december 7 & 8 1976

sponsored by the BTL microprocessor steering committee

BTL/WE microcomputer symposium

BTL/WE microcomputer symposium

Prepared and Published by
the Technical Publication Department

ACKNOWLEDGMENTS

Symposium Sponsor

The Microprocessor Steering Committee

W. E. Danielson - Chairperson
R. M. Allen (WE)
R. C. Fletcher
G. L. Hermansen (WE)
J. W. Schaefer
D. G. Thomas
R. L. Wagner

Symposium Chairperson

R. B. Hawkins

Symposium Assistant Chairperson

C. L. Semmelman

Technical Program Chairperson

J. R. Mc Eowen

Session Chairpersons

C. R. Baugh
A. H. Bobeck
F. H. Henig
W. N. Toy
C. D. Weiss

Technical Arrangements

M. J. Krumm
D. Manolio

Graphic Arts

Technical Publication Department
J. A. Walker - Coordinating Editor

CONTENTS

CONTENTS (Continued)

SESSION 5 - APPLICATIONS II

SESSION 6 - PERIPHERAL TECHNOLOGY

INTRODUCTION

Speaker: I. M. Ross
Executive Vice President

Microcomputer Symposium

The intent of the symposium is to present the latest Bell System experience and expertise in microcomputer applications, to encourage communication among microcomputer users, and to stimulate engineers and designers with the potential of this new technology.

IN-HOUSE MICROCOMPUTERS

A MAC-8 STATUS REPORT

L. C. Thomas, BTL Dept 4391, HO, NJ

ABSTRACT

This report describes the current status and future direction of the components of the MAC-8 development project: the devices, support software and hardware, and user training. The Symposium presentation of this paper will include current schedules and pricing information. This information has not been included here because of the publication deadline.

INTRODUCTION

The purpose of this paper is to describe the development and interaction of the components that are essential for providing a Bell System microprocessor capability. These components are: the MAC-8 microprocessor, software support, hardware interfaces, and user education and application support.

MAC-8 MICROPROCESSOR

Complementary metal oxide semiconductor (CMOS) technology has been used to implement the MAC-8 chip. One of the advantages of this technology is that p-channel and n-channel devices can be used on the same chip. The combination of these devices has produced a variety of innovative circuit configurations that provide the MAC-8 chip n-channel density (about 8000 transistors on a 220- by 220-mil chip) with the low-power dissipation associated with CMOS (less than 150 mW). The details of this design approach are covered in the paper by Cooper and Krambeck.[1]

Another reason for the choice of CMOS technology involves its track record in previous development projects. In a company whose large scale integration (LSI) experience is limited, judgments of track records must be extremely subjective. However, after a 1-1/2 year involvement with the MAC-8 project, we in the Systems Area continue to feel that we have chosen a technology and circuit implementation that promise to become industry standards.

SOFTWARE SUPPORT

In the software development for the MAC-8 microprocessor, two aspects have been emphasized. The first of these is the development of an efficient high-level language to describe applications to the MAC-8. The second item of emphasis is to develop debugging tools in this same high-level language. Possibly a hardware analogy is appropriate here. Obviously, it is much simpler to design logic circuits with medium scale integration (MSI) devices, such as counters and multiplexers, rather than quad gates. A high-level language brings the same advantages to software development. Similarly, a system designed with MSI devices would be difficult to debug on a gate-by-gate basis. Thus, a high-level language should be debugged in its own symbolic, human-oriented terms, not at a lower level. As obvious as these concepts appear, they have not been incorporated into commercial support software. While the high-level language PL/M has been implemented for some commercial microprocessors, it is notoriously inefficient in byte processing and execution time. Its usefulness is restricted to those cases that must emphasize short schedules. The design philosophy regarding software aids, emphasized in papers by Kirby and Rovegno[2] and Shupe, Johnson, and Hofmann,[3] translates directly to reduced development costs and an increased ability to respond to competitive pressures in minimum time. Maintenance and current engineering costs should also be substantially reduced.

HARDWARE INTERFACES

The hardware interface between the MAC-8 dual in-line package (DIP) and the outside world is described in the papers by Blahut[4] and Torres.[5] In these papers, instruction timing, control signal timing, and the utilization of commercially available memories and peripheral devices have been emphasized. The only in-house peripheral device being specifically developed for the MAC-8 is a device

containing read-only memory (ROM), random-access memory (RAM), and input/output (I/O) on a single DIP. This device is described in a paper by Ukeiley and Slemmer.[6] With the exception of this device, emphasis has been placed on developing the central processing unit (CPU) and the user support facilities described in this paper. The demand for any additional members of a MAC-8 family of devices will be determined by system requirements over the next 12 to 24 months. The successful development of a MAC-4[7] device would provide an ideal building block for MAC-8 peripheral devices of all types.

USER EDUCATION AND APPLICATION SUPPORT

The MAC-8 project is confronting the user education and application support problems in a variety of ways.

- Users Manual: A Users Manual will be written by the various hardware and software designers involved in the project and edited by E. J. Angelo. This manual will be available originally in a loose-leaf format; when documentation on all phases of the project becomes complete, a hard-bound version will be published.
- Conventional Courses: MAC-8 courses will be presented in-hours, out-of-hours, in short-course format, and in self-teaching formats beginning in the spring of 1977.
- User Group Application Notes: A MAC-8 user group will be formed and supported. This group will facilitate the exchange of information among MAC-8 users via specific application notes, as well as hardware and software exchanges.
- The MAC-TUTOR: The hardware keystone of the MAC-8 education and applications support effort will be the MAC-TUTOR apparatus. This apparatus will consist of a complete MAC-8 microcomputer system at an objective cost of $350. This system will include a MAC-8 CPU, RAM, a programmable ROM (PROM) debug monitor system coupled with a 20-button keyboard, and I/O capability. The power supply will be included with the apparatus as well as an RS-232 connection for more elaborate program development. A self-teaching booklet will accompany the MAC-TUTOR. The MAC-TUTOR apparatus will be supplied to each student in in-hours, out-of-hours, and short courses to be given at Bell Laboratories. The apparatus will be

considered as the textbook for these courses, and will be paid for by the student's department. For self-teaching purposes, they may be purchased separately. Initial availability of the MAC-TUTOR is expected in the spring semester 1977, with full production quantities expected in June 1977.

REFERENCES

1. J. A. Cooper and R. H. Krambeck, "Design and Fabrication of the MAC-8 Microprocessor Chip," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

2. D. B. Kirby and H. D. Rovegno, "The Programming Languages of MAC-8," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

3. B. B. Hofmann, K. W. Johnson, and C. F. Shupe, "Testing and Debugging of MAC-8 Systems," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

4. D. E. Blahut, "Mac-8 Device Description," paper presented at Microcomputor Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

5. F. B. Torres, "The MAC-8 in a User Environment," paper presented at Microcomputor Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

6. R. L. Ukeiley and W. C. Slemmer, "A ROM-RAM-I/O Device for the MAC-8," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

7. D. C. Stanzione, "MAC-4," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

IN-HOUSE MICROCOMPUTERS

MAC-8 DEVICE DESCRIPTION

D. E. Blahut, BTL Dept 4391, HO, NJ

ABSTRACT

The MAC-8 microprocessor is a single-chip, bus-structured, 8-bit microprocessor. A single 16-bit address bus accesses memory and input/output (I/O), including the sixteen 16-bit working registers and the pushdown stack contained in the random-access memory (RAM). Other features include interrupt, direct memory access (DMA), and reset capabilities.

This paper provides a review of the internal MAC-8 architecture and a detailed description of various input and output signals that will be useful to the hardware designer when interconnecting the 40-pin MAC-8 dual in-line package (DIP) with memory and I/O and peripheral hardware in general.

The Symposium presentation of this paper will include final electrical characteristics and timing waveforms. This information has not been included here because of the publication deadline.

INTRODUCTION

With its innovative architecture, the MAC-8 microprocessor is a powerful device when compared to outside alternatives; however, a detailed description of this structure, its salient and powerful capabilities, and the generous set of instructions and memory accessing modes will not be presented here.[1] The purpose of this paper is to provide a device description primarily for the hardware system designers. Only a brief review of the architecture will be presented in order to more easily describe the interaction between the MAC-8 and its peripherals.

ARCHITECTURE

The MAC-8 is partitioned into four major functional blocks. See Figure 2-1. The control logic array is the nucleus of the microprocessor. It performs the task of directing the circuitry through the sequence of states required to execute each of its instructions and functions. A second logic array controls the function to be performed by the arithmetic and logic unit (ALU), if called upon during the course of each sequence. It also controls the operation of the condition register in latching the flags generated by the ALU. A separate address arithmetic unit (AAU) is provided to expedite the many required address calculations. The last major portion of the microprocessor contains four 16-bit pointers: the program counter (PC), the stack pointer (SP), a temporary register (T16), and the register pointer (RP). The RP points to the location in the 65K memory space allocated to working registers.

One of the advantages of off-chip registers is that a large number of registers (16 in the case of the MAC-8) may be used. Other advantages are not so obvious. For example, a time-critical routine might push the RP and load a new RP value, which, in essence, pushes all registers in one operation. Upon completion of the routine, popping RP returns the original registers to the main program, leaving the registers of the routine untouched until next called upon. Also, a new RP might be selected that partially overlaps the previous set, providing a convenient

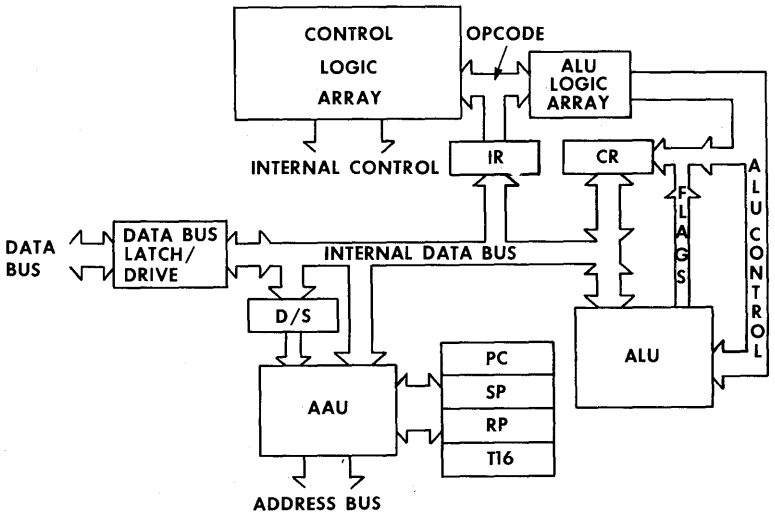


Figure 2-1 - MAC-8 Block Diagram

method for passing parameters and a simplified procedure for deep nesting of calls. The point to be noted is that both the RP and SP might vary over a large range, the extremes of which might be difficult if not impossible to predict. Off-chip knowledge of these pointer values might be fundamental to the administration of proper memory allocation.

A typical MAC-8 instruction consists of two bytes: the first byte defines the instruction; the second byte provides the four bits required to specify the source register and the four bits that specify the destination. These 4-bit nibbles and the RP are combined by the AAU to form the appropriate memory pointers.

PINOUTS

The MAC-8 is capable of accessing 65K of memory or I/O via its 16-bit address bus and its 8-bit data bus. Table 2-1 shows the pinouts. The microprocessor is equipped with DMA, one interrupt, and a reset. Two clock inputs are provided, and a resonator connected between them will activate an internal clock. One of the pins serves as the clock input if an external clock is used. A third pin (clock output) provides a clock for use by peripheral hardware. The data ready lead is instrumental in providing wait states for accessing slow memory or I/O. Finally, three status outputs determine the status of internal components of the chip, such as the values of RP and SP. These status signals are described in more detail in a subsequent section of this paper.

ELECTRICAL CHARACTERISTICS

The MAC-8 is a static microprocessor capable of operating at clock frequencies in the 0 to 2 MHz range. All pins are transistor-transistor logic (TTL) compatible, with each output capable of driving one standard TTL load. Total power consumption is below 200 mW at 2 MHz.

TIMING

Preliminary timing waveforms are shown in Figure 2-2. The address bus and read signal are updated at the start of each machine cycle, which for sufficiently fast memory is a single clock cycle. The read signal specifies the direction of data flow on the data bus and, therefore, remains low for two successive read cycles. The data bus is strobed at the end of each read cycle.

TABLE 2-1

MAC-8 DIP PINOUT

Function		Pin No.
Address	a15	12
	a14	11
	a13	10
	a12	9
	a11	8
	a10	7
	a09	6
	a08	4
	a07	3
	a06	2
	a05	1
	a04	40
	a03	39
	a02	38
	a01	37
	a00	36
Data	d7	14
	d6	15
	d5	16
	d4	17
	d3	18
	d2	19
	d1	20
	d0	21
!Read Clock		22
!Write Clock		23
Data Ready		35
!DMA Request		33
!DMA Acknowledge		34
!Interrupt Request		24
!Reset		25
Clock	Out	32
Clock	Resonator Pin 0	31
Clock	Resonator Pin 1 or In	30
Status	s2	28
Status	s1	29
Status	s0	27
Vcc	+5V	5
Vdd	+12V	26
Vss	0V	13

! is logical negation, indicating active low signal.

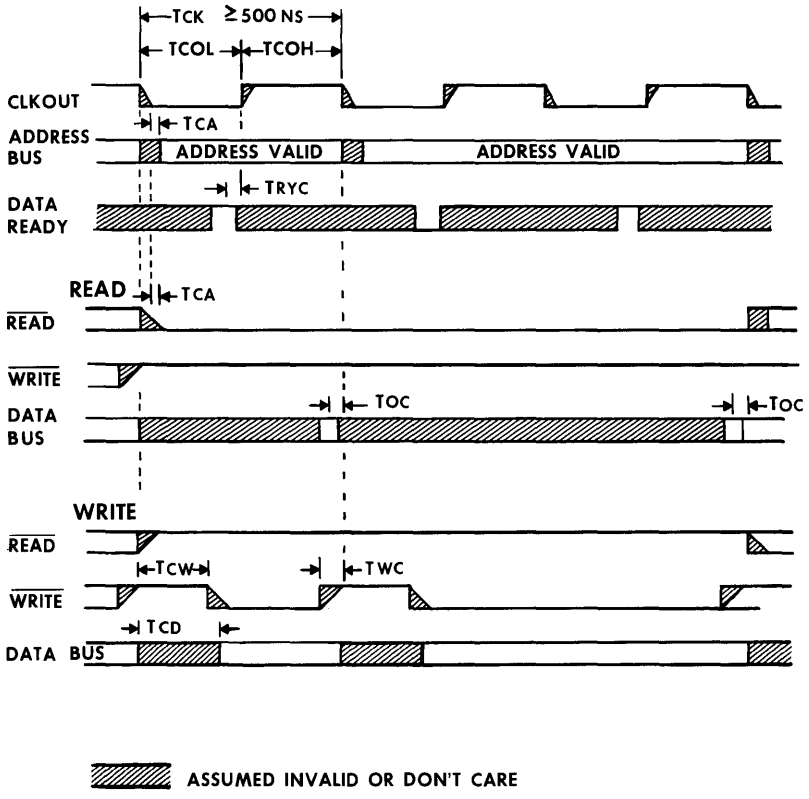


Figure 2-2 - Preliminary Timing Waveforms

A write access cycle includes a write pulse that is activated after the address has stabilized and returns high 50 ns (nominal) before the end of the cycle. Both write and nonaccess cycles have a high-logic level on the read pin, indicating that the on-chip data bus drivers are activated.

Slow memory or I/O ports are accessed via the data ready input. Prior to each midcycle, the data ready input is strobed by the chip. Detection of a low-logic level causes the machine cycle to be extended until a high level is strobed. The machine cycle is terminated during the second half of that clock cycle. It should be noted that the data ready input is ignored during nonaccess clock cycles.

DMA, interrupt, and reset inputs can be changed at any time, and are appropriately strobed into the MAC-8. Interrupt and reset inputs are strobed at the start of the last machine cycle of each instruction or function. A low level on either input begins a sequence of states that pushes the control register (CR) and PC onto

the stack, clears the interrupt enable flip-flop, and applies address X(FFFF) to the address bus. If interrupting, the peripheral hardware should respond to this address with a vector on the data bus that, when combined with a high byte of X(00), forms the pointer to the interrupt handler. Reset automatically vectors to address X(0000). The similarity of the reset and interrupt operations is intentional, allowing the reset to be used as a nonmaskable interrupt.

The logic level on the DMA request lead is monitored at the beginning of each machine cycle. If a request is detected, a low-going DMA acknowledgment is generated as the read and write drivers and the address bus and data bus drivers are switched to their high-impedance states.

STATUS

The remaining MAC-8 pins are the status outputs. These signals indicate when the address and/or data bus contains information about internal status (such as RP and SP) or if an internal condition of special significance exists. Since the above situations cannot always exist, one of the status indications represents "no other reportable status." Seven other independent internal events are each assigned a 3-bit status code. See Table 2-2 .

TABLE 2-2

STATUS ASSIGNMENTS

Function	Status (S2, S1, S0)
PCT Opcode Fetch	000
PC Opcode Fetch	001
RP Change	010
PCF Opcode Fetch	011
No Reportable Status	100
Halt	101
SP Change	110
Trap	111

The RP and SP codes indicate that, during the next machine cycle, the RP or SP, respectively, will appear on the address bus. Each instruction that can result in

a change in either pointer will contain at least one appropriate status signal. The last SP or RP status signal of any instruction or function will correspond to the final correct internal pointer value.

Similarly, status signals PCT, PC, and PCF indicate that an operation code (opcode) fetch is scheduled for the next cycle. As mentioned previously, this is also the cycle in which the interrupt and reset inputs are strobed. If the machine cycle following one of these status indications is a read access cycle, the following statements are true.

- A new instruction has begun.
- The address bus contains the pointer (PC) to the opcode.
- The data bus contains the opcode.

If it is not a read access cycle, the beginning of an interrupt-reset sequence is indicated.

The PCT signal differs from the other two indications because it indicates when the fetch causes a discontinuity in the program flow. This makes it useful for program tracing. The PC and PCF signals correspond to in-line fetches (i.e., no discontinuity). The PCF signal is unique in that it indicates when the otherwise idle data bus is used to output the condition register.

One of the MAC-8 instructions is the HALT instruction, which switches the bus drivers to the high-impedance state and puts out the halt status indication. The status signal remains as long as the state is maintained. The halt state can be terminated only by a reset or an interrupt.

The final status output is trap. The signal is generated when the control logic array gets lost in the execution of an instruction. The only predictable use is when an unassigned or illegal opcode is encountered. A trap results in the chip going through the same sequence of states as an interrupt or reset (the CR and PC are pushed) until the interrupt acknowledge. At this point, control is transferred to address X(0008). Interrupt-reset acknowledge is not generated. The interrupt enable flip-flop, however, is reset.

The unassigned opcodes, of which there are 53, can be used with their functions defined by the trap handler. For example, one-byte calls can be implemented

easily. One application already in use is the insertion of one-byte break points. It should be noted that the value of PC pushed on the stack is two past the location of the invalid opcode.

SUMMARY

The MAC-8 is a bus-structured microprocessor having access to a 65K 8-bit memory space. The simple bus structure makes the accessing of commercially available medium and large scale integration (MSI and LSI) peripherals, including memories, straightforward. The microprocessor is equipped with DMA, interrupt, and reset features. In addition, power-on reset, static operation, low power, and a single-phase internal clock make the microprocessor a viable candidate for a broad spectrum of applications. The memory accessing modes and instructions are generous compared to outside alternatives.[2] A convenient set of status outputs is not only attractive from a hardware point of view for applications such as trace tables, memory allocation administration, sanity logic, and usable unassigned opcodes; but also contributes significantly to the testability (in reasonable detail) of the chip.

REFERENCES

1. D. E. Blahut and R. C. Brainard, MAC-8 Microprocessor Hardware Manual, TM-76-4391-3, TM-76-1353-7, June 17, 1976.

2. S. T. Campbell, MAC-8 Microprocessor Summary, Memorandum for File, Case 38565-2, March 8, 1976.

IN-HOUSE MICROCOMPUTERS

DESIGN AND FABRICATION OF THE MAC-8 MICROPROCESSOR CHIP

J. A. Cooper, Jr., and R. H. Krambeck, BTL Dept 2261, MH, NJ

ABSTRACT

The MAC-8 microprocessor chip was designed and fabricated using silicon-gate complementary metal oxide semiconductor (CMOS) technology. However, many of the circuits are not CMOS, but n-channel metal oxide semiconductor (NMOS) with p-channel load devices. These circuits, called pseudo-NMOS, dissipate more power than CMOS but have layout advantages for gates having more than four inputs. The performance characteristics of CMOS versus pseudo-NMOS as used in the MAC-8 are discussed first.

Next, an overview of the major circuits in the MAC-8 is presented. Internal timing waveforms are derived from CMOS delay elements which partially compensate for processing variations. Control is accomplished primarily by a programmed logic array (PLA), implemented in dynamic pseudo-NMOS. The PLA sequences the microprocessor through the state diagram and provides internal control signals. The control signals produced by the PLA are encoded and must be decoded by CMOS gates which control the register, arithmetic, and logic unit (RALU) section. The RALU contains an 8-bit arithmetic logic unit (ALU), implemented in static pseudo-NMOS, a 4- by 16-bit static CMOS random-access memory (RAM), a 16-bit CMOS address arithmetic unit (AAU), and a master multiplexer implemented with complementary transmission gates.

INTRODUCTION

The object of the MAC-8 microprocessor program is to produce a microprocessor uniquely suited to Bell System needs. The development of the architecture and proposed program set, therefore, involved inputs from many parts of the Bell System. Similarly the choice of a technology to implement the MAC-8 was based upon inputs from several Bell Laboratories departments.

The technology that was chosen was CMOS. The main reason for this choice is the great flexibility in circuit design that is possible when both p- and n-channel transistors are present on the same chip. For example, simple one- and two-input gates, such as those found in the latches and on-chip registers, are made with complementary circuitry. Thus a great advantage in power consumption, compared with NMOS, is gained without using significantly more area. Meanwhile, highly complex combinational gates found in the ALU and PLA are implemented using pseudo-NMOS, just as in NMOS except that the load transistors are p-channel. As a result the areas of these circuits are also comparable to those of their NMOS counterparts. Moreover, since the outputs of these circuits are transferred to static latches, they need not consume static power. Therefore, overall chip power is still comparable to the power consumed by an all-CMOS chip.

Consequently, the gate density of the chip is comparable to NMOS but has a power consumption more typical of CMOS. Flexibility in supply voltages is also typical of CMOS. The chip is expected to operate at 2 MHz with a supply of 12V and with supply voltages of between 4.5 and 13V. A second 5V power supply is required to operate the on-chip transistor-transistor logic (TTL) interface circuitry. Some special features of the chip are TTL-compatible inputs and outputs and a crystal oscillator circuit.

CIRCUIT DESIGN

The flexibility of CMOS permits the use of different kinds of circuits for different parts of the microprocessor. Figure 3-1 shows how the area of the chip is occupied and the kind of circuit used in each section. There are three: CMOS for registers and simple gates, dynamic pseudo-NMOS for the PLA, and static switched psuedo-NMOS for the ALU. These circuits have been chosen for the following reasons.

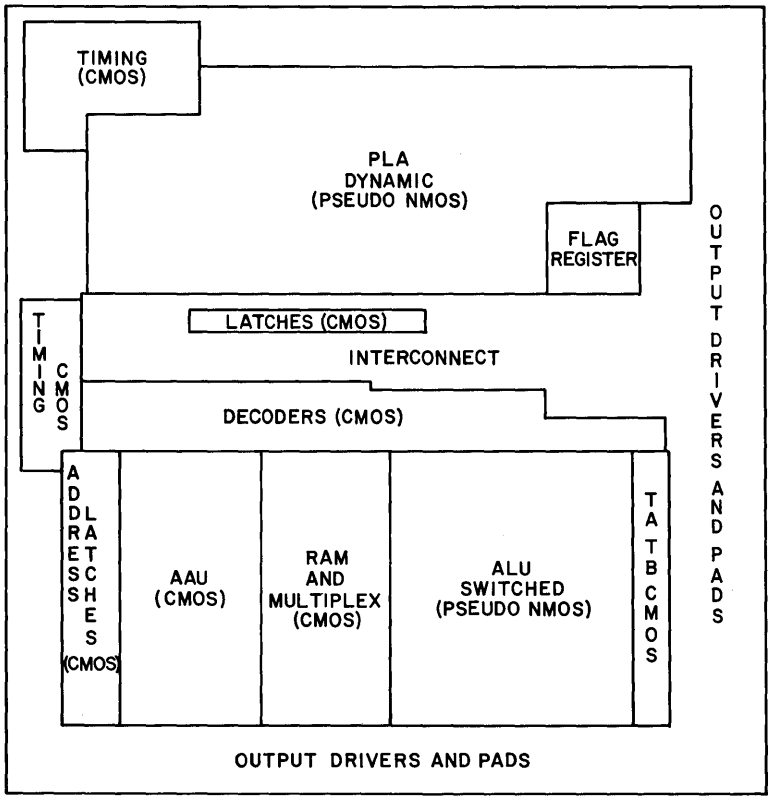


Figure 3-1 - Block Diagram of Chip

A substantial part of the chip area holds registers for storing data and drivers for providing buffering. Figure 3-2(a) shows the circuit used for a pair of cross-coupled inverters using CMOS. The corresponding circuit for NMOS is shown in Fig. 3-2(b). As can be seen, both require four transistors and would take about the same area. However, although both are static, only the NMOS circuit requires significant current to hold its data. It is therefore advantageous to use CMOS for making inverters and registers. There is little penalty in area and negligible static power consumption. For most of the random logic in the control part of the chip, CMOS is also used. A two-input NAND or NOR gate uses four transistors in CMOS, compared with three in NMOS, so some additional area is used, but overall impact on chip area is slight. CMOS is also used for the AAU where most of the gates are also relatively simple.

The gates in the ALU were not well suited to CMOS because of their great complexity. Figure 3-3(a) shows a typical ALU gate using CMOS and Figure 3-3(b) the corresponding circuit using pseudo-NMOS. The CMOS has almost twice as

many transistors and would take about twice the area. However, the pseudo-NMOS circuit has two important disadvantages. First, it consumes power when its output is low. Second, the requirement that the output be low when any n-channel path is in use forces the choice of a poorly conducting load transistor. As a result pullups are much slower than pulldowns. The static NMOS circuit would therefore be relatively tardy.

To alleviate the first problem, a switch was added to the pseudo-NMOS circuit so that it could be turned off when not being accessed. An n-channel transistor

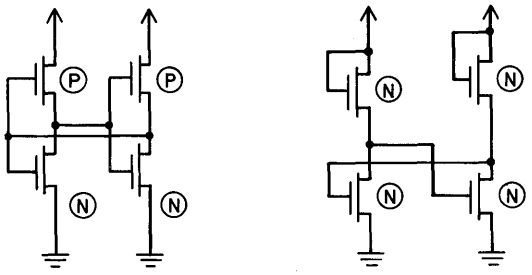


Figure 3-2 - CMOS and NMOS Memory Elements

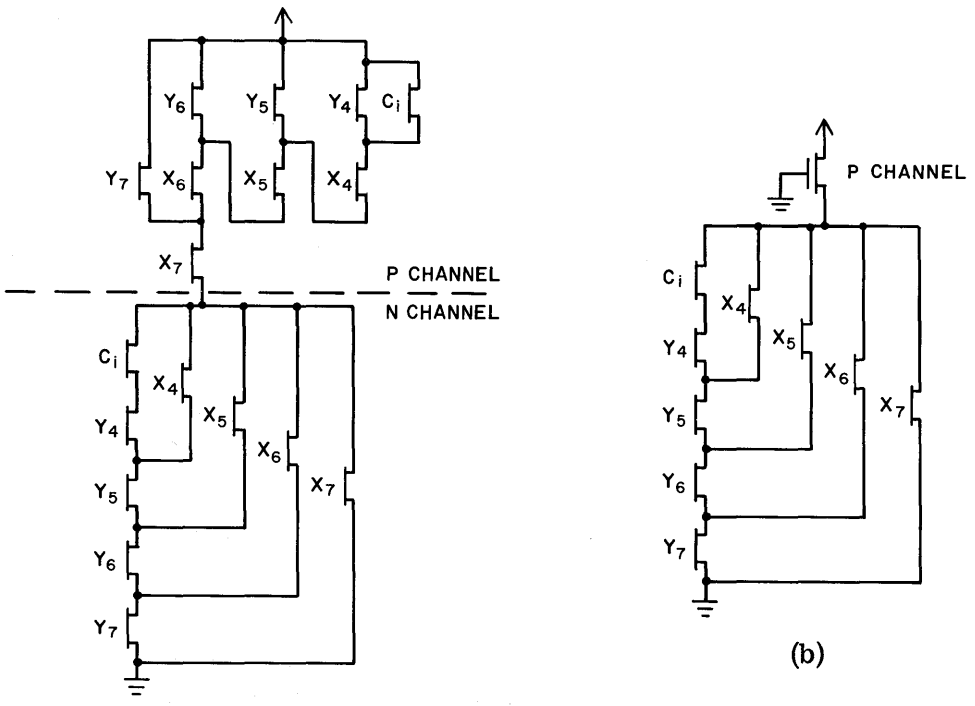


Figure 3-3 - CMOS ALU and Pseudo-NMOS Gates

was used between the circuit and ground. Since the ALU is accessed only once about every six cycles, most of the power drain has been eliminated in this way. The problem of slow pullups is handled by arranging the circuit so that all pseudo-NMOS nodes are precharged high during the intervals between accesses and no pullups ever occur during access. Such an arrangement is illustrated in Figure 3-4. The first stage is turned on only after all inputs are stable. Therefore, its output either stays high or is pulled down. The CMOS inverter driven by the first stage converts the output to a rising signal. (CMOS pullups are fast so this pullup does not significantly increase access time.) As a result, the inputs to the second stage are either fixed or make a low-to-high transition. Therefore, the second-stage output, like the first, stays high or is pulled down. The inputs to the third stage make high-to-low transitions. Therefore, third-stage turnon is delayed until all of its inputs are stable. Finally, the output of the third stage is buffered by a CMOS circuit for delivery to other parts of the chip. The result is a circuit with the area of NMOS but with a small power drain and speed considerably faster than a standard NMOS circuit.

The PLA presents a somewhat different problem. It must be accessed every cycle, so the circuit used for the ALU would not be suitable. Its gates are quite complex, so CMOS would waste considerable area. For the PLA the slow pull-up problem was eliminated by the clocking arrangement shown in Figure 3-5. The p-channel

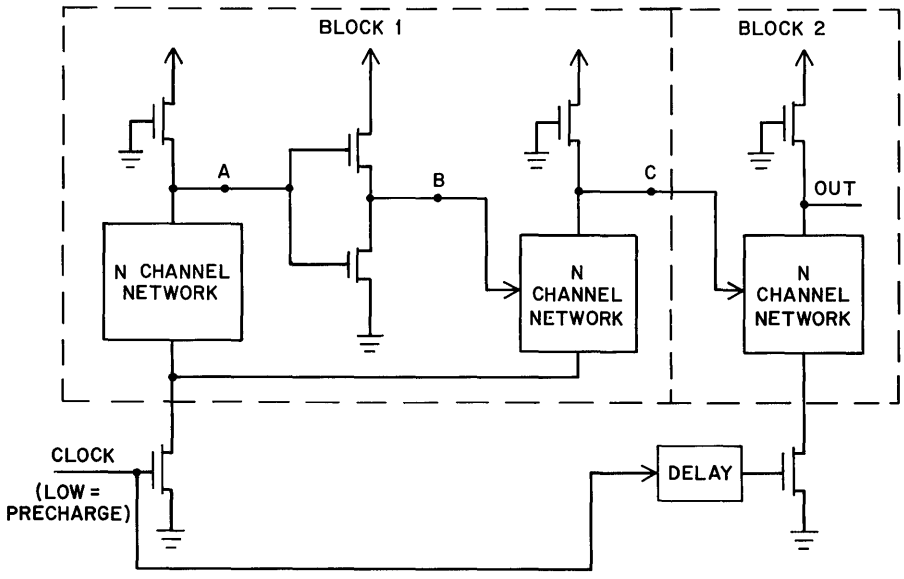


Figure 3-4 - ALU Block Diagram

load is on only when the n-channel switch is off. Each stage is precharged by a low clock signal and accessed by a high clock signal. The delay in access of the second stage is necessary to ensure that its inputs are stable before access begins. The result is a circuit with the area of NMOS but with no static power drain. As in the ALU, speed is maximized by the elimination of pullups during access. Since the outputs are dynamic, each is fed into a latch which is strobed after the second-stage outputs are correct and before precharge for the next cycle begins.

CHIP REALIZATION

The chip was fabricated on the Murray Hill CMOS line. A photograph is shown in Figure 3-6. The chip is 5.9 by 5.55 mm and contains approximately 7500 transistors of which over 80 percent are n-channel. It is shipped in a 40-pin dual in-line package (DIP).

The chip and its constituent parts were tested extensively both on the bench and on the Murray Hill automatic test system, the Sentry 600. Data on access times of various circuits were gathered as an example. Figure 3-7 is a histogram of AAU access time for 16-bit increment. The maximum allowed value is also shown. A sequence of test vectors developed by R. Gallant is also part of the Sentry test and, when complete, will exercise all transistors on the chip. The oscillator circuit has been successfully operated with crystals of 1, 2, 3, and 6 MHz, showing wide margins with respect to actual demands. The power re-

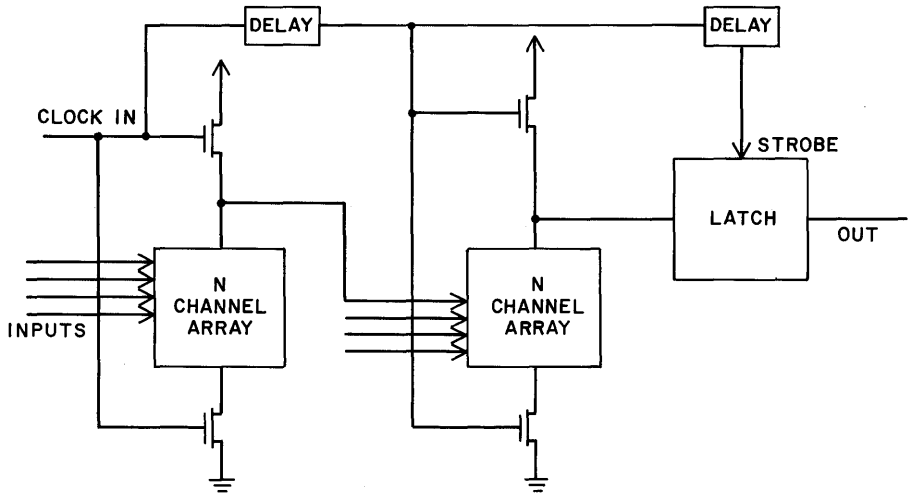


Figure 3-5 - PLA Block Diagram

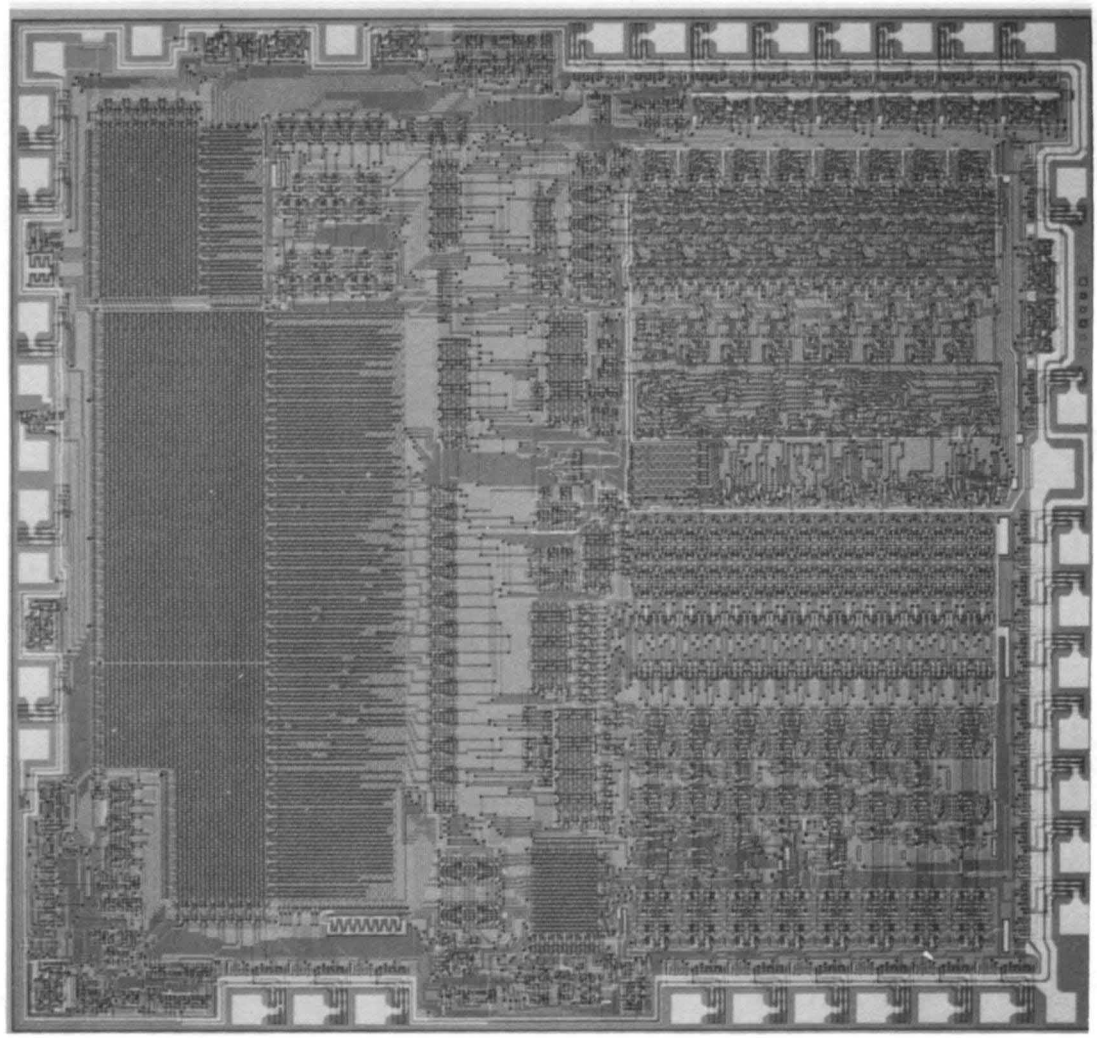


Figure 3-6 - The MAC-8 Microprocessor Chip

quirements as a function of frequency and applied voltage are presented in Figure 3-8. Packaged samples are now available or soon will be, and Bell Laboratories design information (LDI) transmittal is scheduled for January, 1977.

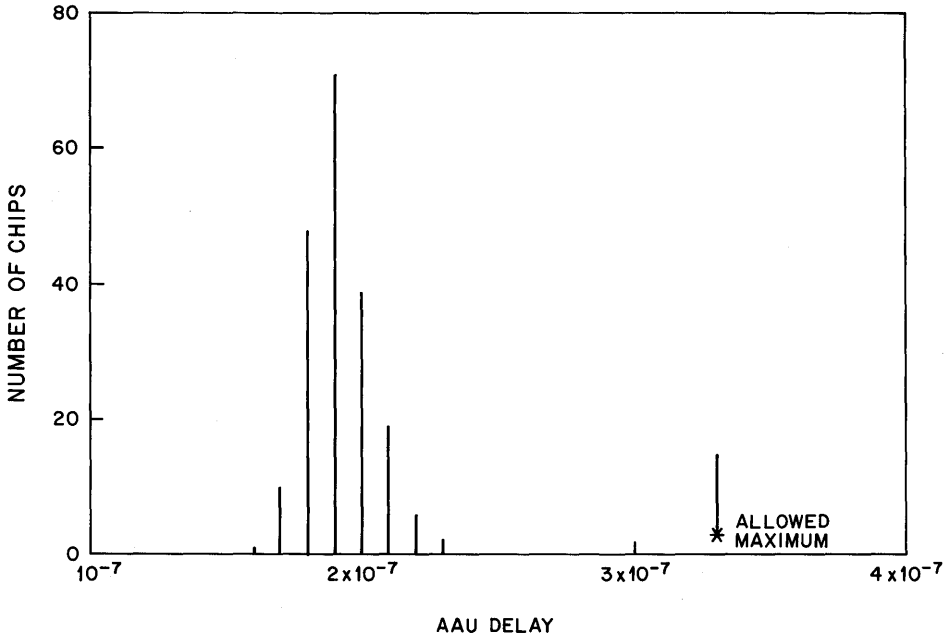


Figure 3-7 - AAU Delay for 16-Bit Increment for 196 Functionally Good Chips

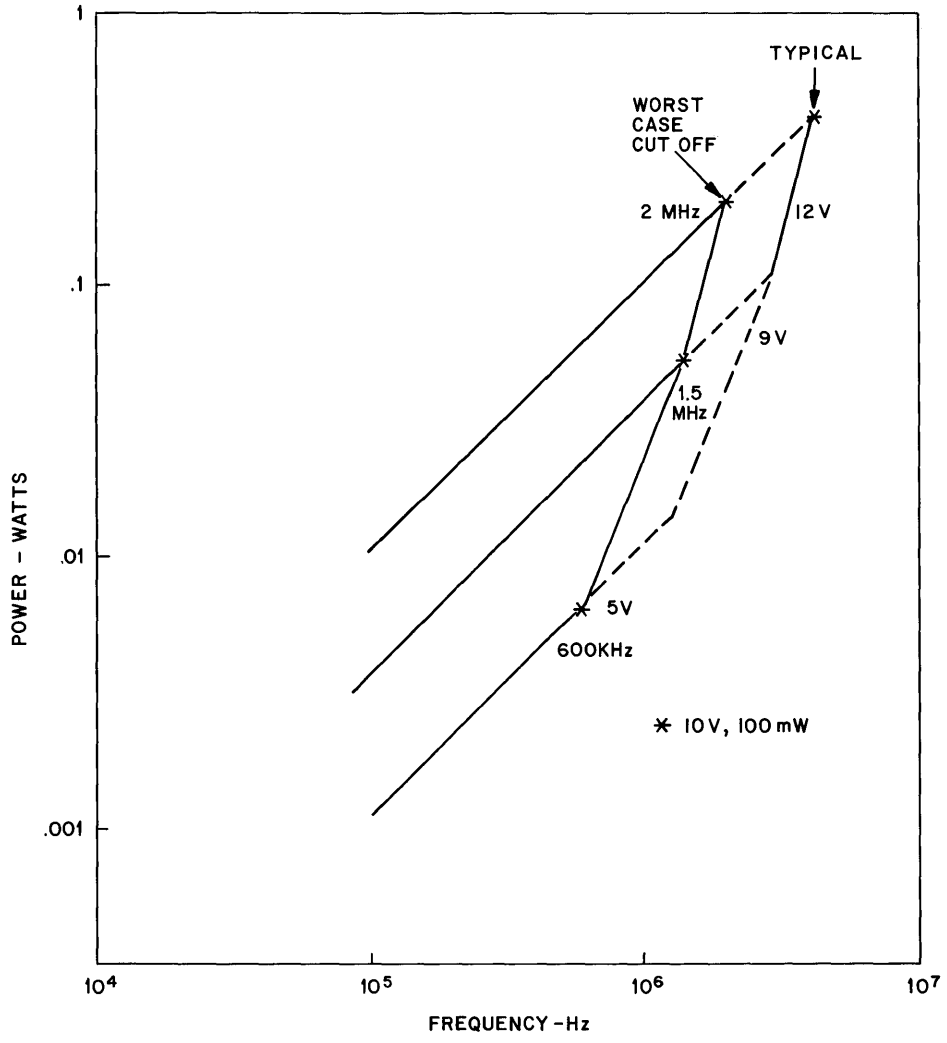


Figure 3-8 - Power vs. Frequency, Showing Worst Case and Typical Frequency Cutoffs

IN-HOUSE MICROCOMPUTERS

TESTING AND DEBUGGING OF MAC-8 SYSTEMS

C. F. Shupe, BTL Dept 8623, HO, NJ
K. W. Johnson, BTL Dept 4391, HO, NJ
B. B. Hofmann, BTL Dept 4393, HO, NJ

Testing shows the presence, not the absence, of bugs.

