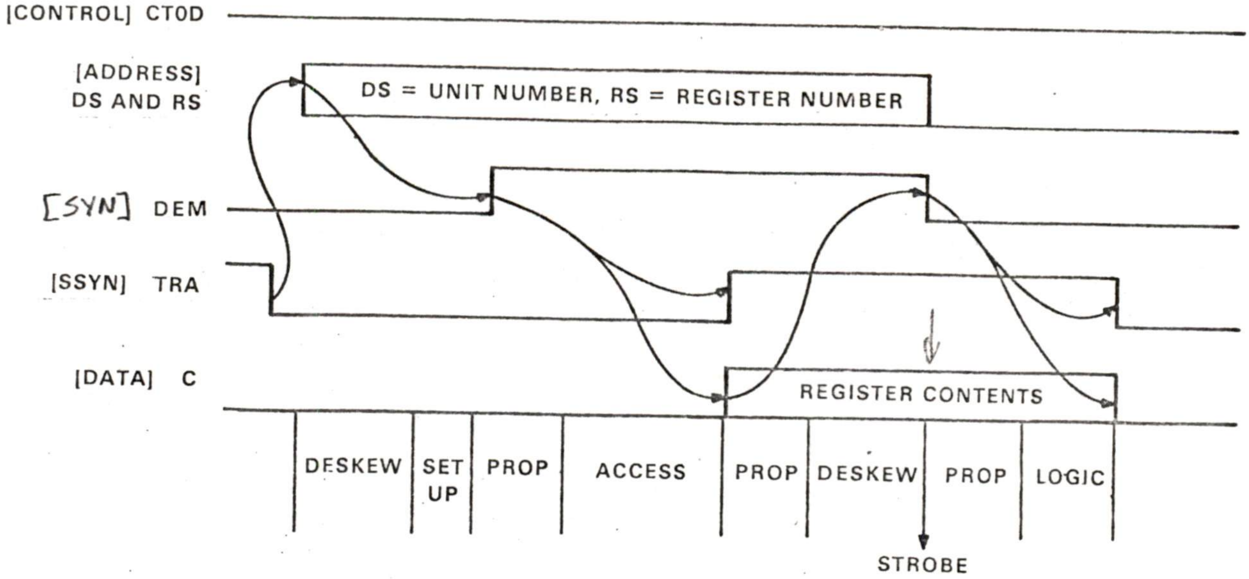
fig. 15



Levy
Fig. 16

Fig 16a

140



Levy Fig. 16a

Levy

Fig. 17

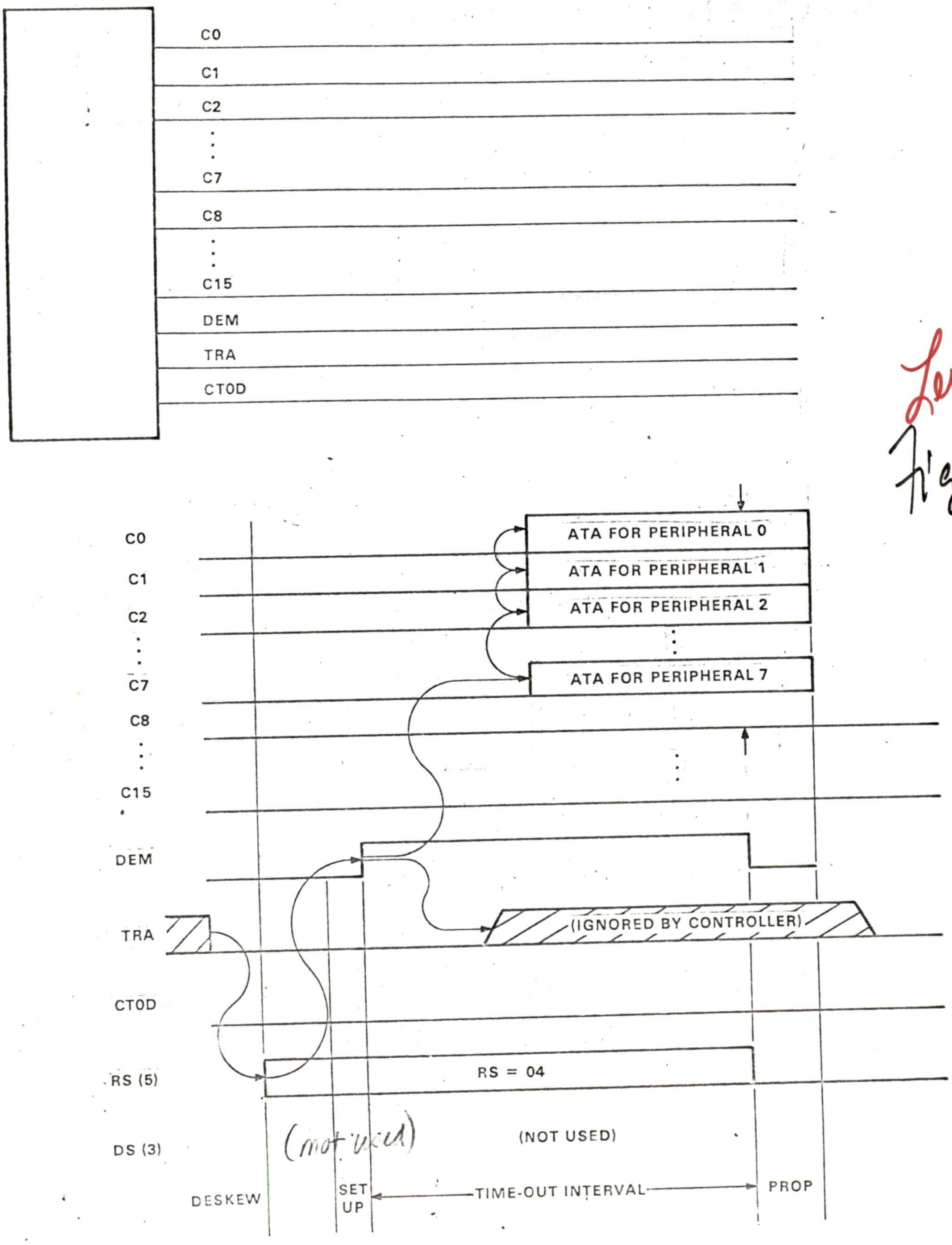
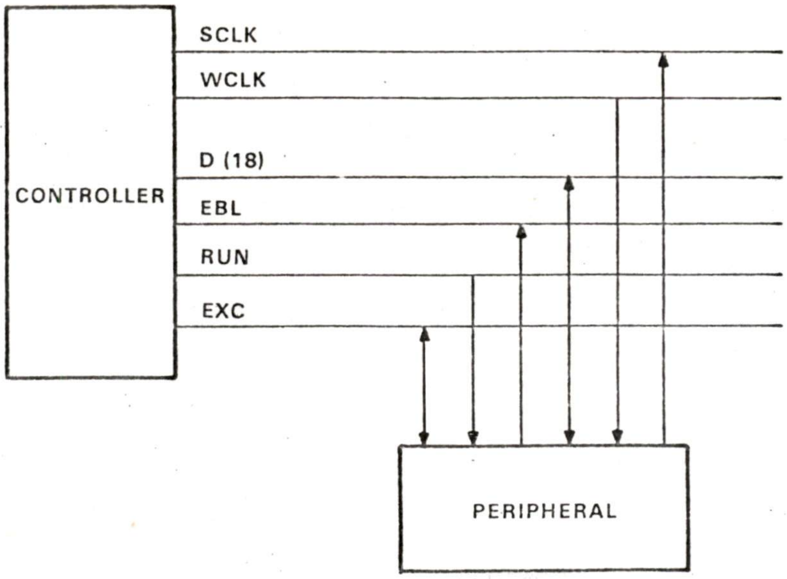
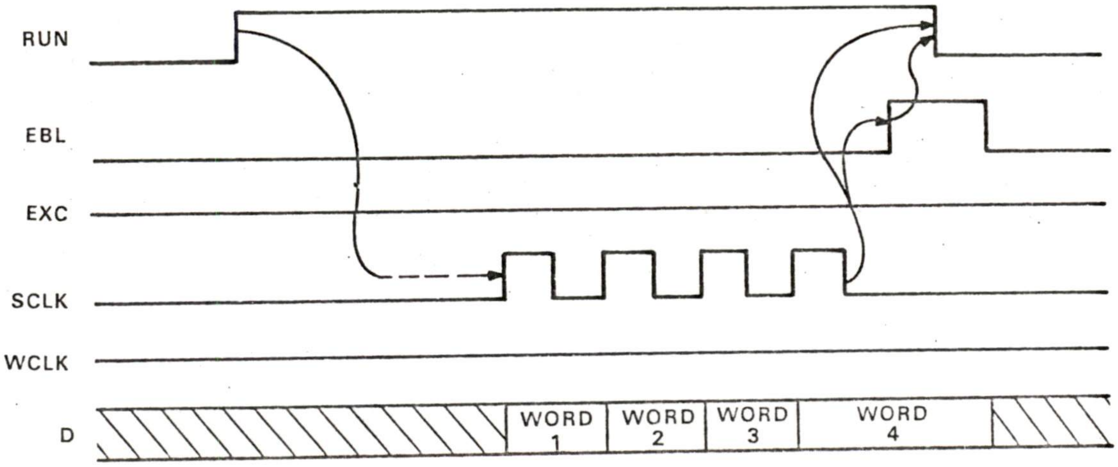

C0
C1
C2
C7
C8
C15
DEM
TRA
CTOD
C0
C1
C2
C7
C8
C15
DEM
TRA
CTOD
RS (5)
DS (3)
ATA FOR PERIPHERAL 0
ATA FOR PERIPHERAL 1
ATA FOR PERIPHERAL 2
ATA FOR PERIPHERAL 7
(IGNORED BY CONTROLLER)
RS = 04
(not used)
(NOT USED)
DESKEW
SET UP
TIME-OUT INTERVAL
PROP

Fig 18 142



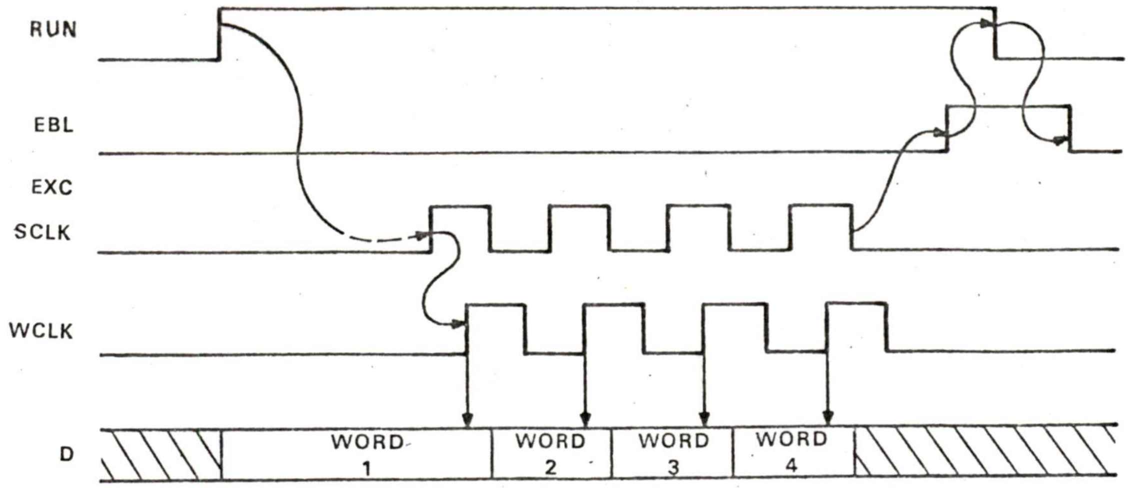
Levy
Fig. 18

Fig 19



Levy
Fig. 19

Fig 20
143



Levy

Fig. 20

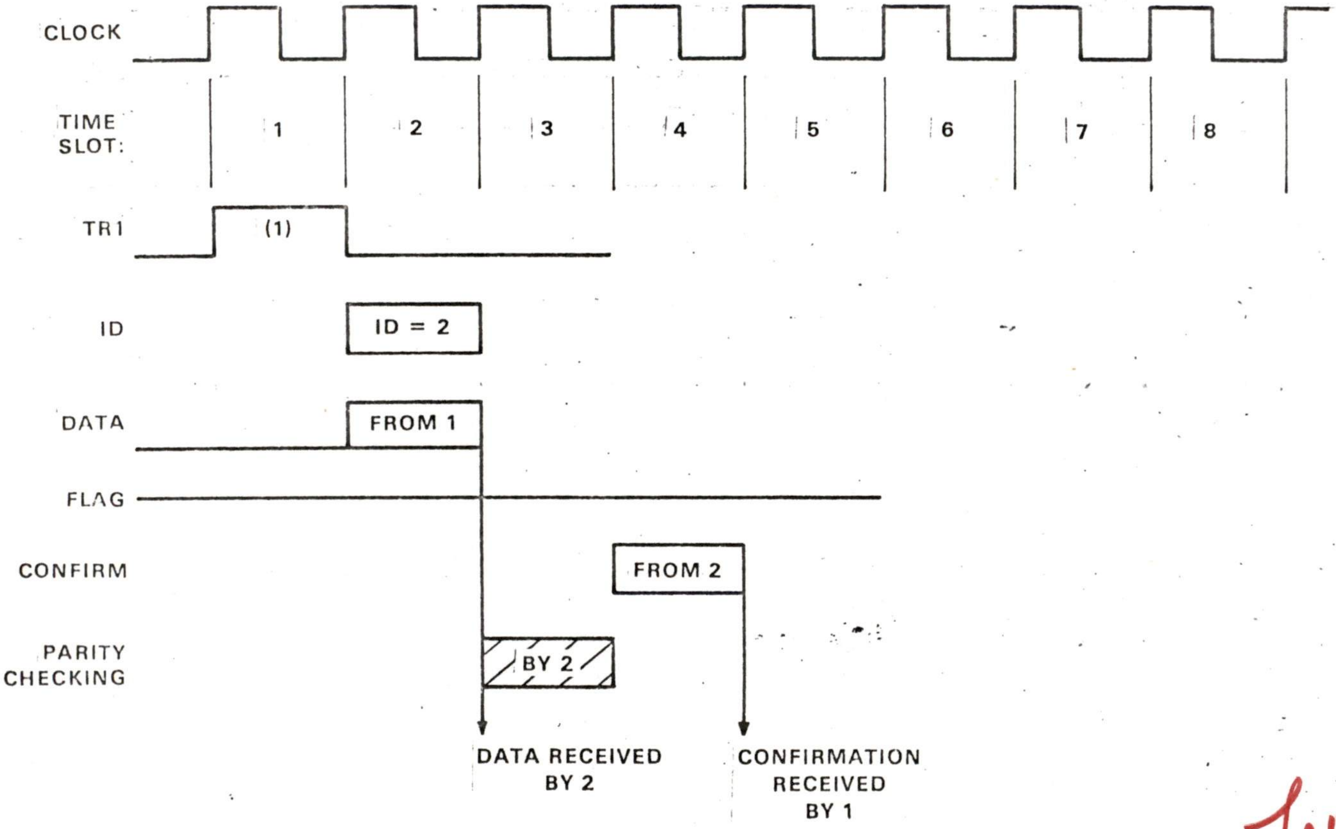