E. W. Dijkstra

A debugged program is one for which you have not yet found the conditions that make it fail.

J. Ogdin

Debugging is an art.

E. Yourdon

ABSTRACT

The main objective in designing the MAC-8 microprocessor software testing tools was to give the MAC-8 user the ability to test at the source code, as opposed to the object code, level. The goal was achieved by using modern software testing techniques from the data processing industry.

The MAC-8 software testing tools are (1) the MAC-8 software simulator (m8sim) and (2) the program logic aid (PLAID) development system. This paper describes the high-level functional design of m8sim/PLAID and its implications for the MAC-8 user. An important aspect of the design is the compatibility between m8sim and PLAID in spite of the differences in their host environments.

INTRODUCTION

The testing of any piece of software is apt to be slowed by the discovery of an error. The discovery of the error's cause (the bug) is termed debugging. Correcting the bug allows testing to resume.

Testing, debugging, and correcting are facts of life in any software project. The confusing aspect of microprocessor software is the obvious hardware nature of the processor itself. After all, when we program a large computer in a high-level language we don't worry about the interrupt pending bit or other real-time considerations. We also tend to think of a microprocessor plus software in terms of its hardware equivalent. But the reality remains that to have a microprocessor perform useful work, we must write, test, debug, and correct software. Any real-time aspects of the application only make these tasks more challenging.

This paper describes the high-level techniques and capabilities that have been designed for the testing, debugging, and correcting of software for the MAC-8 microprocessor. Some features will not be implemented by the time of publication.

In the MAC-8 support system (see Figure 4-1), software source code is first written in the C and/or assembly language and is then compiled and linked into object code. An example of source code is "A=B;" and its corresponding object code is a numerically coded MAC-8 move instruction which includes the absolute addresses for symbols A and B. MAC-8 object code is tested interactively by the software simulator m8sim[1] or, in a hardware environment, by PLAID. The m8sim and PLAID are the MAC-8 debugging aids and the components of the simulator/PLAID (S/P) subsystem for the MAC-8.

Software testing consists of varying certain conditions (such as input data), observing how the software reacts, and comparing this behavior with the expected behavior. With m8sim all inputs to the program are contrived by the user, and the behavior of the program is monitored solely through the output that m8sim provides. In PLAID the user program runs in its intended real-time environment in which the user can modify the inputs and monitor the behavior of the program. The user by means of either debugging aid seeks the cause of erroneous software behavior. Once the cause is found and a cure is devised, the user makes the correction at the source level and obtains new object code. Testing then resumes

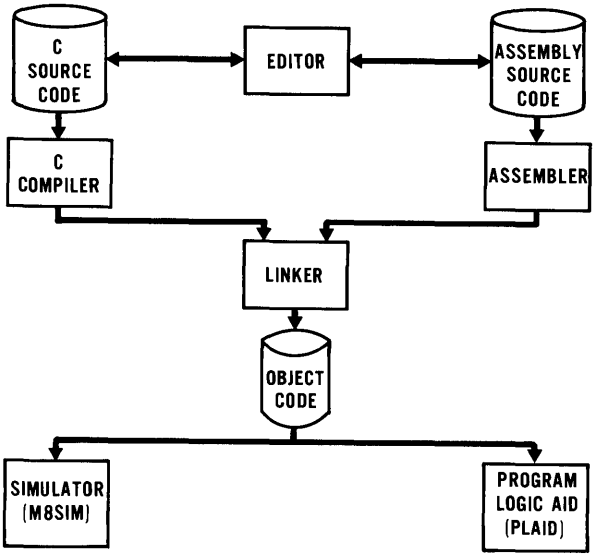


Figure 4-1 - MAC-8 Support System

on the now-modified program. It is important to note that the user software is exercised at the object code level whereas the user is thinking in source code throughout the process.

THE ASPECTS OF HIGH-LEVEL DEBUGGING

Our knowledge of high-level debugging, developed and refined in the large-computer environment, has been applied to the MAC-8 microprocessor.[2]

The Source Code Philosophy

Early in the development of the MAC-8 support system (see Figure 4-1), the source code of the user was defined as the basic unit of the system. This philosophy resulted in the following features.

- The editor, which is used to enter and change the source code, is line-number driven as well as context driven.

- Compiler and assembler diagnostics reference the line number of offensive source code.

- All user software changes are made at the source code level, ensuring that source and object are always in agreement.

- The m8sim and PLAID allow the user to debug using the symbolic names in the source code, in addition to providing a line-number mapping between source and object.

The Command Language

The second aspect of high-level debugging is the necessity for a command language with the following characteristics. First, interactive aids, which require a command language for communication of requests, are necessary for efficient debugging. Second, when there is more than one debugging aid (as is common in microprocessor applications) a single command language is desirable for the sake of user sanity. And finally, the high-level command language for the debugging aid must be consistent with the environment in which it is used.

Of these three requirements, matching the command language to the user's needs as well as to the operating environment is the most difficult. The user acquires a working knowledge of the C language, used in both the MAC-8 C compiler and the assembler and of the UNIX operating system, host of the MAC-8 support system (auxiliary support for PLAID). As m8sim and PLAID are in a UNIX/C environment with which the user is familiar, we deduce that the command language for these debugging aids must somehow be consistent. But since neither the C language nor the UNIX command language alone was suitable as a debugging aid command language, an S/P command language was purposely designed to incorporate the best features of each: the conventions for entering m8sim and PLAID commands, plus the underlying command syntax, are in agreement with UNIX; the C influence appears most obviously in the expressions and value assignments as written in the command language.

Let's consider the C language text "A=B+2*D;", which in English reads: "Assign the value of the expression B+2*D to the symbol A." This is a typical C statement which a user might write for a MAC-8 application in which A, B, and D are symbols known to the C compiler. However, expressions and value assignments are also essential for displaying and changing values in the debugging process. As a result of the S/P command language design, the statement "A=B+2*D;" is also a valid command for both m8sim and PLAID and has the same meaning for them as

it does for the C compiler. Thus a MAC-8 user need know only one way of expressing "A=B+2*D;" to be immediately able to evaluate proposed source-level changes in either m8sim or PLAID.

The S/P command language allows the user to enter explanatory comments in C syntax, along with the commands themselves. Comments are as useful in the debugging process as in the writing of source code and the command language is consistent in this respect with C.

By providing a generalized based-number representation which permits the user to express quantities in any reasonable number base the S/P command language has surpassed even the C language. There are additional important similarities, which we will not discuss, between C and the command language. In summary the consistency of the S/P command language with the source languages and operating system is an important aspect of high-level debugging.

Functions

The third and most significant aspect of high-level debugging is the collection of functions which the debugging aid can perform for the user. In this regard the techniques of large-computer debugging can serve the particular needs of the microprocessor user.

In practice the user enters a command which causes m8sim or PLAID to perform a particular function, the selection and sequence of which must always be under user control. The m8sim and PLAID functions fall into one of the following categories: basic, simulation/execution, or auxiliary functions.

- Basic Functions: The underlying power of m8sim and PLAID is derived from the user's ability to converse symbolically, as well as in absolute terms. The symbols are defined in the source code of the user, by the MAC-8 architecture (including the on-chip registers, such as the program counter, and the user registers in random-access memory [RAM]), or by the user within m8sim or PLAID. The latter type of symbol includes timers and pseudovariables, which are treated like user program symbols but reside outside of the program memory space; the actual use of timers and pseudovariables will be

discussed later. Numerical quantities may be expressed using a based-number convention, which is supported for bases 2 through 36.

Symbols and numbers may be combined into expressions using a subset of the C language arithmetic operators. Examples are the familiar addition and subtraction, the logical operators, and the ever-useful "address of" and "pointed to" operators. In fact, m8sim and PLAID are great based-number calculators because of their expression-evaluation capability, but their power and flexibility really become evident during the simulation (in m8sim) or execution (in PLAID) phase of a user program.

There is, of course, the necessary function for displaying values of symbols, registers, and expressions. Just as important is the value assignment, mentioned previously, by which the user can change the registers, symbols, or pseudovariables. Another function lists all registers.

- Simulation/Execution Functions: The simulation/execution monitoring functions for a user program are concerned with (1) memory management, (2) simulation/execution control, (3) breakpoints, and (4) timers.

  Memory management functions allow the user to define the memory space, i.e., to specify read-only addresses and input/output (I/O) ports, and to load user programs into the simulated or real memory. The control functions cause actual simulation or execution to begin. Breakpoint functions allow the user to define and remove interruptions of simulation/execution. Breakpoint definitions are classified as conditions and actions. Conditions are written as expressions, with the ability to specify user memory read and/or write accesses as operands. The actions are sets of S/P commands, to be executed whenever the conditional expression is true. In addition to the traditional breakpoint sequence in which control is always returned to the user, simulation/execution may continue automatically if the user desires. A pseudovariable, for example, can be automatically incremented into a subroutine; the user may later interrogate this quantity as an indication of subroutine usage.

There are two types of timers: range timers, which are active only when the program counter is in a specified address range, and extent timers, which are turned on or off under explicit user control. Range timers are particularly useful for measuring code efficiency, whereas extent timers are more useful for real-time debugging and critical path evaluation.

A noteworthy feature of the simulation/execution function is the ability to inform the user of how simulation/execution is progressing by printing out the source code which corresponds to the simulated/executed instructions. This code is extracted from the original source file of the user and includes comments made in the source language.

Traces, the past history of simulation/execution, are most useful in PLAID where a program may be run in real time, stopped, and then diagnosed. There are two types of traces. One is a list of recently simulated/executed instructions, and the other gives a history of program control transfers.

- Auxiliary Functions: A high-level function lets the user avoid the typing of repetitious command sequences. The user defines a named set (a block) of S/P commands; instead of a regular command, the user can then enter the block name to execute the set of commands.

  Another high-level feature allows the user conditionally to execute an S/P command. This option is useful for establishing a decision sequence within breakpoint actions. The presence of a conditional in the command language, incidentally, is another similarity to the C language.

  The help function, as its name implies, is designed to give the user reference information about the S/P on demand. It is essentially a built-in reference manual which can easily be updated to make the latest information available to the user.

  Other functions offer the user the ability to: (1) interrupt, and later resume, a debugging session with no loss of effort; (2) exercise control over the content and format of the m8sim/PLAID output; and

(3) draw command input from a file. There are other functions which we will not discuss here.

THE SIMULATOR AND PLAID ENVIRONMENTS

The high-level debugging features, just described, are available in both the MAC-8 simulator m8sim and PLAID (when connected to UNIX). The rationale for using both m8sim and PLAID is, of course, based on (1) the wide availability and low cost of m8sim and its utility in debugging logical programming errors, and (2) the ability of the PLAID user to diagnose real-time, software, and hardware problems and to verify real-world operation. Of necessity, then, there are some differences in the environments and uses of m8sim and PLAID.

The MAC-8 Simulator (m8sim) Environment

The host for the MAC-8 support system (see Figure 4-1) is the UNIX operating system, which provides time-shared computing for multiple users. Each user accesses UNIX through a remotely located ASCII terminal à la modem. In preparing a software program for the MAC-8 the user builds a source code file using the UNIX text editor. The source code is compiled, assembled, and linked, resulting in an object code file. At this point the user may invoke m8sim and evaluate the functioning of the application program. Initially, in the testing phase the incidence of bugs is high, requiring numerous iterations of debugging and correcting the source. Although the user leaves m8sim to make source corrections and to construct a new object file the overhead time involved is minimal. At some later point the user becomes confident that the m8sim testing has revealed most logical programming errors and is thus ready to test in the hardware environment.

The PLAID Environment

Before actually discussing the PLAID environment we first need to mention its architecture and options. PLAID is itself a hardware unit to which the user connects a standard ASCII terminal in order to communicate with PLAID and, through PLAID, with UNIX. There is a built-in modem for establishing the phone line tie with UNIX. Although PLAID may be operated with reduced capabilities without UNIX, we assume in the following discussion that the PLAID-UNIX connection has

been established. An external microprocessor access and control (MAC) cable enables connection to be made with the user application system hardware. (The free end of the cable clamps onto the processor dual in-line package [DIP] socket in that system.) Finally, as an option a floppy disc may be attached for the local storage of object code and object code symbol tables. The user may thus minimize the transmission time of new object code from the UNIX system and utilize the local linker for the object software.

PLAID has an internal master subsystem, which performs executive and command processing functions, and a slave subsystem, which can be used in a variety of useful ways. The slave has a bus to which are connected the MAC cable, a MAC-8 microprocessor, a 65K RAM, and 36 I/O ports. Any combination of 1K RAM segments can be overlaid by 1K programmable read-only memories (PROMs), the procedure followed in the later stages of the development of an application. The slave I/O ports can be enabled or disabled in groups of 12.

PLAID may be operated in the basic mode in which case it functions very much like m8sim. With the slave processor executing the user program from the slave RAM and the program I/O directed through the slave I/O ports, the user will find few functional differences between m8sim and PLAID. The basic mode is typically employed during the initial implementation of a system when hardware development and software development are proceeding simultaneously. The fact that the three components of the slave - the processor, memory, and I/O ports - can be independently substituted for the user processor, memory, and I/O ports gives the user very powerful hardware prototyping capabilities. In a typical situation the user program in slave RAM will be executed by the user processor with a mixture of slave and user I/O ports, while execution is monitored by the PLAID master via the MAC cable. As the user hardware grows during development, less and less of the slave hardware is used, until such time as the system is complete. After completion PLAID is also quite useful in monitoring system operation and diagnosing hardware and software failures. Consequently, throughout the life of the user system, PLAID allows the user to test in real time (with the possible exception of breakpoints) with real I/O activity.

SUMMARY

The MAC-8 support system has proven the initial assumption that state-of-the-art techniques drawn from the large-computer environment can be successfully applied

to microprocessors. Specifically, the high-level facilities in m8sim and PLAID are designed to ease the testing, debugging, and correcting burden by providing the necessary interfaces between the user and the user object code. We emphasize the advantages of a single command language for both m8sim and PLAID. In addition the combination of m8sim for simulation and PLAID for execution monitoring and hardware prototyping gives the user maximum flexibility in the testing and debugging of MAC-8 systems.

We would like to acknowledge the many people who, through their suggestions and criticisms, have contributed to the design and implementation of both m8sim and PLAID. Steve Campbell, in particular, deserves recognition for his participation in the original design process.[2]

REFERENCES

1. C. F. Shupe, MAC-8 Simulator (m8sim) User Manual, Version 1.2, Memorandum for File, Case 39898-14, June 30, 1976.

2. S. T. Campbell and C. F. Shupe, MAC-8 Simulator/PLAID Design Specification, Memorandum for File, Case 39898-14, June 30, 1976.

IN-HOUSE MICROCOMPUTERS

THE MAC-8 IN A USER ENVIRONMENT

F. B. Torres, BTL Dept 4391, HO, NJ

ABSTRACT

The first bona fide application of MAC-8 in a true working environment is the PLAID console. A wire-wrap version of PLAID, which has been built and tested, demonstrates the relative ease with which MAC-8 works with a variety of peripheral devices, including the following:

- Intel 2116, 16K by 1 dynamic random-access memory (RAM) with associated 3242 address multiplexer/refresh counter.
- Intel 2708, 1K by 8 programmable read-only memory (PROM).
- Intel 8255, programmable peripheral interface.
- Intel 8251, universal synchronous/asynchronous receiver-transmitter (USART).
- Intel 8253, programmable interval timer.
- Intel 8259, programmable interrupt controller.
- Vectron external clock oscillator.

PLAID has also shown that MAC-8 functions harmoniously with external circuits providing such features as direct-memory access (DMA) capability, programmable wait state generation for slow memories, write protection for selected RAM locations, and overlay of RAM locations by PROM and input/output (I/O). In addition it has been demonstrated that MAC-8 can indeed be used in a multiprocessor environment.

Other characteristics of the MAC-8 which have been brought to light by PLAID are the ability to latch the contents of the on-chip registers from the address and data buses, to recognize when the off-chip registers are being accessed, and to snapshot the old and new values of the program counter (PC) in cases of such PC discontinuities as JUMP or CALL.

The purpose of this paper is to present selected circuits used in PLAID which illustrate some of these features.

INTRODUCTION

The PLAID console contains two MAC-8 microprocessors and numerous peripheral circuits. The processors work together as master and slave. The peripheral circuits perform functions such as dynamic RAM, PROM, I/O, wait states, write protection, DMA, interrupt priority, user system monitoring, and links to a terminal and UNIX. PLAID demands that the MAC-8 be capable of functioning with a wide variety of peripheral devices, including another MAC-8. The wire-wrap version of PLAID demonstrates that these demands have been met with relative ease.

This paper discusses several circuits controlled by MAC-8 which are used in PLAID.

RAM

The RAM circuit uses the Intel 2116, 16K dynamic RAM and the Intel 3242 address multiplexer and refresh counter. Since timing constraints at a 2-MHz clock rate prohibit both memory refresh and access in the same cycle, another method of refresh had to be used. Because it slows down the processor, cycle stealing was ruled out in favor of memory quadrant rotation, a scheme which causes the MAC-8 to access successive quadrants of memory on incremental PC values (see Figure 5-1). While one quadrant is being accessed, the other three are being refreshed. Address bits A_0 and A_1 are decoded to indicate to the four Intel 3242s which quadrant is being accessed. The Intel 3242s apply address bits A_2 through A_{15} in two groups of seven bits to the quadrant being accessed and the internal counter addresses to the quadrants being refreshed. During

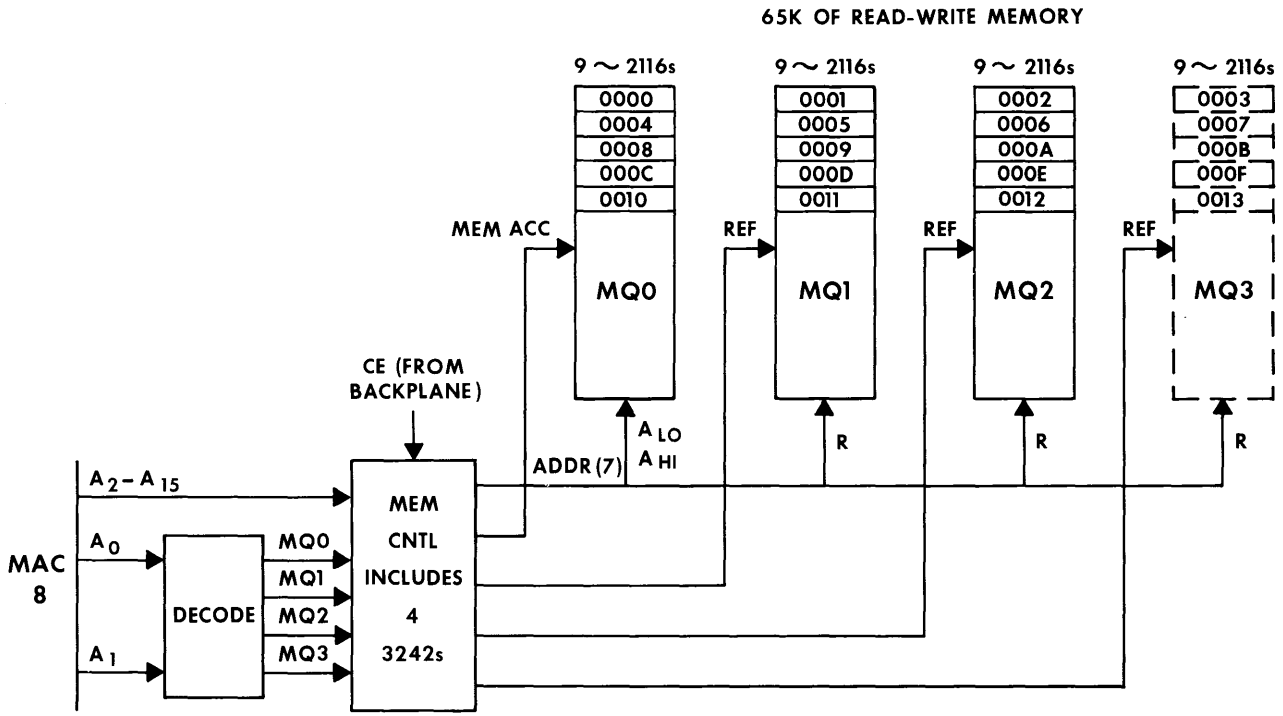


Figure 5-1 - Memory Quadrant Rotation

clock cycles when RAM is not being accessed, all four quadrants are refreshed. This method of refresh control permits operation of the MAC-8 at frequencies well below 2 MHz without danger of losing RAM data. We recognize the fact that successive PC values are not always continuous, and occasionally the same quadrant may be accessed several times in succession. The chip enable (CE) input to the memory control allows RAM to be placed in a high impedance state, permitting PROM and I/O to overlay portions of the 65K available memory space.

Figure 5-2 shows one quadrant of the actual circuit used in PLAID for RAM control. Row address select (RAS), column address select (CAS), row enable (ROWEN), and COUNT are derived from the MAC-8 clock on the central processing unit (CPU) board. The write pulse (WR) is taken directly from the MAC-8. The memory quadrant 0 (MQ0) being accessed is derived from the two least significant address bits. These signals provide the Intel 3242 with the information it needs to control the RAM devices in quadrant 0 in either a memory access or refresh operation.

PROM

The PROM circuit (see Figure 5-3) uses the Intel 2708, 1K by 8 PROM. Address bits A_0 through A_9 select the appropriate word in PROM, bits A_{10} through A_{13} select which of the 16 PROMs to access, and bits A_{14} and A_{15} are compared with the quadrant select switches to determine if, in fact, PROM is being accessed. The PROM present switches are also checked to see if the desired PROM is in its socket and not turned off. If all conditions are met, the PROM circuit is given access to the MAC-8 data bus, and the CE signal is forced low to keep the RAM circuit turned off.

WAIT STATE GENERATOR

Figure 5-4 illustrates a wait state circuit that can be used with MAC-8. The decoder selects the addresses at which wait states are required. In this example a single wait state is provided by the DELAY flip-flop when the address falls in quadrant 0 or quadrant 3.

Assume that, prior to cycle N, A_{15} is high and A_{14} is low. This arrangement forces DCD low and the DATA READY signal high. DELAY is high since the

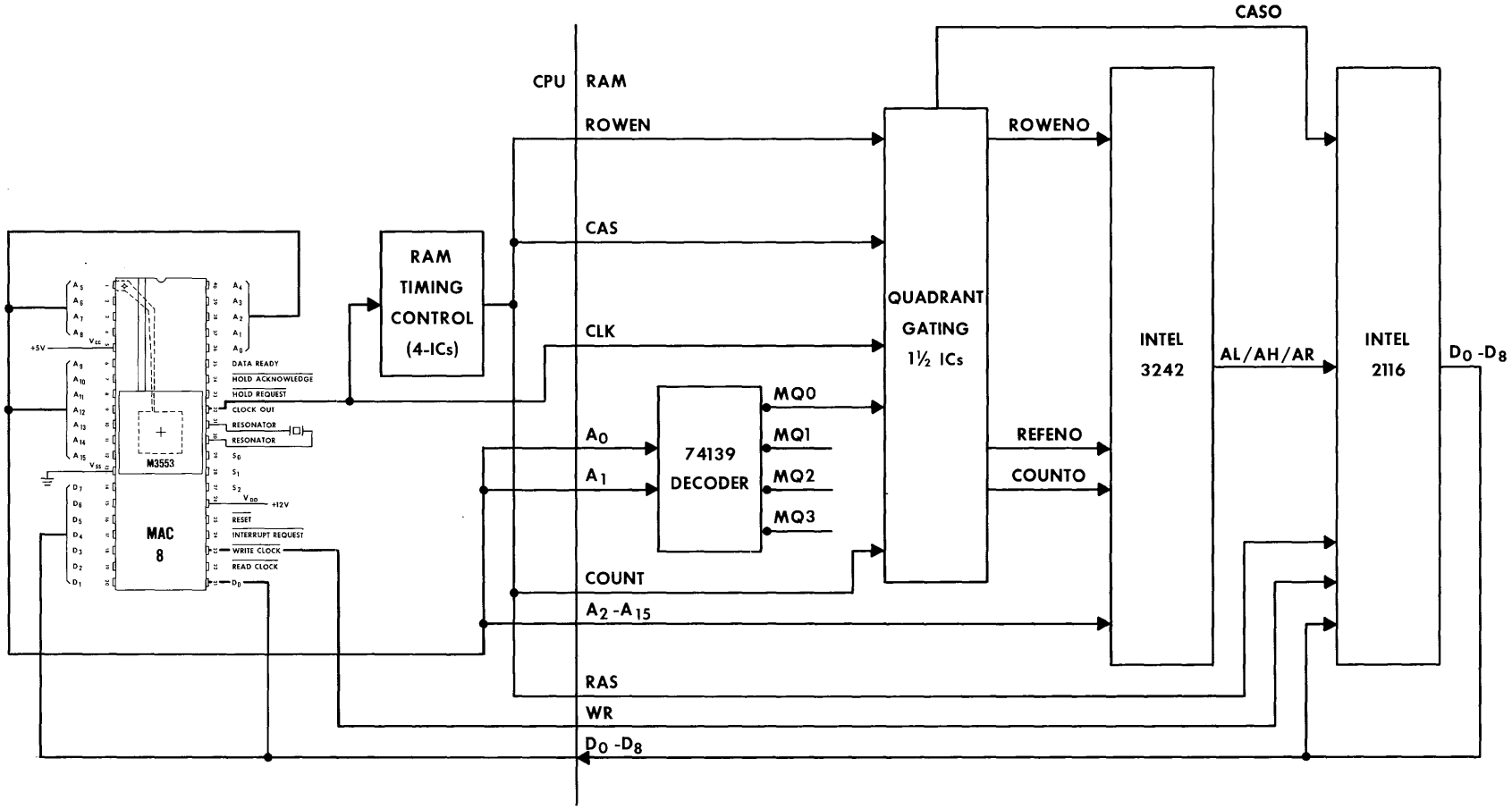


Figure 5-2 - RAM Control

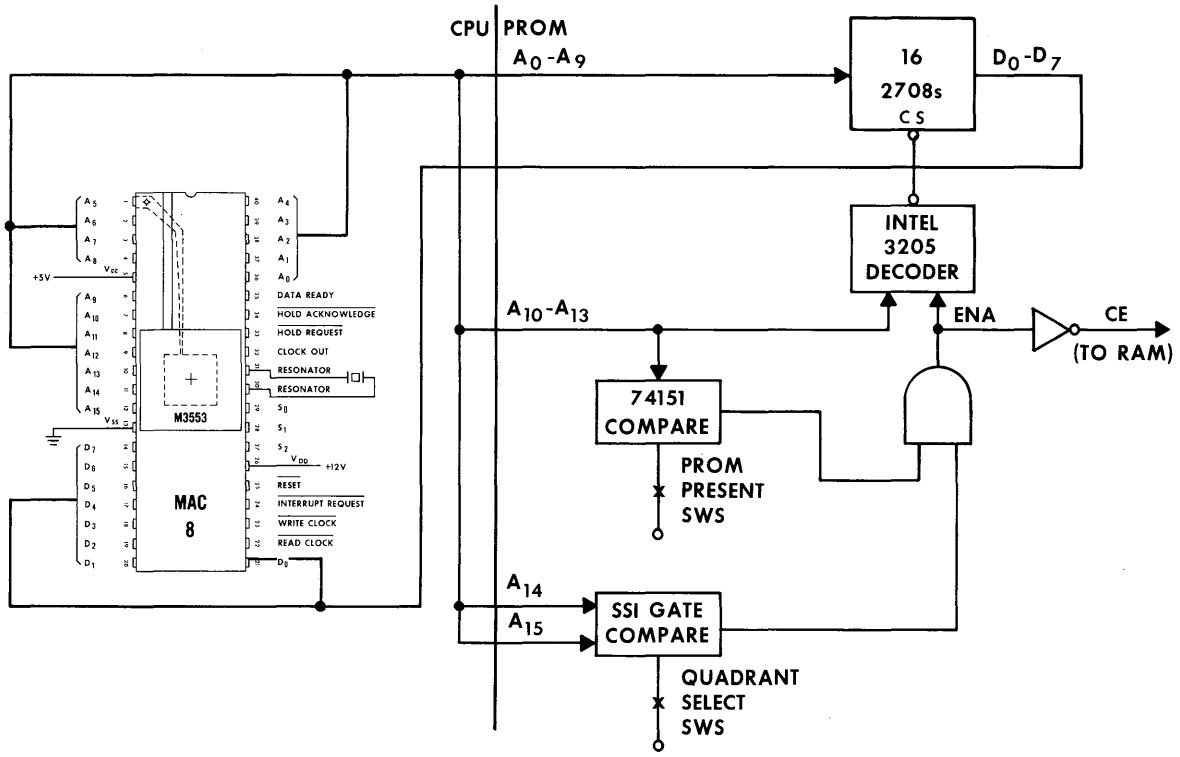


Figure 5-3 - PROM Control

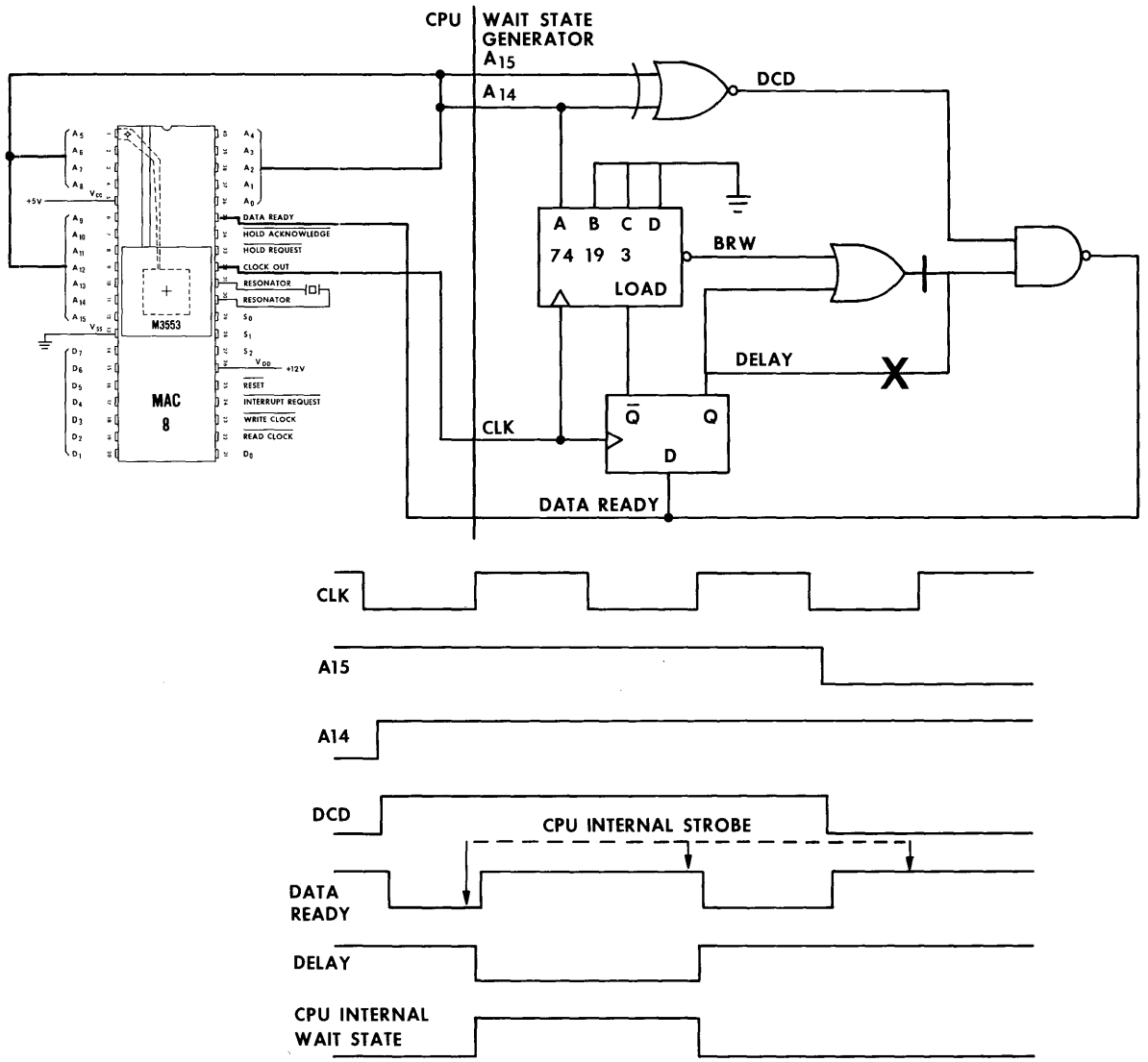


Figure 5-4 - Fixed Wait States

value of DATA READY is clocked into the flip-flop. After the start of cycle N, assume that A_{14} goes high, forcing DCD high and DATA READY low. At mid-cycle the MAC-8 strobes the DATA READY signal, finds it low, and sets its internal wait level. DELAY is cleared at the positive clock transition, and DATA READY goes high again. During cycle N+1 the strobe finds DATA READY high and clears its internal wait level. DELAY is again set, forcing DATA READY low. If, in cycle N+2, the new address falls in quadrant 1 or 2, as depicted in this example, DATA READY is restored to its high state and remains there until the next time a wait state is required. If the new address is still in quadrant 0 or 3, the cycle is repeated. In this example the memory access called for by the MAC-8 in cycle N was delayed until cycle N+1, hence a single wait state.

If more than one wait state is required for slower memory applications, a counter may be inserted into the circuit. Here the number of wait states is determined by the value loaded into the counter, plus one for the DELAY flip-flop. The example shows two wait states being generated, but larger values may be used. The number of wait states may be variable for different address locations and may indeed be under program control.

PLAID uses a wait state generator which is under program control. The circuit shown in Figure 5-5 uses a Fairchild 93L422, 256 by 4 RAM. Three of the four bits are used to provide from zero to seven wait states for each of the 256 locations and are loaded into the counter discussed above. Since the control RAM locations are accessed by the slave MAC-8 with its eight high-level address bits, each location represents a page (block of 256 bytes) of MAC-8 memory locations. The control RAM, which is initialized by the master MAC-8 under control of software, also contains a memory disable bit which is used to turn off preselected pages of slave memory. This feature can be used to implement write protection or to allow the user to simply replace part or all of the slave memory with that of the user's own.

DMA

DMA in PLAID is mainly used for the exchange of data between master and slave memories. The circuit, illustrated in Figure 5-6, makes use of the National DM8542 I/O register and two counters. The counters, preloaded by the master with the starting addresses of master and slave, may be allowed to autoincrement

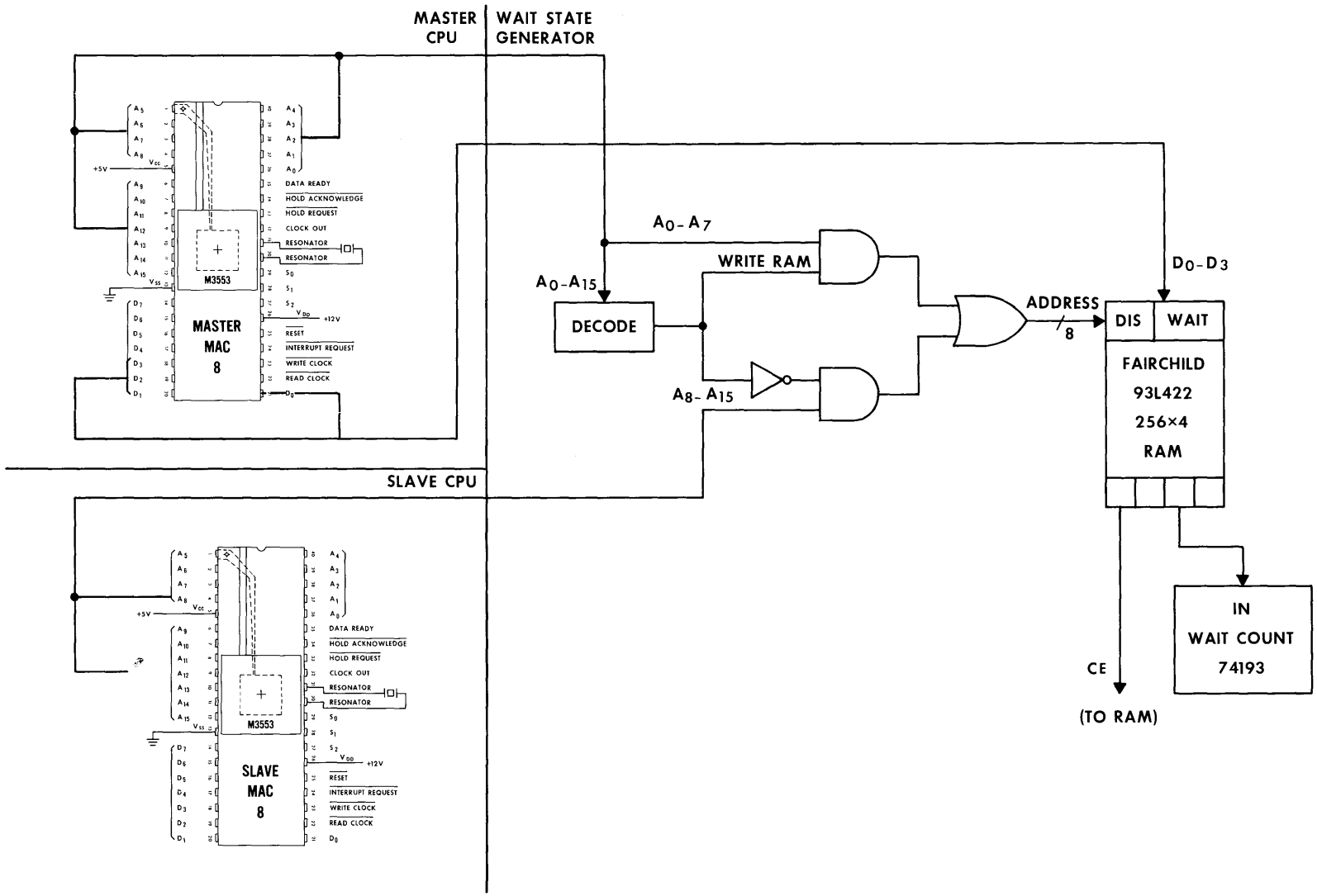


Figure 5-5 - Programmable Wait States

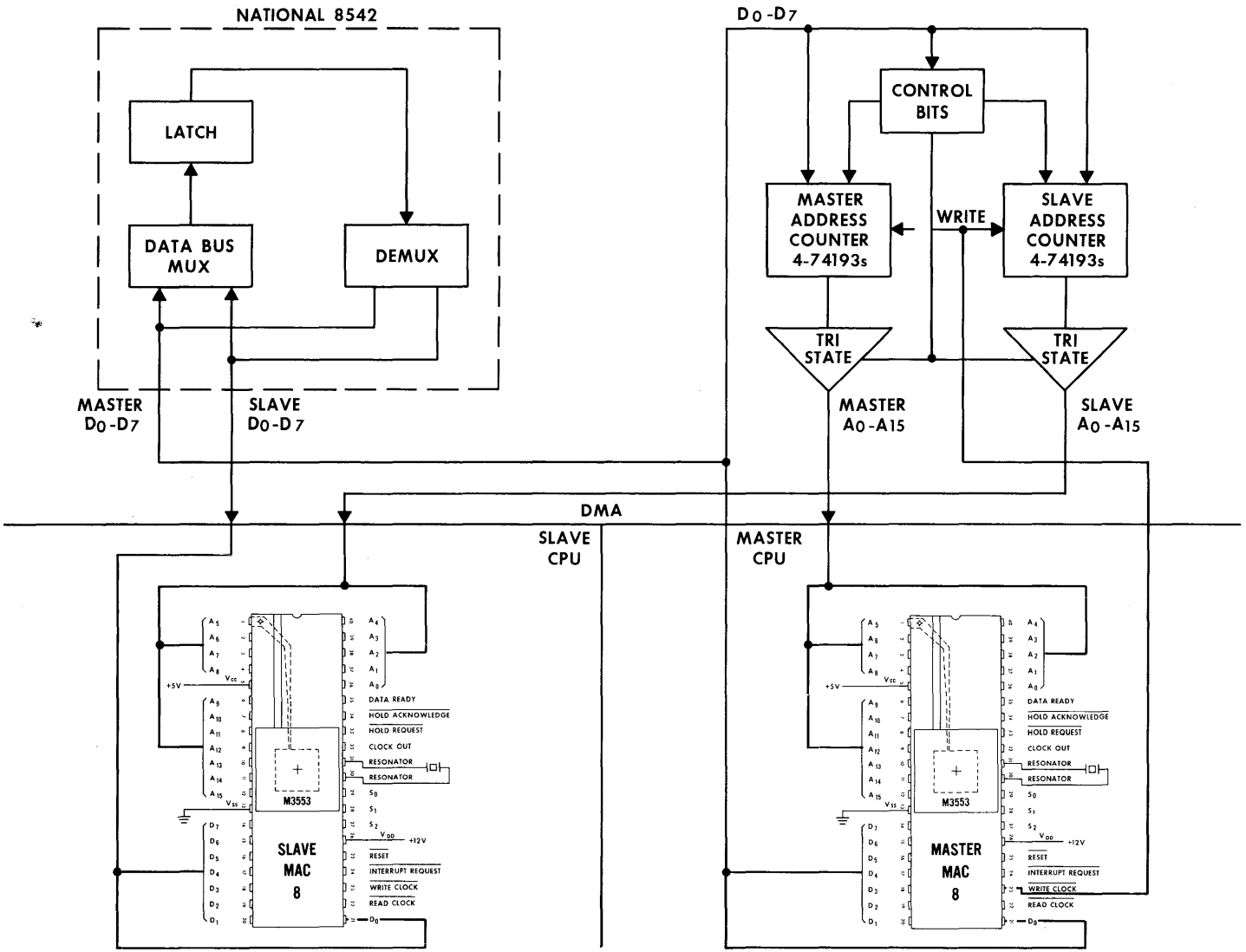


Figure 5-6 - DMA Control

with each transfer, or one may be held stationary while the other is incremented. Having these operations under program control permits the user a large degree of flexibility. For instance, by holding the slave address fixed at an I/O port, data transfer between the master and the I/O device can take place.