fig. 22

Levy
Fig. 22

Fig 23

Levy

Fig. 23

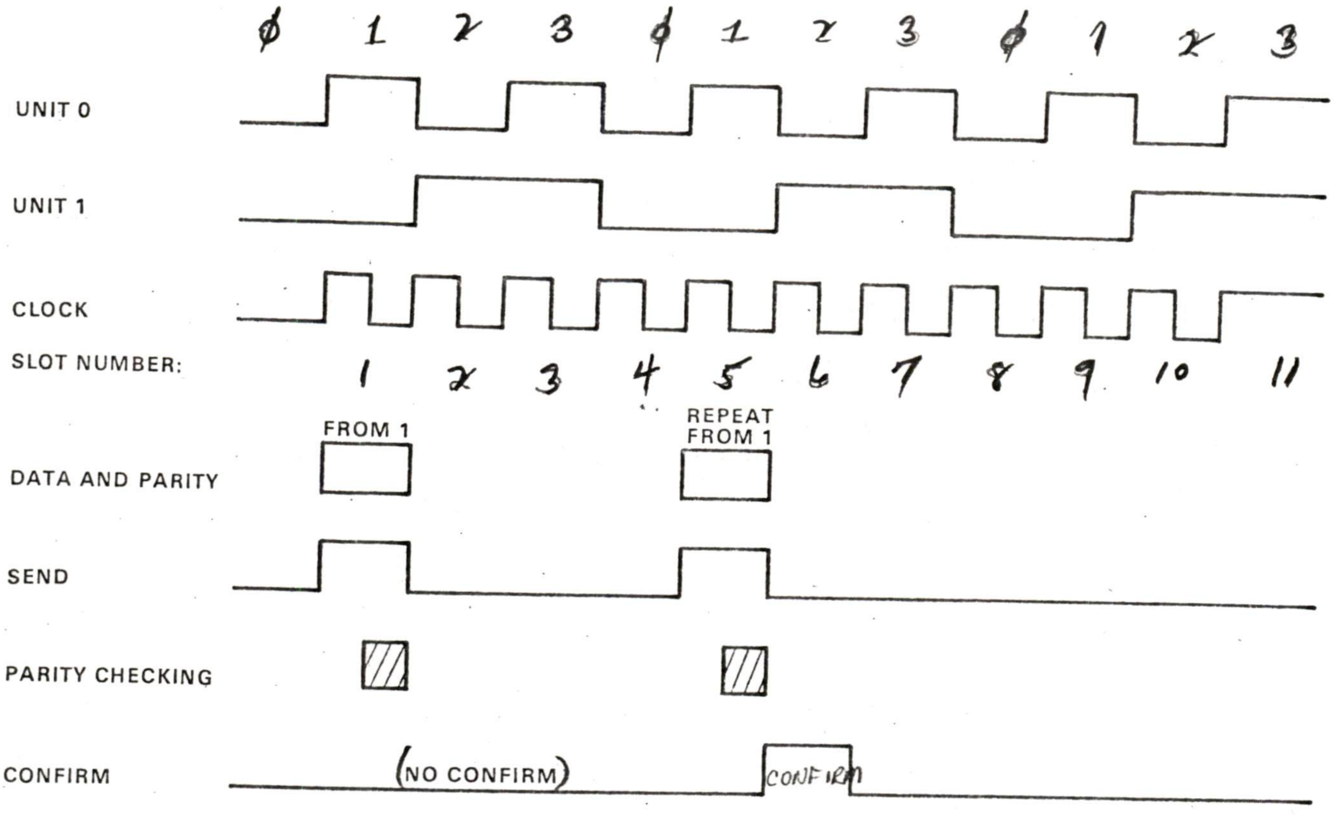
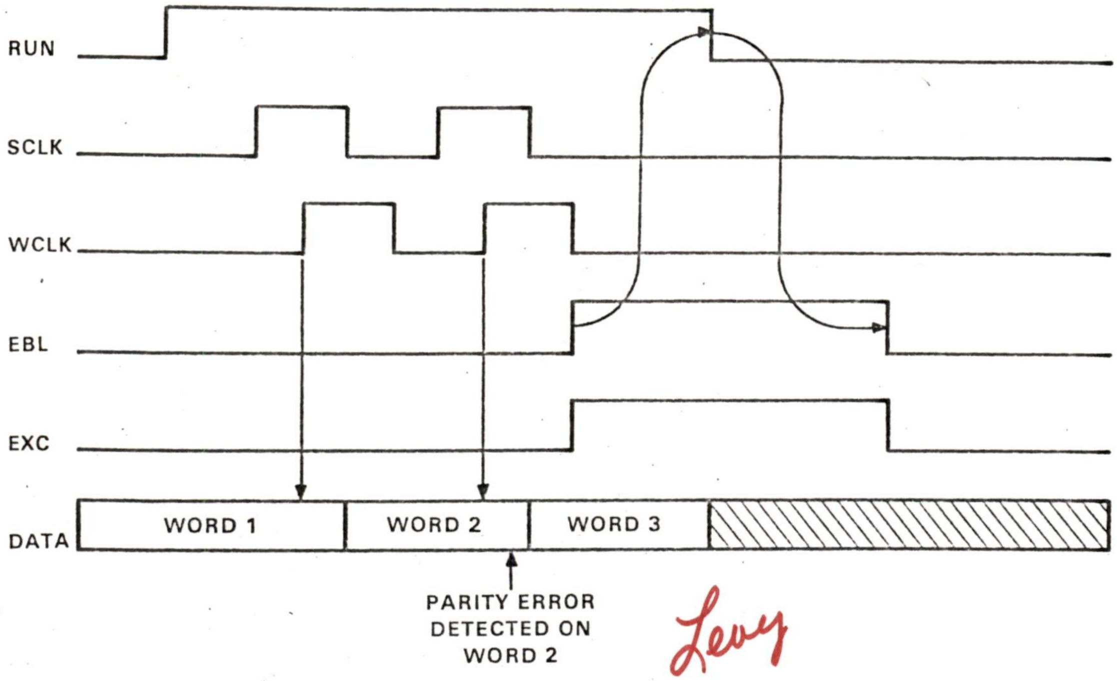