Data transfer occurs through the multiplexer-demultiplexer-latch circuitry of the DM8542. The latch is necessary since timing constraints require two clock cycles per word transfer between master and slave. Control signals are decoded in a 74139 device to direct the mode of operation and to generate the appropriate master and slave write and read signals.

MONITORING MAC-8 REGISTERS

The contents of the on-chip registers appear on the data and address buses at certain times during MAC-8 operation. In order to know what is present on the buses at any given time, it is necessary to know the status of the CPU during the current clock cycle when the operation is under control of the interrupt handler (IH) and a subroutine return (SUBRET) instruction is being executed. In the circuit illustrated in Figure 5-7, CPU status is determined by decoding the three status bits from the MAC-8. IH control begins when the expected read signal from the CPU is not received in the cycle following an IR fetch status. IR fetch status includes PC change, IR fetch with condition register (IRFC), and IR fetch without condition register (IRFC'). IH control ends with the reception of the next PC status. A SUBRET is detected by decoding each instruction that appears on the data bus during the cycle following an IR fetch status. The instruction ends with the next PC or IRFC status indication.

The contents of the RP or SP registers appear on the address bus during the cycle following the RP or SP change status indication. The contents of the PC register appear on the address bus during the cycle following an IR fetch status, provided control has not been transferred to the IH. The contents of the condition register are present on the data bus under the following conditions:

- During the same cycle that an IRFC status is received.
- During the same cycle that a PC status is received, provided that control is not transferred to IH and a SUBRET is not being executed.
- During the cycle preceding that in which a PC status has been received if a SUBRET is being executed.

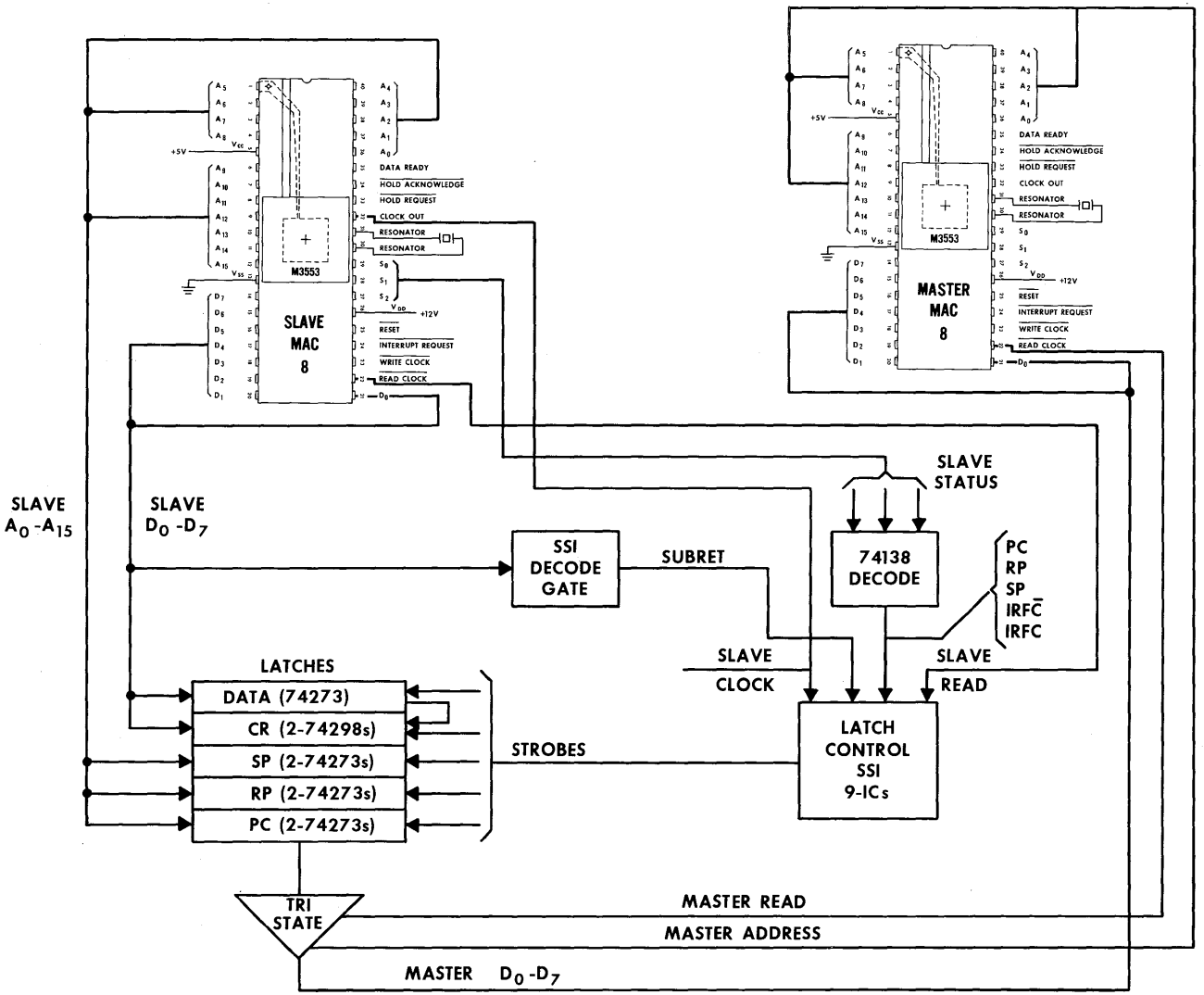


Figure 5-7 - Snapshotting MAC-8 Registers

In order to handle the last condition the data bus is latched at each clock cycle. One other operation affects the condition register monitor: each time an IH operation is detected, the interrupt enable (INTE) bit must be cleared.

The monitor registers are updated each time a change is indicated in the on-chip registers and, if enabled, the circuit interrupts the master when a change occurs. The contents of the monitor registers can be read by the master at any time.

CONCLUSION

I would like to acknowledge the work of D. E. Blahut, R. L. Ferch, and H. B. Greene with whom I shared the circuit design for the PLAID console. Clearly, there are many other interesting circuits in PLAID not covered in this paper. Please call upon us at any time for further discussion of PLAID circuitry. It is our feeling that the MAC-8 microprocessor is not only a device with powerful capabilities but also one that is interesting and enjoyable to work with.

IN-HOUSE MICROCOMPUTERS

MAC-4

D. C. Stanzione, BTL Dept 4391, HO, NJ

ABSTRACT

As the sophistication of our microprocessor technology increases and our skill in the use of that technology matures, we are finding that the benefits of memory-based large scale integration (LSI), customized with software, are applicable to a wide range of Bell System products. In particular, applications which require relatively few semiconductor devices and have previously been considered either too cost sensitive or lacking the complexity to justify use of a microprocessor are now candidates for a single-chip microprocessor system - a single-chip device containing central processing unit (CPU), memory, and input/output (I/O).

The internal structure and characteristics of such a system containing CPU, program memory, data memory, and a sophisticated I/O structure are discussed in this paper.

INTRODUCTION

A phenomenal growth in the development of integrated circuit technology has occurred over the last several years. A familiar statistic is the yearly doubling of the number of components on a single silicon chip and the forecast that this rate of growth will be maintained for at least the next five years.[1] Confronted with this capability in LSI, system designers are pressing to define functional blocks which will utilize it. The identification of the blocks is not simple. The high developmental expense for LSI circuits requires that there be substantial justification for a particular device. Development of a device for a specific customer, usually referred to as custom LSI, normally offers an implementation which

provides the lowest device cost for a given function, but it is accompanied by lengthy development times and changing system specifications. These problems become more acute as improved technology provides the opportunity for even greater circuit complexity. Ideally, an LSI circuit would be catalog or off-the-shelf, rather than custom, so that it might have a wider market base and be more adaptable to changes in device requirements. Some early examples of successful catalog LSI were memory components and calculator devices.

ENTER THE MICROPROCESSOR

As integrated circuit technology evolved, a revolutionary, rather than evolutionary, phase in the utilization of the technology occurred. The advent of the microprocessor solved the LSI dilemma of lengthy development times versus changing system requirements and also permitted the use of cost-effective LSI in products with relatively low production volumes. The devices used in a microprocessor system were still customized but based on an entirely different concept: software- as opposed to hardware-customized LSI. The microprocessor system was now tailored to a particular function by altering its program memory instead of configuring hardware. Designs could be rapidly implemented and easily changed.

With the change in design technique to software-customized systems, real innovation in hardware development shifted from the design of controller logic to that of sophisticated I/O interfaces to the processor and system memory. Most of the difficult problems in microprocessor design involve the application of sequential, stored-program logic in areas where we have traditionally used parallel logic. Most of the solutions to these problems are found in innovative I/O interfaces.

SINGLE-CHIP SYSTEMS

As LSI maintained its seemingly unimpeded rate of growth, processors became more complex; memory sizes continued to increase; and I/O devices gained in sophistication. As the capability of the technology increased, it became apparent that an entire system (CPU, read-only memory [ROM], random-access memory [RAM], and I/O) could be combined on a single chip. The first devices of this type, available about mid-1975, were revisions of existing calculator chips.

Instruction sets had a strong orientation toward binary coded decimal (BCD) arithmetic with little emphasis on program control and I/O capability. The I/O structures of these devices were slanted toward keyboard input and light-emitting diode (LED) output. We are beginning to see newer systems with strong emphasis on increased processor capability but with still fairly simple I/O structures. The trends in commercial, single-chip systems are toward increased processor capability (more efficient instruction sets, higher speed, etc.) and larger program and/ or data memory, but I/O architecture continues to be neglected despite the significance of I/O in system implementation. The importance of I/O in single-chip systems is emphasized since adding additional devices will significantly impact system cost.

The solution to the I/O probem is not an obvious one. While systems may have commonality in the controller implementation of the system (CPU, ROM, RAM), the I/O interfaces normally differ from system to system. The customization that has been obtained through programmability for controller logic has not been achieved for I/O interfaces.

MAC-4

The MAC-4 is a single-chip microprocessor device and is designed to extend the design commonality of the microprocessor beyond controller architecture and into the I/O structure of a system. The I/O structure of the MAC-4 provides a wide range of flexibility and programmability which has not previously been achieved. This capability is coupled with a powerful CPU and an instruction set which is optimized for memory efficiency in control-intensive applications. (In the preliminary description of the MAC-4 device and architecture which follows, the target specifications which were available when this paper was written are discussed. The paper is based upon work done by the Low End Microprocessor Study Group.[2])

CHARACTERISTICS

The MAC-4 architecture is structured around a 4-bit internal bus and 4-bit memories. However, data operands may be any widths which are multiples of 4-bit nibbles. Arithmetic and logic operations are performed in nibble-serial fashion within the processor. The operand length is determined by an internal register which is set by the user program.

The MAC-4 instruction set is optimized for I/O- and control-intensive applications. Particular attention has been given to memory efficiency since memory space is an especially critical parameter in single-chip systems. Memory efficiency is enhanced by various memory address modes, including several memory-to-memory operations. The standard device has a 2K by 4 program memory and a 64 by 4 data memory, excluding internal registers. The total available address space is 4K by 4 bits.

The MAC-4 will be fabricated in Si gate, Al metal, complementary metal oxide semiconductor (CMOS) technology. Target characteristics for the device include a required supply voltage of 3 to 10V; an operating power of approximately 6 mW to permit line powering of the device; a standby current (for retention of RAM data) of 50 nA at 2V; and a clock rate of 2 MHz at 10V operation and 400 kHz at 3.5V operation. The standard MAC-4 device will be packaged in a 40-pin dual in-line package (DIP).

I/O ARCHITECTURE

The I/O architecture of the MAC-4 is designed to permit a wide range of flexibility and to minimize package count in many types of applications. Features within the I/O subsystem make possible data rates in the megabit range and allow responses to input stimuli to occur in less than a microsecond. The capability of the MAC-4 I/O subsystem impacts significantly on applications which previously would have required either discrete logic or custom LSI.

One of the most innovative features of the MAC-4 is a programmable logic array (PLA) (see Figure 6-1). In this configuration the state of the PLA is determined by the user program and is a part of the PLA inputs. The other inputs are clocked directly from device input pins. PLA outputs are latched every clock cycle, thereby achieving submicrosecond response times to input signals. The contents of the

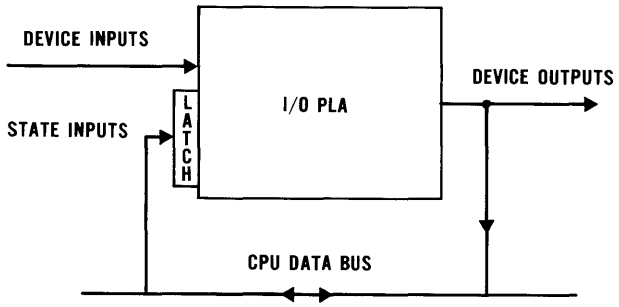


Figure 6-1 - MAC-4 I/O PLA Structure

PLA are programmed when the device is fabricated in the same mask step that determines the contents of the system ROM.

Since a PLA may be used to implement specific logic functions (the CPU controller logic is implemented with a PLA), a user may program custom combinatorial logic for a specific I/O function. This feature may then be coupled with the state input provided by the CPU to achieve complex sequential logic functions tailored to a specific application under program control.

A direct-memory access (DMA) capability is also provided in the I/O subsystem. The ability to perform DMA operations allows the MAC-4 to send and receive bursts of data at megabit rates with no external hardware. The interrupt feature on the MAC-4 is implemented using the system DMA capability.

I/O transfers may also be made under program control. These transfers can be accomplished in either of two ways: special bit manipulation instructions provide I/O capability at the bit level, or data may be moved in parallel using standard data transfer instructions.

APPLICATIONS: SHORTCUT TO CUSTOM LSI

It is anticipated that in most applications the standard MAC-4 device will be used. Those in which high-volume production is expected will also use the standard device for prototyping, field trial, and initial production. However, the MAC-4 has been designed so that it may be optimized for cost-sensitive products with high-production volumes, a course which may be followed with relatively small, additional development effort.

Since approximately one-half of the standard chip is occupied by program and data memory, reduction of memory for simpler applications is an excellent area for possible cost reduction. Other areas include changes in I/O circuitry, the use of smaller packages when changes in the I/O make this appropriate, and even optimization of the CPU instruction set. These cost reduction techniques, coupled with the flexible system architecture of the MAC-4, compare favorably with device costs achievable with custom LSI. However, the problems of extensive chip development and changing system requirements, associated with the latter, have been dispensed with.

SUMMARY

As LSI technology continues to evolve, complete microprocessor systems are becoming single chips. The tailoring of a catalog LSI device by means of software, rather than hardware, offers a number of substantial advantages. The MAC-4 is a single-chip microprocessor system which provides an extremely sophisticated, flexible I/O structure not previously available. This structure is combined with a powerful CPU which has been optimized for memory efficiency and is especially well-suited for I/O- and control-intensive applications. The introduction of this and similar devices into Bell Laboratories will allow designers to take full advantage of the rapidly advancing LSI technology.

REFERENCES

1. Gordon E. Moore, "Progress in Digital Integrated Electronics," Proceedings of ELECTRO 76, Boston, Massachusetts, May 11, 1976.

2. Final Report of the Low End Microprocessor Study Group (D. E. Blahut, I. A. Cermak, W. F. Chow, J. A. Copeland, D. H. Copp, J. I. Frederick, R. C. Hansen, V. K. L. Haung, C. A. Merk, J. R. McEowen, D. R. Morgan, P. O. Schuh, D. C. Stanzione, and L. C. Thomas), Bell Telephone Laboratories, Holmdel, New Jersey, May 28, 1976.

SUPPORT FACILITIES

THE PROGRAMMING LANGUAGES OF MAC-8

H. D. Rovegno and D. B. Kirby, BTL Dept 4393, HO, NJ

ABSTRACT

The MAC-8 hardware design and software design have proceeded in parallel, with strong interaction between the two groups. Each has influenced the other. The result is a total package of well integrated parts, which have been designed to work together.

There are two C-based languages being offered with the MAC-8: a compiler language and an assembly language. Unlike other microprocessor code development, it is anticipated that the compiler language will be the primary language used for MAC-8. This is possible because the compiler is capable of generating very efficient code, even though it is designed to handle code generation in a systematic way. For the unusual cases where some advantage can be gained, the assembler is available. In appearance, the assembly language resembles the compiler language, and offers some compiler-like features.

As the hardware and software have been designed together, so have the software languages been designed to form an integrated family of software support. This paper briefly describes the two MAC-8 programming languages.

INTRODUCTION

The rapid evolution of microprocessors has resulted in the availability of increasingly sophisticated hardware without the corresponding sophistication in the

software support systems. Initial software offerings from commercial vendors included only absolute assemblers and primitive simulators. Although several vendors now offer compilers and some also feature relocatable code generation, the microprocessor industry is only now starting to appreciate the impact of good software support tools.

The MAC-8 has been designed from the beginning as a system in both hardware and software aspects. The software subsystem consists of an integrated set of tools featuring a compiler, assembler, linker, simulator and various utilities. These UNIX-based[1] tools all feature UNIX or C-based[2] languages as user input. This paper discusses the use of two of the tools: the compiler[3] and the assembler.[4]

COMPILER

Even though microprocessor users tend to avoid using high-level languages because of apparent "inefficiencies," surveys show that use of an appropriate high-level language dramatically increases programmer productivity.[5] The high-level language, "C", is a good tool for microprocessor users because it gives them more control over the resources of the machine, as well as the power of high-level constructs.

C gives the user more control over the machine by allowing specification of the following primitive types of variables:

- Character variables which allow the user to manipulate 8-bit portions of memory easily.
- Pointer variables which enable the user to do manipulations with addresses.
- Register variables which enable the user to specify which variables are important in a given subroutine, so the compiler can allocate storage for these variables in registers.

C also allows the user to allocate storage for high-level data aggregates such as arrays and structures; these are essential in all types of table lookup. An example of a structure and an array is contained in Figure 7-1.

```
/*
 * Structure used to input data on a port
 */
struct port {
        char status;
        char *buffptr;
        char nextchar;
};
/*
 *  Table of port entries
 */
struct port intable[9];
```

Figure 7-1 - Input Data Structure

The structured programming constructs, such as "switch", "do", and "while" are also part of the C language. These constructs are fundamental to the formation of a well structured program, easy to read and debug. Constructs such as switches are very useful in table lookup. The compiler uses special techniques depending on the nature of the switch. For example, the switch in Figure 7-2 would generate code that would directly calculate an index based on "d" and use it in a jump table. The switch in Figure 7-3, on the other hand, would generate a short linear search of a table containing the possible values of d. The resultant index would be used in another jump table.

In switches with a large number of cases and a large spread in values, hashing is used to identify a short subtable. This subtable is linearly searched to generate an index into a jump table.

Boolean, shift, and rotate operators, as well as the usual arithmetic operators, can be used in C. These operators make bit manipulations easier in a high-level language, as demonstrated in Figure 7-4.

```
main ()
{
     register int d;
     .
     .
     .
     switch (d)
     {
     case 31:
          ...
          break;
     case 28:
          ...
          break;
     case 30:
          ...
          break;
     case 27:
          ...
          break;
     }
}
```

Figure 7-2 - Switch 1

```
main ()
{
     register int d;
     .
     .
     .
     switch (d)
     {
     case 27:
          ...
          break;
     case 122:
          ...
          break;
     case 124:
          ...
          break;
     case 211:
          ...
          break;
     }
}
```

Figure 7-3 - Switch 2

```
/*
 * Input character from port 5, saves the low order 2
 * bits, shifts them left 2 positions, and uses the
 * result to index into a table.
 */
char data, table [50];
data = table [(in(5) & 03) << 2];
```

Figure 7-4 - C Bit Manipulation

Even though the features of C, as mentioned above, give the user more control over the machine resources, C has all the advantages of a high-level language:

- Allocation of various types of storage, such as automatic, static, external, and register.
- Evaluation of expressions which includes mode conversions, register assignment, and code generation.
- Passing of arguments, saving registers upon function invocation, and restoring of registers and return values upon function return.

Since a compiler must handle code generation in a general and systematic way, it must of necessity build inefficiencies into the code. However, by following the few general rules below one can minimize these inefficiences without sacrificing the benefits of a high level language:

- Declare as many variables as possible to be register variables. Registers are a more efficient type of memory with respect to an instruction set, so by maximizing the use of register variables, the compiler will produce more efficient code. The compiler assigns the indicated register variables to registers as it encounters them. If there are more register variables than available registers, the compiler gives the remaining register variables an automatic storage class.

- Use character variables whenever possible. (This pertains to the MAC-8 version of the C compiler only.) The MAC-8 is a byte (8-bit) oriented machine. Though the MAC-8 contains many 16-bit oriented instructions, they are not as efficient with respect to time.
- Use pointers and the four operators - post/pre increment/decrement - to avoid array calculations. There are several ways to write code to initialize an array, as illustrated in Figures 7-5 and 7-6.

```
/*
 * Initializes an array of 10 elements to 5
 */
main ()
{
        char c[10];
        register char i;
        for (i=0;i<10;i++)
                c[i] = 5;
}
```

Figure 7-5 - Array Initialization by Subscript Calculation

```
/*
 * Initializes an array of 10 elements to 5
 */
main ()
{
        char c[10];
        register char *p, i;
        p = c;
        for (i=0;i<10;i++)
                *p++ = 5;
}
```

Figure 7-6 - Array Initialization by Pointers

In Figure 7-5, the array is initialized by calculating an index each step of the way. In Figure 7-6, on the other hand, the array is serially swept one element at a time, using the pointer variable "p". Use of p as a pointer is indicated by the unary operator "*". More calculations have to be performed by the compiler in Figure 7-5 than in Figure 7-6 to implement the code sequence. When these two cases were run through the MAC-8 compiler, Figure 7-6 required 3 percent less memory and executed 24 percent faster.

To illustrate the "efficiency" in the compiler-generated code, a bubble sort routine[6] was coded both in C and MAC-8 assembly language. The C version is

shown in Figure 7-7 and the MAC-8 assembly language version in Figure 7-8. Not only is the byte count for both versions approximately the same, but the compiler version was coded in a much shorter time and is easier to modify.

```
/*
 * Bubble sort in C
 */
bubble (v, n)
char *v, n;
{
        register char  *pi, *pj, k;
        register char  *w;

        w = v;
        for (pi=w+n-1;pi>w;--pi)
                for (pj=w;pj<pi;++pj)
                if  (*pj > *(pj+1))
                        {
                        k  = *pj;
                        *pj = * (pj+1);
                        *(pj+1) = k;
                        }
}
```

Figure 7-7 - C Bubble Sort

Situations will arise in which ineffiencies, however minimal, are not tolerable. For these cases, the assembler can be used to generate code at the machine level.

ASSEMBLER

The assembler gives complete control over the MAC-8 instruction set and architecture, while offering most of the high-level constructs available in C. The assembly language for the MAC-8 has purposely been fashioned after the compiler language. The inherent addressing modes of the MAC-8 have made this unified approach easy and natural. The major advantage of having similar assembler and compiler languages is that the time needed to master one, when

the other is known, will be at a minimum. As an example of the similarity of the two languages, the assembly language version of the bubble sort of Figure 7-7 is shown in Figure 7-8. As a further illustration, an assembly language version of the array initialization of Figure 7-6 is shown in Figure 7-9.

```
/*
 * Bubble Sort In MAC-8 Assembly Language
 */
bubble ()
{
        extern    char  *v, n;
        a3 = n;
        do
          {
          a2 = a3;
          --a3;
          if (zero) return;
          b5 = v;
s1:       --a2;
          }
        while (zero);
        if ( *(b5+0) > * (b5+1))
          {
          a4 = *b5;
          *(b5+0) = *(b5+1);
          *(b5+1) = a4;
          }
        ++b5;
        goto s1;
}
```

Figure 7-8 - Assembler Bubble Sort

```
/*
 * Initialize an array of 10 elements to 5
 */
main ()
{
        char  c[10 ];
        b0 = &c;
        for  ( a0 = 0; a0 < 10;  ++a0)  *b0++ = 5;
}
```

Figure 7-9 - Assembler Array Initialization

Although the syntax of the assembly language resembles C, it is a true assembly language. That is, there exists one assembly language statement for one machine instruction. Every machine instruction, and every addressing mode for each instruction, is completely specifiable in the assembly language. So for those cases in which every byte and cycle are important, the assembly language - with its consistent syntax - is available.

In addition to the traditional one-for-one nature of an assembly language, this assembler offers compiler-like facilities not generally found in traditional assemblers. For example, it allows high-level constructs which have become popular in structured programming. They tend to make a program more readable and easier to debug. Also, data layout and data initialization are completely specifiable by the user. The language for handling this task, however, is in the form of C declaration statements. The necessity for assembler pseudo-ops disappears with this approach. As in C, the assembler forces the structure of every program to be a collection of functions (or subroutines) which eases and encourages modular program structure.

As has been implied by the above examples, both the compiler and assembler produce relocateable code. The MAC-8 linking-loader serves to tie together all of the separately developed routines and assign absolute origins to the final product.

So, regardless of the source language chosen for an application on the MAC-8, all of the facilities for good programming practices are available. In addition, by

adhering to some simple conventions, it is possible to call an assembly language routine from a C program, and vice-versa. Therefore assembly language can be used in a critical routine while compiler language can be used in less critical ones.

SUMMARY

The programming languages available for the MAC-8 microprocessor have been designed to cover all the needs of microprocessor code development. The high-level compiler language, C, will be the primary programming language for the MAC-8. Programming productivity, the usual advantage of a compiler language, is evident in the MAC-8 along with code generation efficiency and machine resource control. The MAC-8 assembler language provides total access to the MAC-8 instruction set as well as providing a high-level statement syntax and high-level programming constructs. Both languages facilitate the production of readable, well documented, and easy-to-debug programs.

REFERENCES

1. K. Thompson and D. M. Ritchie, "The UNIX Time-Sharing System," Communications of ACM, July, 1974, p 365.

2. D. M. Ritchie, C Reference Manual, TM-74-1273-1, 1974

3. H. D. Rovegno, Microprocessor C for Beginners, Bell Telephone Laboratories unpublished work.

4. D. B. Kirby, MAC-8 Assembler User's Manual, TM-76-4393-16, November, 1976.

5. E. A. Nelson, "System Development Corp. Report TM-3225," Management Handbook for the Estimation of Computer Programming Costs, pp 66-67.

6. B. W. Kernighan and P. J. Plauger, Software Tools, Addison-Wesley, p 105.

SUPPORT FACILITIES

A SUPPORT SYSTEM FOR THE INTEL 8080

D. J. Hunsberger and J. J. Molinelli, BTL Dept 3231, HO, NJ

ABSTRACT

This paper describes a comprehensive set of development tools for developing and debugging the hardware and software for the Intel 8080 family of microprocessors.

For software designers, the set offers tools for the description (a structured assembly language), construction (link editors, library facility), testing (symbolic debugger), production (programmable read-only memory [PROM] writer), and documentation (text formatting macros) of the target software system. The tools are particularly well suited for project implementation by a group of designers, facilitating as they do the partitioning of the system into many small independent pieces.

For the hardware designers, the set includes a single-board utility system which is plugged into the target system, enabling the software tools to manipulate the target hardware. Further development will allow this capability to be transported into the field via a field test unit (FTU).

For the most part, the software tools are project independent and would be applicable to any Intel 8080-based development; they are available in an interactive mode under the UNIX operating system. The hardware development tools interface the target system to the minicomputer; hence, they tend to be more project oriented.

MOTIVATION

Recently Laboratory 323 at Holmdel undertook the development of an Intel 8080 microprocessor-based electronic key telephone system designated the 32A Communications System.[1] This application has some characteristics unusual in the current microprocessor software art. It is large, about 20K to 30K bytes of program, ten people are developing the program, and it is very sensitive to real-time usage.

Good programming requires good tools. Vendor-supplied tools tend to be primitive and oriented towards small applications. For example, assemblers do not generate relocatable code; debuggers tend to be at the bit level; higher level languages are usually very inefficient; development support systems are geared toward a singler user.

This paper describes a comprehensive set of 8080 development tools designed for a complex application. They rely on interfacing the target application with a PDP-11 minicomputer, operating under the UNIX time-sharing system. In this manner, the designers have access to their development tools in an interactive mode.

SOFTWARE TOOLS

This section outlines the tools available to the software designer. They are organized in a sequence corresponding to the order of software development: design, code, test, and document.

Structured Assembly Language

All of the 32A programs are written in SMAL2. It is a structured assembly language which combines the ease of use of a compiler with the efficiency of assembly language. The compiler features of SMAL2 allow the user to describe the algorithm in "natural" notations; the language also contains a rich set of control structures (if-else, while, do-while, switch). SMAL2 has the syntax of the programming language C. As an assembly language, its machine registers may be manipulated directly; it can be used in a mode in which one SMAL2 statement generates one machine language instruction.

Standard Assembly Language

The 8080 assembler available to our designers uses mnemonics and instruction formats similar to those of other 8080 assemblers. It differs from most others in the following ways:

- It generates relocatable code.
- The programmer has three relocation counters available.
- The programmer can declare the existence of undefined names within the assembly.
- Temporary labels are available.

Link Editor

The link editor is used to combine object modules generated by different assemblies (or SMAL2 compiles). Hence, it relocates object files and resolves external references. The link editor can also search libraries of object modules; such libraries are created and maintained with the standard archive facility available on UNIX.

Symbolic Debugger

The symbolic debugger allows the programmer to control both the state and the operation of the 32A microprocessor via a hardware interface between it and the host minicomputer. The function and design of this hardware is described under HARDWARE; we concentrate here on the capabilities of the debugger itself.

The debugger allows the programmer to initialize, save, restore, and verify the contents of the 8080 memory. That is, the programmer can load a program, verify that a program has not been mutilated (by a bug), save the current state of the microprocessor (e.g., to go to lunch), and restore a previously saved state.

The programmer can control the execution of the 8080 in a number of ways: by resetting the processor, causing the processor to commence running from any memory address, halting or holding the processor, or causing the processor to continue from the halt or hold location.

The program-testing features of the debugger include the ability to examine and modify memory, to initialize the registers, and to set up address and data breakpoints.

In each function of the debugger, the programmer communicates memory addresses and register names symbolically, so there is not the constant shuffling between the bit level and the programming language level. Indeed, the debugger is capable of disassembling the contents of memory into SMAL2 statements.

Intellec MDS Interface

For smaller projects not requiring interface between the target microprocessor and the PDP-11, tools are available to interface with the Intellec MDS development system. Specifically, it is possible to transmit an object module to the MDS system via a dial-up data link. This capability requires that a small monitor be resident in the MDS; its function is to check the incoming data for consistency and to load memory.

PROM Writing Tool

A Data I/O PROM programmer V is interfaced to the minicomputer system. This programmer is capable of writing a wide variety of PROM devices. A software tool allows the user to select and write individual PROMs from any object module. This software is capable of handling many PROM types; it can also generate standard Western Electric (WE) PROM map (PK) documentation.

Documentation

A number of tools exist to extract and print information from object modules. The size of an object module can be obtained as well as the contents of the symbol table.

Finally, a set of instructions for the UNIX text formatting program (NROFF) is available to our designers. These macros enable them to generate standard WE program listings (PRs) from source code. The macros also permit the extraction of information relevant to the program descriptions (PDs) required by WE.

HARDWARE

Three hardware developments were undertaken to provide development and debugging capabilities for our designers:

- A utility system designed specifically for software development.
- A single-board utility system for hardware debugging which may be plugged into any 32A.
- An FTU designed to remote the utility system functions for use in the factory or on customer premises.

Software Utility System

The software utility system interfaces the PDP-11 minicomputer with the 32A microprocessor. This approach allows the connection of the 32A periphery as it appears in the field, enabling software development to occur on an exact model of the final product.

As shown in Figure 8-1, the software utility system is interfaced to the PDP-11 through a standard PDP-11 interface. The hardware consists of a system control unit, the writable program store (WPS) and some specialized debugging logic. The control unit allows the PDP-11 to initialize and control both the execution of the microprocessor and the special debugging hardware. The 32K-byte WPS is accessible by both processors. The hardware debugging logic includes matchers, transfer trace memory, and an elapsed time counter.

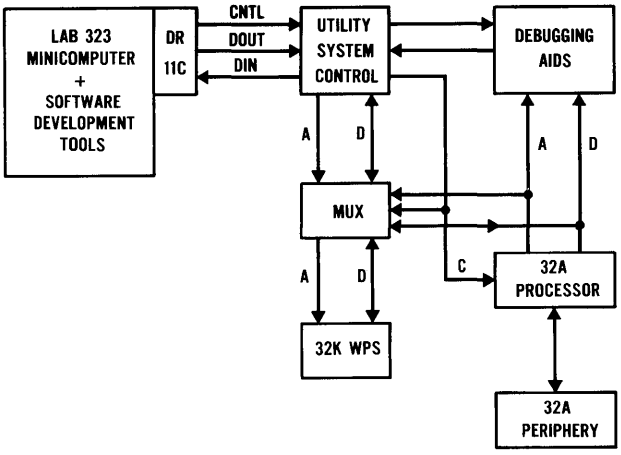


Figure 8-1 - Software Utility System

The minicomputer has direct control of the reset, hold, and ready inputs of the microprocessor, as well as two of its interrupts. By manipulating these signals, the minicomputer can load, save, and modify the WPS and cause the microprocessor to run, halt, reset, and single-step at will. Additional capabilities provided by the debugging aids include:

- Two maskable address matchers which, upon access to a programmed memory address, can hold the microprocessor, cause it to enter a wait state, or supply a trigger pulse to an external device.
- A maskable data matcher capable of halting the microprocessor or supplying a trigger pulse when a specific data word is read or written.
- A transfer address stack operating under control of the matchers to provide the last eight program branch addresses.
- An elapsed time counter, also operating under control of the matchers, to provide an indication of the time required to execute a segment of code.

Single-Board Utility System

The 32A system will be installed throughout the Bell System as the next generation key telephone system. For such a product, maintainability is a major concern. Hence, a separate connector, the FTU slot, is an integral part of all 32A systems and allows access to major control and data points within the system.

A single-board utility system, which plugs into the FTU slot, was designed to provide a means by which any 32A system can interface with the PDP-11. This facility was developed to assist hardware designers in debugging initial versions of the 32A backplane, memory, and processor boards. Figure 8-2 is a block diagram of this utility. It differs from the software utility system in two ways:

- It does not have the hardware debugging aids.
- It uses tristate buffers to interface with the 32A memory boards. Hence, it has direct access to the actual memory of the 32A system.

The single-board system has local memory (1K random-access memory [RAM], 1K PROM) to control the 32A microprocessor. The board is powered by its own power supply, and thus places no load on the 32A power supply.

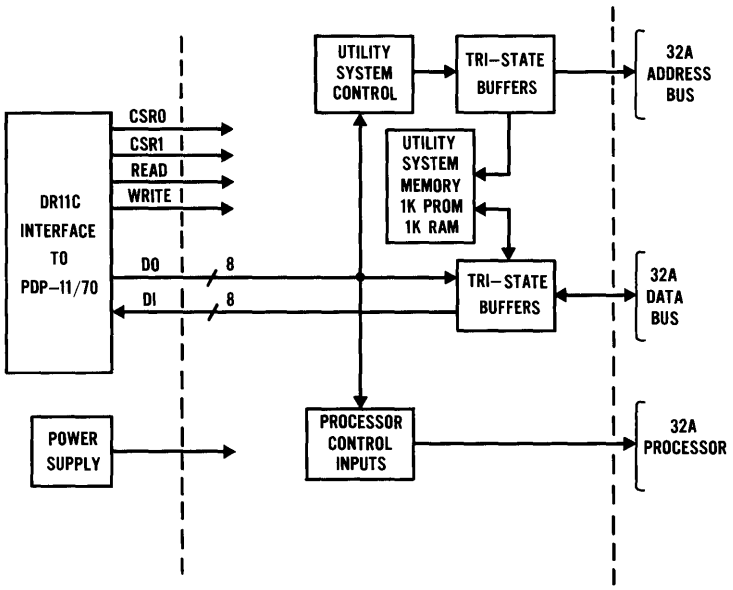


Figure 8-2 - Single-Board Utility System

Field Test Unit (FTU)

The FTU, a portable tester, provides a remote utility system in the factory or on customer premises. Its users require only limited reeducation since it is consistent with the laboratory utility systems.

The FTU is a microprocessor-controlled test set which connects to the 32A through two 50-wire flat cables and an interface board plugged into the 32A FTU slot. The FTU supplies power for the interface board, and thus prevents any drain on the 32A power supply. The FTU has two serial data ports, the first of which connects to a local terminal for user interaction. The second port links the FTU with the laboratory minicomputer for transfer of test programs.

Figure 8-3 is a block diagram of the FTU. The interface board and the sanity control and receiver board contain the high-speed buffers required to send bi-directional addresses, data, and control signals over the 5 feet of cable connecting the two boards. The matcher, elapsed time clock, and transfer trace functions are enhanced versions of those provided on the software utility system. There are four data and/or address matchers in the FTU, and the transfer address stack is 128 locations deep.

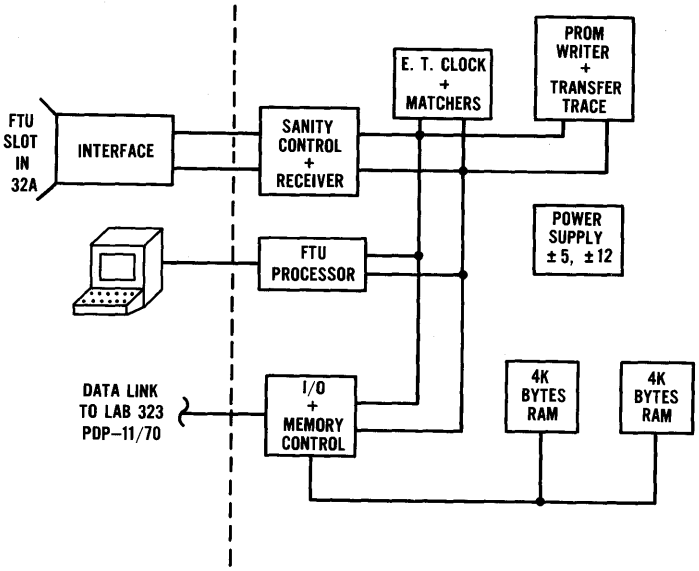


Figure 8-3 - Field Test Unit

The I/O and memory control board controls 8K of RAM which is accessible by either the 32A microprocessor or the FTU microprocessor. This RAM is used to store the software required to implement the utility system functions and to store test programs transferred to the FTU by the laboratory minicomputer. Patches to the 32A PROMs can be implemented in this memory by using matcher control of the high-priority interrupt line and a software link to the code to be patched.

The PROM writing circuit in the FTU enables remote programming of Intel 2708-type PROMs. The PDP-11 controls this process to ensure adequate documentation before a new version of PROM is used.

The processor used in the FTU is on a general-purpose 8080 microprocessor board. The board contains the 8080, 8224, and 8228 chip set along with an 8-level priority interrupt structure, 1K of RAM, 8K of PROM, and a serial I/O port.

SUMMARY

All of the previously described development tools are operated in the Laboratory 323 minicomputer system. The software tools are also available on Department

4393 microprocessor support machine. Most are exportable with no changes to any UNIX-based machine. The symbolic debugger requires that the auxiliary hardware be available on the PDP-11. The design is available for replication. The PROM writing utility would require a compatible PROM writer. Finally, both the PROM writing program and the MDS system transmit functions require special teletype modes on the corresponding ports of the PDP-11.

CONCLUSION

The combination of PDP-11 resident software tools with three hardware utility systems has produced a comprehensive set of tools for microprocessor system development. The ability to transport these tools to the field will simplify the difficult task of maintaining a complex, stored program system in the field.

ACKNOWLEDGMENTS

The authors are acting as reporters for the work of many individuals. David H. Copp wrote the SMAL2 compiler, which is based on a previous language by Popper.[2] The assembler is a modified version of a previous 8080 assembler by Bayer.[3] The link editor is based on the UNIX link editor and was implemented by John J. Molinelli. The symbolic debugger was written by Robert C. Radcliff and based on earlier work by Molinelli. The Intellec MDS interface was designed and interfaced by Stuart Tartarone. Robert J. Council implemented the PROM utility and the text formatting macros. Dennis Hunsberger designed the FTU and the utility system hardware. Dana Runyon designed the specialized debugging hardware. Finally, James H. VanOrnum did the original system design work for the entire utility system.

REFERENCES

1. W. M. Johnson, et al, "32A Communications System," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Incorporated, Holmdel, New Jersey, 1976.

2. C. Popper, "A Structured Macro-Assembly Language for a Microprocessor," COMPCON 74 Fall, IEEE Computer Society, International Fall Conference, 9th, 1974, Washington, D. C., pp 147-151.

3. P. Bayer, Assembler User's Manual, Bell Telephone Laboratories unpublished work.

SUPPORT FACILITIES

UNIX ON THE LSI-11

H. Lycklama, BTL Dept 1352, MH, NJ

ABSTRACT

The LSI-11 microcomputer[1] has many potential applications in Bell System projects. The LSI-UNIX system (LSX system),[2] with 20K words of memory and floppy discs for secondary storage, is a modified version of the UNIX system that has been written to run on the LSI-11 microcomputer. This system allows most of the UNIX user programs to run on the LSI-11 microcomputer. The main programming language used is the higher-level C language.[3] Other features of the LSX system are: a background process may be run as well as foreground processes; a set of subroutines interfaces with the file system on the floppy discs; asynchronous read and write routines are available; currently, a 2-dimensional text editor is being written for a TV terminal interfaced with the LSI-11 microcomputer; total system cost is less than $7000, which makes the system appealing as a stand-alone system for dedicated applications. Also, the total system can be utilized as an intelligent terminal.

INTRODUCTION

The LSX system is a modified version of the UNIX operating system that runs on the LSI-11 microcomputer. It uses 20K words of primary memory and floppy discs for secondary storage. This configuration permits most of the UNIX user programs to run on the LSI-11 microcomputer.

The purpose of this paper is to describe some of the objectives of the LSX system as well as some of its more important features. Its capabilities are compared

with the powerful UNIX time-sharing system that runs on the PDP-11/40, 11/45, and 11/70 computers. A number of current and planned applications are described in some detail. A summary and some thoughts on future directions are also presented.

WHY UNIX ON A MICROPROCESSOR?

There are several reasons that support the development of a microprocessor-based UNIX system. The hardware costs of a computer system have decreased substantially over the last few years (even over the past few months). Microprocessors on a chip are a reality. The cost of primary memory (e.g., dynamic metal oxide semiconductor [MOS] memory) is decreasing rapidly as 4K-bit chips are being replaced by 16K-bit chips. There is a large amount of expertise in PDP-11 hardware interfacing, and the similarity of the Q-bus of the LSI-11 microcomputer to the UNIBUS of other members of the PDP-11 family of computers makes this expertise available.

The increasing cost of software development has dominated the total cost of a computer system. With the LSX system, this cost has been minimized because, as mentioned previously, the system supports most of the UNIX user programs that run under UNIX time-sharing[4] and, therefore, most of the software is already available.

It is conceivable that, in approximately 5 years, the power of a minicomputer system will be available in a microcomputer. When this occurs, it will be possible for a user to have access to a dedicated microcomputer instead of a minicomputer time-sharing system. The LSX system is a step in this direction. This system provides the user with a cost-effective, interactive, and powerful computer system with an instantaneous response time to requests. There are no unpredictable time-sharing delays. Also, the system has applications in areas where security is important. Access to the data is physically limited to the area in which the system is located.

The LSX system has provisions for local text editing and text processing. Other features can be added easily. Special input/output (I/O) equipment can be interfaced with the Q-bus for dedicated experiments. Using floppy discs as secondary storage gives the user a rather small data base; however, a link to a larger

machine can provide access to a larger data base. Interfaces such as the DLV-11 (serial interface) and the DRV-11 (parallel interface) can provide access to other computers.

One of the main benefits of using the UNIX software base is that the C compiler is available for writing application programs in the structured higher-level C language. The use of the powerful command interpreter (sh) is a great asset, and a general hierarchical file system is available.

The LSX system has two main areas of application:

- Control of dedicated experiments.
- Intelligent terminal.

Using the LSX system as a dedicated experiment controller, special I/O equipment can be interfaced with the Q-bus and the experiment can be supported and controlled within the system.

The applications of the LSX system as an intelligent terminal are:

- Development system.
- General text processor.
- Form editor.
- Two-dimensional cursor-controlled text editor.

HARDWARE CONFIGURATION

The hardware required to build a useful LSX system is minimal. The absolute requirements are:

- LSI-11 microcomputer (with 4K memory).
- 16K memory (e.g., dynamic MOS).
- Extended instruction set (EIS) chip.
- Floppy disc controller with one drive.
- DLV-11 serial interface.
- Terminal (e.g., TTY-33).
- Power supply.

A more flexible and powerful system is shown in Figure 9-1. As illustrated, the floppy disc controller may contain up to four drives. Another serial or parallel

interface is often useful for connection to a larger machine that provides software support. The system terminal may be either a TTY-33 teletypewriter or an inexpensive cathode ray tube (CRT) terminal. Various types of memory (core, MOS or random-access memory [RAM]) are available either from DEC or from other outside vendors (16K boards are available from Monolithic Memories, Incorporated). The floppy disc controller may be bought either from DEC (no direct-memory access [DMA] capability) or from other outside vendors, e.g., Advanced Electronics Design, Incorporated (AED) (with DMA capability). The actual floppy disc controller (one controller per disc[5]) used in the LSX system has been built to satisfy specific Bell System requirements. DEC floppy diskettes are formatted according to IBM standards. The comparative data for floppy diskettes are:

Features	DEC	BTL	AED
Sector Size (Bytes)	128	512	512
Sectors per Track	26	8	16
Number of Tracks	77	77	77
Total Capacity (Bytes)	256256	315392	630784
DMA Capability	no	yes	yes

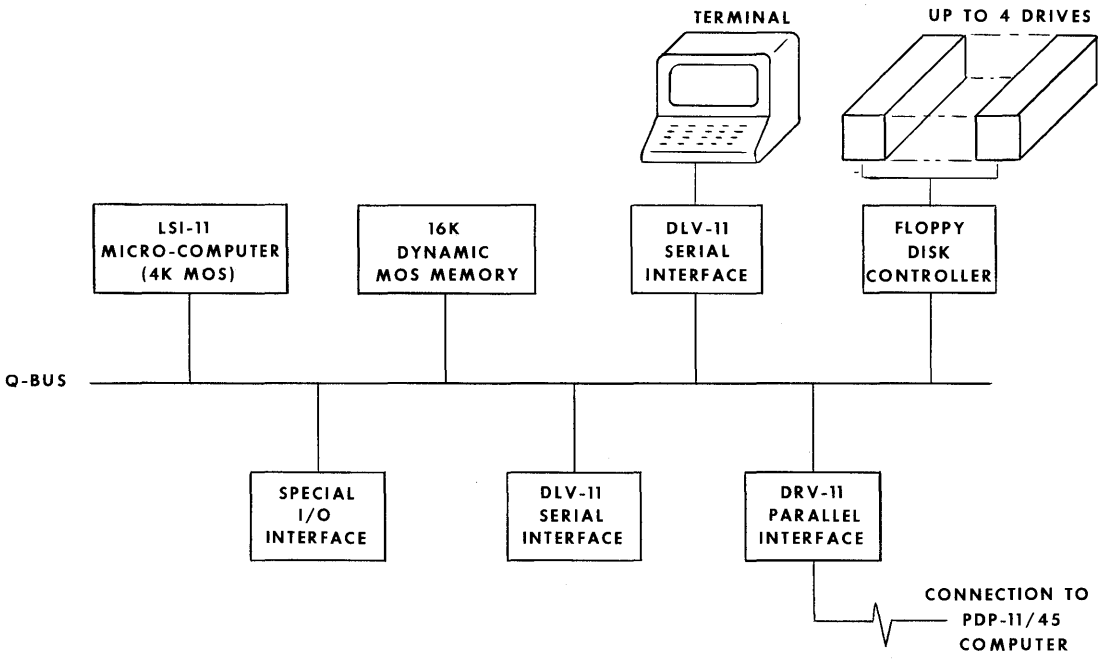


Figure 9-1 - LSI-11 Configuration

The outside vendor (AED Systems) supplies dual-density drives for an increase in storage capacity.

If necessary, a bus extender may be used to interface special devices with the Q-bus. One such device that has been interfaced with the Q-bus is a special-purpose TV terminal with cursor control and user-labeled buttons.[6] Applications of this terminal are discussed in later sections. Other systems are interfacing special signal processing equipment with the Q-bus. Additional applications will be developed when DEC provides more of the interfaces for standard I/O peripherals.

LSX CAPABILITIES

The total system occupies 8K words of memory and has room for six system buffers. The C compiler requires up to 12K words of user address space; therefore, the C compiler can run using 20K words of total memory. The system size may be increased if more capabilities are required since the total memory space available to the system and user is actually 28K words. More system buffers can be provided. If the system is kept to 8K words, a 20K-word user program can be run; however, this requires more swap space, which is at a premium.

Although the LSX system is a dedicated, single-user system, a process may fork up to two levels deep, giving rise to a total of three active foreground processes. The last process forked will run to completion first. Pipes are not supported in the system, but pseudopipes are supported in the command shell. This is accomplished by expanding the shell syntax | to > ._pf;< ._pf. If sufficient disc space exists, the pipe implementation is transparent to the user.

The hierarchical file structure of UNIX is maintained; however, there is no read and write protection on files. File protection is strictly the user's responsibility. Essentially, the user is given superuser permissions. Only execute and directory protection is given on files. Group IDs are not implemented. File system space is limited to the capacity of the diskette in use. Each file system contains only 96 files (i.e., six inode blocks each). The list of available inodes is not dynamically created by the system, but is created when the file system itself is created or salvaged. The system automatically mounts a user file system on a second diskette, if desired. The mount and unmount commands are not available to the user. Thus, a reboot of the system is necessary to mount a new user diskette. Normally, the system diskette is configured with swap space and temporary file

space. User programs and files may reside on the system diskette if a user diskette is not mounted.

The size of available memory and the lack of memory protection (i.e., memory segmentation unit) have put some restrictions on the capabilities of the LSX system; however, these are not severe in the single-user environment. Profiling is not provided in the system. Timing information only becomes available if a clock interrupt is provided on the LSI-11 event line at 60 times per second. At present, only one character device driver and one block device driver are allowed. No physical I/O is provided for. There is also no read-ahead on file I/O. Only six system buffers are provided, and the buffering algorithm is much simpler than in UNIX. All user programs must be relocated to begin execution at 8K in memory. This requires modifications to the UNIX link edit (ℓd) and debugger (db) programs. Most other differences between LSX and UNIX systems are transparent to the user.

BACKGROUND PROCESS

It is possible to run a background process on the LSX system while running a number of foreground processes. The background process is run only while the current foreground process is in an input wait state. The system calls bground and kill have been added to the LSX system to enable the user to run and remove a background process. Only one background process is allowed to run and it is not allowed to fork another child process; however, it may execute another program. The background process may be compute-bound or may perform some I/O functions, such as outputting to a hard-copy terminal.

STAND-ALONE ROUTINES

Under the LSX system it is possible to run a dedicated program (less than 12K words) in real time, using all of the conveniences of the UNIX system calls to communicate with the file system. For programs that require more than 12K words of memory or that require more flexibility, a set of subroutines has been written to provide a UNIX-compatible interface with the file system without using the LSX system calls.[7] With this set of subroutines, the user is given more control over the program. Disc I/O issued by the user is buffered using the read-

ahead and write-behind features of standard UNIX. More system buffers are available. Eight of the standard file system interface routines are provided. The arguments required for each routine and the calling sequence are identical to those required by the UNIX system C interface routines. These include read, write, open, close, creat, sync, unlink, and seek. Three unique routines, saread, sawrite, and statio, enable the user to perform asynchronous I/O directly into buffers in the user area rather than into system buffers. These additional routines allow a user to start multiple I/Os to and from multiple files concurrently, do some computation, and when convenient, wait for completion of a particular outstanding I/O transfer. A load program under LSX enables the user to load the stand-alone program, which must start execution at location 0 in memory.

A DEVELOPMENT SYSTEM

One system disc has been configured to contain a fairly complete program development system. The development programs include:

- Editor.
- Assembler.
- C compiler.
- Link editor.
- Debugger.
- Command interpreter.
- Dump.

Also, a number of libraries are included that contain routines frequently used by the link editor. Thus, it is possible to compile, run, and debug application programs completely on-line without access to a larger machine. In a typical application, the contents of the system disc remain quite stable, whereas all user programs are maintained on a permanently mounted user diskette. For minimal systems, it is possible to run with only one diskette. Due to the lack of protection, it is possible for the system to crash. In practice, however, the use of the higher level C language minimizes the number of fatal bugs that actually occur, since the stack frame and program counter are quite well controlled.

In the particular installation described here, it is often convenient to use the satellite processor system[8] to aid in the running and debugging of new user programs. This is possible since programs running in the LSI-11 satellite microcomputer behave as if they are running on the central machine with access to its file system. This emulates the environment on LSX quite closely. Thus, a program may be compiled on a central machine supporting the C compiler and run on the LSI-11 microcomputer and debugged. When the program has been completely debugged, it is possible to load the program onto the floppy file system using the stand-alone routines (described previously) and the satellite processor system. The program may then be run under the LSX system.

TEXT PROCESSING SYSTEM

When the LSX system is used as a personal computer system for text processing, files may be prepared using the editor program and run off, using the UNIX nroff command, on a hard-copy device. This system disc includes programs such as:

- editor
- cat (Output ASCII Files)
- pr (Print ASCII Files)
- od (Octal Dump Files)
- roff
- nroff
- negn (Mathematical Equation Formatter)

The file transfer program referred to previously provides the ability to transfer files to and from a machine with a larger data base. User files may be maintained on a personally mounted diskette. If a hard-copy device, as well as the user interactive terminal, is attached to the computer, hard-copy output can be obtained using a background process while editing another file in the foreground.

FORM EDITOR PROGRAM

When a special-purpose TV terminal is interfaced with the Q-bus, the LSX system can be used for the entry and retrieval of data records by computer-naive users. The terminal[6] has some advanced features particularly suitable for this application. It has scrolling capabilities, cursor control, and user-labeled buttons below the TV screen. The buttons can be used for cursor positioning as well as

other dedicated functions defined by the user program. This terminal is suited for the input of data into computer-displayed forms. Protected fields are implemented in software rather than in hardware as for the TTY-40 terminal. A general-purpose form entry program has been written for this terminal.[9] Another program, mkform, is available to enable a user to compose a form on the TV screen interactively. The form is then used with the fentry program to create, update, and delete entries in a data base whose record structure depends only on the structure of the form but is independent of the fentry program. The LSX system provides the underlying support and file system for these programs. The programs are designed to be easy to use.

TWO-DIMENSIONAL SCOPE TEXT EDITOR

The TV terminal described previously can perform also as a 2-dimensional text editor. The interactive 2-dimensional cursor-control features allow the cursor to be moved anywhere on the face of the TV screen. The editor available on the UNIX system has some very powerful features, and it is desirable to use these in the scope editor as well; therefore, the scope editor features have been imbedded in the existing UNIX text editor. This allows the user to go back and forth between the standard UNIX editor features and the additional scope editing features. The labels on the buttons below the TV screen tell the user what the mode is and what functions are available. Complete cursor control is available. A window into the file being edited is displayed on the TV screen. The user has the ability to insert, remove, and replace a character at the current cursor position or delete the remainder of the line to the right of the cursor on a per-line basis. The user may also insert or delete whole lines or break a line in two. Block deletes, copies, and moves are also available by means of three marks that may be set in a file. A position command is available to move any section of the file onto the TV screen window (26 lines).

SUMMARY

The LSX system is being used currently in Center 135 for research in intelligent terminals and in stand-alone dedicated systems. There are plans to use this system in other areas of Bell Laboratories. Hard-copy features have yet to be incorporated into the system. Currently, the system is connected to a larger machine using the satellite processor system. Additional connections to larger

machines, or possibly to a network of machines, have to be investigated. Also, the LSX system has potential uses in multiterminal or cluster-control terminal systems where multitasking features are important. These applications have been studied only superficially and warrant further investigation.

ACKNOWLEDGMENTS

The author is deeply indebted to H. G. Alles for designing and building the initial PERTEC floppy disc controller and the TV terminal. These two pieces of hardware have provided much of the motivation for developing the LSX system and for doing research in the area of intelligent terminals. Many of the application and support programs described here were written by two summer employees, G. A. Gladney and E. W. Stark. The driver for the AED disc was written by J. S. Thompson. The author is grateful to J. C. Swartzwelder and D. R. Weller for their efforts in building the first LSX system.

REFERENCES

1. DEC LSI-11 Processor Handbook, Digital Equipment Corporation, 1976.
2. K. Thompson and D. M. Ritchie, "The UNIX Time-Sharing System," Communications of ACM, July, 1974, p 365.
3. D. M. Ritchie, C Reference Manual, TM-74-1273-1, January 15, 1974.
4. K. Thompson and D. M. Ritchie, UNIX Programmer's Manual - 6th Edition, Bell Telephone Laboratories, May, 1975.
5. H. G. Alles, "An LSI-11 Controller for the PERTEC Floppy Disk," private communication.
6. H. G. Alles, "A TV Terminal for the LSI-11 Microcomputer," private communication.
7. E. W. Stark, LSI-UNIX Information, Bell Telephone Laboratories unpublished work.
8. H. Lycklama and C. Christensen, Emulation of UNIX on Peripheral Processors, TM-75-1352-2, January 9, 1975.
9. E. W. Stark, System for Entering Data Through Computer-Displayed Forms, Bell Telephone Laboratories unpublished work.

SUPPORT FACILITIES

MANUFACTURING MICROCOMPUTER-BASED SYSTEMS

L. A. O'Neill, BTL Dept 4393, HO, NJ

ABSTRACT

CADRE (computer aided design for reliability and economy) has been used to simplify assembly of the program logic aid (PLAID) prototype console for the MAC-8 microprocessor. CADRE automatically wires models that verify performance and testability before layout is started. Go/no-go and diagnostic tests are developed with a simulator. The test results are passed to an automatic tester in executable form. Computer-generated schematics can be obtained from the same circuit description used for models and simulation. If performed on the Applicon system, layout, which must be in agreement with the schematic, can be verified by audit of a computer with the same circuit description, since layout and testing data are available in machine-readable form. The time needed to develop manufacturing information is reduced, accuracy is ensured, and expensive redesign is avoided. By developing a fully integrated system for printed wiring board (PWB) layout and documentation for both BTL and WE, maximum benefit will be achieved.

INTRODUCTION

One of the primary advantages of the MAC-8 microprocessor is its ability to reduce the time involved in designing a new product by replacing complex hardware with software. Its full benefit can only be realized, however, if the time required to assemble the microprocessor components and interface circuits to the rest of the system is also shortened. One of the most important considerations in the PLAID[1] project was reaching the potential users of the MAC-8 as soon as possible. The marketing of the MAC-8 is considerably enhanced by the

availability of the prototype console, PLAID, because software development can now be coordinated easily with the hardware design of the system. Although funds were limited, we sought means of reducing the elapsed development time within BTL, as well as simplification of information transfer to WE in order to compress their schedule.

CADRE OVERVIEW

The CADRE system for integrated design of PWBs satisfied the above objectives. CADRE is an interconnected set of computer aids, shown in Figure 10-1, that provides the following capabilities:

- Simplified construction of wired models.
- Logic simulation for verification and test evaluation.
- Generation of acceptance and diagnostic tests.
- Machine-drawn schematics.
- Verification of machine-aided PWB layout.

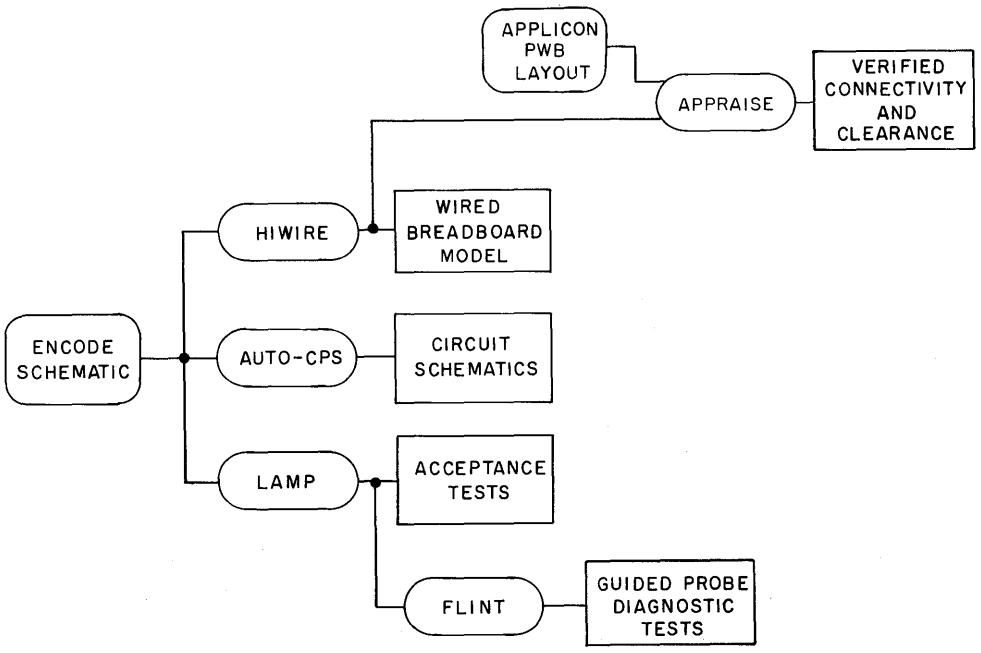


Figure 10-1 - CADRE System for Logic Circuit Pack Design

CADRE has been used by the echo suppressor terminal (EST) project at Holmdel and is thus a proven process, requiring a minimum of training. The programs are used jointly by engineering and drafting so that each is able to work in an efficient manner; yet communication is simplified by maintaining a single data base.

When the development schedule for PLAID was adopted, it was assumed that high-confidence wiring (HIWIRE)[3] would be used to construct the prototype system from wired breadboard models, rather than PWBs, saving five or six weeks of time and drafting effort. When CADRE, which depends on early and frequent interaction between engineering and drafting to reduce redundant effort, was chosen, the estimates of drafting cost were reduced by an additional 25 percent. We are a long way into the schedule, and costs are running even lower than estimates. To understand how CADRE could have this much effect on a project, one must be familiar with the system philosophy and the details of individual capabilities.

MODULAR CAPABILITIES

Computer aids, in order to be useful to a project with a new technology and a close schedule, must be able to be easily modified so as to accommodate new devices and programmed capabilities. This flexibility is available in CADRE because existing programs are interconnected through interface routines that can be changed readily to provide for new programs. In fact, the system was assembled originally from programs that had been written for other applications and new programs were added only when no alternative existed. In addition, all unique information, such as device codes and board descriptions, was put into libraries, which form the heart of the system. It was a relatively simple operation to add the many new devices needed for PLAID. The coordination, maintenance, and documentation of these libraries are simplified by the structure and data formats which were chosen to facilitate maintenance.

The libraries are defined for those items that are used repeatedly in the design process but which change with the technology or the application:

- Component Library: One entry per device code (e.g., TI 7400).
- Package Library: One entry for each pin configuration (e.g., 16-pin dual in-line package [DIP]).

- Board Library: One entry for each board type (breadboard or PWB).
- Function Library: One entry for each nonatomic logic function (e.g., D flip-flop, counter, etc.).

Entries for most of these libraries are obtained in a routine manner from data sheets. Only the function library, which contains simulation models, requires significant effort when adding a new device. Since it is needed only for simulation, the time for developing these models does not delay breadboarding.

The contents of these libraries are determined by the data transformations and mappings that must occur in the design process. For example, functional interconnections must be converted to a set of commands for an automated wiring machine. The engineer indicates on the schematic what functions are to be interconnected, and the draftsman translates them into machine-readable form, partitioning each function to a DIP and placing the DIPs on a board. The HIWIRE program, using the library information, determines from the functional data the connections needed and then calculates the corresponding DIP pins and the location of the board pins into which the DIPs will be inserted. For each connection the program issues the commands necessary to drive the wiring machine to the correct locations and make the connections. In this manner, tedious, error-prone, manual operations are eliminated, saving not only the time spent on inputting data but also on locating the errors that would have occurred.

DESIGN VERIFICATION

At each step in the design process, the data must be verified either manually or by machine. Verification ensures that the data is correct and complete, thereby minimizing repeated passes through the design process. The interface programs in CADRE are used to reformat the data so that its error-prone reentry is reduced. It is possible to accumulate a complete circuit pack description in a series of steps. First, the interconnection of functions is encoded. Then the physical design information is added and, finally, the testing data. The outputs of the system are: (1) a machine-readable description of the board layout, (2) a schematic drawing, and (3) testing data that can be used without change at the manufacturing location.

Of course, the full benefit of this information can only be realized if the manufacturing location is able to verify this consistency and eliminate extensive manual checking. For this reason, it is important to establish a working relationship with the manufacturing location as early as possible in the project so that the verification equipment will be available when needed. With regard to PLAID, we have worked closely with Western Electric-Merrimac Valley (WE-MV) to make sure that efforts are not duplicated. To that effect, WE engineers have been working with us to generate the test information during our design phase. By establishing a joint schedule, it has been possible to send information to WE as it becomes available, instead of waiting until all data are complete in the Bell Laboratories design information (LDI). We are hoping to have the first tool-made sample produced only 4 months after the release of the last specification to WE. This initial work is being performed on a Bell Laboratories experimental (BLX) order so that the usual review of product potential is not needed. Justification of potential market to obtain funding for a small project like PLAID is inappropriate and would have unnecessarily delayed its introduction.

SIMULATION

The construction of wired breadboard models, a step performed to avoid repeating the layout process, is a physical simulation of the actual PWBs. Simulation is very important in the design verification process because the sooner a problem is detected, the fewer the operations that must be repeated to correct the error. Computer simulations are the fastest means of verifying the operation of, and obtaining test data for, each circuit pack. The CADRE function begins with the encoding of circuit connectivity into a language that is used as input to the logic analyzer for maintenance planning (LAMP) simulation program.

There are three distinct phases in the use of the simulation capability of LAMP.[4,5] In the first phase the nominal behavior of the circuit is verified. This step can be performed concurrently with the initial design so that as each section is completed it may be checked. The simulation can be performed either at the gate level, using detailed models of the components, or at a functional level, using simplified models of complex functions and memory units. The functional simulator was used to model the MAC-8 chip at a register transfer level to evaluate the completeness of the design before the chip was available. In a second phase, timing simulation can be used to ensure that, in addition to proper

logical behavior, the sequence of operations will be correct. This step is particularly important for sequential circuits where several functions may have to occur simultaneously to obtain the proper operation. A third phase, fault simulation, is used to evaluate a set of tests so that any incorrect operation can be detected at the terminals of a circuit pack. This simulation should be performed as soon as possible after a circuit design is complete because otherwise it may be impossible to detect some faulty components without providing additional test terminals in the layout.

MODEL CONSTRUCTION

To wire a breadboard, the partitioning of functions to DIPs and their placement on the board are added to the circuit connectivity previously encoded for simulation. Then, as described earlier, the data are transformed into the commands necessary to direct a wiring machine. The library capability of HIWIRE also generates the power and ground wiring for the DIPs that are not explicitly included in the LAMP input.

Wired breadboard models are used to avoid the long delays encountered in routing the paths and fabricating the boards for PWBs. Early in the design process, models are provided that can be checked out in the laboratory and easily modified if the design intent is not realized. Furthermore, the PWB routing, when performed, is less likely to require extensive modification. It is the reduction in this change activity that allowed the drafting estimates for PLAID to be reduced by 25 percent.

In addition, a complete set of board models can be assembled and tested as a system. The time necessary to assemble a complete system using prototype PWBs is extremely long because of queues encountered in the graphical layout and fabrication process. Thus, even system level problems can be detected and corrected before the PWB layout process is begun.

The paper tapes or cards that drive standard semiautomatic or fully automatic wire-wrapping machines are automatically produced. Several companies will wire boards directly from this data. PLAID uses an in-house semiautomatic wire-wrapping machine. Other projects have used commercial, fully automatic machines with an elapsed time of one week from coding until a wired board was

available. PLAID also generates manual wiring lists that can be used for hand-wiring or checking the models.

TESTING

The philosophy of microcomputer testing is undergoing rapid change; the amount of testing that can and should be done at the board level versus the amount done with self-diagnostic programs is particularly being questioned. The CADRE system is capable of generating both acceptance and diagnostic board level tests. It has been used with PLAID, assuming that the chips would be pretested and that most memory tests would be programmed on PLAID itself.

The acceptance and diagnostic tests were developed with the LAMP fault simulator, which uses the breadboard circuit description as input. In the first step a sequence of input test vectors is generated that will allow all classical faults to be detected at the outputs. The fault simulator aids in evaluating the completeness of the tests, but the design engineer is responsible for choosing the test patterns.

The manufacturing process must not only detect bad packs but also diagnose and repair them. The diagnostic approach used in PLAID depends on a guided probe that can access internal nodes on a pack. The probing operation is performed under computer direction until the bad device is found. The additional data needed for the probing program is obtained from LAMP (the internal state) and HIWIRE (the physical location of all components). The flexible interface for testing program (FLINT)[6] is used to reformat and combine the data with the commands needed to operate the Datatron[7] automatic test equipment. Identical Datatron equipment is available at BTL and WE, permitting engineers to develop tests at BTL and see that they are run correctly at WE. The tests can therefore be generated during the design phase and used directly at WE with no duplication of effort.

AUDITING

Once the system operation has been verified using wired boards, the layout of the PWBs can begin. In our system the most important auditing of circuit connectivity occurs at this point. In the layout process the circuit schematic is manually converted into a physical layout and then digitized into the Applicon[8] graphical

editing system. The connectivity audit ensures that the layout corresponds to the data for wiring the model and that a working circuit will be manufactured. Of course, the audit should be used each time the layout is edited to make certain that no errors have been introduced inadvertently.

An audit may be a simple comparison of two similar data sets; however, the one in the APPRAISE[9] set of programs is more sophisticated. It will accept data prepared by different programs and recognize logically equivalent, though not physically identical, circuits. Before the audit is performed a program is run on the Applicon interactive editing station to extract the connectivity and write an output tape for comparison with the reference data. This extraction program also performs a metal-to-metal clearance check[10] to make sure that manufacturing tolerances are maintained. The output of this clearance check is put into a file which indicates individual errors to the Applicon operator so that they may be corrected by editing. In addition to the connectivity data which is furnished, geometric data accounts for differences in library specifications between the two sources. By including data necessary to map one set of component geometries into the other libraries, it is unnecessary to maintain identical data in each library. They can then be constructed to satisfy a particular application, and the different programs easily be accommodated in the audit process. In the layout process, the draftsman is usually permitted to interchange equivalent input or output leads as well as equivalent functions. These changes do not affect the operation of the circuit but do require a more complicated audit capability in order to recognize that the changed connections are not errors. When equivalent changes are made, it is necessary to repeat the test data preparation step to achieve a manufacturing test that corresponds to the actual layout.

Although we have not yet reached the auditing step on PLAID, it was found in the EST project that none of the 30 boards audited had any errors when checked by WE. In addition, each connectivity audit replaced 7-1/2 to 10 hours of meticulous manual checking. The cost of the audit is low. It requires (1) about 5 minutes of Applicon editing, (2) that the extraction program run unattended on the minicomputer (approximately 40 minutes), and (3) 1 minute of main frame computer time for the actual audit. The auditing process thus provides a relatively inexpensive way to improve the quality of information transferred. WE can also use the program to verify consistency.

MACHINE-DRAWN SCHEMATICS

The engineer's hand-drawn schematic is adequate for much of the design process but it must be redrawn and standardized for use in WE and the telephone companies. The object of much of WE checking is to determine whether the manually drawn circuit pack schematics (CPSs) agree with the layout of the PWBs. In the EST project, errors on the manual schematics accounted for the only inconsistencies since the audit was used on the boards. It is difficult to extract connectivity from a schematic for auditing, but machine-generated schematics from the same source as the audit ensure consistency.

The automatic circuit pack schematics (AUTO-CPS)[11] program will be tried on PLAID to generate schematics on the computer from the HIWIRE input data. The schematic is divided into pages, and the aesthetics are applied interactively, but the connectivity is maintained. The editing takes about 4 hours, and computer costs are under $100 for each schematic, considerably less expensive than the week of drafting effort required for a manual schematic. Thus a less expensive means of obtaining schematics which are consistent with the audited layout may prove to be effective.

CONCLUSION

The development of PLAID on a close schedule at low cost has been greatly aided by the CADRE system. The various components of the system have saved significant amounts of time by reducing manual operations and eliminating redundant work. Moreover, costs have been kept low by ensuring that repeated passes through the more expensive operations, e.g., PWB layout, are kept to a minimum. The close cooperation with WE-MW throughout the project promises a smooth transition from development to manufacture. It appears that CADRE can greatly enhance our chances of realizing the full potential of the MAC-8, as well as getting the product to the market in the shortest possible time.

ACKNOWLEDGMENT

The author wishes to acknowledge the work of all those who have contributed to the development of CADRE. They include not only the writers of the various pro-

grams listed in the references but all the users whose suggestions have made it an effective system. In particular, the contributions of J. Van Zweden, who coordinated the physical design of PLAID, are gratefully acknowledged.

REFERENCES

1. C. F. Shupe, K. W. Johnson, and B. B. Hofmann, "Testing and Debugging of MAC-8 Systems," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey 1976.

2. D. S. Evans and L. A. O Neill, CADRE: An Integrated System for the Design of Printed Wiring Boards, TM-76-4393-15, September 8, 1976.

3. D. S. Evans and L. C. Thomas, HIWIRE User's Manual, Memorandom for File, Case 39898-13, October 12, 1976.

4. C. H. Elmendorf and R. B. Schmidt, LS LAMP System User's Manual, TM-74-5334-1, TM-74-5331-2, July 22, 1974.

5. S. G. Chappell, P. R. Menon and A. M. Schone, Functional Simulation in the LAMP System - User's Manual - Version 1, TM-75-5331-5, June 19, 1975.

6. D. S. Evans and F. W. Kerfoot, FLINT D User's Manual, TM-76-4393-12, TM-76-4392-10, July 30, 1976.

7. Digital Module Tester Operator's Manual (Model 4400-128), Datatron, Incorporated, Santa Ana, California, January 26, 1973.

8. Computerized Graphic Processing Systems User's Manual, Applicon Incorporated, Burlington, Massachusetts, September, 1973.

9. R. M. Allgair, Connectivity Audit User's Manual for Version A, Memorandum for File, Case 39898-13, January 26, 1976.

10. R. C. Fairbrother, Field Trial Release of Printed Circuit Metal-to-Metal Clearance Checking Programs, Memorandum for File, Case 36361-7, August 6, 1976.

11. Solar Flare No. 1, Bell Telephone Laboratories, Dept 8623, July, 1976.

SUPPORT FACILITIES

SWAT - A DEBUGGING SYSTEM FOR THE DIMENSION PBX

T. B. Cannon, BTL Dept 3222, DR, CO

ABSTRACT

The CSS 201 switch analysis tool (SWAT) is a debugging system designed to aid in the development of the DIMENSION private branch exchange (PBX) product line. The development of SWAT is discussed, and the advantages of using a stored program controller, i.e. microcomputer, to provide a sophisticated portable test system on a short schedule are demonstrated. Several concepts which allowed a smooth evolution of SWAT while encompassing the ever-present new and changing requirements are described.

SWAT is a debugging system, designed to aid in the development of the DIMENSION family of stored program control PBX systems. SWAT is biased toward the testing and debugging of software in real time, but it is also a powerful tool for debugging firmware (microprograms) and hardware.

The major SWAT features are:

- READ/WRITE

  Memory, registers, input/output (I/O), flags.

- PROCESSOR CONTROL

  Halt, go, single-step, multistep.

- MATCHERS

  Four programmable hardware matchers.

- TRACE MEMORY

  Programmable, real-time history memory.

- REAL-TIME MEASUREMENT

  Two programmable timers/counters.

SWAT users may reference the CSS 201 memory in an absolute, relative, or indexed addressing mode. Eight tables are provided for storage of user-defined base address, offset, block length, and reference tag, allowing quick and easy display of different tables and programs stored in memory. When registers are read, the contents are displayed along with a mnemonic header for easy referencing.

The matchers monitor the CSS 201 system for user-specified operations. The user may stipulate that a match should occur on any combination of the following operations: memory read, memory write, I/O read, I/O write, or instruction fetch. In addition, the user may require that the address bus, data bus, and miscellaneous flags have any specific bit pattern including don't cares. Thus, the user can specify complex events, such as writing a 1 in bit 6 of memory location $0121F_{16}$. The user may also designate the match response mode to halt the processor operation, enable/disable automatically any of the four matchers, execute a programmable task queue, or pass. The pass mode blinks a light-emitting diode (LED) indicator and provides a pulse to synchronize an oscilloscope, etc.

The trace memory provides a 255- by 20-bit memory array which will store up to 255 program addresses in real time. When it is used as a program trace memory, the address of the last 255 instructions executed, the address of the last 255 branch instructions executed, or the last 127 branched from-to program address pairs may be stored. For conditional branch instructions, only those which take the branch are saved. When it is used as an event trace memory, any one of the matchers may be used to specify an event to be traced, i.e., the program address of the last 255 instructions writing a 1 in bit 6 of memory location $0121F_{16}$ may be stored. The user may also specify when the trace memory should function. Normally, it does so continuously, and its contents are displayed after a match event halts the processor operation. Alternatively, the user may specify a start trace event and a stop trace event with the matchers in order to take a snapshot trace without altering the CSS 201 operation.

The real-time measurement option provides two 7-digit decade counters plus a passes counter for averaging. Any event which can be specified by the matchers can be measured. Thus the execution time of a single instruction or an entire

task may be measured. The user may also specify a master/slave mode to allow subtracting time spent on other tasks such as the interrupt handler. The clock for each counter may be specified as an internal 10-MHz crystal or an external clock for measuring time, or as a matcher output for counting the number of match events up to 9999999.

The CSS 201 family utilizes two different processors. Each is microprogrammed and executes a similar dialect of the 3A language. The MC3 processor, used in CSS 201VS and CSS 201S systems, is designed with medium scale integration (MSI) transistor-transistor logic (TTL) technology and has a 24-bit microinstruction, a 16-bit address bus, a 16-bit data bus, and a typical microcycle of 1.2 μs. The 201CC processor, used in CSS 201L systems, is designed around the Intel 3000 series microprocessor chip set and has a 32-bit microinstruction, a 20-bit address bus, a 16-bit data bus, and a microcycle of 225 ns.

A SWAT system, shown in Figure 11-1, interfaces the CSS 201 system buses, internal processor microbuses, and control leads via an interface circuit board. The interface board plugs into a dedicated test slot in the CSS 201 control carrier. A different interface board is provided for each of the two different CSS 201 processors.

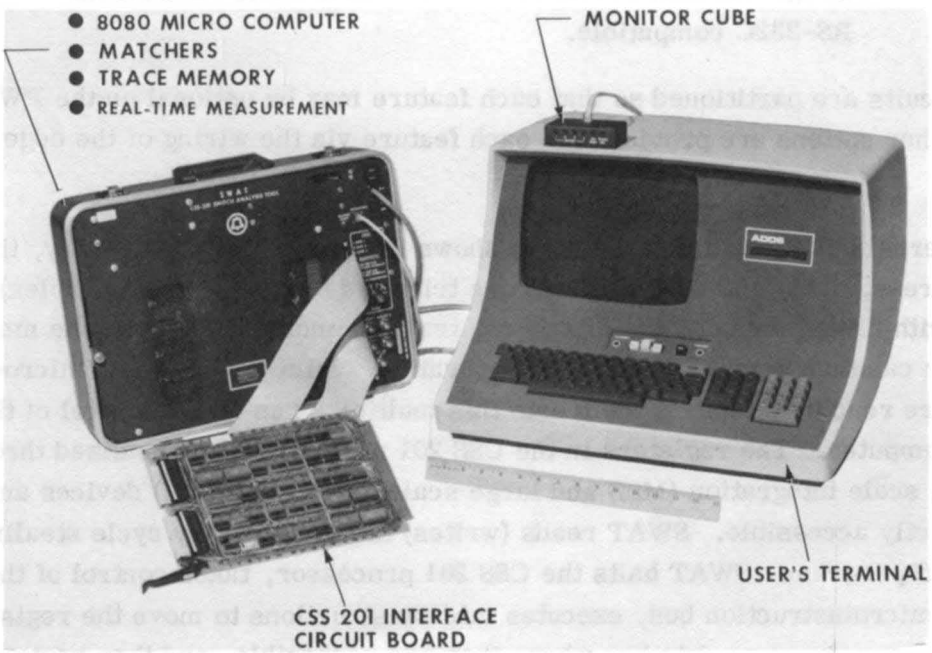


Figure 11-1 - CSS 201 Switch Analysis Tool (SWAT)

The typical user interface is via a cathode ray tube (CRT) terminal, but in general, any RS-232C compatible terminal may be used. A simple and easy-to-use command language provides communication with the SWAT functions and CSS 201 system. This language is common to all of the CSS 201 systems since any differences between the two processors are automatically rectified by SWAT.

At the heart of the SWAT system is a microcomputer based on the Intel 8080A microprocessor.[1] This microcomputer has been laid out on a printed wiring board (PWB), approximately 8-1/2 by 11 inches, which has a 100-pin edge connector. This general-purpose, single-board microcomputer provides the following features:

- Intel 8080A microprocessor, system controller, and clock.
- Programmable read-only memory (PROM)/ROM, 8K words in 1K-word increments (Intel 8708/8308).
- Random access memory (RAM), 1K words in 256-word increments (Intel 8101).
- Priority interrupt structure, 8 levels, programmable.
- Programmable serial interface, 7 selectable baud rates, RS-232C compatible.

The circuits are partitioned so that each feature may be optional on the PWB, and many other options are provided for each feature via the wiring of the edge connector.

The internal bus structure of SWAT is shown in Figure 11-2. Normally, the CSS 201 address, data, and control buses are tristated onto a 44-bit (multiplex) bus. Along with timing signals, the matchers, trace memory, and real-time measuring features can then monitor the CSS 201 operation. Microaddress and microdata buses are read by tristating them onto this multiplex bus under control of the 8080 microcomputer. The registers in the CSS 201 processors are realized through medium scale integration (MSI) and large scale integration (LSI) devices and are not directly accessible. SWAT reads (writes) the registers by cycle stealing from the microprogram. SWAT halts the CSS 201 processor, takes control of the processor microinstruction bus, executes microinstructions to move the register contents to the address or data bus where they are accessible, and then restores the CSS 201 processor. Memory, flags, and I/O are handled in a similar fashion.

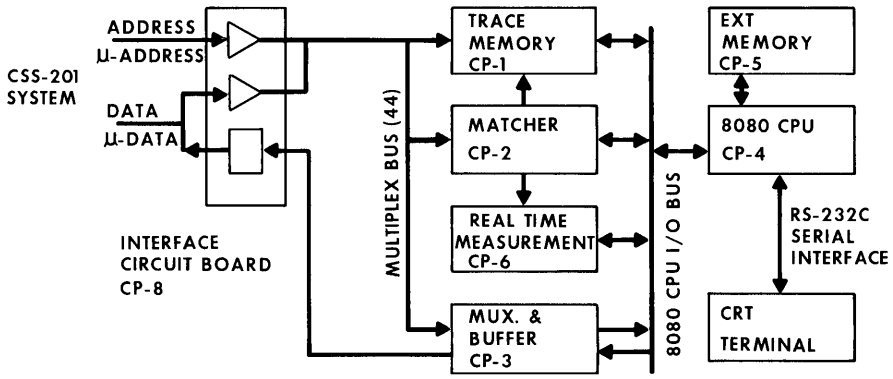


Figure 11-2 - SWAT Bus Structure (CSS 201L Interface)

SWAT development began in July of 1975 and ran concurrently with the development of the CSS 201L system. A breadboard SWAT system with simple read/write and processor control features was operating in October. The breadboard only required two of the circuit boards shown in Figure 11-2, the interface (CP-8) and the multiplex/buffer (CP-3). An Intel Microcomputer Development System, MDS-800, was used as the central processing unit (CPU), providing read/write program store, monitor program for debugging SWAT software, and time to design the SWAT CPU later in the schedule. The matchers, trace memory, and real-time measurement circuits and software drivers were added in November, January, and February, respectively. The SWAT CPU design was also completed in January, and the first two CSS 201L SWAT systems were delivered in April. The interface circuit (CP-7) and software for the CSS 201S systems were completed and the first CSS 201S SWAT system delivered in May.