Fig 24



Levy

Fig. 24

147

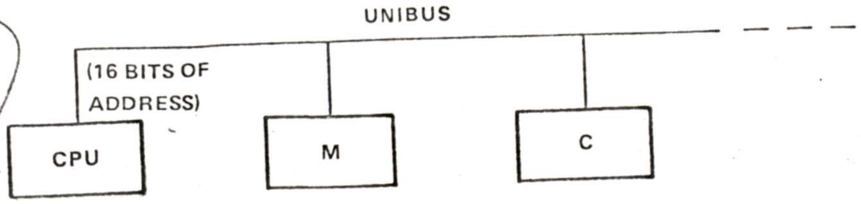
Levy

Fig 25 Fig 25

will be in caption

PDP-11/20 CONFIGURATION

2



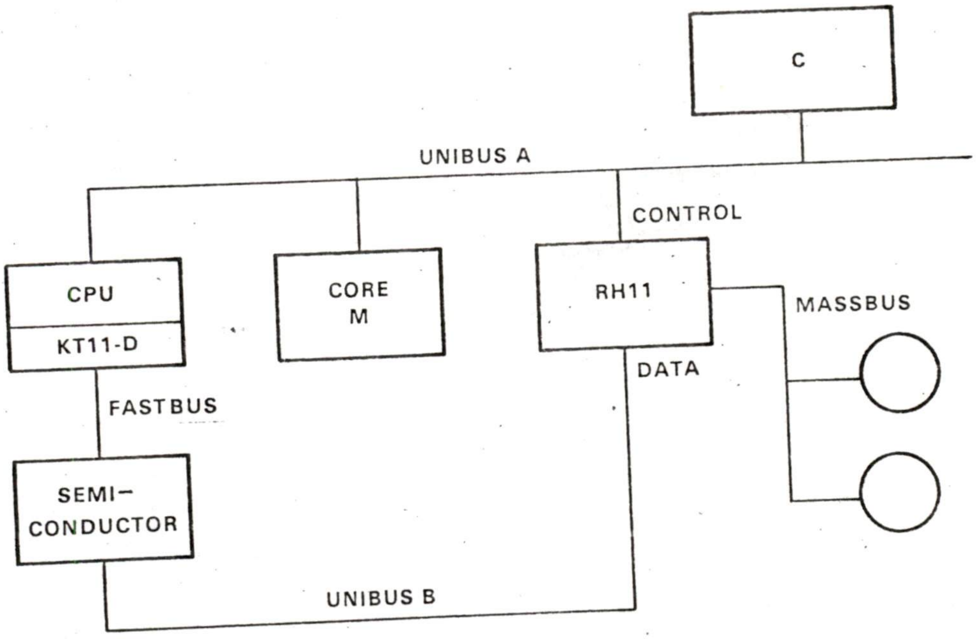