CONCLUSIONS AND SUMMARY

A very powerful and portable test and debugging system was designed and tested in a short time by using stored program control, i.e., a microcomputer. An estimated two EMTS years of effort were expended, divided approximately equally between hardware and software.

The decision to use a standard CRT terminal as the user interface contributed to achieving the short schedules. Considerable design effort was eliminated by not integrating this function directly into SWAT, especially for a limited-production test system.

Throughout the development of SWAT, a basic design philosophy was used: If it can be done in software, it shall be. This proved to be an important decision when incorporating the ever-present changes in the requirements, features, and the CSS 201 itself.

An Intel MDS-800 with the in-circuit emulator, ICE-80 option, was used for debugging software and testing the design of the SWAT CPU during the second half of the SWAT development. It is the author's opinion that the money spent was recovered many times over in saved time and effort.

The structured macro assembly language $(SMAL)^2$ was used to write the SWAT software. This saved considerable effort over writing in assembly language. Much more time would have been saved if SMAL had generated relocatable program modules or had been able to handle large programs (10K bytes). Software support and choice of language are very important in the development of microcomputer-based systems.

REFERENCES

1. Intel 8080 Microcomputer Systems User's Manual, Intel Corporation, Santa Clara, California, September, 1975.

2. C. Popper, SMAL-A Structured Macro-Assembly Language for a Microprocessor, TM-74-3233-3, August 1, 1974.

APPLICATIONS I

THE LOOP SWITCHING SYSTEM

N. G. Avaneas and J. M. Brown, BTL Dept 4533, WH, NJ

ABSTRACT

A new line concentrator, the Loop Switching System (LSS), has been developed. It uses PROCON, a programmable controller, to control the call processing, traffic measurement and display, system maintenance, and manual testing.

INTRODUCTION

The LSS concentrates 96 subscriber lines onto 32 voice-frequency trunk pairs by means of a graded multiple space-division switching network at the remote terminal (RT). The trunks connect with a central office terminal (COT) which expands the trunks into the individual line appearances. An additional 96 lines and 32 trunks can be added, either at the same RT or at a second RT on a different cable route, using the same COT common control equipment.

In addition to the trunk pairs, each RT requires two standard voice-frequency pairs for a full-duplex data link. This 4-wire data link, operating at 1250 b/s, is used to transmit concentrator connection orders from the COT to the RT and to transmit service request information from the RT to the COT.

The LSS system configuration for 1- and for 2-RT operation is illustrated in Figures 12-1 and 12-2, respectively.

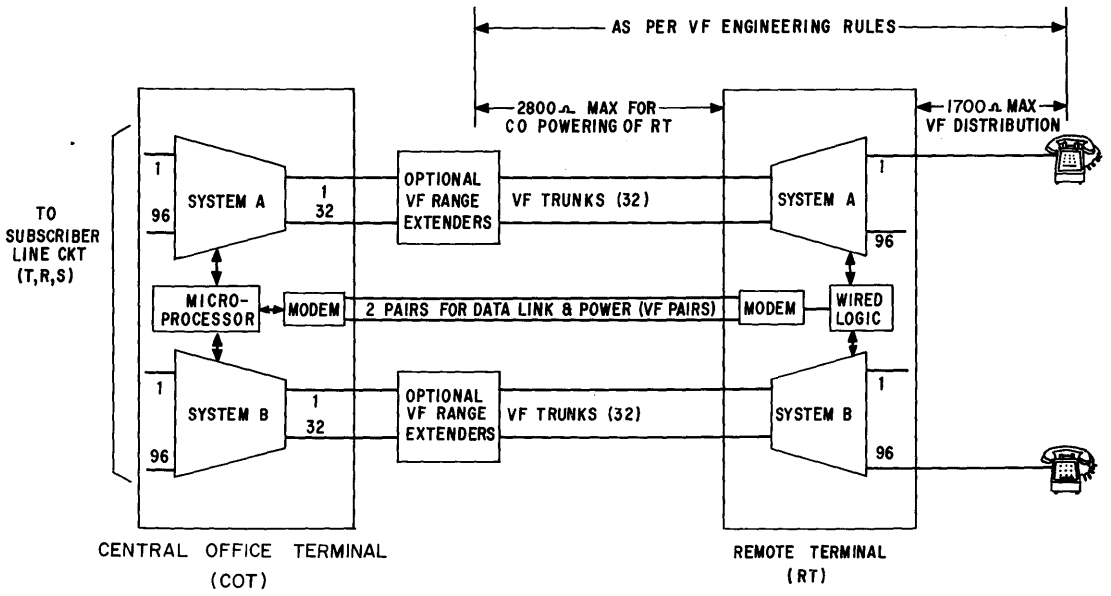


Figure 12-1 - Typical LSS System Layout - One RT

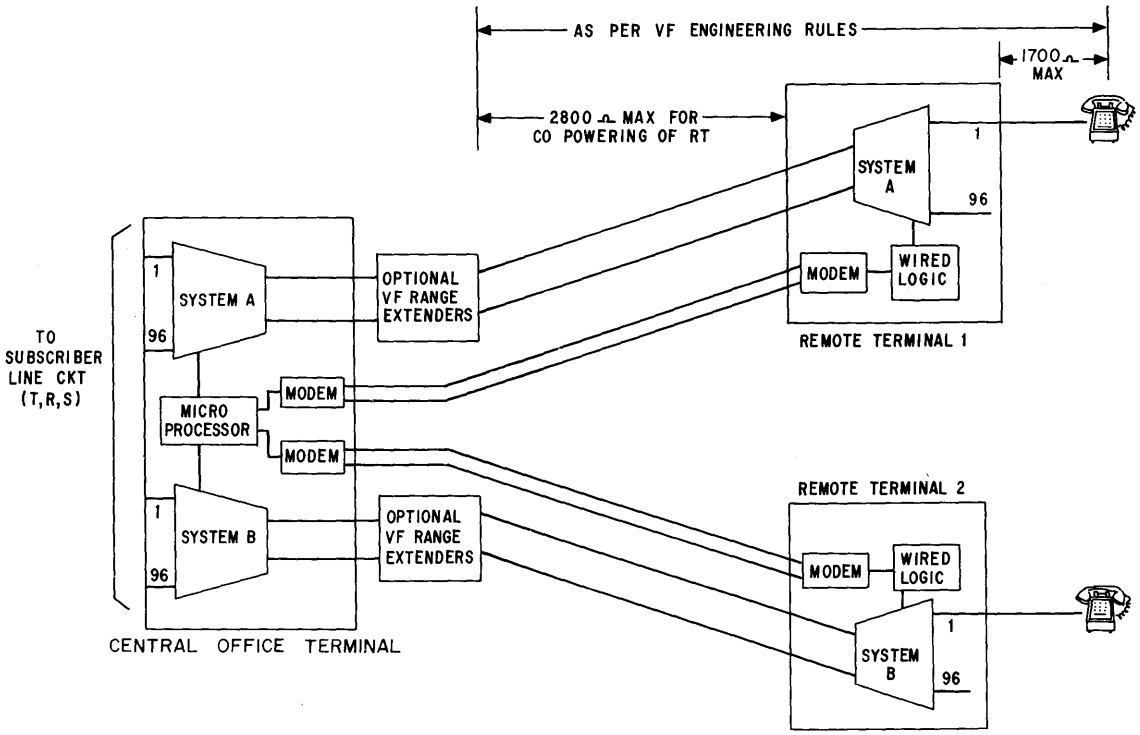


Figure 12-2 - Typical LSS System Layout - Two RTs

SWITCHING NETWORK

The LSS uses a graded multiple switching network to minimize common equipment, and hence minimize getting-started costs and spare plug-in unit stocking costs. The graded multiple is less costly but less efficient in terms of traffic than a 1- or 2-stage network; 32 trunks are required instead of 28 for the same traffic handling capability.

Each of four lines, in the graded multiple network, has full access to only seven trunks. These seven trunks are multipled to other groups in an appropriate manner to minimize trunk commonality between line groups, and to maximize traffic capacity. Thus the entire switching network is available on a per-line basis and is added as a function of lines loaded on the concentrator.

The Western Electric (WE) BL7 multicontact, magnetically latching, miniature, wire-spring relay is used as the switching element. A latching switching element is desirable to minimize power drain. This is a particularly important consideration for a batteryless, simplex-powered RT which is constrained to consume less than 10W of power. As shown in Figure 12-3, three of these multicontact relays (K1, K2, K3), wired in a tree configuration, provide each line with access to one of seven trunks. The eighth port on this tree network is connected to a per-line service request detector at the RT and to a ring trip and overflow circuit at the COT. Overflow is applied to a line in the COT when all of the seven accessible trunks are busy. A fourth relay (K4), a nonlatching, miniature, flat-spring relay, isolates the line from the switching network during the switching of the three per-line relays. This will avoid objectionable transients on the line and on the working trunks. Contacts on the K4 relays at the RT and COT are also used to check the trunk and the switching network for leakage and continuity just before the relays cut through the line. This preconnection trunk testing ensures that a defective trunk is not assigned to a subscriber.

CALL PROCESSING

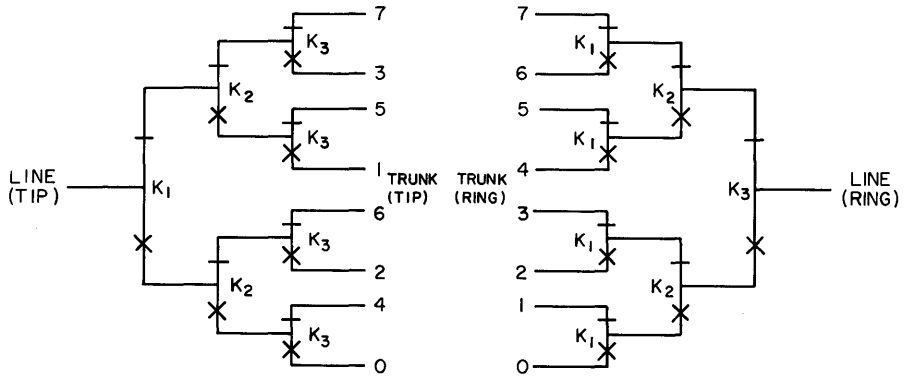
The chief constituent of the concentrator is a stored program controller at the COT. This controller implements the call processing algorithm and communicates, via the data link, with a relatively simple wired logic processor at the RT. The common control circuitry connects a trunk in accordance with the results of

both an RT scan of the service request detectors and a COT scan of the sleeve lead status. A trunk is disconnected only when a sleeve lead is ungrounded.

The processor transmits scan, network, and activate orders to the RT. The response of the RT to each of these orders is discussed below.

Upon receiving a scan order, the RT transmits the number of the line requesting service and, for redundancy, its binary complement. If differences occur between the line number and its complement, the COT rejects the scan result and initiates another scan order.

When a network order is received by the RT, it is stored in a register and is also returned to the COT for comparison with the original transmission. If no errors are present, the COT transmits an activate order to the RT. This triggers the



NOTE: WHEN NO TRUNK IS CONNECTED, THE K_1, K_2, AND K_3 RELAYS ARE LATCHED: I.E., MAKE CONTACTS ARE CLOSED

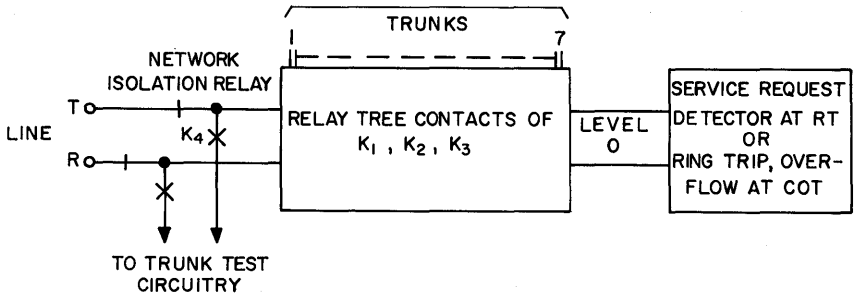


Figure 12-3 - Relay Tree

RT to implement the network order, e.g., connect a trunk to a line. The activate order must be received by the RT within 20 ms of the network order so that the network order is implemented.

After the trunk is connected to a line, the RT returns the confirmation answer, which indicates the actual trunk level connected to the line, to the COT. The confirmation answer is compared with the original network order for consistency, and corrective action is initiated if differences exist.

If two RTs are connected with one COT via two separate data links, the COT scans one RT and then the other for call originations. If an origination is discovered in one RT, the necessary network order is transmitted to that RT only. Scanning then resumes with the other RT.

Data transmission between the COT and RT is asynchronous, i.e., information is not transmitted in predetermined and regular time slots. Instead, orders and answers to orders are recognized by a unique series of information bits. These orders and answers are shown in Figure 12-4.

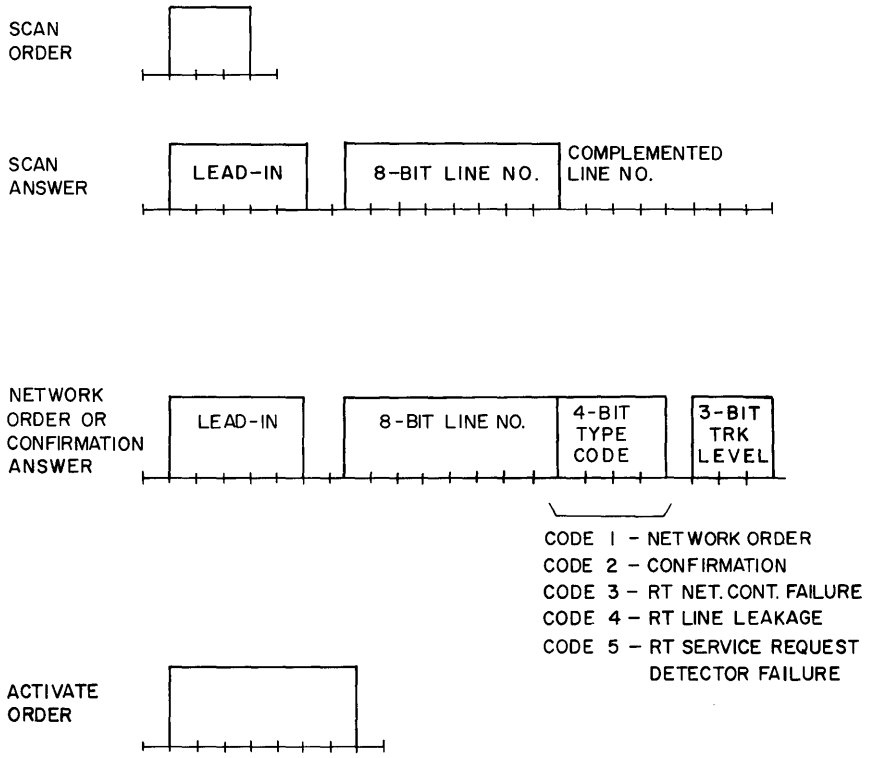


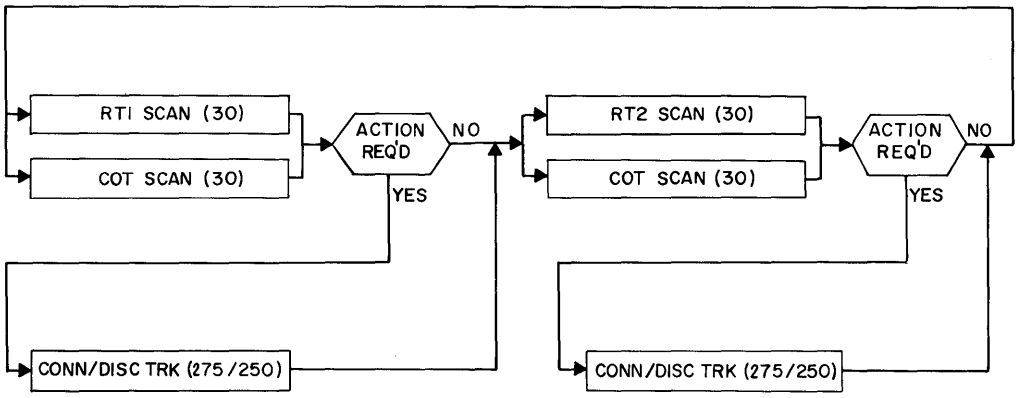
Figure 12-4 - Data Organization for Orders and Answers

The timing associated with the LSS call processing algorithm is shown in Figure 12-5. For a terminating call, a trunk is connected within 275 to 335 ms after the sleeve lead is grounded. For an originating call, a trunk is connected within 375 to 435 ms after a station set goes off-hook. Originating call times include 100 ms for the service request detector, which is heavily filtered to reject 60-Hz longitudinal signals.

PROCESSOR

The LSS is controlled by PROCON, a programmable controller manufactured by WE. The PROCON used in LSS is an 8-bit data, 24-bit instruction, nonself-checking unit with multiparity checking and a 500-kHz clock. Every instruction is fetched and executed in one clock period (2 μs).

To control LSS, 5000 words of read-only memory (ROM) (instructions) and 500 words of random-access memory (RAM) (data) are required. Approximately 2000 words of program instruction are used for basic call processing, and 3000 words are used for automatic trouble locating, manual testing, and traffic measurements.



NOTE : ALL NUMBERS IN MILLISECONDS (MS)

CONNECTION TIME*	275 - 335
DISCONNECT TIME	250 - 310

* FOR CALLS ORIGINATING AT THE RT, ADD APPROXIMATELY 100MS FOR THE TIME DELAY OF THE SERVICE REQUEST DETECTOR. IF TRUNKS ARE DERIVED FROM SLC-40 OR USE THE 5A RANGE EXTENDER, ADD THE TIME DELAY CONTRIBUTED BY THESE UNITS.

Figure 12-5 - Call Processing Times

As shown in Figure 12-6, the separate 8-bit input and output buses of PROCON are combined into an 8-bit I/O data bus using hardware external to PROCON. Data is transferred between PROCON and the peripheral units by means of this I/O bus.

Thirty-two output control signals are used to selectively transfer data via the I/O bus from PROCON to various other peripheral units. The PROCON destination signals D0 and D2 and four device selection signals INi (i=0, 1, 2, 3), are used to generate these 32 control lines by means of a decoder external to PROCON.

A different set of 32 input control signals is used to selectively transfer data from the peripheral units to PROCON via the I/O bus. These control signals are generated from the PROCON source signals S1 and S2, and from the same four device selection signals INi used for the output control.

The 512-word RAM is divided into an upper and lower half, corresponding to the ninth bit of the 9-bit RAM address. As long as successive RAM addresses are confined to either the upper or lower half of the RAM, the RAM can be addressed with eight bits (one PROCON instruction) plus the ninth bit previously stored. But, if the RAM address changes from one half of the RAM to the other, two instructions are required to generate the 9-bit RAM address. After the RAM address is stored in the RAM address register, the RAM is accessed in one instruction which enables either the S0 read control signal or the D1 write control signal.

A timing signal generator external to PROCON generates all of the timing signals necessary for the operation of LSS. This generator is driven by the PROCON clock and is controlled by the stored program.

MAINTENANCE FEATURES

The LSS processor controls a number of maintenance features such as continuous monitoring of performance, alarm displays at the COT - which indicate system status and defective plug-in units, automatic troubleshooting procedures - which aid in maintaining and restoring service, and manual maintenance procedures. Some of the routine maintenance functions performed automatically by LSS include the following:

- Each time a trunk is assigned to a line, the trunk assignment is verified, and the trunk and switching network are tested for leakage (less than 30KΩ) and continuity.
- Data transmission between the COT and the RT is checked for accuracy.
- Performance of the alternate data link pairs is tested periodically.
- All relays are operated and released once a day.
- Much of the per-line and common control circuitry is tested for proper performance on a routine basis.

The following troubleshooting procedures are implemented automatically by the LSS processor, on a selective basis, when the routine maintenance tests show a failure:

- When the LSS is unable to process a call, a means is provided to transfer automatically from the main data link pairs to an alternate set of pairs. If it is not possible to communicate with the RT via the main or alternate data link pairs, all of the critical common control plug-in units in the COT are tested automatically, and the appropriate plug-in and CO alarms are raised.
- When COT power is restored after a failure or a critical plug-in unit is replaced, the LSS automatically reinitializes itself by disconnecting all idle lines and reconnecting all active lines (sleeve grounded).
- When a line unit is plugged in at the COT or at the RT, all idle lines are disconnected in order to ensure that the switching network is in a known state.

The COT is equipped with an alarm plug-in unit containing 20 indicator lights which display the status of the LSS as well as indicate which plug-in units are defective. A COT test and display plug-in unit contains numeric displays and indicator lights which show LSS traffic, the number of the line and trunk being manually tested and the results of the tests, the number of busy trunks, and the number of the trunk connected to a particular line. Pushbuttons are used to select a particular display. Two lever-wheel switches select a particular line number for test and display.

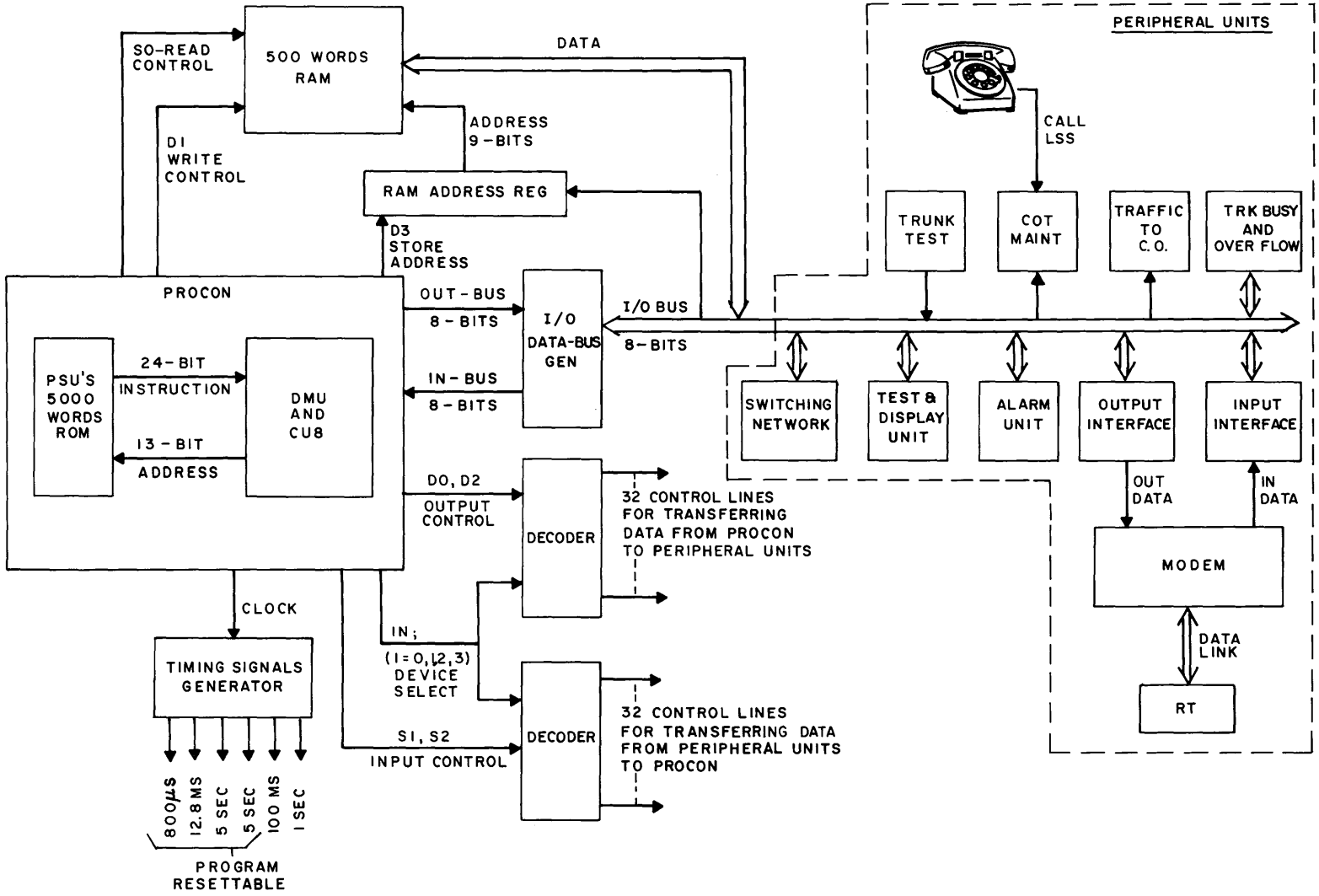


Figure 12-6 - Block Diagram Showing Communication Paths Between PROCON, COT Units, and the RT

A maintenance plug-in unit at the RT initiates line-trunk tests from the RT and indicates the results of these tests. Four other indicator lights show system status and plug-in failures at the RT.

PHYSICAL DESIGN

As shown in Figure 12-7, the COT of the LSS requires a maximum of 61 inches of vertical space on a 23-inch wide miscellaneous bay and consists of up to four assemblies. The 17-inch common control assembly consists of a fuse and alarm panel, two power units, PROCON, and an 8-inch shelf for 13 common control plug-in units.

The common control assembly for RT2, required whenever a second RT is provided, consists of a 4-inch high shelf which accommodates a power unit and a modem.

The 20-inch line unit assembly consists of a 14-inch high tray, which accommodates up to twelve 8-line line units, and a 6-inch high field of terminals which provides test access to the line, trunks, and other pairs.

A second line unit assembly is used for a second set of 96 lines.

As shown in Figure 12-8, the RT of the LSS is mounted in a cabinet 48 inches high, 29 inches wide, and 13 inches deep, and consists of up to four assemblies.

The common control assembly consists of an 8-inch high shelf for eight common control plug-in units.

The line unit assembly consists of a 14-inch high shelf which accommodates up to twelve 8-line line units. It is equipped with gas tube protectors for the trunks, lines, data link, and order-wire pairs.

A second line unit assembly is added for a second set of 96 lines.

A 3-inch panel contains a thermostatically controlled heater and a duplex power outlet. It is required only when it is necessary to control the humidity level within the cabinet.

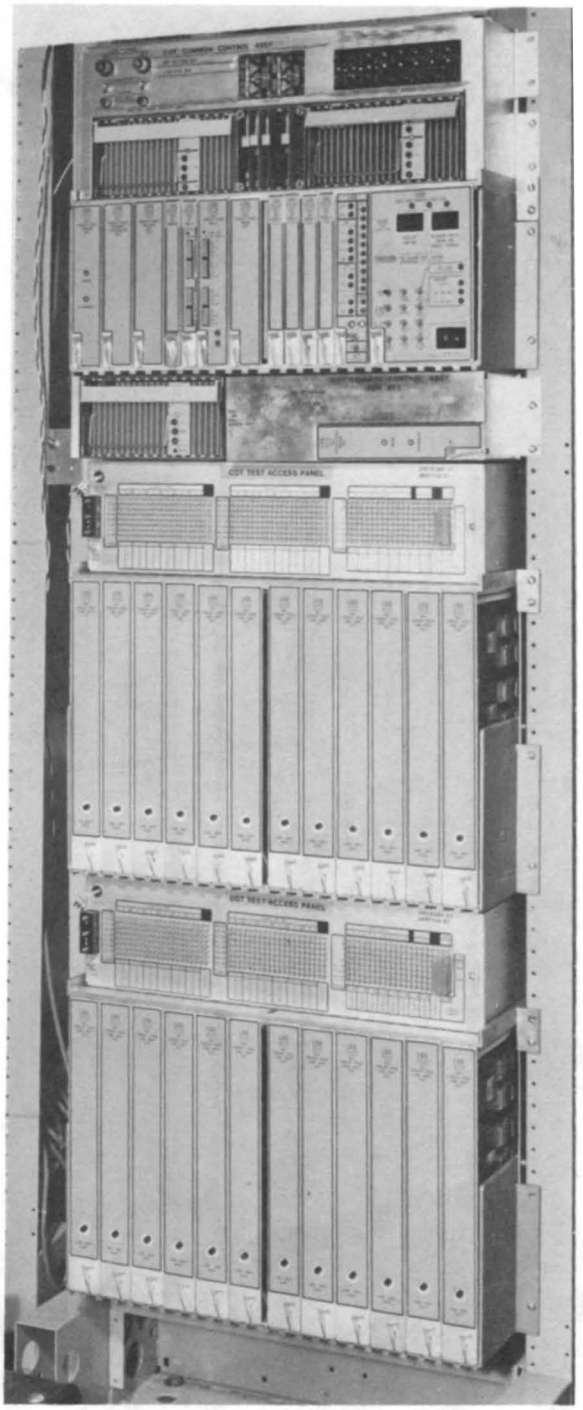


Figure 12-7 - Central Office Terminal

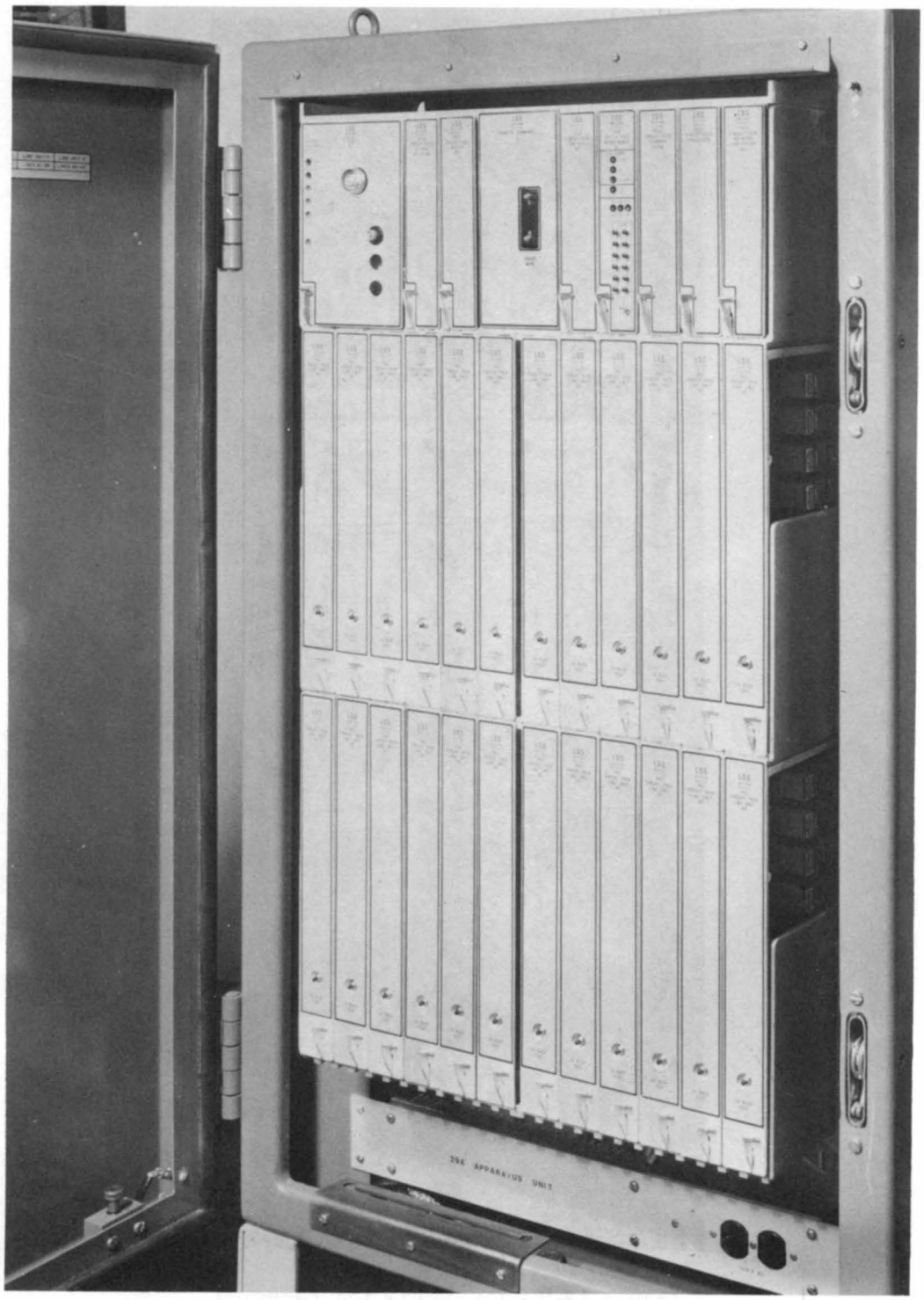


Figure 12-8 - Remote Terminal

SUMMARY

The Loop Switching System described in this paper utilizes the PROCON stored program controller, which permits a wide variety of operational features to be included in the system. This flexibility would not have been possible if a hard wired logic approach had been used. By storing the history of system performance, a more accurate and thorough diagnosis of system trouble is possible, and a more flexible call processing algorithm can be implemented. In addition, the turnaround time for correcting system bugs and adding features is greatly reduced when such changes are implemented by software modifications.

The LSS field trial began during August, 1976 near San Diego, California. Production systems will be available from Western Electric during 1977.

APPLICATIONS I

TASI-E SOFTWARE DEVELOPMENT

J. C. Selbo, BTL Dept 4393, HO, NJ

ABSTRACT

This paper discusses software development for the microprocessor controller for the experimental time assignment speech interpolation (TASI-E) terminal. TASI-E is a system which takes advantage of normal pauses in telephone conversations to increase the circuit capacity of existing domestic analog transmission facilities. The overall system design and the functions performed by the microprocessor are outlined. Various factors affecting the organization of the software for this interrupt-driven system are explained. Memory and real-time requirements are discussed as well as the decision to exploit memory in order to save real time. The use of such development tools as Intel PL/M compiler, simulator, microprocessor development system (MDS), and in-circuit emulator (ICE) equipment to meet a tight schedule is also explored.

INTRODUCTION

TASI-E is a system which takes advantage of normal pauses in human speech during a telephone conversation to increase the circuit capacity of existing domestic analog transmission facilities.[1, 2, 3] When speech activity ceases on any one of 120 trunks, its particular channel, 1 of 48, may be taken away and given to another trunk which has become active. All processing of speech is done in sampled digital form and all trunk-channel assignments are made under control of the microprocessor.

SYSTEM OVERVIEW

The primary functions performed by the microprocessor at the transmitting end include obtaining data from the trunk status detector, making new trunk-channel assignments, directing the time slot interchange (TSI) to connect a specific trunk to its assigned channel, and signaling the assignment to the far end. The receiving microprocessor then processes the incoming signal and directs the receiving TSI to make the corresponding connection. A block diagram of the microprocessor interfaces is presented in Figure 13-1.

The microprocessor also performs such secondary functions as updating noise and volume for each trunk, gathering traffic information which is the output of a thermal printer, and monitoring overload and failure conditions. The test panel displays the status of the system and permits craftspersons to interact with it.

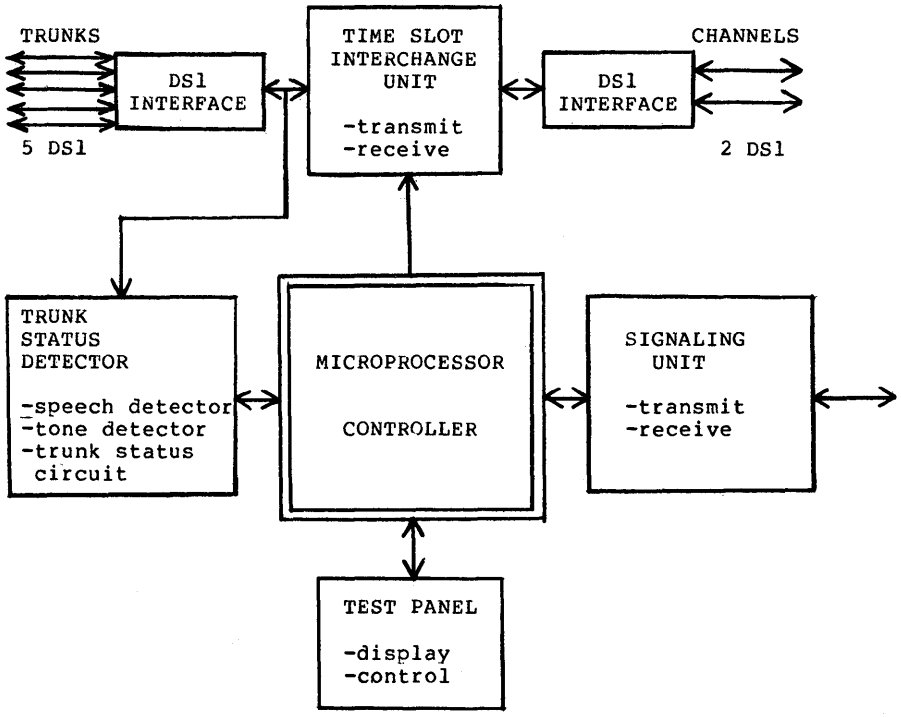


Figure 13-1 - TASI-E Microprocessor Interfaces

MICROPROCESSOR HARDWARE

TASI-E uses the Intel 8-bit single-board computer (SBC) 80/10 with the 8080A central processing unit (CPU). This 80/10 board is expanded with an additional SBC-416 programmable read-only memory (PROM) board, an SBC-016 random-access memory (RAM) board, an SBC-508 input/output (I/O) board, and an interrupt board. The present system has a capacity of 15K PROM and 17K RAM. The current version of the program requires 13.5K PROM and 4K RAM.

REAL-TIME REQUIREMENTS

TASI-E is a completely interrupt-driven, real-time system with four priority interrupt levels:

- Channel A receive.
- Channel A transmit.
- Test panel.
- One-second clock.

The real-time pace of the system is set by the rate of the signaling channel. The microprocessor receives a transmit interrupt from the signaling channel every 6.7 ms. The system objective is to prepare a new-connect or an update signal to be the output during the appropriate window for each transmit interrupt. This signal preparation process may also be interrupted by a higher priority receive signaling channel whose information must be processed immediately. Therefore, real-time requirements are immediate and critical.

PROGRAM ORGANIZATION AND USE OF A HIGH-LEVEL LANGUAGE

Because of a tight 5-month software development schedule for the experimental TASI-E terminal, the decision was made to use PL/M, the Intel high-level language, to write the microprocessor software. Its high-level constructions (if-then-else, do case, do while, etc.) facilitate the writing of structured code which, with in-line comments, is self-documenting and easily maintained and modified. With the continuous evolution of the system throughout the development interval, numerous system changes and additions were made. Ease of software modification was essential.

Typically, one line of high-level code results in more machine processing than one line of assembly language code; however, both take the same amount of programming time.[4] Therefore, the necessary system code can be written in a much shorter time using a high-level language such as PL/M.

Code produced by a compiler tends to be less efficient than that written by a skilled assembly language programmer. This inefficiency must be weighed against a longer development time which might cause the schedule to be missed completely. One way to help alleviate the problem is to develop the system software using a high-level language, then identify time-critical paths and recode these as assembly language routines. Perhaps an even more effective course is to consider alternate algorithms or structures which would require less execution time.

With TASI-E it was possible to meet the real-time requirements using PL/M by careful organization of the system. The decision was made to exploit memory in order to save real time. Table look-up techniques were used to avoid on-line computations. For example, a table of total noise (combined trunk and channel noise) was created and stored in PROM. During real-time system operation, the trunk noise bits are logically ORed with the channel noise bits and the result used as the entry address into the total noise table. A similar table was used to perform gray-to-binary code conversion. Additional RAM was also utilized to save real time. Much of the input to the microprocessor consists of several parameters packed into a single byte. The individual parameters are isolated by the software and stored in RAM in separate machine bytes. They are then available for use without requiring the repeated bit manipulation of a packed byte for each parameter reference.

During preliminary studies of system performance it was discovered that real-time limits were being exceeded during system overload. Because the only variable factor dependent on system load was the size of the queues, the queue structure was reorganized from a simple linear list to a doubly linked list. It is frequently necessary during system operation to remove an element from the middle of a queue. This process becomes time-consuming for a long linear list. The doubly linked list required more memory but greatly reduced the processing time for the removal of an internal element. It made the time independent of queue size because it eliminated the need for searching and for sliding elements to close the gap.

The software functions were organized, as much as possible, to minimize the time periods during which interrupts are completely disabled. Operations involving queue handling, for example, are performed sequentially within one priority interrupt level. Therefore, the queue-processing routines are interruptable, but the system organization ensures that the interrupting procedure cannot be one which will also require queue operations. Therefore, the need for reentrant code is eliminated, and such problems as the possible destruction of the chains of a doubly linked list (which could occur if nested interrupts attempt to process the same queue) are also prevented.

LIMITATIONS IMPOSED BY INTEL SYSTEMS

It was discovered that the Intel SBC 80/10 board has only one CPU interrupt level. The priority interrupt structure required was achieved by adding an interrupt controller on the interrupt interface board and modifying the SBC 80/10 board to make the necessary CPU signal available for connection to the interrupt controller. Still, only one restart level is sent into the CPU; therefore, it was necessary to write an assembly language utility to save the processor status, to read the interrupt level from the input port of the interrupt controller, and to transfer control to the appropriate service routine. This process added approximately 30 μs to the execution time required to honor an interrupt. Under these conditions it was not appropriate to utilize the interrupt procedure facility available in PL/M.

Another limitation encountered involved the layout of PROM and RAM in the system. The SBC 80/10 board contains 4K PROM and 1K RAM both with fixed address ranges. The PROM and RAM boards have restricted selectable base addresses. It is advantageous to make full use of the memory on the 80/10 board because it can be accessed by the CPU without the additional wait states associated with off-board memory references. Also, the present version of the PL/M compiler does not allow the placement of a segment of RAM between two segments of PROM. Therefore, the only configuration possible is the placement of the PROM board so that it overlaps the address range of the PROM and RAM on the 80/10 board, as shown in Figure 13-2, thus making these portions of the PROM board unusable and limiting the available system PROM to 15K. An attempt was made to arrange the program so that critical, frequently used portions of memory reside on the faster SBC 80/10 board.

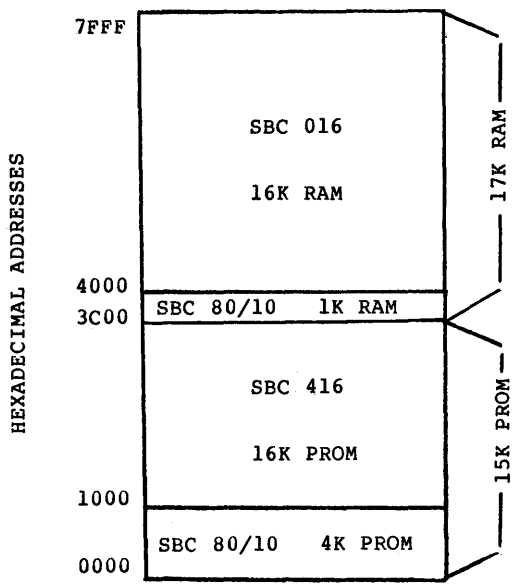


Figure 13-2 - TASI-E Microprocessor Memory Map

SYSTEM DEVELOPMENT AND TESTING

The use of the PL/M language for program development has already been discussed. This code was stored and compiled on the Control Data Corporation (CDC) Cyber 72 machine, the in-house Division 40 computer. During the course of program development it was necessary to expand the resident version of pass 2 of the compiler to increase the size of the symbol table to handle the 330 symbols used in the TASI-E program and also to increase the amount of memory available for assembled code from 12K to 15K bytes.