Levy

Fig 26

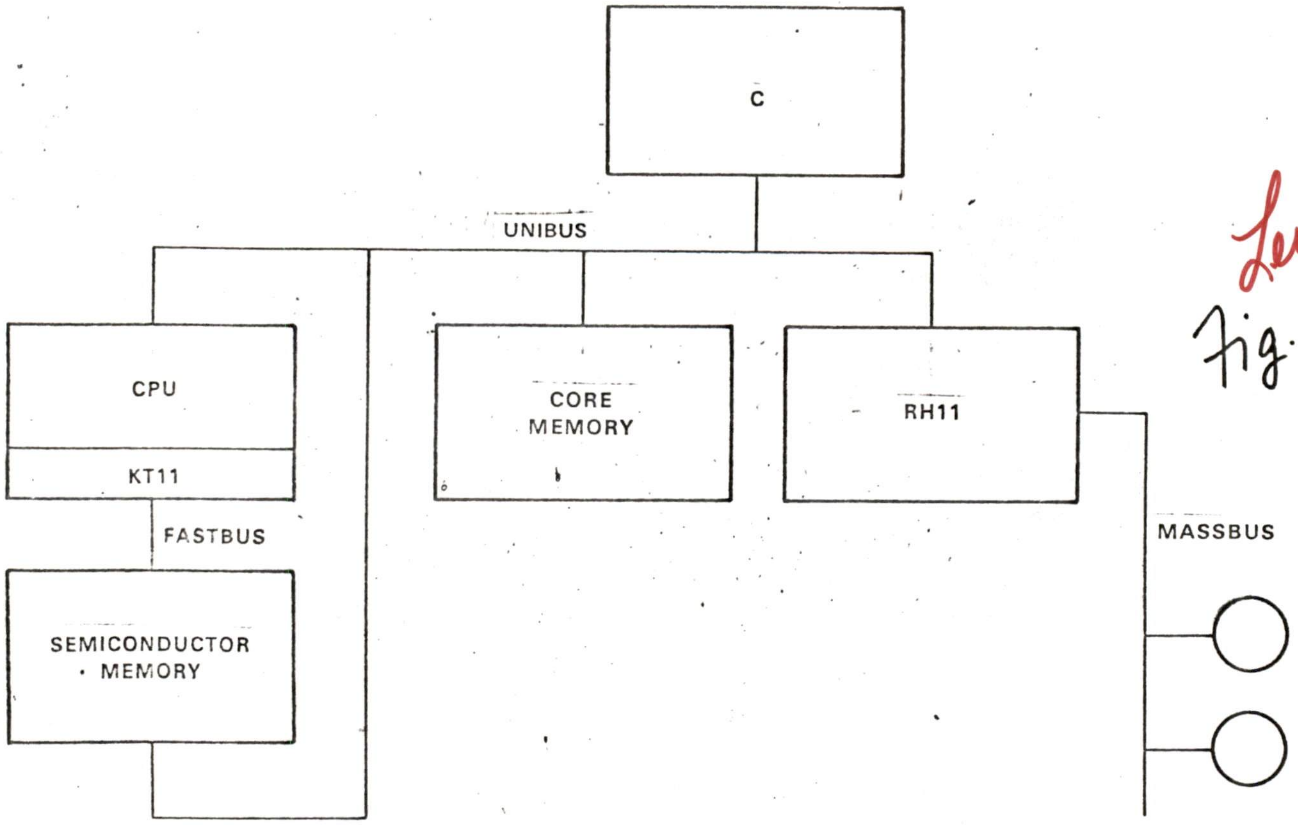
Fig 26

PDP-11/45 CONFIGURATION

2



Levy
Fig. 26a


C
UNIBUS
CPU
KT11
FASTBUS
SEMICONDUCTOR
MEMORY
CORE