All initial testing of software algorithms was performed using the Intel simulator INTERP/80, which is also available on the CDC computer. It provided an approximation of the real processing time required, thereby allowing the comparison of timing requirements for alternate algorithms. Also, the use of the simulator permitted the correction of most software bugs without the necessity of numerous transfers of the program to the Intel MDS-ICE system for testing. Considerable handling time was thus eliminated because no direct link was available from the CDC to the MDS-ICE. The only means of transfer required the punching and reading of a large paper tape.

The software was checked with each hardware interface individually, using the MDS-ICE system to monitor performance with the program residing in the RAM of the MDS. For complete real-time system testing it was necessary, of course, to write the program into PROMs and run the entire system with all hardware interfaces and interrupts. The ICE umbilical cord was used to study performance and detect problems. With the individual software paths already tested, it was much easier to isolate problems in the integrated system. The software also proved to be very useful in debugging the hardware. Once a working version of the software exists it is more reliably reproducible than hardware components which are susceptible to numerous physical conditions.

CONCLUSION

The Intel SBC 80/10 computer board has been used successfully as the heart of the experimental TASI-E terminal. The use of a microprocessor controller has greatly reduced the amount of hardware required for the system.

Through careful program design the real-time requirements of the system have been met using a high-level programming language. Use of available development tools has made it possible to design, construct, and test the entire system software in less than 5 months from its inception and has facilitated any system modifications recommended as a result of performance evaluations.

REFERENCES

1. R. B. Robrock, TASI-E - A Domestic TASI for Analog Carrier, Memorandum for File, Case 38723-20, December 29, 1975.

2. V. I. Johannes, A Domestic TASI Proposal - TASI-E, Memorandum for File, Case 38723-20, February 20, 1976.

3. R. B. Robrock, An Experimental TASI-E Terminal - Overview, Memorandum for File, Case 38723-20, August 17, 1976.

4. W. M. Taliaffero, "Modularity. The Key to System Growth Potential," Software Practice and Experience, Vol 1, No. 3, July, 1971, pp 245-257.

APPLICATIONS I

A MICROCOMPUTER TEST FACILITY FOR A MOBILE TELECOMMUNICATIONS SYSTEM

S. A. Tartarone, BTL Dept 3141, HO, NJ

ABSTRACT

The design of a facility to exercise the call processing algorithms of the high-capacity mobile telecommunications system (HCMTS) mobile logic unit is presented. The system architecture, the use of a finite state machine model for the test-planning activity, and the resulting software design structure are explored.

INTRODUCTION

It is becoming quite apparent that the facilities required to test microcomputer-based designs thoroughly are as complex as the very systems that they are testing. The mobile logic system exerciser (MOLOSEX) sustains this basic maxim. It is a system designed to test a microcomputer-based logic unit which will be the master controller for the mobile telephone equipment to be used during the technical trial of HCMTS.[1, 2] The mobile logic unit,[3] designed around an Intel 8080 microprocessor, must execute a complex set of call processing sequences to set up, monitor, and release calls. As Figure 14-1 displays, it must communicate over a 10-kb/s data channel, respond to inputs and drive call status indicators associated with the user console, and control certain functions of the transceiver section. Testing of this design required several steps: (1) strip testing of small pieces of software, (2) string testing of conglomerate pieces of software, and (3) system testing of the integrated hardware/software design to determine if it was meeting its call processing requirements. The purpose of this paper is to report on this final stage of system testing and the design of the equipment required to undertake it.

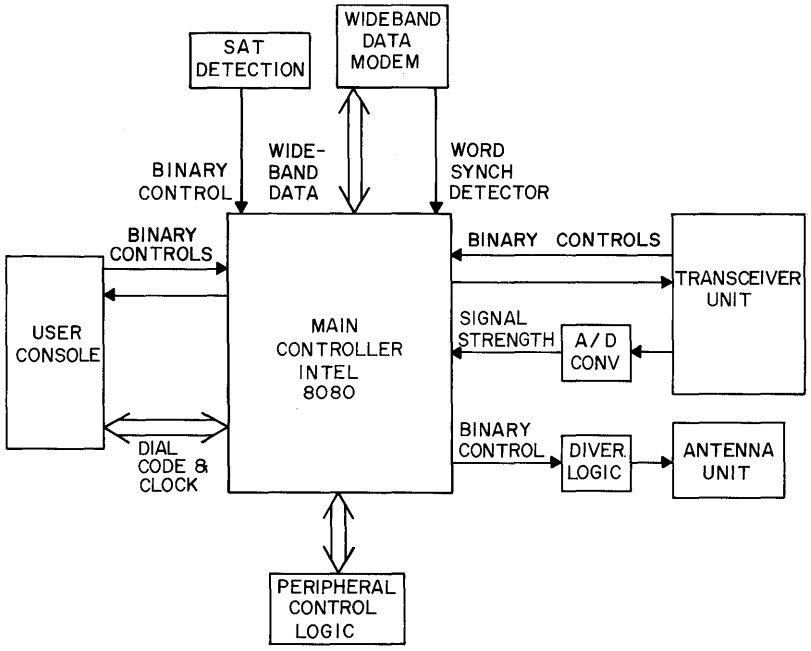


Figure 14-1 - Mobile Logic System

SYSTEM ARCHITECTURE

Figure 14-2 illustrates the architecture of the testing facility. The plan is to simulate the external environment of the main controller using a driver processor which will communicate with the logic unit through a hardware interface.

The requirements for the driver processor were that it be relatively inexpensive and portable, that it provide rather simple interfacing capability, and most important, that it have a short procurement lead time to meet the tight development schedules. The Intel INTELLEC MDS closely met these requirements and was selected as the driver processor.

The hardware interface was designed around the I/O port structure of the INTELLEC. Two universal synchronous/asynchronous receiver-transmitters (USARTs) were employed to provide the 10-kb/s data transmission-reception capabilities. A third USART was used to provide the dial code stream from the user console. Latches were used to establish the interfaces with the binary controls, and a digital-to-analog (D-A) converter was chosen to serve as the input to the signal strength measurement device.

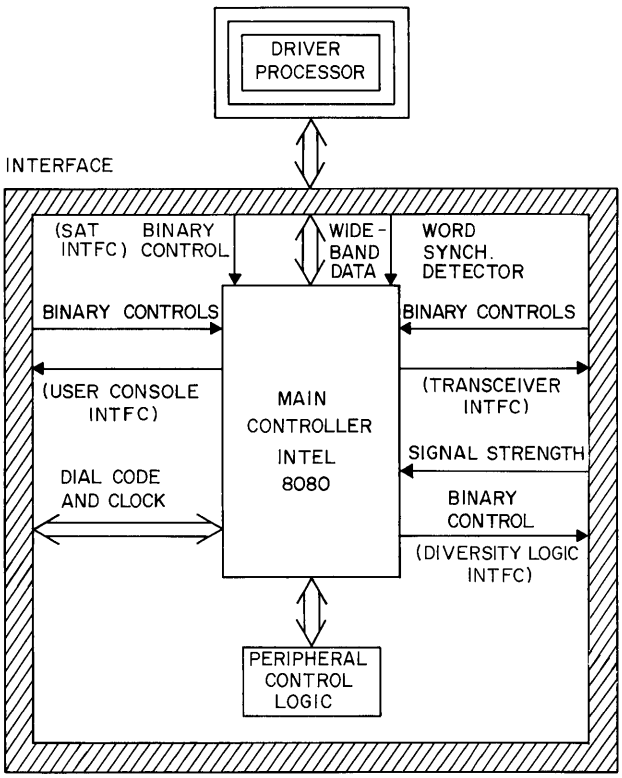


Figure 14-2 - Test Facility Design

TEST PLANNING

The test planning and subsequent software design are based on a finite state machine implementation of the logic unit call processing requirements. A simplified model is illustrated in Figure 14-3. It consists of a number of states which represent break periods during which outside stimuli designated as inputs may occur. An input will cause a response specified by a transition routine to be executed before transferring to the next state. The detailed call processing model was formulated using the invaluable finite state support software (FS^3), developed in Department 5331.[4] The resulting model consists of some 40 states, 30 inputs, and 60 transition routines linked by over 300 arcs which interconnect the various states.

The test plan was to attempt to cover the finite state machine model by a judicious selection of sequences which would walk through enough arcs of the model to validate the integrity of the design of the logic unit. Once the sequences were se-

lected, test vectors based upon the finite state machine definitions were carefully specified in detail in a high-level PL/I-like language.

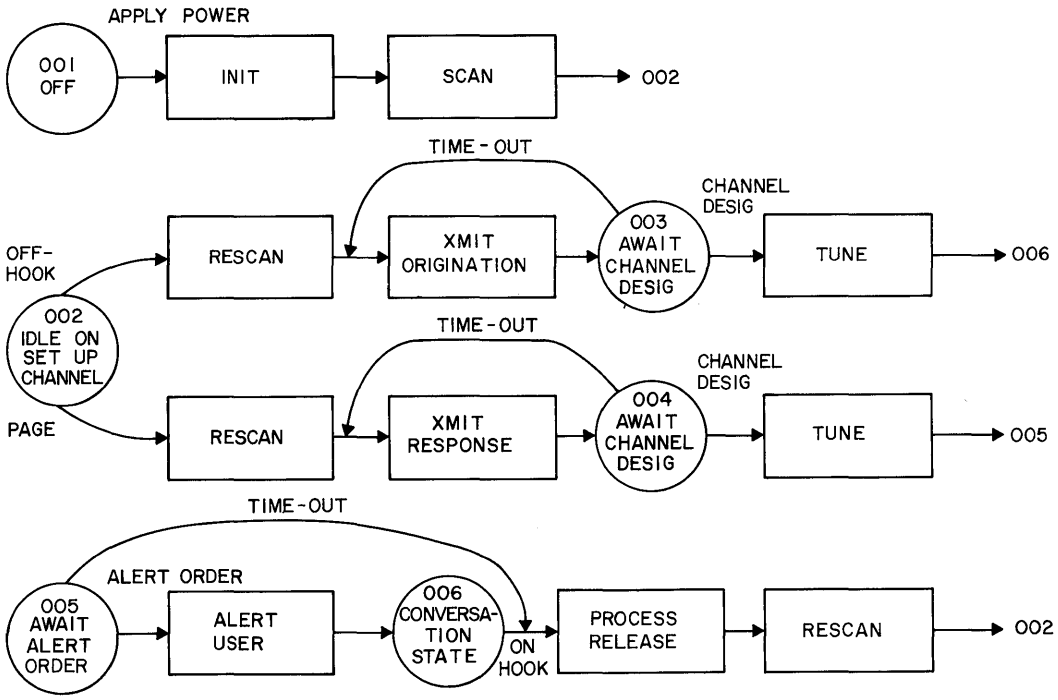


Figure 14-3 - Simplified State Diagram

SOFTWARE

Software Structure

The software structure is based upon the finite state machine concept with the inherent capability of automatically linking together arcs of the state table model to perform the required tests. Figure 14-4 illustrates the software structure.

- The operating system provides communication with the system console, monitors commands, handles test errors, provides data examination and parameter change facilities, and stores library functions for lower level routines.

- The test sequence generator (TSG) receives arc number inputs from the test operator to build a test sequence. This module is the master linker which pages arc controller programs into a load module and generates a master driver program.

- The test sequence driver (TSD), created by the TSG, receives control from the operating system to initiate a test and sequences through the selection of arcs by calling the appropriate arc controller programs.

- An arc controller is created for each arc and contains the necessary calls to the lower level stimuli generators and response detectors to process an arc.

- The interface drivers are called by the higher level routines to interact with the hardware interface and perform the required communication with the logic unit for transmission-reception of the 10-kb/s data stream and control of the binary leads.

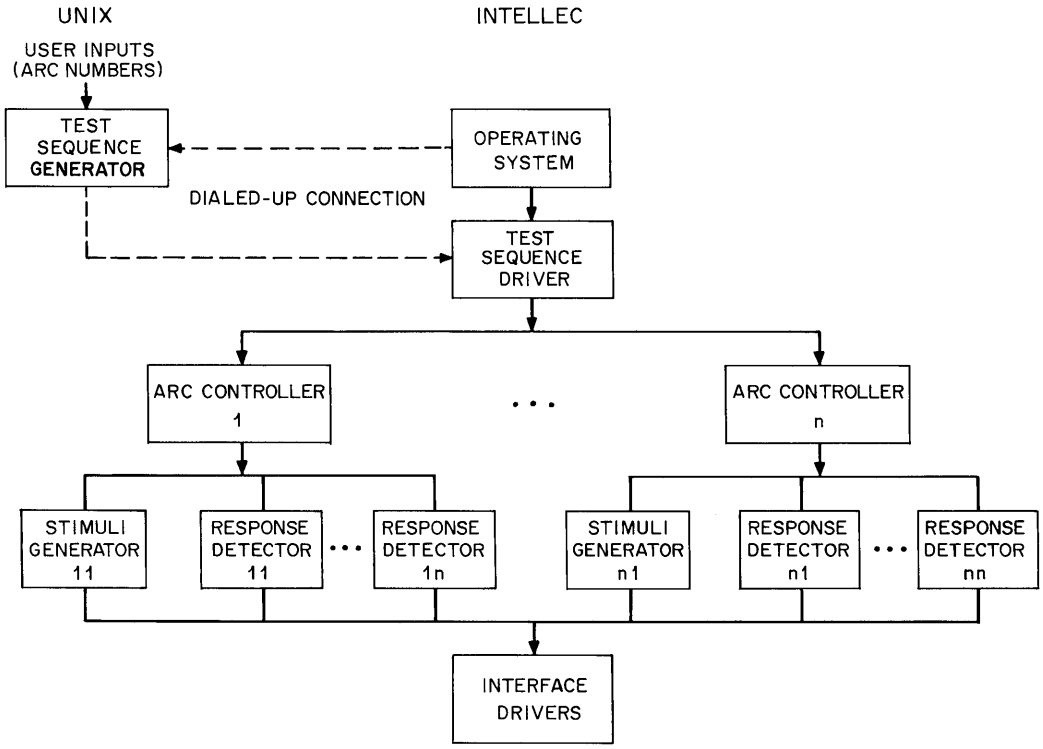


Figure 14-4 - Software Structure

System Interaction

To perform a test, a sequence of arcs, each designated by an arc number, is chosen. Certain arcs have variable parameters and these too must be preselected. A dial-up connection is then established with UNIX and the TSG program is executed. Inputs to this program consist of arc numbers and parameter values.

Up to 10 individual tests may be specified. The TSG builds up a complete load module, including the master driver program, for each test. The dial-up connection is then temporarily maintained while the MOLOSEX operating system contained in the INTELLEC is entered. Control is passed to a loading program, and the TSG load module is retrieved over the dialed-up link. Communication with UNIX is no longer needed. A test start command is entered to commence execution of a test and control is passed to the TSD. It calls the arc controllers for each arc. They, in turn, call the stimuli generators (SG) to provide the external inputs to the logic unit and the response detectors (RD) to determine if the logic unit has properly processed the stimulus. The SG and RD programs depend upon the interface drivers (ID) to establish the necessary communication with the logic unit.

Software Design and Coding

The design activity was centered around structured techniques using high-level languages and macros to ease the development of the application programs and mask the complicated interfacing with the hardware. Early in the development it became apparent that the high-level language capabilities (SMAL2), link-editing facilities, and powerful support facilities provided by UNIX[5] made it the optimal environment for program development. Furthermore, the other capabilities of UNIX, in particular the high-level C language, were instrumental in allowing certain functions of the test facility, such as the complex TSG, to be performed off-line on UNIX. The operating system was written in SMAL2 and provides several system macros and subroutines to be called by the lower level programs which interface with the hardware. The arc controllers, stimuli generator, and response detector programs were also written in SMAL2 with judicious use of macros to call the lower level interface drivers written in the Intel assembler.

CONCLUSIONS

The use of a unique testing model embodied in the finite state machine concept, coupled with structured programming techniques making use of high-level languages and sophisticated program development facilities, produced a powerful system testing facility for exercising the technical trial HCMTS logic unit. The facility was used to detect software errors and requirement misinterpretations. Extension of this type of equipment to testing of successive generations of mobile telephone equipment is expected with this facility as a prototype. In addition, the general concepts illustrated here could be extended to test hardware other than the HCMTS mobile logic unit.

ACKNOWLEDGMENT

Joseph E. Lencoski was most instrumental in the advancement of this system by contributing significantly to its hardware and software design. Steven M. Silverstein and David L. Sobin provided the author with many useful suggestions which are reflected in the system design. David H. Copp and John J. Molinelli provided most valuable assistance in the development of powerful Intel software development tools and, in particular, in their response to the specific needs of this project.

REFERENCES

1. R. H. Frenkiel, "A High-Capacity Mobile Radio Telephone System Model Using a Coordinated Small Zone Approach," IEEE Transactions on Vehicular Technology, May, 1970, pp 173-177.

2. Z. C. Fluhr and E. Nussbaum, "Switching Plan for a Cellular Mobile Telephone System," IEEE Transactions on Communications November, 1973, pp 1281-1286.

3. M. R. Karim, "A Controller for the Mobile Logic Unit in the Bell Laboratories High-Capacity Mobile Telecommunications System," Proceedings of the 26th Annual Conference of the IEEE Vehicular Technology Group, March 24-26, 1976, Cat. No. 76CH1056-1VT.

4. H. Y. Chang et al, FS^3 (Finite State Support Software) System Overview, Bell Telephone Laboratories unpublished work.

5. D. J. Hunsberger and J. J. Molinelli, "A Support System for the Intel 8080," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Incorporated, Holmdel, New Jersey, 1976.

APPLICATIONS I

THE DESIGN OF A SELF-CHECKING MICROCOMPUTER

M. Liron, BTL Dept 5317, IH, IL

ABSTRACT

The high availability requirements dictated by the No. 1 ESS environment make it necessary for a continuously on-line microcomputer to be self-checking. A microcomputer architecture which is capable of providing immediate fault detection and high-fault location resolution is presented. Because of the continuous reduction of hardware costs, selective hardware duplication becomes increasingly more attractive. The method employed is based on duplicating a subset of the microcomputer, called the hard core, which is used to bootstrap diagnostics from the hard core to the nonself-checking hardware. In order to meet the availability requirements, a spare microcomputer is used which is switched on-line upon detection of faults in the active on-line microcomputer.

INTRODUCTION

A microcomputer system (μc) is currently being designed for use in the No. 1 ESS periphery. The incorporation of an intelligent stored program controller as an integral part of a peripheral frame marks a deviation from the traditional centralized control approach to a more distributed control approach. The first peripheral system to utilize a μc for its control will be the digital carrier trunk (DCT) with other μc-based peripheral frames expected to follow.

The wired-logic controllers currently in use are sufficiently simple to be fully maintained by the ESS central control (CC). The μc on the other hand, being considerably more complex, cannot be maintained by the CC in real time, and must therefore be capable of autonomously carrying out the various maintenance

functions. These maintenance functions can be classified as: fault detection, fault location, reconfiguration, recovery, and in the case of a duplicated and matched μc, the capability to initialize a repaired μc before the transition from a simplex to a duplex mode (i.e., the update process).

The μc, a continuously on-line unit, must be fully operational for the entire 40-year lifetime expected of the ESS machine. Even in a μc of moderate complexity (on the order of 150 dual in-line packages [DIPs]), hardware faults are likely to occur long before the mission time is completed (probability of 0.99). In order to meet the continuous availability requirement in the presence of hardware faults, a spare μc is used, thus increasing reliability (i.e., the probability that at least one μc will be operational) to above 0.999.

This paper presents the duplex μc architecture currently being implemented for the DCT system that is designed to meet the maintenance and availability requirements necessary in the ESS environment.

OVERALL ARCHITECTURE

The duplex μc system integrated into a No. 1 ESS peripheral frame is depicted in Figures 15-1 and 15-2. CC of the input/output (I/O) units is indirect, via the active on-line μc. The μc receives and translates the relatively high-level CC orders into the primitive control signals necessary to control the operation of the wired-logic I/O units. The CC orders are received over the peripheral unit bus (PUB), while data to the CC is transferred to a memory unit scanned by the CC via the scanner answer bus (SCAB). In the DCT system the I/O units represent 20 T-carrier units, called digroups, used to encode and to multiplex 480 analog channels for transmission over 20 T-1 digital trunks. The digroups also perform the receiving functions of demultiplexing and decoding the digital trunks into analog channels.

Hardware techniques make each μc partially self-checking; hardware techniques augmented by internal diagnostic routines make each μc fully self-checking. Although one μc is used to control the peripheral system (the active unit), both active and spare μcs run in a fully synchronized and matched mode simultaneously executing the identical instruction sequence. Those faults not detected internally by the μcs will be recognized (within microseconds) by a mismatch between the two μcs. The internal diagnostic routines resident in each μc are triggered by

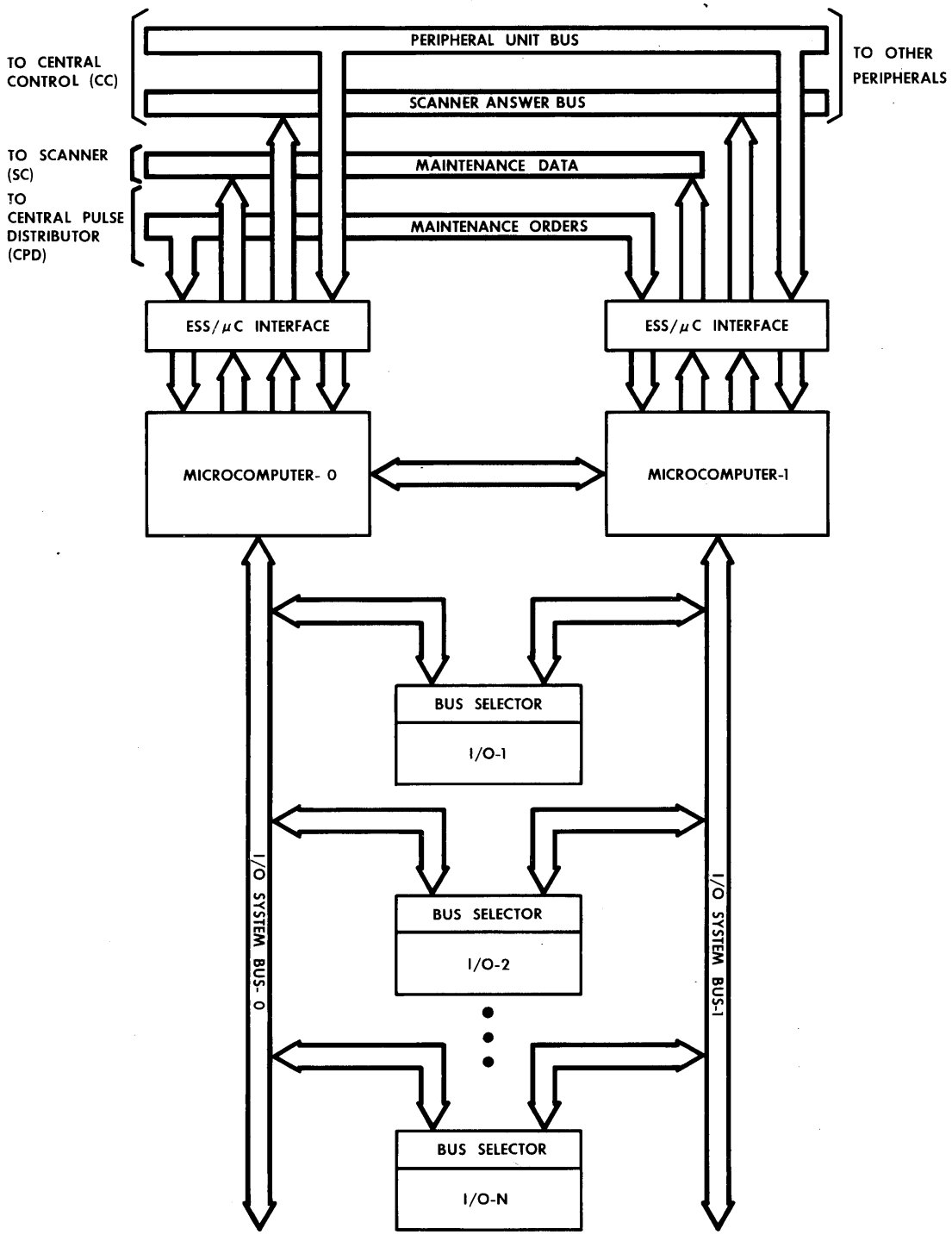


Figure 15-1 - Peripheral Frame Block Diagrams

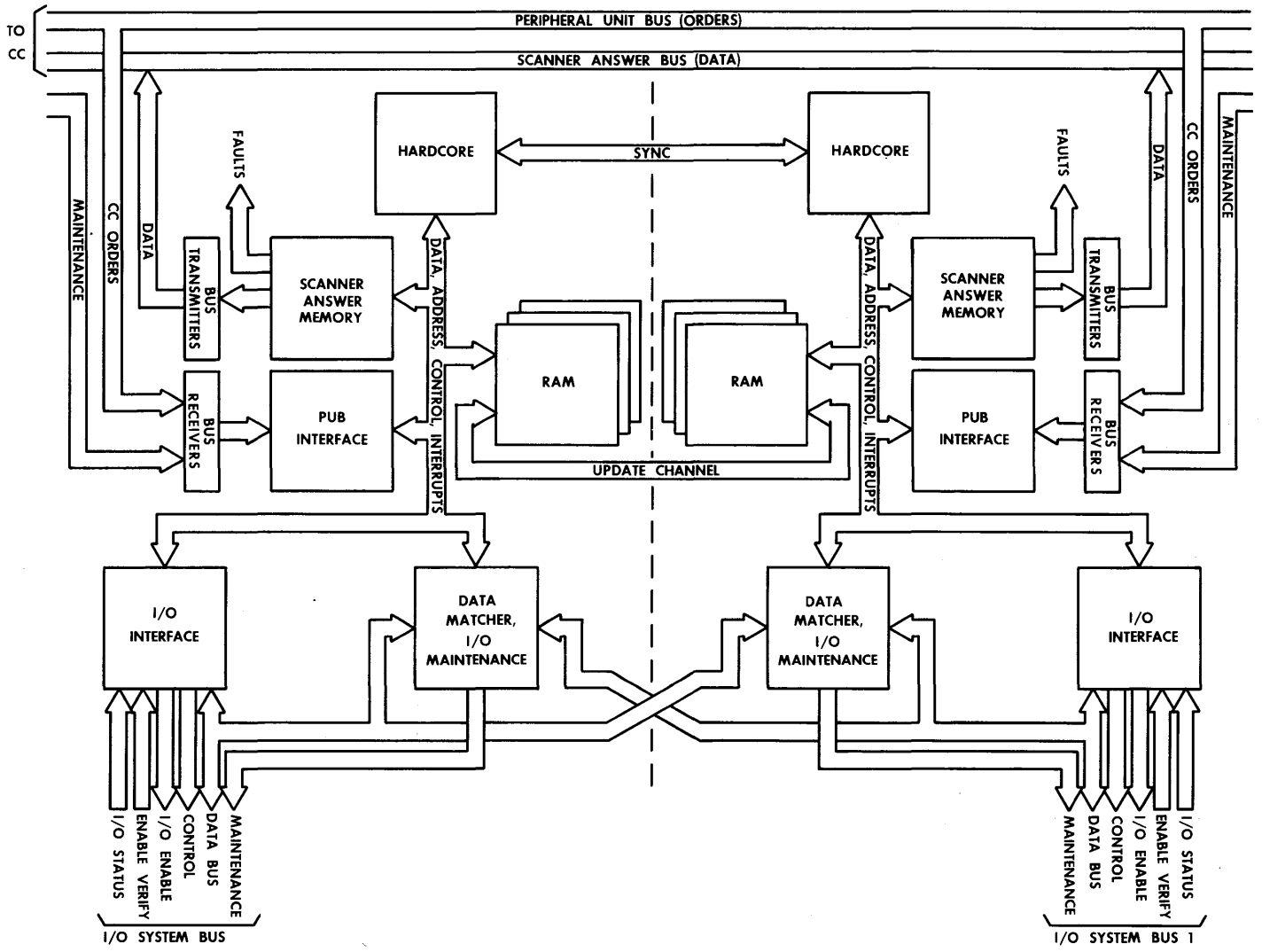


Figure 15-2 - Duplicated Microcomputer Block Diagram

the mismatch and are responsible for isolating the fault to the specific failing μc. It should be pointed out that, although it is theoretically possible to implement a fully self-checking computer entirely with hardware techniques, it is not economical for systems of this type which utilize standard off-the-shelf large scale integration (LSI) building blocks.

In order to keep the spare μc fully synchronized with the active unit, all input information is received and simultaneously acted upon by both μcs. (The input information consists of orders from the CC and data from the I/O units.) Output information, though generated by both μcs (data to the CC and orders to the I/O units), is selected from the active μc only. The fully synchronized mode of operation, in addition to enabling a continuous matching by the hardware, also forces the spare μc to be completely updated as to the status of the I/O units. Hence, in the event of an active μc failure, the reconfiguration to a simplex mode (i.e., spare switched to active) is accomplished within milliseconds, thus enabling rapid recovery.

Faults internally detected by the μc reveal which of the two μcs is to be switched off-line, and in many cases, the specific faulty circuit pack (i.e., fault location). However, isolation of a failing μc detected by a mismatch and the fault location process are implemented through software self-diagnosis.

Since the correct execution of the diagnostic routines depends on the correct operation of the μc hardware, the internal hard core (HC), a fully self-checking subset of the μc, is used to bootstrap the diagnostic routines from the HC to the nonself-checking hardware. The interrelationships among the various maintenance functions are depicted in Figure 15-3.

THE HARD CORE

Theoretically, only a minimal set of functions, those sufficient in bootstrapping the diagnostics from the HC to the rest of the system, need be assigned to the hard core. In practice, however, it is advantageous to implement an additional set of functions (e.g., self-checking which, therefore, can be considered as part of the HC) for the sake of fewer and simpler diagnostic programs. The continuous reduction of hardware cost and the increasing functional density of LSI components (hence the difficulty of their diagnosis) make hardware duplication

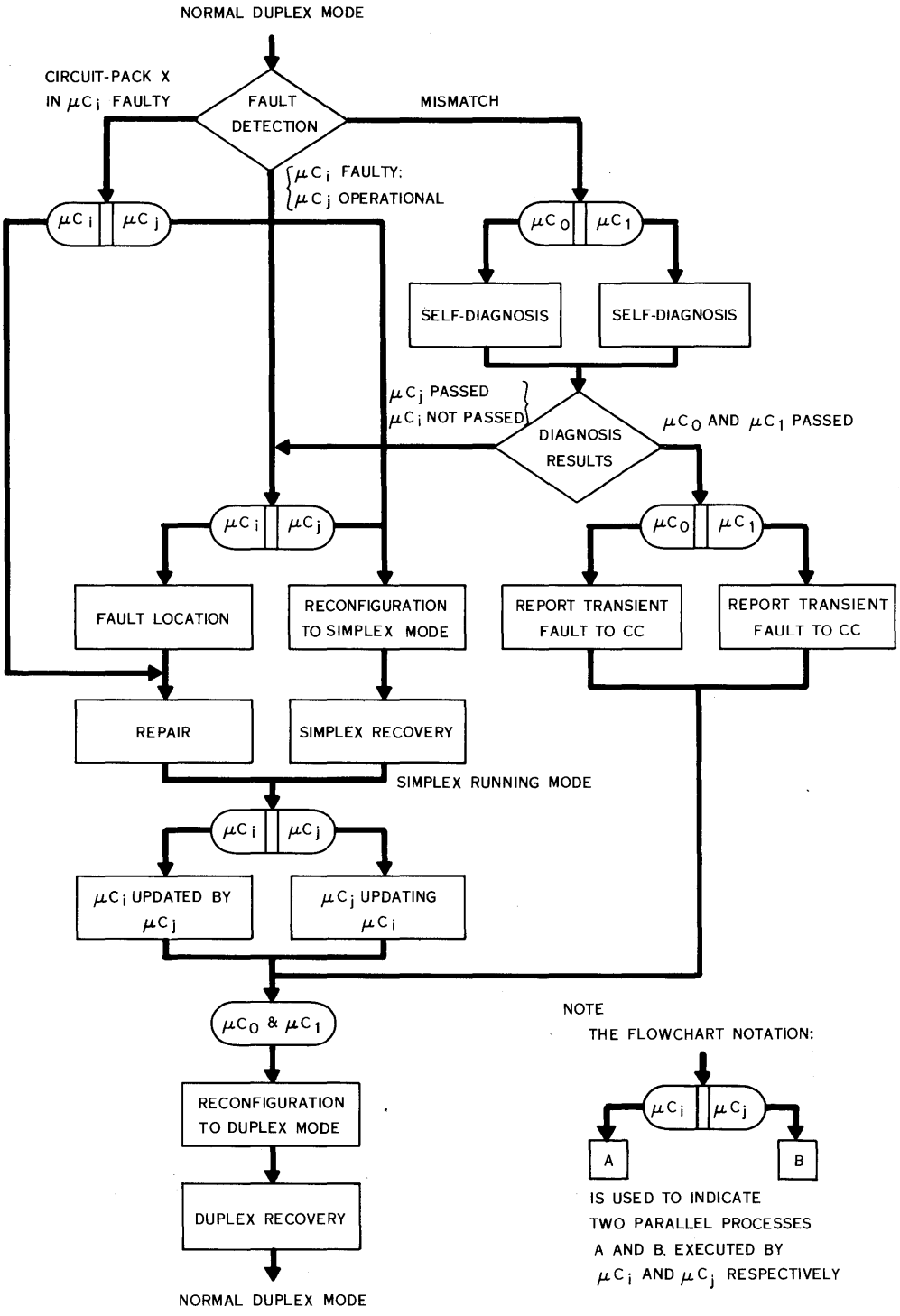


Figure 15-3 - Maintenance Functions

increasingly more attractive. The criteria adopted in assigning functions to the HC, in addition to those needed for diagnostics bootstrapping, is based on the tradeoffs between the diagnosability of a function and the cost of duplicating the function. The term diagnosability is used here rather loosely and is meant to serve as a measure of the effort required to design the diagnostic program, its memory space needs, the fault location resolution it can achieve, the execution time, additional hardware needed to make the function diagnostic, etc.

The functions implemented in the HC are shown in Figure 15-4. Self-checking is achieved through duplication and matching. The lab model system utilizes the Intel 8080 microprocessor (μp); the final production version will utilize the MAC-8. Because of an insufficient number of internal μp registers (none in the MAC-8), a local 256 by 8 random-access memory (RAM) is added to each μp. This local RAM space is used for the general purpose registers and temporary storage locations needed in the diagnostic process. The 8K bytes of read-only memory (ROM) are used to store the application and diagnostic programs; an additional 8K bytes of address space are reserved for future ROM growth.

The real-time clocks are used for task scheduling and are initialized by the μps (under program control) to generate interrupts at requested points in time. Sixteen priority-interrupt levels are used for normal program execution and for fault identification. The assignment of different interrupt levels to different fault indications eliminates polling, thereby speeding up and simplifying the fault location process (e.g., an interrupt at the level assigned to RAM parity faults automatically reveals the faulty circuit pack). All modules transmitting data to the HC generate a parity bit, along with the data, which is checked at the HC during a read cycle. The parity check detects faults in non-HC modules as well as in the μc internal data bus shared by all modules. Internal HC buses are buffered from the μc internal bus to avoid μc bus faults from propagating into the HC.

HC - CLOCK CIRCUITS

As stated previously, matching is performed on two levels: between μc0 and μc1, and internally in the HC of each μc. To enable a continuous matching, all four μps (two in each HC) are driven by the same clock. However, a clock fault will not cause both μcs to fail. Figure 15-5 depicts the clocking circuitry which is distributed over each of the HCs.

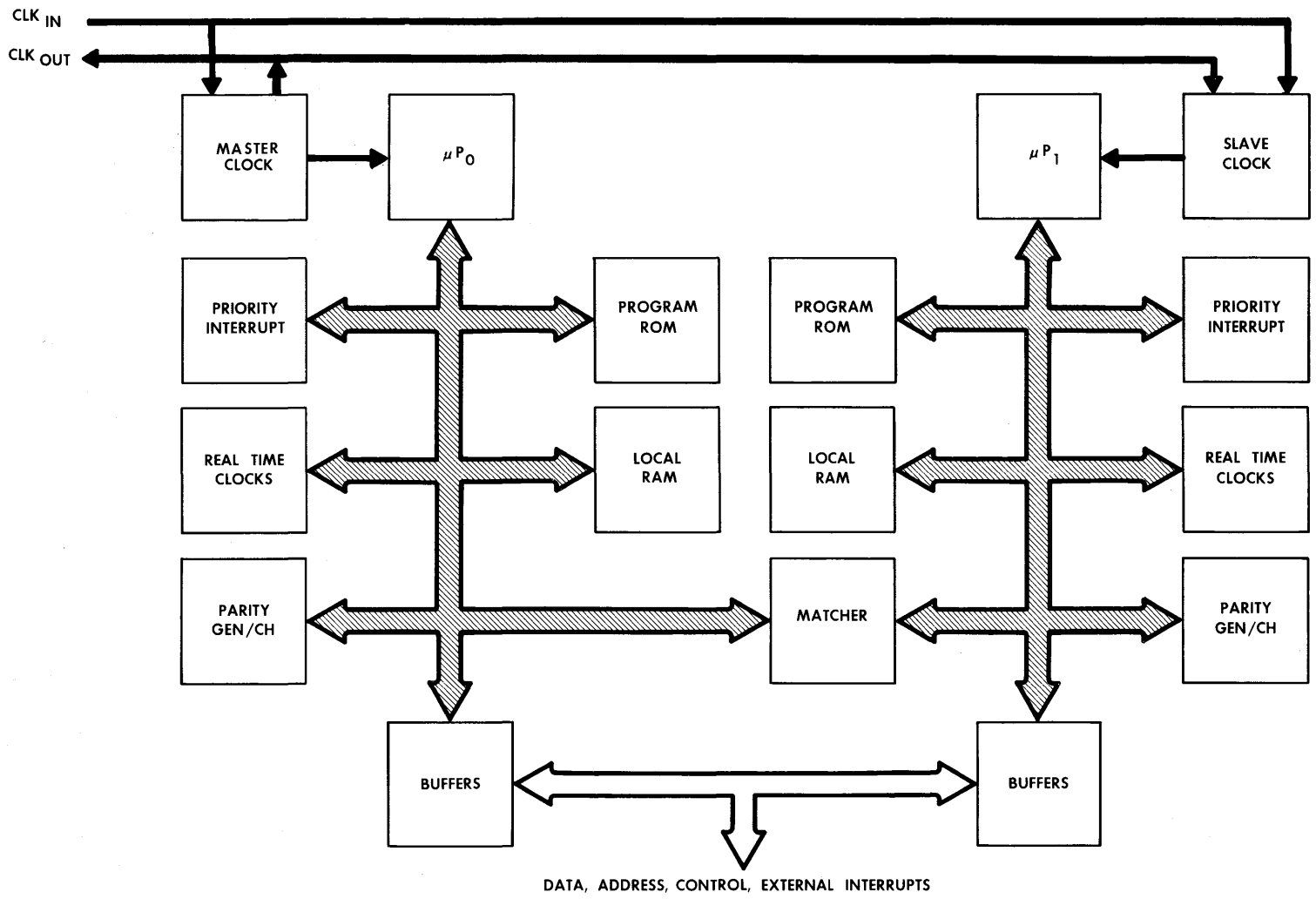


Figure 15-4 - Hard Core Block Diagram

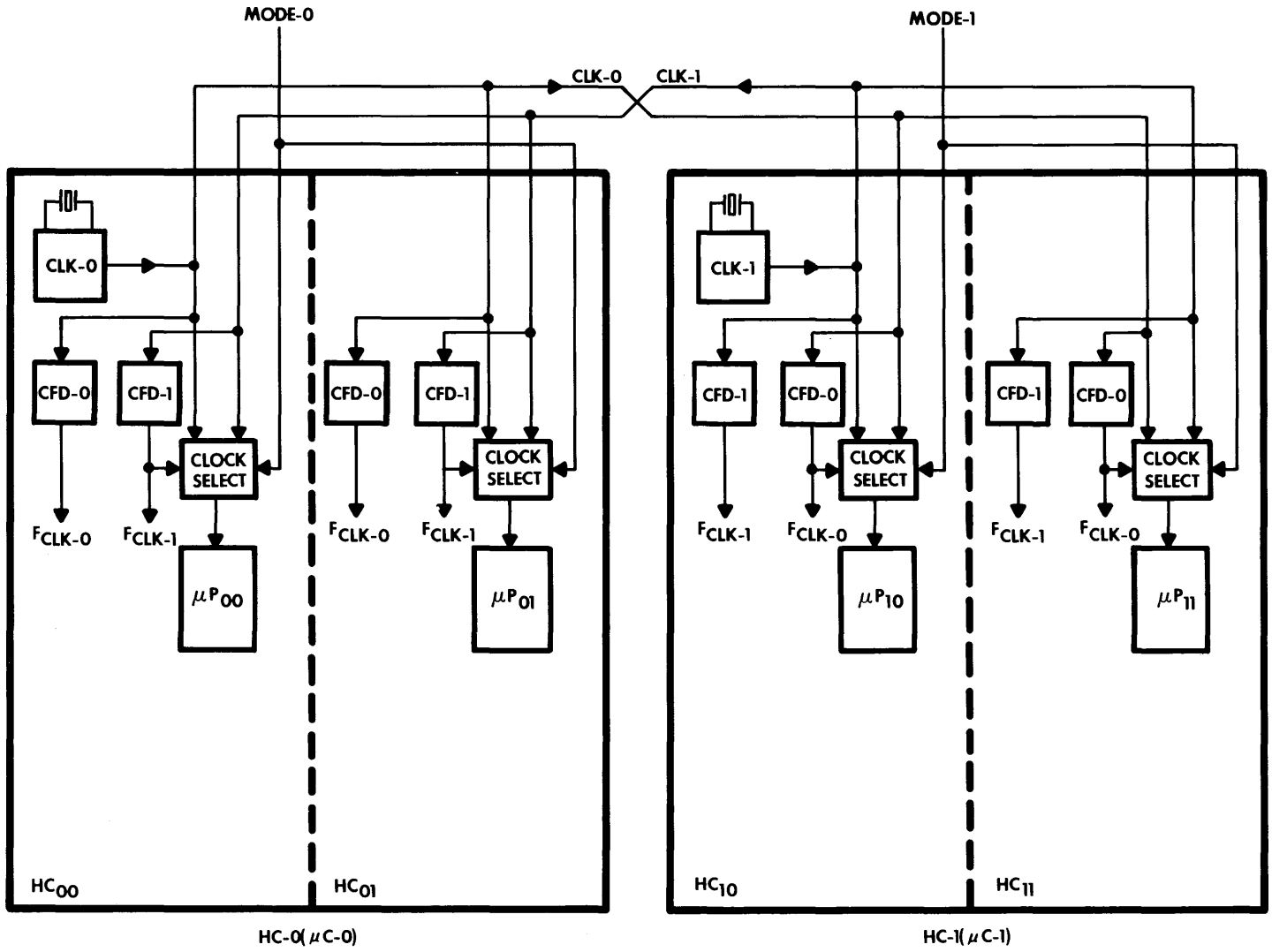


Figure 15-5 - Hard Core Clocks

The MODE is an input controlled by the CC declaring the μc status as active or spare. The clock fault detectors, CFD-0 and CFD-1, are retriggerable monostable multivibrators used to detect faults in CLK-0 and CLK-1, respectively. The clock, which has a cycle period T1, triggers a monostable adjusted to generate a pulse of duration $T2 = T1 + \Delta t$. Since $T2 > T1$, the output F of the monostable will be at 1 for as long as the clock is functional. Upon a clock fault, the monostable will not be retriggered, resulting in $F = 0$. Typical waveforms of a clock stuck-at-1 and of the F signal are shown in Figure 15-6.

A clock selector (one associated with each μp) selects either CLK-0 or CLK-1 to drive its respective μc. The clock selection is controlled by the MODE, F_{CLK-0} and F_{CLK-1} signals, in accordance with the following logical statements:

in HC_{00} or HC_{01}:

If $[\overline{\text{MODE-0}} \wedge F_{CLK-1} \wedge F_{CLK-0}]$, then CLK-1 else CLK-0

and in HC_{10} or HC_{11}:

If $[\overline{\text{MODE-1}} \wedge F_{CLK-1} \wedge F_{CLK-0}]$, then CLK-0 else CLK-1.

where:

MODE-i = 1 means μc-i is active. $F_{CLK-i} = 1$

means no faults detected in CLK-i.

Under normal conditions, with no clock faults, MODE-0 = $\overline{\text{MODE-1}}$ and the active μc clock drives all four μps. If any clock fault is detected, the selectors in HC-i will switch to their own clock CLK-i, regardless of the MODE input.

All inputs receiving the same clock signal (selectors and CFDs) are isolated from each other with a high-input impedance buffer gate in series with a resistor. This buffering avoids stuck-at-0 or stuck-at-1 inputs from affecting all other inputs (see Figure 15-7).

A failing CFD, clock selector, or input buffer can affect only one μp. Faults of this type, however, will induce a mismatch within the HC containing the failing component, and will lead to an immediate detection and location of the failing HC. The same buffering technique is used in other cases where an input stuck-at-1 or stuck-at-0 fault would lead to faults not resolvable to a specific circuit pack.

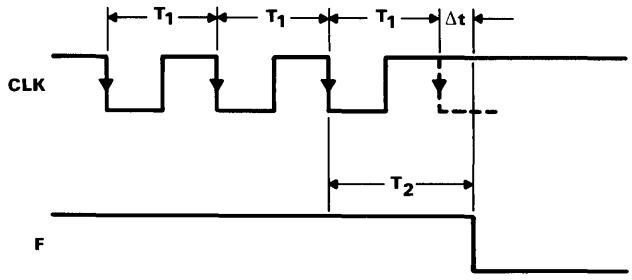


Figure 15-6 - Clock Fault Detection

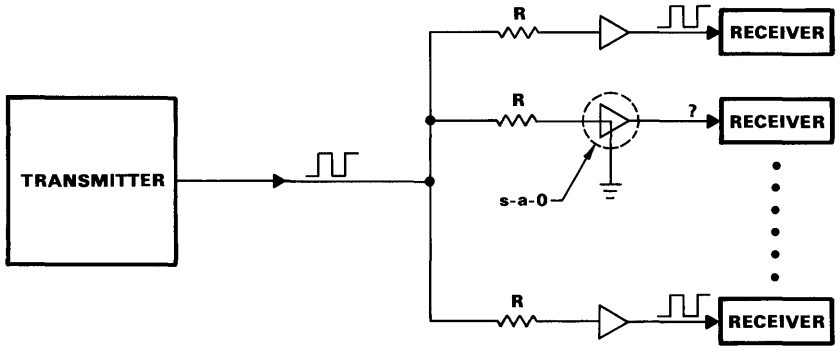


Figure 15-7 - Isolation of Input Faults

THE RAM MODULE

The RAM module is organized in blocks of 4K 9-bit words, with each word containing eight data bits and one parity-check bit. RAM blocks are bit sliced using 4K by 1 static RAM DIPs; consequently, the parity check is highly effective in detecting most typical RAM faults (i.e., only one bit of a word is changed by a failing RAM DIP therefore, the fault is immediately detected by the parity check). Since the control circuitry is byte oriented, faults not detectable by the parity check may occur (e.g., an even number of bits failing in a data buffer). These faults however are easily recognized by the diagnostic program.

A general purpose bidirectional I/O port is provided on each RAM block. The I/O port on the first RAM block, the update channel, is the only one used in DCT. It provides a communication link for data transfers between the μcs (Figure 15-2) to update the volatile storage locations of a repaired μc reinstalled into service prior to the reconfiguration from the simplex to the duplex mode (Figure 15-3). The I/O port serves as a mail box into which the on-line μc inserts the data, and from which the repaired μc retrieves the data. This update approach provides a loose coupling between the μcs (as opposed to direct-memory access [DMA]), and prevents fault modes which could cause both μcs to fail simultaneously, (i.e., a functional μc can simply ignore a faulty μc attempting to access it via the update channel).

MICROCOMPUTER INTERFACES

The Peripheral Unit Bus Interface

The peripheral unit bus interface (PUBI) is the module that receives the 24-bit CC orders over the PUB via a set of bus receivers. Reception of an order is acknowledged by returning a verification signal to the CC. Since orders could arrive at a rate higher than the μc execution rate, a first-in/first-out (FIFO) circuit is used to store the orders, forming an execution queue for the μcs. A validity check by the μcs precedes the execution phase of the orders read from the FIFO, which checks the parity and the legality of the received order. Legality is checked by verifying that the order received does correspond to one of the orders existing in the CC I/O instruction repertoire. If the check passes, an all-seems-well signal is reported to the CC, and the execution phase starts. A failure to return the all-seems-well signal within 25 ms of receiving the order triggers a CC resident diagnostic program that checks the PUBI and orders the μc to perform self-diagnosis.

Scanner Answer Memory

The scanner answer memory module (SCAM) is a 128 by 16 dual-port memory used as a buffer for returning data to the CC. The μc writes data into the memory in 8-bit words, while the CC autonomously retrieves the data in 16-bit words. The CC and μc access priority to the memory is on a "first-come/first-served" basis. The μc reports to the CC, via the memory, the current state of each I/O

unit, the execution state of the CC orders, and the maintenance-related data such as diagnostic results. In addition, this module contains drivers which set the ESS scan points (i.e., magnetic ferrods) to report severe fault conditions (e.g., HC fault). These scan points are automatically set upon detection of such faults and are not under program control.

I/O Unit Interface

The I/O interface module (Figure 15-2), interfaces the I/O units to the μc via an 8-bit bidirectional common data bus. The I/O module also interfaces the various control signals needed to address and access the I/O units. A handshaking procedure is used in all I/O instruction transfers to the I/O units to protect the integrity of communication in the noisy environment. When an order from the μc is loaded into an I/O unit input buffer, the addressed unit returns an order received status bit. Each I/O unit has its own data-ready status bit which signals the μc that data in the I/O unit output buffer is to be unloaded. The data, along with a parity bit, is simultaneously returned to both μcs on their respective I/O system data bus, and is matched for consistency. I/O unit addressing is implemented with a separate enable line to each unit. The I/O address generated by the μcs is decoded in the I/O interface module by a 1-out-of-N decoder whose outputs are reencoded and checked for consistency with the μc-generated address.

DIAGNOSTICS

The diagnostic program isolates faults detected by a mismatch to one of the two μcs, and locates the faulty circuit pack within the failing μc. The macro-test is used for rapid fault isolation, while the micro-test is used in the fault location process.

In the macro-test, large portions of the hardware are tested for the presence or the absence of faults. It has the logical structure of a maze with one legal exit, and returns a pass/no pass-type result. The maximum testing period is on the order of 5 ms, but testing is terminated if an intermediate result indicates a fault. If both μcs pass the macro-test, the fault causing the mismatch is attributed to noise and the event is reported to the CC. However, if the number of unresolved faults per unit time exceeds a predetermined threshold, the detailed micro-test (normally used for fault location) is executed.

The micro-test utilizes a bootstrap approach, where the first test starts with the smallest amount of circuitry possible. Each additional test adds a small increment to the circuitry tested. When a given test fails, the assumption is that the failing circuit is within the group of circuits added by that test. Figure 15-8 illustrates the general test flow starting at the self-checking HC (for which diagnostics are not needed), and gradually bootstrapping the tests to all modules accessible by the μc. Tests are designed to check the actual hardware and the functions provided by the hardware. Fault detection hardware is checked by purposely injecting faults (e.g., forcing a mismatch in a matching circuit) and verifying that the simulated faults are actually detected.

Bus bits stuck-at-zero could be a result of stuck-at-zero faults in the bus interface hardware of any circuit pack sharing the common bus. These faults, though rapidly detected by the parity check or the macro-test, cannot be isolated to the specific faulty circuit pack. The method usually employed to locate such faults is manual removal of the circuit packs from the bus one at a time until the fault is removed. Essentially, the same approach is used in the micro-test but at electronic speed. The equivalent of removing a circuit pack is achieved by breaking the ground lead of the bus interface circuit of each circuit pack with a transistor-transistor logic (TTL) compatible DIP relay. These relays are located on the circuit packs and are controlled by the HC during the diagnostics. This approach is used to isolate faults in the I/O system bus and the internal μc buses.

SUMMARY

A microcomputer architecture has been presented which is capable of immediate fault detection through hardware techniques, and high-fault location resolution through software self-diagnosis. The architecture is implemented with standard building blocks and is sufficiently general to be applicable to any microcomputer system requiring high availability.

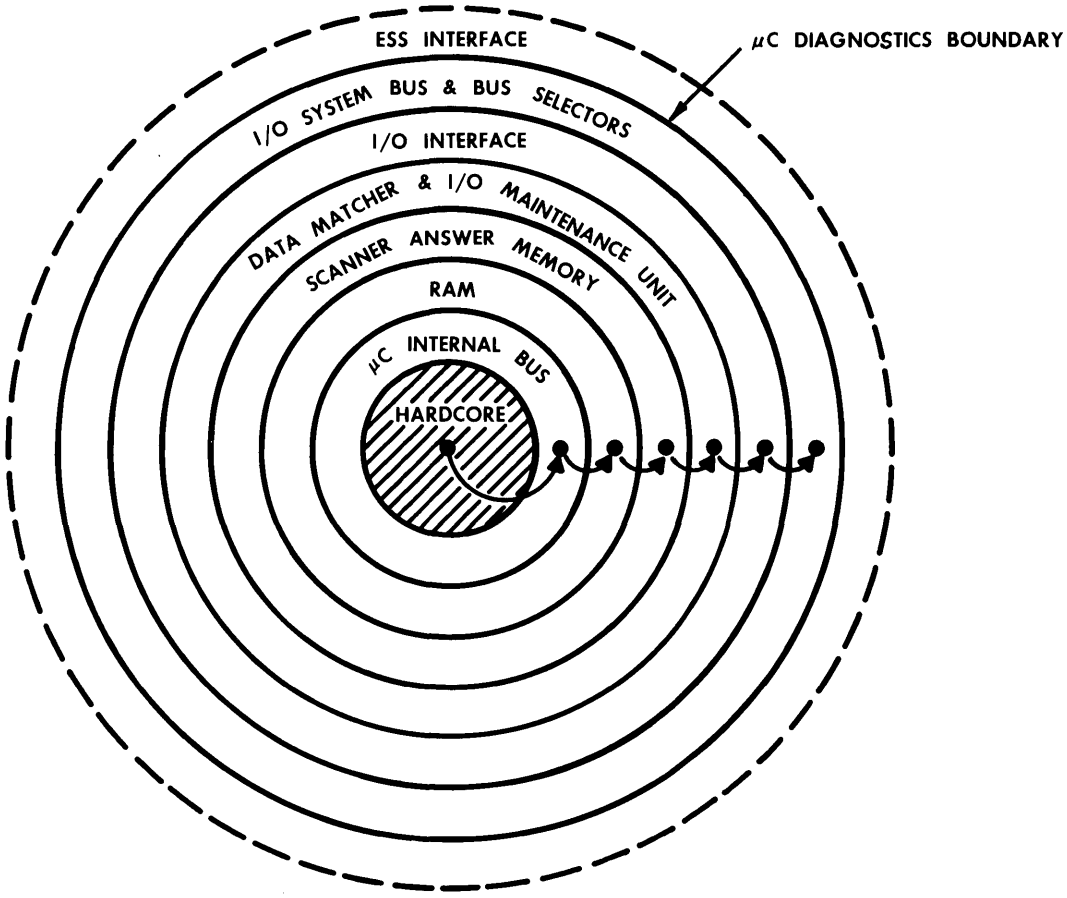


Figure 15-8 - μC Bootstrap Diagnostics

APPLICATIONS I

DESIGN OF AN LSI MICROPROCESSOR

J. C. Moran, BTL Dept 3222, DR, CO

ABSTRACT

The design of a processor which emulates the No. 3 language is described. The design is based on Motorola's M10800 family of emitter coupled logic (ECL), large scale integration (LSI) circuits, which is a microprogrammable processor family of 4-bit slice circuits. The principal design goals were the execution of the No. 3 language and high throughput. The throughput required was a minimum of twice that of the No. 3. A description is given of the basic system architecture and the more significant design decisions. In addition, the M10800 family is evaluated for suitability in emulating the No. 3 language.

INTRODUCTION

The DIMENSION family of stored program private branch exchanges (PBXs) consists of the DIMENSION 100, 400, 2000, and Custom (4000 lines), the number designating the approximate maximum number of lines controlled by each system. All four systems execute the No. 3A language; however, the DIMENSION 100 and 400 use the MC-3 processor, and the DIMENSION 2000 and Custom use the 201CC processor. A fifth member of the DIMENSION family, capable of handling up to 10,000 lines, is also presently being considered. The unit for this proposed 10,000-line system is designated the 201VL processor.

The basic goal of the 201VL is to achieve a minimum of twice the throughput of the 3A processor. Two conditions must be realized to reach this goal. First, the bandwidth of the memory system which stores the program must be improved. This improvement is accomplished by using a pseudocache memory. Second, the

microinstruction execution time must be less than half that of the 201CC (225 ns). The reduced time is attained by using Motorola's MECL M10800 family of LSI circuits.

GENERAL DESCRIPTION

A block diagram of the 201VL processor, memory system, and input/output (I/O) structure is shown in Figure 16-1. The system architecture utilizes the fact that the parts of the program and data which affect real time can be placed in the pseudocache memory, which has a cycle time of 200 ns. The main memory has a cycle time of 600 ns. Although a true cache memory is dynamically loaded, the program and data in the pseudocache memory (capacity: 16, 384 words) are permanently assigned.

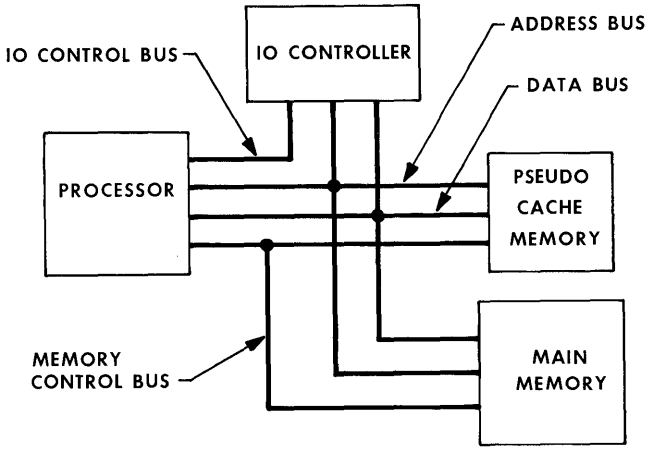


Figure 16-1 - System Structure

M10800 FAMILY

The M10800 is a 4-bit slice family, consisting of four basic parts: (1) arithmetic logic unit (ALU) (MC10800), (2) microprogram control function (MC10801), (3) timing function (MC10802), and (4) memory interface function (MC10803).

The MC10800 (see Figure 16-2) is capable of performing logic operations, binary arithmetic, and binary coded decimal (BCD) arithmetic on combinations of one, two, or three variables. The A, O, and I buses are the input buses for this cir-

cuit. The A bus is unilateral; however, the O and I buses are bilateral. The MC10800 can be disconnected internally from either the O or I bus. The latch and accumulator provide on-chip storage. Much of the power of the MC10800 is provided by the latch, A-input MUX, mask MUX, Y-input MUX, and complementor sections. They provide a great deal of flexibility in the operation of the MC10800 with regard to both internally stored variables and those coming in on the A and O buses. For example, note the number of paths from the accumulator into the basic adder section.

The MC10800 provides the following arithmetic and status outputs: group propagate, group generate, carryout sign, zero detect, parity of carries, parity of results, and overflow. The two parity outputs permit parity checking of ALU operations. The outputs also can be used for parity generation of the result.
The MC10801 is shown in Figure 16-3. CR0 holds the address of the word being fetched from the microstore. The output of CR0 can be disabled by the CS5 input. CR1 is best used for repeating microinstructions or for calling subroutines. The section labeled INC is a fast incrementor for CR0. CR2 is normally used as the macroinstruction register, and CR3 is a status register which can be selectively loaded from the DIN line. The output of the CR3 register is available in parallel on the CR3 bus. CR4, 5, 6, and 7 form a last-in first-out (LIFO) push-pop stack, used for subroutine calls. The IB, OB, CR0, CR3, and NA inputs are 4-bit buses. The remaining input lines are one bit.

Inputs IC0, 1, 2, and 3 determine which microinstruction will be executed by controlling the next address logic section, which selects the address to be jammed into CR0. These microinstructions include increment, branch, jump, and subroutine call. In addition, microinstructions are provided to decode the macroinstruction.

The features of the MC10802 are a programmable number of phases (timing pulses), selectable double-width duration of a particular phase, output enable, master reset, cycle-complete output, single-phase stepping, and single-cycle stepping. The MC10802 is also cascadable.

A programmable number of phases allows a microinstruction to determine its own timing and also allows multiple cascaded 10802s to give numbers of phases which are not multiples of four. The ability to select a double width for a particular

phase is very useful when there is a worst case timing path in the system. By using this capability, the system need only be slowed when the worst case path is executed. The initial prototyping and maintenance operations are aided by the single-phase and single-cycle stepping capabilities.

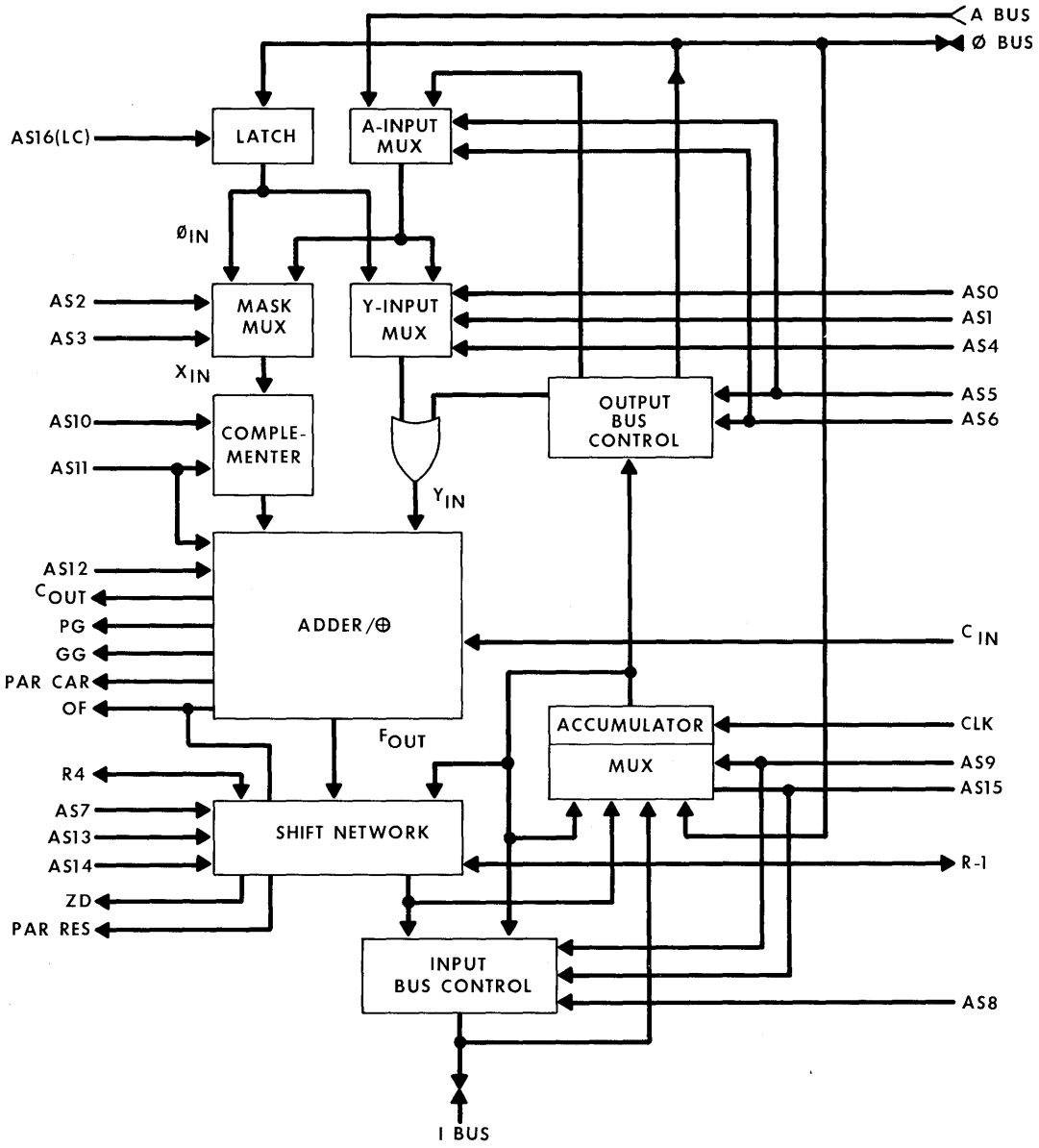


Figure 16-2 - Arithmetic Logic Unit

The MC10803 (see Figure 16-4), the most powerful part in the family, contains the logic for both data-routing and memory-addressing operations. All data paths and data storage elements are four bits wide. Data information and address information are transmitted to the memory system or I/O system via the data and address buses, respectively. The I and O buses transmit and receive data from the rest of the processor. The pointer inputs are used for mask and arithmetic functions by the ALU. The MC10803 is comprised of two main functional sections, the data matrix and the ALU, capable of independent operation. Note that the I bus, the O bus, and the output of the register file (RF) connect directly with the ALU and do not go through the data matrix section.

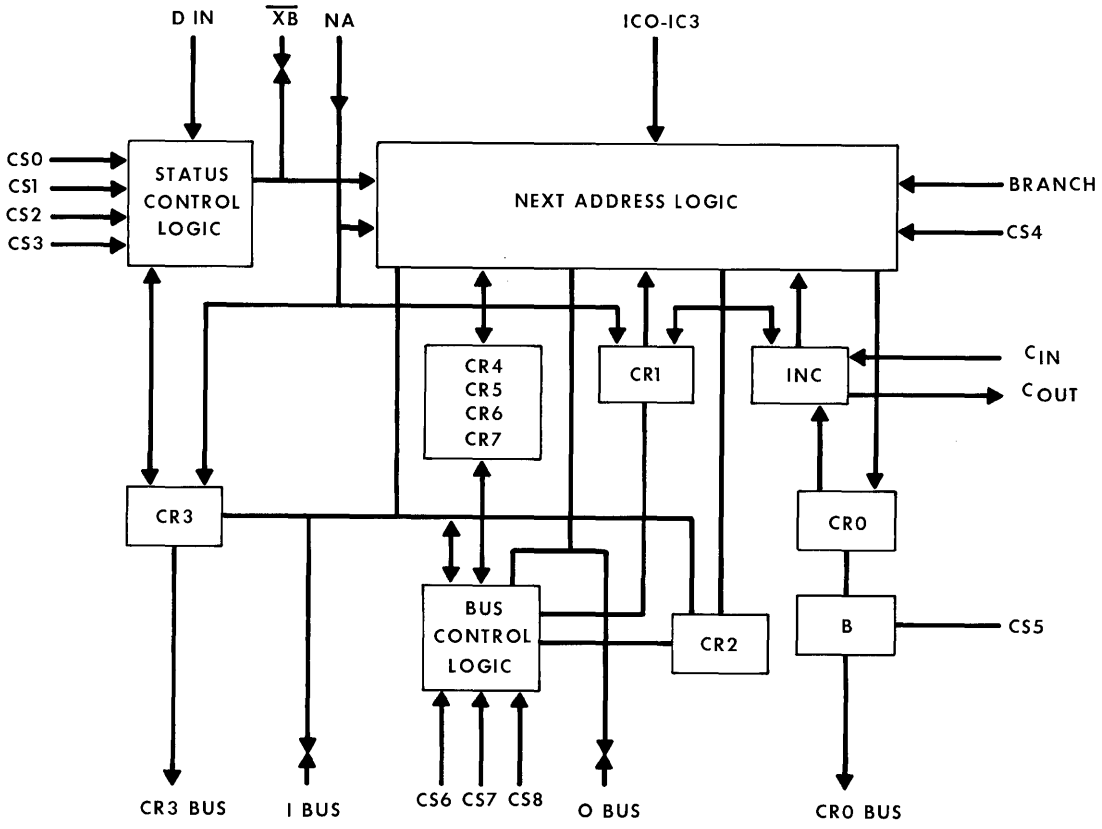


Figure 16-3 - Microprogram Control Function

The RF is a 4-word file in which RO, the program address (PA) register, has special functions. The DR register stores information to be used predominantly on the data bus. The AR MUX block selects the data to be loaded into the AR. The AR MUX can select the data bus, the output of RO, the output of the ALU, the

output of the RF, or the output of the AR. The selection decoding is provided by the microfunction and designation decode section. The clock signal times all of the above registers.

PROCESSOR DESCRIPTION

A block diagram of the 201VL processor is shown in Figure 16-5. There are four main buses: address, data, I, and O. These four bilateral buses are 20 bits wide. The address and data buses are used to interface with I/O devices as well as with the memory system. The I and O buses are the internal data buses. The A bus allows data movement from the constant memory (CM) to the ALU block.

The memory interface block is composed primarily of MC10803s. All address manipulation is performed in this block, allowing the generation of 20-bit addresses to be handled in a straightforward manner in a 16 data-bit machine. An example of such manipulation is a branch relative (BY in 3A code) with respect to the PA register, which is a one-word instruction. The eight least significant bits (LSBs) of the instruction are the bits to be added or subtracted from the PA. When the BY instruction is fetched from memory, it is stored in the DR register of the MC10803s. The PA is the RO register of the MC10803s. RO and DR (with the pointer inputs masking out the 12 most significant bits [MSBs] are added or subtracted in the ALU of the MC10803. The result of the operation is moved to the address register (AR) which controls the address bus.

The CM, a 64 by 16 memory, contains constants used by the microprogram. The CM can be addressed either directly by a microinstruction or by the 6-bit L register. The L register can be loaded either directly from the microinstruction or from the I bus.

The register memory (RM) contains the 16 general-purpose registers used by the 3A instruction set. These registers are 20 bits wide; however, only registers 13 and 15 are used by the 3A instruction set as full 20-bit registers. The others are used as 16-bit registers. The design of the RMs is very similar to the random-access memory (RAM) of the 201CC processor. In both processors the microcode has the capacity to deal with the 4 MSBs of the memory without affecting the 16 LSBs and vice versa. This capacity enables the microcode to form the 20-bit addresses.

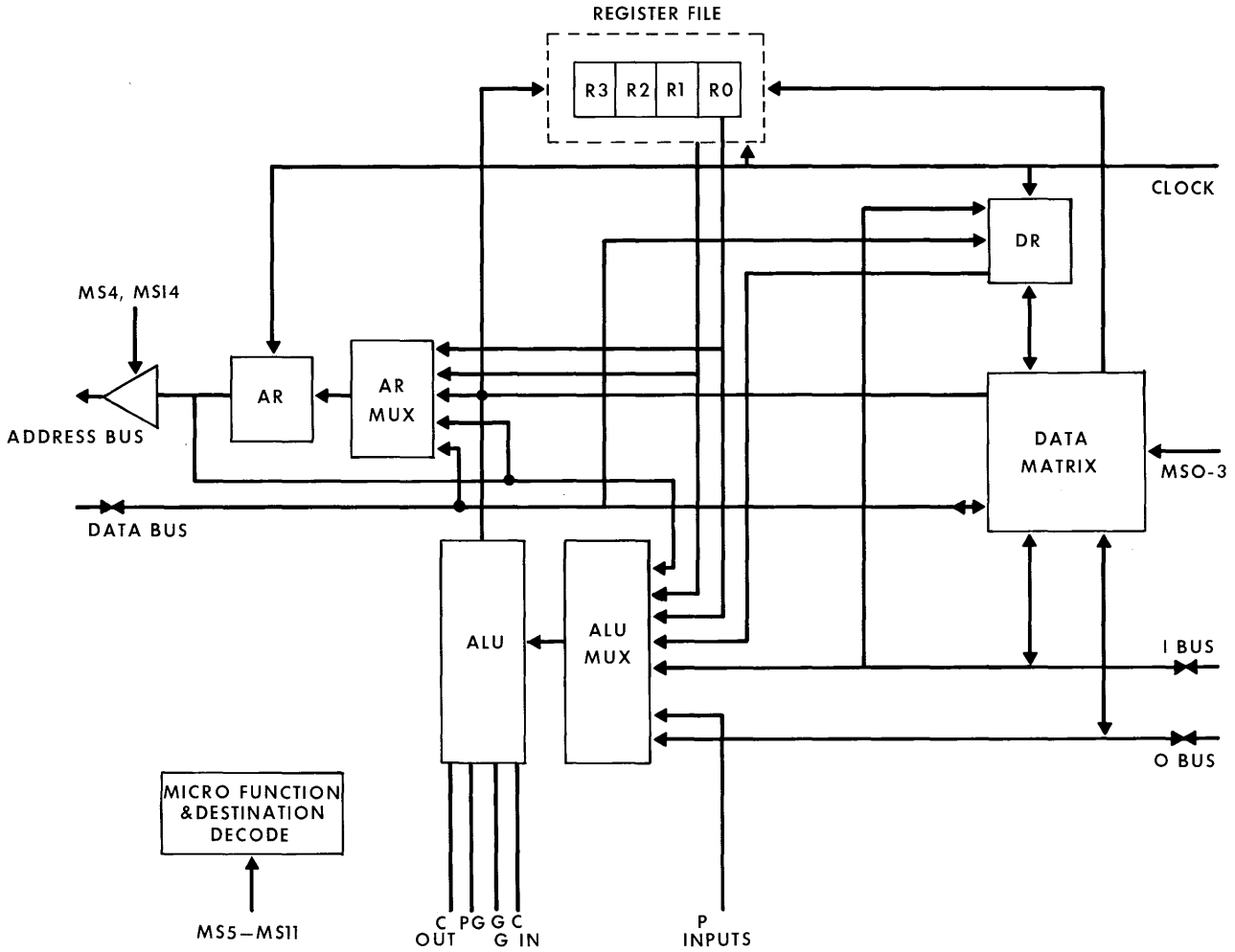


Figure 16-4 - Memory Interface Function

The ALU block is mainly comprised of MC10800s. This unit performs only the logic and arithmetic operations on data, all address operations being performed by the memory interface block. The ALU block is 16 bits wide.

The temporary memory (TM) is a 16 by 20 memory which stores data used by the microcode. The TM is addressed directly by a microinstruction.

The timing sequencer primarily made up of three MC10802s, provides all the timing for the processor. During operations with the main memory system, the execution of microinstructions and memory access can be overlapped. However, the test for memory complete is performed by hardware in the timing sequencer rather than by a microinstruction. The processor does not execute microinstructions during the test for memory complete, so it runs efficiently with memory systems whose access time is not a multiple of the microcycle.

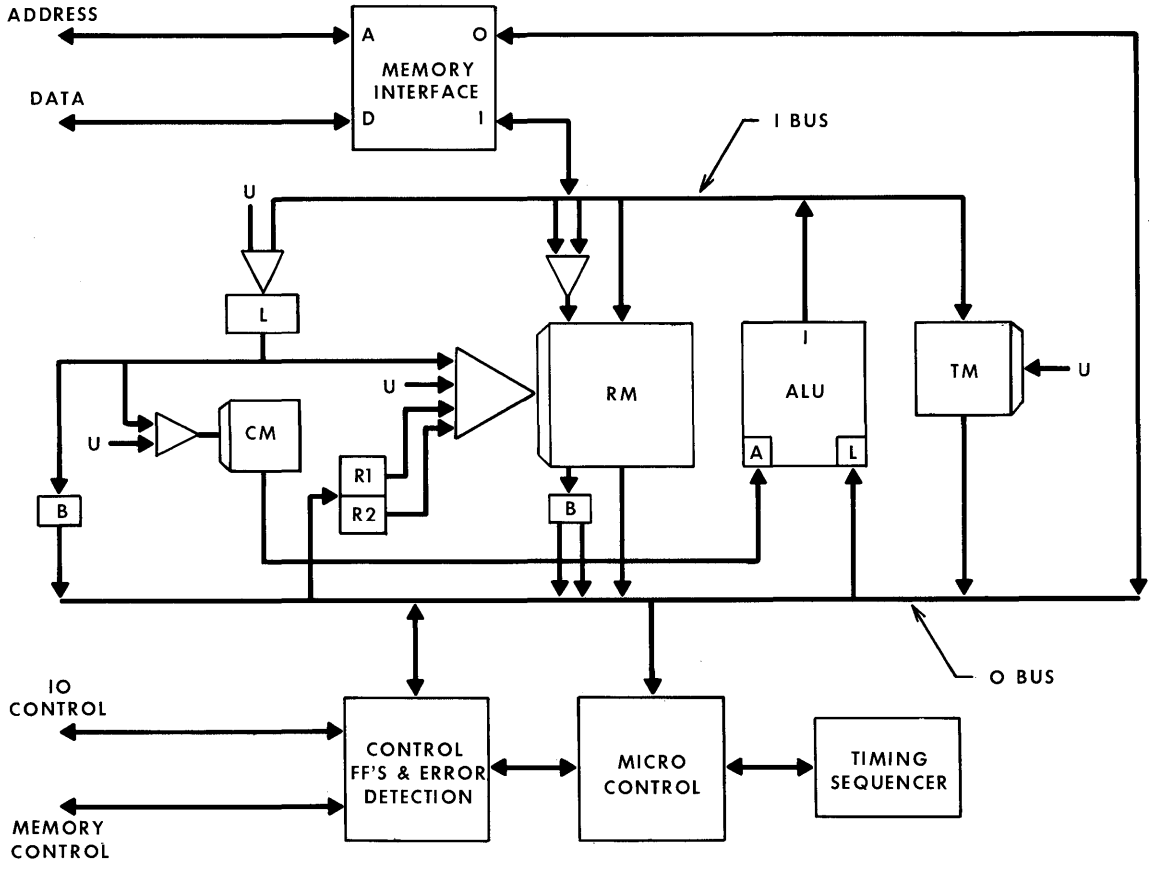


Figure 16-5 - 201VL Processor

The microcontrol block consists of the microstore (2048 words), microlatches, and MC10801s (which control microaddress generation). Most of the decoding of the microinstructions is done internally in the MC10800s, MC10801s, MC10802s, and MC10803s. The microinstruction is a 61-bit word with an execution cycle of 100 ns. Figure 16-6 shows the relationship of the microstore, the MC10801s, and the microlatches. Three MC10801s are used. When the microinstruction is fetched from the microstore (the microaddress is generated by the MC10801s), it is stored in the microlatches. The MC10801s then initiate the fetch of the next sequential microinstruction. Hence, the fetching and execution of microinstructions overlap.

The manner in which a macrointerrupt and macroinstruction fetch is handled is shown in Figure 16-6. When the first word of a macroinstruction is fetched in a microprogrammed processor, it is necessary to jump to the first step of the emulation routine for that macroinstruction in an efficient way so that minimal delay is introduced. The method used here is as follows. When the first word of the macroinstruction is fetched, the INST enable signal (from the timing sequencer) is made true, which disables the output of the MC10801s and enables the output of the OP programmable read-only memory (PROM). The OP code of the macroinstruction addresses the OP PROM. The output of the OP PROM addresses the microstore and is loaded into the CRO of the MC10801s. The OP PROM output is the starting address of the emulation routine for the macroinstruction.

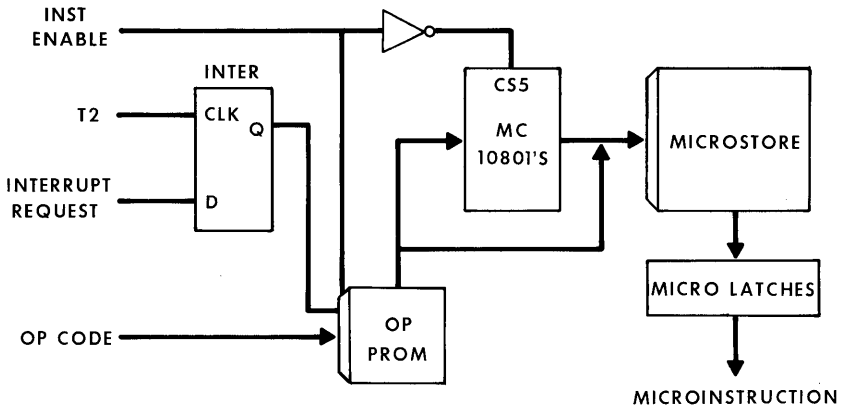


Figure 16-6 - Microstore Structure

Macrointerrupts are recognized when a macroinstruction is fetched. If the INTER flip-flop is set when a macroinstruction fetch occurs, the output of the INTER flip-flop causes a word to be accessed from the upper half of the OP PROM. This word contains the starting address for the macrointerrupt handler routine. Microinterrupts are handled by initializing the timing sequencer to the proper state.

CONCLUSIONS

The 201VL processor, including the microstore, would be contained on five multilayer, printed wiring boards and would consist of approximately 140 integrated circuits with a maximum power dissipation of 106W. Because of the long microword and the availability of only 4096-bit PROMs to implement the microstore, the latter would be contained on two multilayer, printed wiring boards with a maximum power dissipation of 41W.

The cost of the components for the processor based on a low-volume procurement would be approximately $5000. The microstore accounts for $2800 of this figure. Hence, it becomes the dominating factor with respect to cost, as well as to physical space and power dissipation.

Based on the instruction mix of the DIMENSION 2000 call processing program, in addition to its software architecture which allows the use of the pseudocache memory, the performance of the 201VL is 2.0 to 2.6 times faster than that of the 3A processor. Other applications, however, may be able to duplicate the performance produced with the DIMENSION.

APPLICATIONS II

SPEECH OUTPUT FROM A MICROPROCESSOR

M. Baumwolspiner, BTL Dept 4391, HO, NJ

ABSTRACT

This report presents several techniques which can be used to produce microprocessor output in the form of human speech. Speech output, unlike display (visual) output, can easily be transmitted over phone lines and directed to multiple users. Particular emphasis is placed on the formant generation and waveform generation techniques. The microprocessor software algorithms, processing time, and hardware constraints are also discussed.

INTRODUCTION

Advances in technology along with strides in the understanding of the speech production process have made voice output from microcomputers increasingly practical.

Utilizing voice output to replace or complement display output has many advantages. One of these is that voice output is generally easier to transmit and can easily be directed to multiple locations. Another advantage is practicality. For example, while craftspersons are working their sight is usually directed toward the task at hand and not available to observe a visual output. Their hearing, however, can be utilized to receive the computerized information.

There has been much concern over whether the advantages of the synthetic voice mentioned above can match the clarity and unambiguity offered by display output. The purpose of this paper is to describe several techniques used to perfect the synthetic voice. These range from the high bit rate and high quality techniques

(based on the digital encoding of the actual speech waveform) to the low bit rate techniques (based on the reproduction of speech in terms of fundamental models of the voicing process).

WIDEBAND SYNTHESIS

Wideband synthesis techniques are based on a digital encoding of the actual speech waveform. The three techniques belonging to this category are pulse-code modulation (PCM), differential pulse-code modulation (DPCM), and delta modulation (DM). Each of these methods approximates the time-domain waveform without regard to the fundamental parameters of speech reproduction derived from the voice generating organs. These methods produce good quality voice output with low coder complexity at the cost of a high bit rate.

Adaptive versions of these coding techniques (APCM, ADPCM, ADM) have also been investigated. They offer improved performance but at the cost of increasing coder complexity.

Further improvement is possible by the use of logarithmic quantization - generally referred to as companding. Through companding, toll quality speech can be attained with 7 levels of quantization; a uniform quantizer would need 11 bits for similar performance.

The bit rate required in all of these techniques ranges from 12 to 48 kb/s. Figure 17-1 compares the best of these techniques[1] in terms of signal-to-noise ratio (SNR).

In terms of hardware the synthesis method is relatively simple but has expensive memory requirements. The microprocessor using this method can control and select the required sections of memory containing the given sentence. Through the use of stored articulation algorithms, the microprocessor can also organize complete sentences by selecting different versions of the same word. The selected version depends upon sentence placement. In the adaptive versions of wideband synthesis, the microprocessor can be used to implement in software the adaptive algorithms which would normally be implemented in hardware.

Floppy disc systems are well suited for wideband synthesis applications. Recent advances in magnetic bubble technology, however, may provide a significant cost improvement.

Further information on the specifics of these techniques may be obtained from references 2 through 4.