MEMORY
RH11
MASSBUS

149

Fig 37

Levy

PDP-11/70 CONFIGURATION

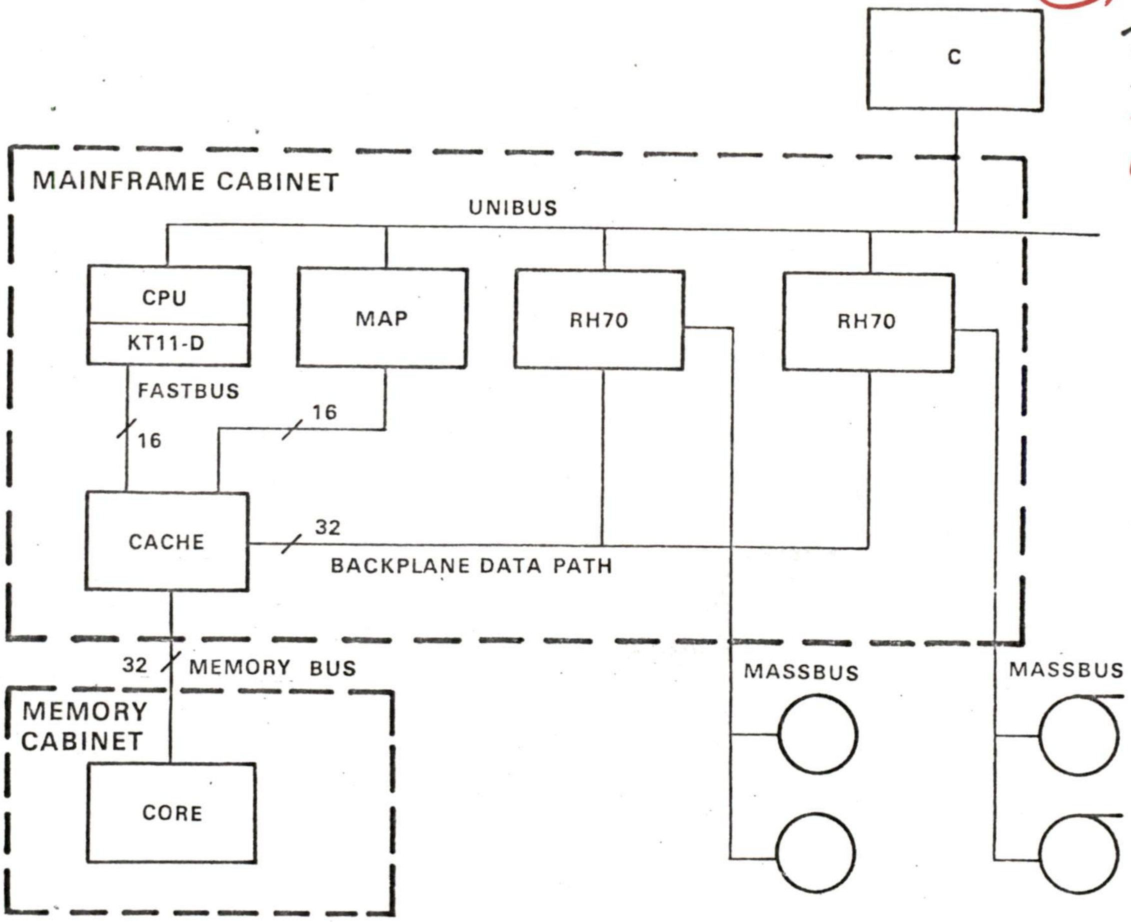
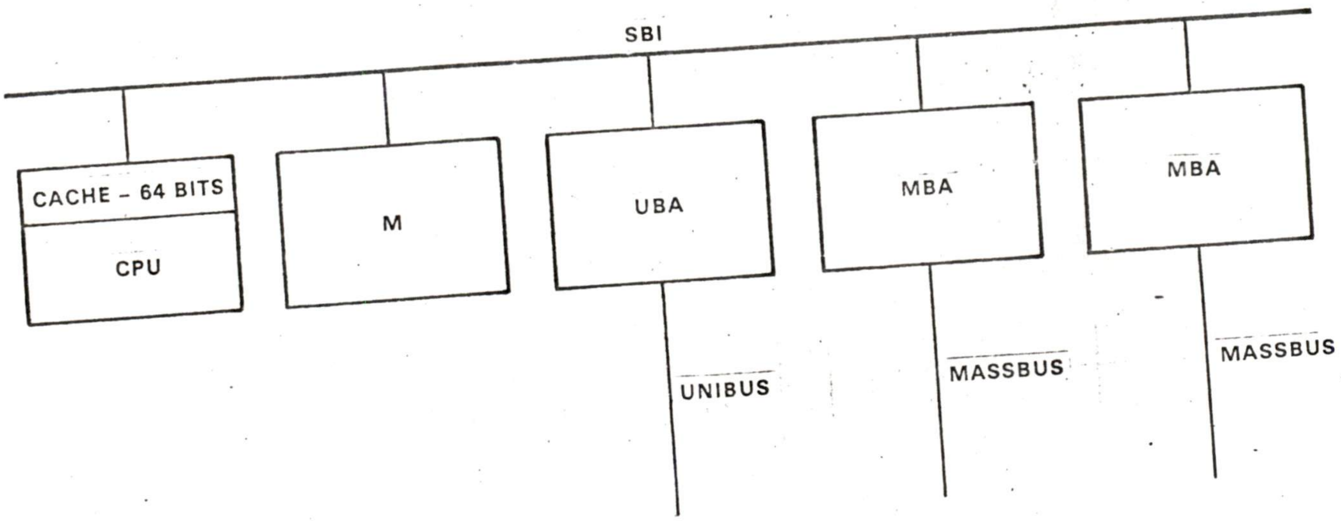


Fig 28

Levy

150



(151)

Fig 29 ~~Levy~~

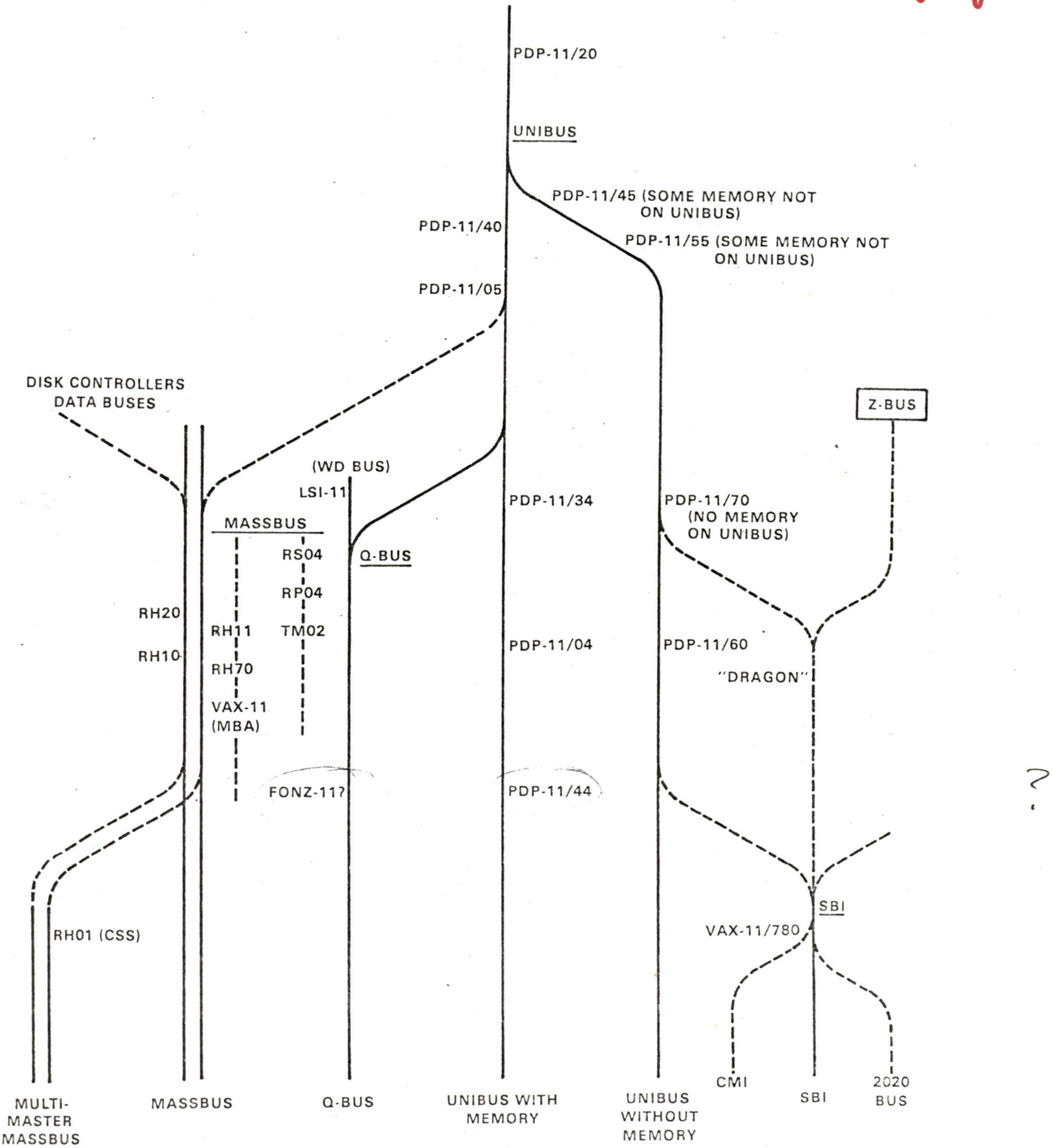


Fig. 29

Levy

157

Table 30

Key design goals of high-performance PDP-11 family systems.

CPU	Date introduced	Design Goals	Resulting evolution relating to buses
DP-11/20	1970	1. Modularity 2. Standardized interfacing 3. User-designed interfacing 4. Mixed memory technology	Unibus
DP-11/45	1972	1. More performance	Dual-Unibus capabili
		2. Introduce MOS +Bipolar memory	"Fastbus"
		3. Larger memory ~~??~~ size	KT11-D memory segmentat unit
	1973	4. ~~High~~ Introduce high speed, high capacity (3330 eq.) disks	Massbus, RH11 controller
DP-11/70	1974	1. More performance	Cache memory, 32 bits wid
		2. Larger memory size	Extended KT11 memory seg. un (also forced to add Unibus map)
		3. Higher ~~speed~~ total DMA bandwidth	Internal 32 bit wide data path for DMA from Massbus Controllers
~~DP-11/55~~	~~1975?~~	~~1. Maximum performance w/ limited mem. size~~	~~all bipolar memory, expanded size (64K bytes)~~ ??
AX-11/780	1977	1. Higher performance	Cache memory, 64 bits wide
		2. Larger Virtual Address Space	New instruction set, 32 bit addresses
		3. Higher total DMA bandwidth	SBI @ 13.3 Megabytes/se +64 bit transfer
		4. Compatibility with PDP-11 peripheral interfaces	MBA Massbus Controllers UBA Unibus adapter I/O page concept retained