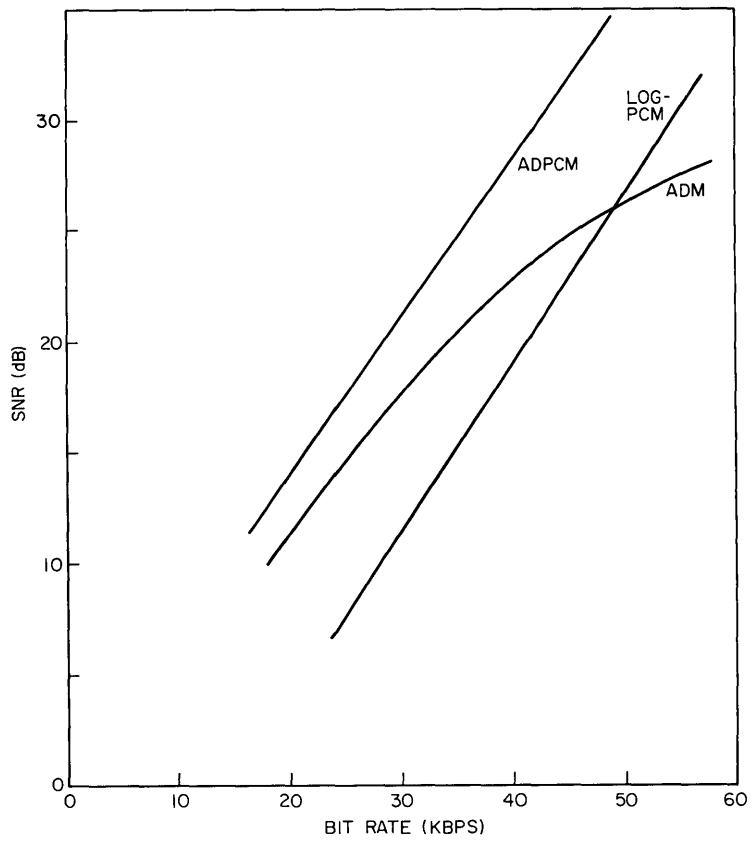


Figure 17-1 - Wideband Coding Comparisons

FORMANT SYNTHESIS

The formant synthesis technique simulates the vocal tract and associated vocal organs through an electrical equivalent circuit. Comprised of a set of resonant (formant) networks, this circuit is excited by a variable pulse (pitch) generator. The formant trajectories in time are of great importance in recognizing a word or phrase. However, time-localized breaks are often needed to perceive consonant sounds such as "b" or "d."

Voice spectograms reveal two sources of excitation: voiced and unvoiced sounds. The voiced sounds are produced by the glottis when it emits pulses having a base frequency of 60 to 250 Hz. The unvoiced sounds, e.g., the "s" and "f" sounds, are synthesized by passing a noise source through a variable pole-zero network.

Although many parameters are required in a formant synthesizer, they are nevertheless relatively constant during short time intervals. As a result the bit rate required is approximately 700 b/s. This compares favorably with PCM systems requiring 24 to 64 kb/s.

A microprocessor can be employed to interpolate and interpret the basic parametric data stored in read-only memory (ROM). The microprocessor can then output this information, through a multiplexed bus structure, to the formant networks, pitch generator, and output attenuator.

A block diagram of a formant voice synthesizer linked to a microprocessor is shown in Figure 17-2. The analog section (or voice channel) contains a set of second-order, variable-frequency, active filters driven by a pulse generator. This section realizes the "voiced" sounds. A simple noise generator followed by a second-order pole-zero network can be switched into the signal path to realize the "unvoiced" sounds. The microprocessor controls the analog section through "memory mapped I/O." That is, part of memory space is reserved for the voice channel outputs; a transfer to memory at those locations will output the desired information. The microprocessor ROM can contain a user application program in addition to the voice synthesis program. Hence, a user may employ the microprocessor for an independent task and call as a subroutine the voice synthesis program whenever a voiced output is required.

A call to the synthesis program requires the word or phrase number (ID) to be passed along through one of the registers. The voice synthesis program will look up in the "word dictionary" the starting and ending locations of ROM where the actual parametric data are contained. The microprocessor will decode and interpolate this data and control the appropriate analog elements. When the word or phrase is completed it will return to the user program.

Most microprocessor based systems can be augmented to provide voice output through the addition of software and a voice channel card. An Intel 8008-1 system can accommodate one real-time voice channel whereas an Intel 8080A system

can handle up to eight real-time voice channels. The software requirements typically consist of 0.5K to 1K bytes for the synthesis algorithm and 1K bytes per 18 words of voice storage.

Additional information on formant synthesis techniques may be obtained from references 5 through 7.

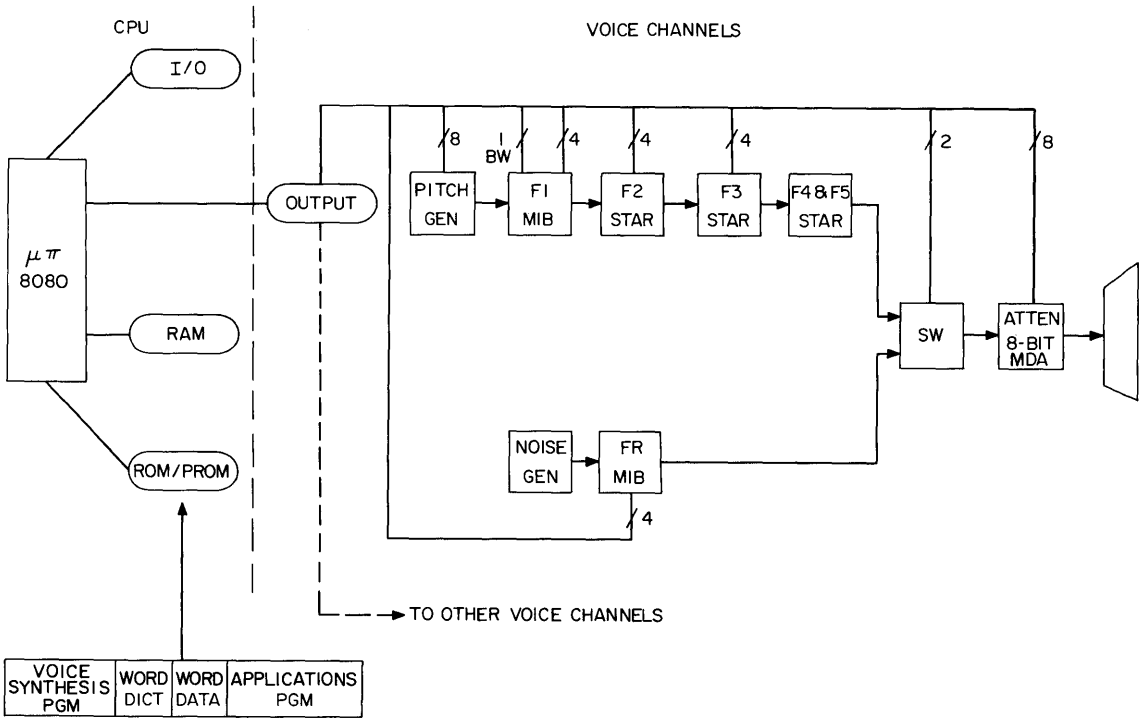


Figure 17-2 - Voice Synthesizer

WAVEFORM SYNTHESIS

The waveform synthesis technique is based on a time-domain realization of the speech waveform. A set of basis functions in the time domain serves as the primary waveforms on which the entire synthesis strategy is based. This set of basis functions, in conjunction with a time-compression (and expansion) operation, spans the parameter space of the vocal tract model.

The vocal tract can be represented[5] in the complex frequency domain by the infinite product:

$$H(s) = \prod_{m=1}^{\infty} \frac{w_m^2}{s^2 + b_m s + w_m^2} .$$

When the different sound patterns are produced, the formant frequencies (w_m) and to a lesser extent their bandwidths (b_m) attain differing values. Hence, if we revert to the time domain and select a wide enough span of formant frequencies we obtain a set of basis functions which can be used to approximate the speech waveform.

Given

$$f_1(t) = L^{-1}[H_1(s)], \quad f_2(t) = L^{-1}[H_2(s)], \ \ldots$$

where L^{-1} = Inverse Laplace transform

then

$$g(t) = \sum_{n=1}^{\infty} f_k(a_n t - c_n)$$

where g(t) is an arbitrary speech waveform and k is some function of n (see Figure 17-3 for a pictorial representation of this equation).

The significance of this last equation is that the summation over the basis functions f_k does not involve a weighted summation. The operation involved is a time compression or expansion which is easily accomplished in a microprocessor realization.

The variable c_n in the last equation represents the pitch period interval. That is, the vocal tract - which gets excited periodically by the glottis - will produce a new additive component to the output waveform at time c_n. It is interesting to note that the basis functions generally decay in two, or at most three, pitch cycles. Hence, the addition in the last equation involves only two or three basis functions. We can more directly visualize this synthesis technique by drawing the vowel formant trajectories in the log F1-F2 plane, as in Figure 17-4. In the conventional formant synthesis the first and second formants are independent and have a fixed range associated with each parameter; hence, they can attain any value within the upright rectangle as shown in Figure 17-4.

As an alternative, for each given set of formant locations, we can derive the pulse response for the formant network and store away the corresponding time function. As a result, a speech pattern can be produced in the time domain by reading out the appropriate time function at each pitch period interval, as in Figure 17-3. Since the time function may be longer than a pitch period, the trailing edge of this time function is superimposed upon the next pitch period. This is easily achieved, for it is an additive process without multiplications. The difficulty with this synthesis strategy is that it requires a large amount of storage to store the time functions corresponding to all the combinations of allowable formant frequencies.

By employing a well known Laplace transform operation namely,

$$1/a\ f(t/a) \quad <----> \quad F(as),$$

we can significantly reduce the number of basis functions required. This equation indicates that by time compression (and time expansion) we can linearly scale the frequency domain; hence, the formant frequencies can be scaled up or down.

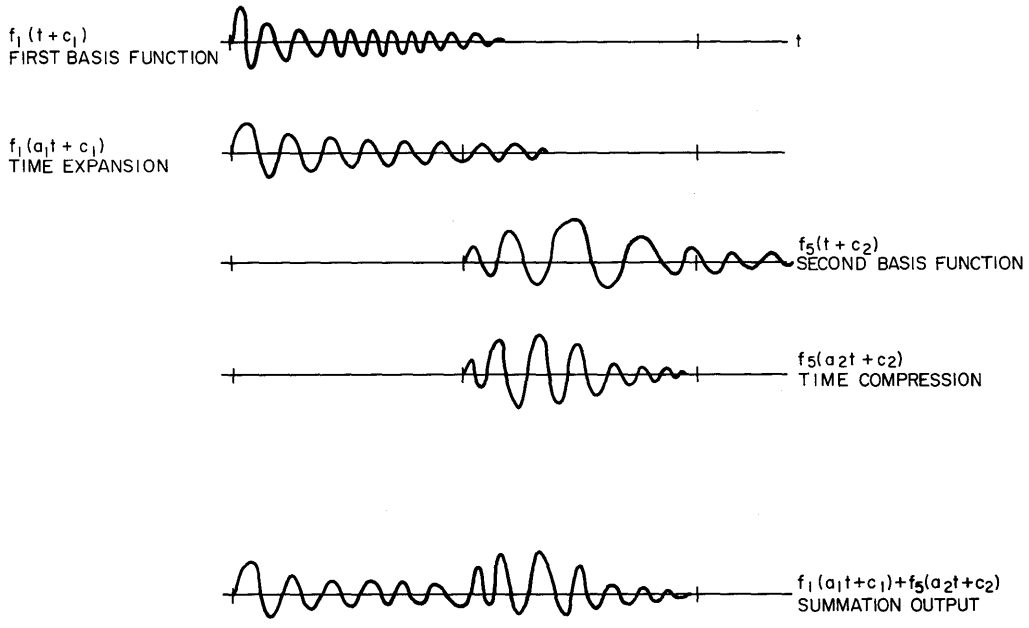


Figure 17-3 - Waveform Construction

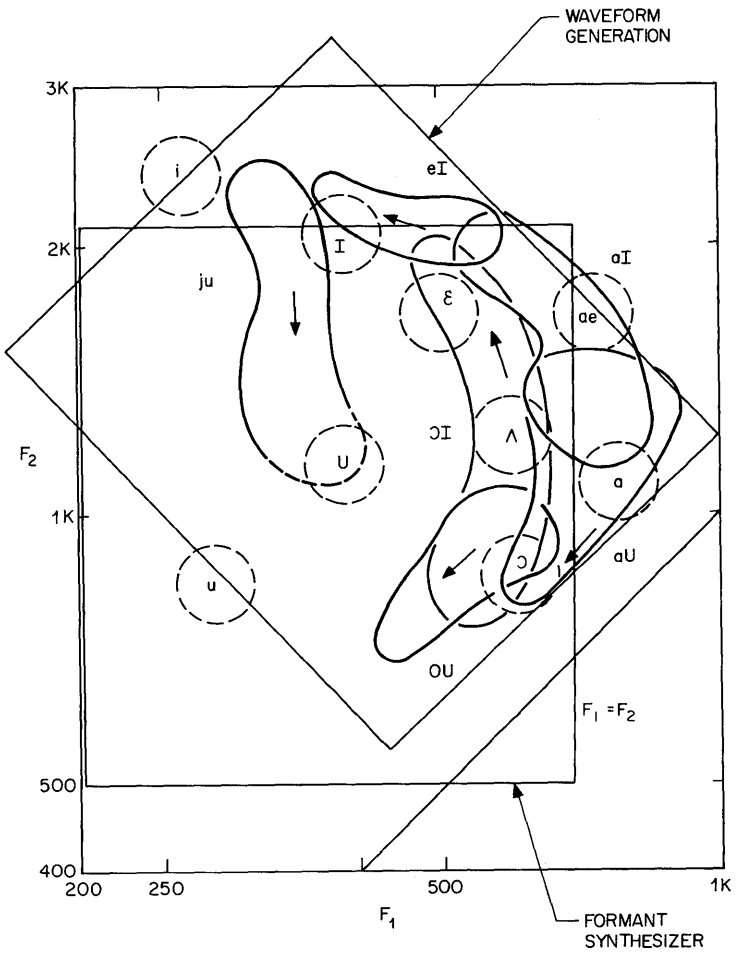


Figure 17-4 - Frequency Span of Formants One and Two in Vowels and Diphthongs

Thus, by employing this technique, we choose a sloping rectangle as shown in Figure 17-4. The basis functions are chosen along a negative sloping line. By time-scaling, the formant positions are varied along a positive sloping line; hence, a rectangle can be spanned.

It is noteworthy that from typical vowel trajectories, more efficient coverage is attained by a sloping rather than an upright rectangle. Figure 17-4 shows the central regions of variation for the F1 and F2 formants for diphtongs. With the exception of the "u" vowel the region is efficiently covered by a sloping rectangle.

The synthesis strategy for the third formant is also simplified. In conventional synthesizers it is essential that the third formant network be variable in its resonant-frequency position. If it is not variable an objectionable tone effect will be perceived by the listener when the second formant approaches the fixed position of the third formant. According to the strategy, the time-scaling will automatically shift the third format as well as the first and second formants. From listening tests, this seems to be a valid assumption.

From a physical viewpoint, time expansion or compression within a pitch period can be interpreted as lengthening or shortening the effective length of the vocal tract. It is well known that the resonant nodes of a straight pipe are:

$$f_m = (2m-1)\ v/4\ell$$

where v is the velocity of sound and ℓ is the length of the straight pipe. As shown in this equation, all the formant frequencies are inversely proportional to the length of the tube. Hence, the basis functions depict the different shapes which the physical vocal tract can attain and the time-scaling depicts the variations in the effective length of the vocal tract.

The waveform synthesis method can also be used to generate "unvoiced" sounds. Only one basis function of approximately 20 to 30 ms is needed. This basis function is the white noise response of a fricative, pole-zero network. By time-scaling the above basis function, we can achieve the effect of a variable pole-zero network and are thus able to generate the different frication sounds such as "s" and "f".

The overall scheme of this waveform synthesis technique is to generate the time domain waveform for each pitch period. It is a synchronous technique in which the waveform is initiated in the beginning of the pitch period. The appropriate basis function is applied to a digital-to-analog (D-A) converter at a sampling rate which corresponds to the basic sampling rate (10 kHz) multiplied by the time-scaling parameter. At the end of the pitch period interval the new basis function is superimposed onto the remaining portion of the first basis function as shown in Figure 17-3. In addition, the waveform may be amplitude-scaled on a long-term basis to accommodate normal variations in amplitude level. This can be accomplished through a multiplying D-A converter or digitally, through scaling by factors of two.

Memory and Hardware Requirements

Twelve basis functions of 15-ms duration each and eight levels of time-scaling are necessary to span the voiced sounds. Another basis function of 20- to 30-ms duration is needed for the unvoiced sounds. We will assume that each basis function is stored in PCM format at a rate of 48 kb/s. Thus, to store the entire set of basis functions, 0.210 second is required for a total bit count of 10,080 bits.

In addition, for each word synthesized the following are required in a 20-ms frame rate:

	No. of Bits	b/s
Basis Function	4	200
Time Scale	3	150
Amplitude	4	200
Pitch Period	4	200
Total Bit Rate		750 b/s

A microprocessor is needed to decode the information, provide the time-scaling, select the appropriate basis function from ROM, and produce the pitch cycle. External to the microprocessor, a D-A converter and a programmable timer are needed. Time-scaling can easily be achieved by outputting to the timer the time interval to the next interrupt. In this manner, the microprocessor can be free to perform the other control tasks while the timer performs the time interval count.

CONCLUSION

Several techniques which can be utilized for voice synthesis have been briefly introduced here.

The wideband synthesis techniques (12 to 48 kb/s) offer high quality and simple circuitry but have expensive memory requirements. On the other hand, low-band synthesis techniques (400 to 2000 b/s) offer an order of magnitude reduction in memory but also degrade the quality of speech.

Among the low-band synthesis techniques, the formant synthesizer (500 to 1000 b/s has been widely investigated and can be produced at relatively low cost - $50 to $300 for a 45-word vocabulary. The linear predictive coder (LPC) technique (not mentioned earlier in this paper) produces better quality but requires more expensive hardware and a bit rate of 1800 to 3600 b/s.

The relatively recent waveform synthesis technique (500 to 1000 b/s) is particularly well suited for microprocessor implementation and has the potential of producing good quality sound. The use of time compression or expansion in the synthesis strategy makes this technique viable and cost effective.

REFERENCES

1. N. S. Jayant, "Digital Coding of Speech Waveforms," Proc. IEEE, Vol 62, pp 611-632, May, 1974.

2. H. R. Schindler, "Delta Modulation," IEEE Spectrum, Vol 7, pp 69-78, October, 1970.

3. H. S. Black, Modulation Theory, Princeton, New Jersey, Van Nostrand, 1953.

4. P. Cummiskey, et al, "Adaptive Quantization in Differential PCM Coding of Speech," BSTJ, pp 1105-1118, September, 1973.

5. G. Fant, Acoustic Theory of Speech Production, Manton and Company, 1960.

6. L. R. Rabiner, et al, "A Hardware Realization of a Digital Formant Speech Synthesizer, IEEE Trans. Comm. Tech., Vol COM-19, No. 6, December, 1971.

7. M. Baumwolspiner, A Microprocessor Controlled Voice Synthesizer, TM-76-4391-4, June 24, 1976.

APPLICATIONS II

A SMALL DIGITAL TIME DIVISION SWITCH USING MICROPROCESSOR CONTROL

W. L. Aranguren and R. E. Langseth, BTL Dept 1344, HO, NJ

ABSTRACT

We describe a small digital time division switch which could be used in future digital satellite systems. The switch is designed to interface with up to 30 T-carrier systems on the terrestrial side. Its architecture is that of a common-control, bus-oriented switch, using a common time-slot interchanger for simultaneous switching of up- and downlinks. The controller is implemented as two Motorola M6800 microcomputer chips and associated memory. The switch can communicate with a variety of terrestrial originating switches, including those using either register-sender or cut-through operation. With proper software, a simultaneous mix of such originating switches can be handled, and either delay-dial, wink-start, or dial tone-start outpulse control can be utilized in either direction of call setup (i.e., to or from a terrestrial switch connected to the satellite earth station).

INTRODUCTION

Recently there has been considerable effort in Department 1344 to develop system concepts for a second generation domestic communication satellite system. The model system would utilize digital transmission and operate in the 12- to 14-GHz bands. Current thinking tends to place this system in the 1980s private line environment. Because of the digital transmission, it would be desirable to perform as much of the terrestrial interfacing as possible in a digital fashion and thereby enjoy advantages of time-shared interface processing.

In this paper we discuss work designed to demonstrate the feasibility of a small (approximately 700 circuits) digital interface which could be useful in an envisioned private line satellite system serving the large tandem tie trunk network (TTTN) customers among others. The interface was conceived for the purpose of accepting several T-carrier terrestrial inputs and switching the circuits (8-bit bytes) on them to the proper output buffer for transmission to the desired destination. Control is performed by multiple (currently two) microprocessors, permitting a very compact controller. The entire interface could be mounted in a corner of the customer's premises. Such an interface could be the building block for a satellite system in which many relatively small earth stations share the satellite capacity via time division and/or frequency division multiple access.

Because there may be more than one customer sharing an earth station, efficient utilization of each satellite circuit is improved if the satellite-terrestrial interface can provide a switching function. This function also makes efficient use of those T-carrier circuits available to a given customer. No prededication of the destination of terrestrial access circuits is required, thus permitting maximum employment of the satellite circuits assigned to a particular destination. (The number of the latter can, of course, vary on a demand-assignment basis to accommodate diurnal demand peaks.)

We will now describe briefly the basic operation of the earth station. In essence, the output bit stream is a time division, multiplexed collection of digital trunk circuits. On a short-term basis, at least, these satellite circuits will be grouped by destination, meaning another specific earth station. The earth station will contain a set of buffers corresponding to each of the possible destinations. (Of course, there will be separate sets for transmit and receive functions.)

The essential function of the interface will be to sort or switch input circuits from the T carriers into the proper output buffer. The proper buffer will be determined by customer-dialed digits, using appropriately defined codes for all of the possible locations in a given customer's network. In this way a TTTN customer can dial all the necessary digits at once, and the interface, operating essentially as a register-sender common-control switch, will automatically complete the connection through the satellite to a similar interface at the other end.

ARCHITECTURE AND OPERATION

Figure 18-1 shows the overall block diagram of the interface. Customer T1 lines are multiplexed by bytes (8-bit words) onto an 8-bit wide data bus. This method of multiplexing permits use of slower, smaller power logic families than would have been required by an equivalent serial multiplexer. Because of the particular configuration of the time-slot interchange (TSI) hardware, the data corresponding to the two directions of a given circuit appear on the up-link and down-link buses simultaneously. Such an architecture is a prerequisite for digital common-control echo suppressors. In addition to customer T1 lines, the multiplexer also receives two equivalent T lines which carry telemetry information for call setup, breakdown, and satellite switch control in the case of a switched multibeam system. This total of 32 equivalent T lines must be brought into synchronization before multiplexing takes place.

The TSI control is simply a map stored in the random-access memory (RAM), which is capable of being altered or updated by the controller. The map relates terrestrial to satellite circuits, i.e., it relates a T1 and time-slot number to its desired satellite TDM burst number and time-slot number within that burst. It may also map a given terrestrial to a desired service circuit. The mapping controls boxes are labeled TSI & DESTINATION SORTING. The three boxes taken together can connect any terrestrial circuit to any service or satellite circuit. Note that both path directions are mapped simultaneously by the single TSI controller.

During the last frame of the 12-unit framing sequence on the T1 lines, the least significant bit of the 8-bit pulse code modulation (PCM) byte is replaced with a signaling bit which carries on-hook, off-hook information pertaining to the corresponding voice circuit. Dial pulse information, because it is simply a series of on-hook, off-hook pulses, is also carried by the signaling bit. The line scanner extracts these signaling bits from the bus and passes them to the controller, which will be described in more detail later. The controller maintains status of all T lines and checks each new signaling bit for a change in state.

Let us trace through a telephone call setup with particular emphasis on hardware activity. We will begin at a private branch exchange (PBX) using cut-through operation. The PBX, in response to a request for service, seizes a circuit on the T1 line, which causes the appropriate signaling bit to be set. The controller recognizes this request and updates the TSI controller so that a digitally generated

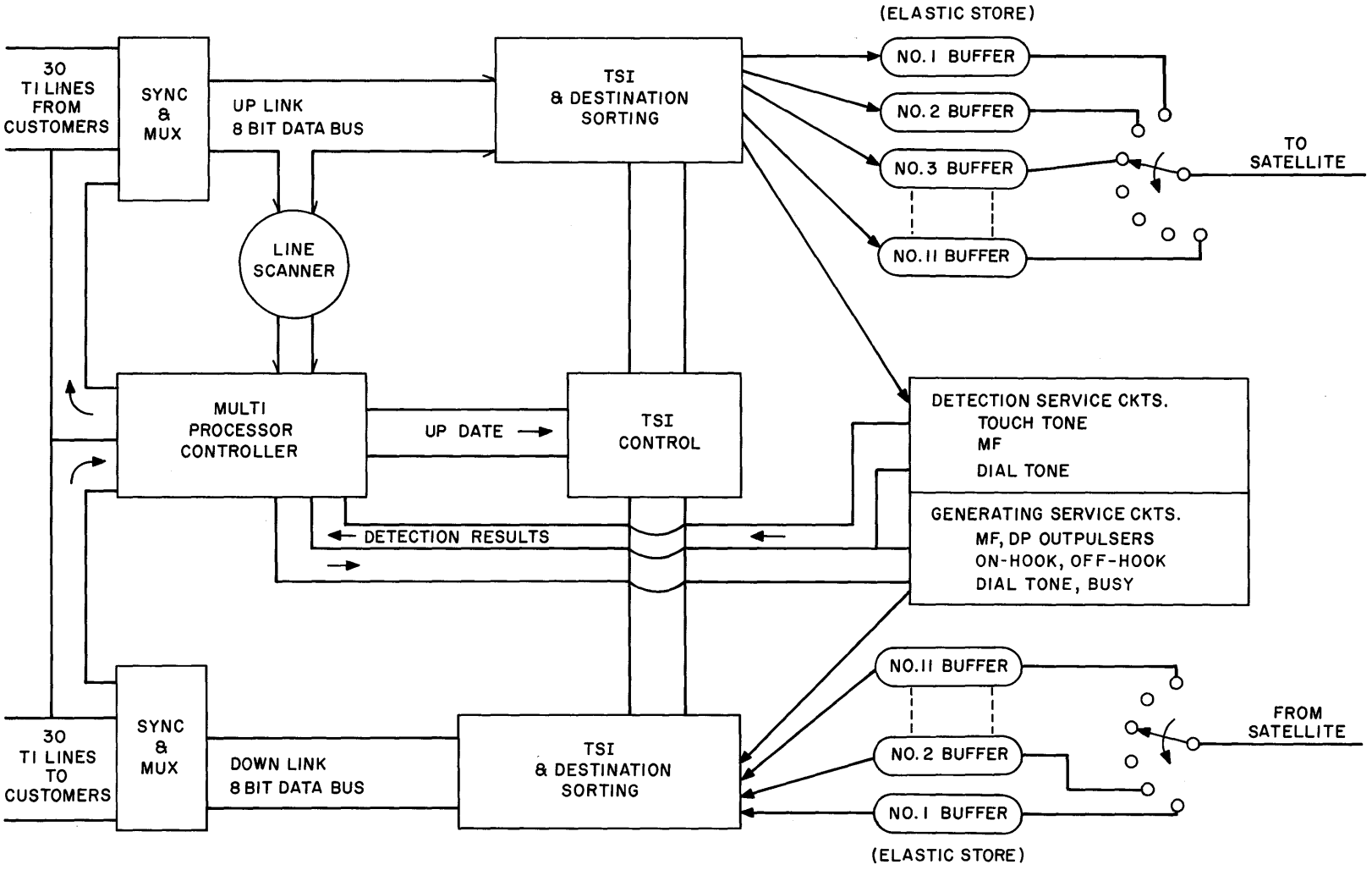


Figure 18-1 - Interface Architecture

dial tone is connected to the return half of the seized voice circuit. The customer, upon receipt of the dial tone, dials the appropriate digits. Upon receipt of the first digit the dial tone ceases. When the remaining digits have been received, the controller determines whether a valid access code was dialed. If not, a digitally generated, fast busy signal is returned to the customer. A valid code is translated into an output buffer number, and a satellite circuit is chosen. The request - the number dialed and the number of the satellite circuit chosen - is forwarded to the desired terminating earth terminal via the forward-error-corrected telemetry channel.

The terminating controller now locates the appropriate T1 line and seizes a circuit on it by changing its outgoing signaling bit. The bit is changed by removing the on-hook generating service circuit and connecting the off-hook generating circuit (assuming E&M or loop-type signaling interfaces at the PBX). Upon detection of the returning go-ahead signal from the PBX (either dial tone-start, wink-start, or delay-dial), a loaded digital outpulser is connected to the line via the TSI. Finally, after outpulsing, the TSI is updated once again to complete the circuit. If no circuit is available, a message is sent back to the originating earth station instructing it to return the trunks-busy signal to the originating customer. Note that the busy signal is not returned directly from the terminating earth station due to a unique characteristic of the service circuits, namely, that a generating service circuit can not only be connected to any T line, but it may be connected to all T1 circuits simultaneously (thus, all customers connected to a given ground station could receive a busy tone simultaneously). Further, any up-link telemetry circuit may be directed to any single satellite circuit, but it may go to only one such circuit at a time. Therefore, if we wanted to return a busy or dial tone over the satellite path it would have to be generated on as many up-link telemetry circuits at the T1 side as would be required to satisfy demand. Consequently, we have chosen to return any terrestrial trunk-busy responses to a particular satellite request as messages over the telemetry link and take advantage of the one-to-many service capability at the originating end.

MICROPROCESSOR CONTROLLER

As has been mentioned, the control function of the satellite-terrestrial interface is performed by multiple Motorola M6800 microprocessors. We will now describe the operating mode of these processors. Operating in the environment described earlier, signaling information appears on the input T lines every 12th

frame; in certain other special services offered by the Bell System, additional signaling states appear during an alternate 12th frame sequence, offset by 6 frames from the first 12-frame sequence. The line-scanning portion of the processor sketched in Figure 18-1 is actuated by a 12th frame interrupt pulse from the synchronizing circuitry; since a frame is 125 μs long, the interrupt interval is 1.5 ms.

At each interrupt, signaling bits which correspond to one of the 24 time slots are read from each of the 30 input T lines. Because of the synchronization, the same time slot is present on all input T lines at a given interrupt. Thus, a given time slot on each T line is scanned every 36 ms (24 by 1.5). This rate appears to be fast enough to follow 10-p/s dialing. Thus, the line-scanning processor also collects dialing information from rotary dial customers. It also maintains status for all input circuits, performs time-out for customers who wait too long to dial, detects wink-start or delay-dial signal, assigns terrestrial circuits in response to requests from other earth stations, etc.

The TSI is controlled by a second processor, P2, which has limited 2-way communication with the first processor, P1, via a 64-byte mailbox. Since the TSI is also involved in providing dial tone to an originating customer, P2 participates in call setup rather early, responding to requests from P1 for dial tone. The second processor also interprets customer-dialed digits, selects satellite circuits, controls terrestrial outpulsers, forms messages for telemetering to the desired terminating earth station, and requests that P1 assign a terrestrial circuit when necessary.

In the case of TOUCH-TONE® customers, we plan to provide digital receiver circuitry connected as shown in Figure 18-1. In this application P1 would inform P2 of the seizure of an incoming circuit; P2 would then connect an available receiver and process the digits as before.

At this point on the learning curve, we have not provided a completely fail-safe system design. Certain obvious software steps have been taken to guard against occasional false signaling bits. Guarding against processor or memory failure would require additional, redundant hardware together with the appropriate software. Our primary goal here is a feasibility demonstration. To this end, a hardware model of the interface sketched in Figure 18-1 is being built. Much of the peripheral apparatus shown has been completed. In addition the essential

software for P1 has been written and burned into type 2708 programmable read-only memories (PROMs). The processor itself has been constructed on two circuit cards, which include the mailbox for communicating with P2. The physical layout is shown in Figure 18-2. Software for P2 is currently under development. P1 software occupies about 2500 PROM bytes, and about 6000 bytes of read-write memory are used to store status information for the terrestrial circuits as well as other temporary data. This memory uses the EMM SEMI 4200 4K static RAM. Additional information for each circuit tells P1 whether it is a delay-dial, wink-start, or dial tone-start circuit and whether it requires multifrequency (MF) or dial-pulse outpulsing.

Therefore, it is comparatively simple to provide the capacity for working into a variety of signaling-outpulsing arrangements. In effect, trunk signaling interfaces have been replaced (at the earth stations) by a common T-carrier interface together with the appropriate software.

CONCLUSIONS

We have described a small, single-stage, bus-oriented time division digital switch suitable for use in a postulated digital satellite system. The switch uses multiple microprocessor common control, with a single TSI memory simultaneously switching up- and down-link circuits. On the terrestrial side, provisions are made for accepting and synchronizing up to 30 T1 lines. Of the two processors in the present design, P1 is primarily involved in scanning for on-hook and off-hook changes on the terrestrial circuits so as to keep track of call progress and in collecting dial-pulse information. P2 controls the TSI and assembles and disassembles signaling messages, which are utilized to establish and break down calls over the satellite link. Under software control, the switch can communicate with originating switches using either delay-dial, wink-start, or dial tone-start outpulse control (the latter in the case of TTTN customers). Additionally, it can respond to seizure of an input circuit with either delay-dial or wink-start signals in the case of register-sender PBXs or with digitally generated dial tone in the case of cut-through PBX operation as in TTTNs.

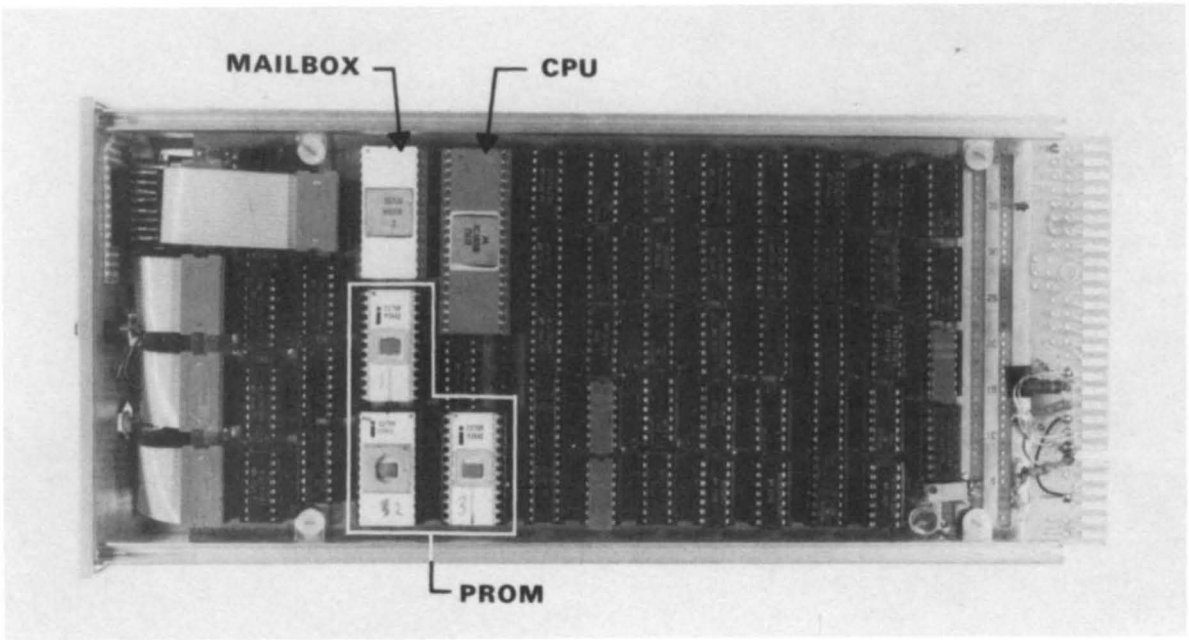


Figure 18-2(c) - Processor "One": CPU Board

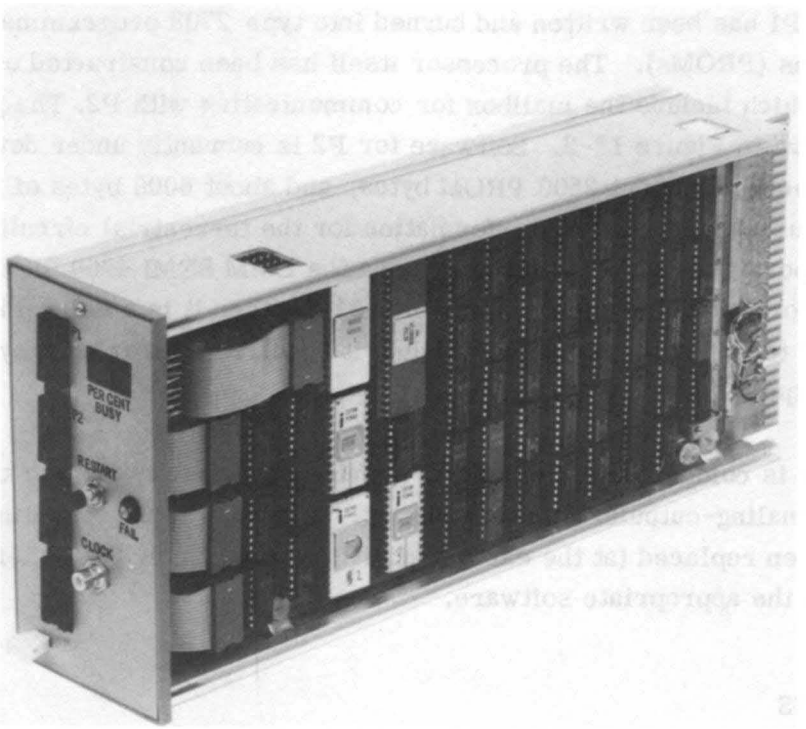


Figure 18-2(a) - Processor "One": Overview

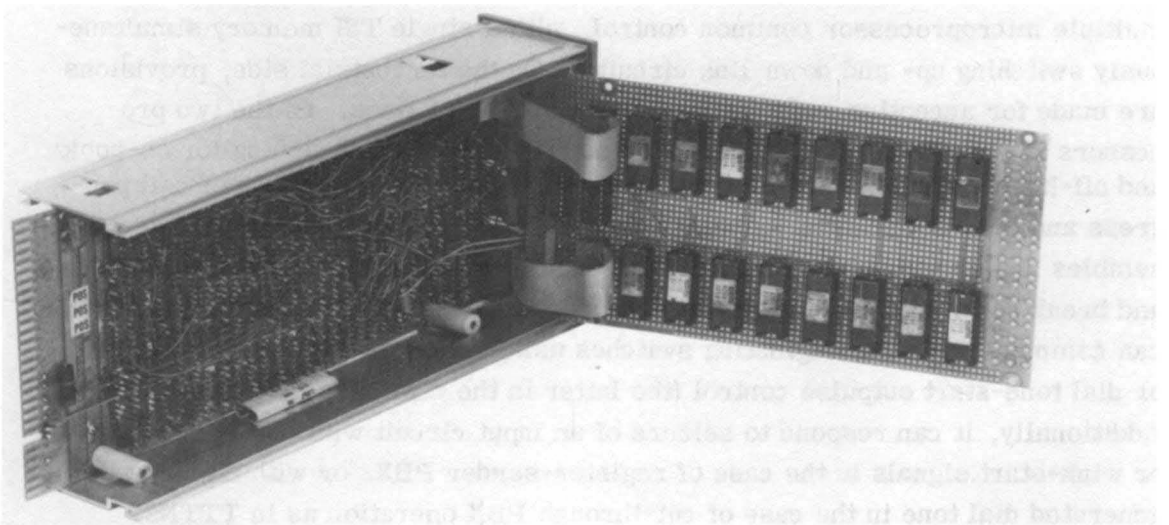


Figure 18-2(b) - Processor "One": RAM Board

APPLICATIONS II

A MICROPROCESSOR-CONTROLLED SWITCHING SYSTEM FOR L5E - DESIGNS, LESSONS, AND EXPERIENCES

P. G. St. Amand, BTL Dept 4225, MV, MA

ABSTRACT

A microprocessor-based terminal protection switching system (TPSS-1), to be used on the L5E coaxial cable routes, has recently been designed. The protection switching monitors 2 sets of 20 multi-mastergroup translators (MMGTs), and in the event of a failure of 1 MMGT, switches a protection MMGT unit into service. A fully loaded TPSS-1 may be responsible for up to 96,000 voice channels. The hardware contains a variety of fail-safe features, including parity and sanity timing. Self-checking diagnostics are included in the program. In the paper special emphasis is placed on the microprocessor game as seen from the user's view:

- What new design talents are needed?
- What support tools are nice and what support tools are necessary?
- What new problems does a designer face?
- What new pleasures does a designer enjoy?

The experiences gained on TPSS-1 may prove useful to a new microprocessor user about to enter a sea of spec sheets, coding forms, and design decisions.

HARDWARE

The L5E, 22 mastergroup, coaxial cable system will have multiplex equipment, MMGT, that carries up to eight mastergroups. The present AT&T policy is that any active unit carrying more than one mastergroup (600 voice channels) be automatically protected. The previous generation of L carrier multiplex, JMX, provided duplicate hardware to achieve protection; with the arrival of microprocessors an inexpensive, common control terminal protection switching system (TPSS-1) could be designed for the MMGT cable. The TPSS-1 uses only one protection unit to safeguard service on up to 20 regular MMGT units, and thus a fully equipped TPSS-1 system may be responsible for up to 96,000 voice channels.

The heart of the TPSS-1 is the microprocessor-based controller. Previous controllers, such as LPSS-3 for L5E or 400A for radio, employed hard-wired logic. The minicomputer alternative would have been too costly. The TPSS-1 has two identical 6-inch shelves, housing input/output (I/O) interface circuits, which are mounted above and below the central processing unit (CPU) shelf (see Figure 19-1).

Each I/O shelf holds up to 21 interface printed wiring boards (PWBs) consisting of one protection MMGT I/O card and from 1 to 20 MMGT I/O cards. Each of the cards in the I/O shelf holds the drive circuits and data input circuits associated with a single MMGT. The regular MMGT I/O card controls the coaxial switches, lights all indicators, monitors the MMGT pilot detector output, and senses the pushbutton of one MMGT unit. Initially, a TPSS-1 system is equipped with only one working MMGT and one protection MMGT. As the office call-carrying needs increase, additional MMGTs are added, and for each MMGT added, an MMGT I/O card is added to the shelf. This per-channel interface strategy gives a simultaneous display of the condition of all MMGTs. Since a typical TPSS-1 system may take over 20 years to achieve full growth, the delayed purchase of interface cards is an economic advantage.

The center shelf is a small, self-contained microcomputer with CPU, random-access memory (RAM), programmable read-only memory (PROM), peripheral I/O, and power supplies. Two dc-to-dc converters transform the -24 Vdc office power to +5 and ±12V. The outputs of the converters are fused on this shelf. There are three I/O circuits on the center shelf: relay I/O, carrier supply I/O, and diagnostic display I/O.

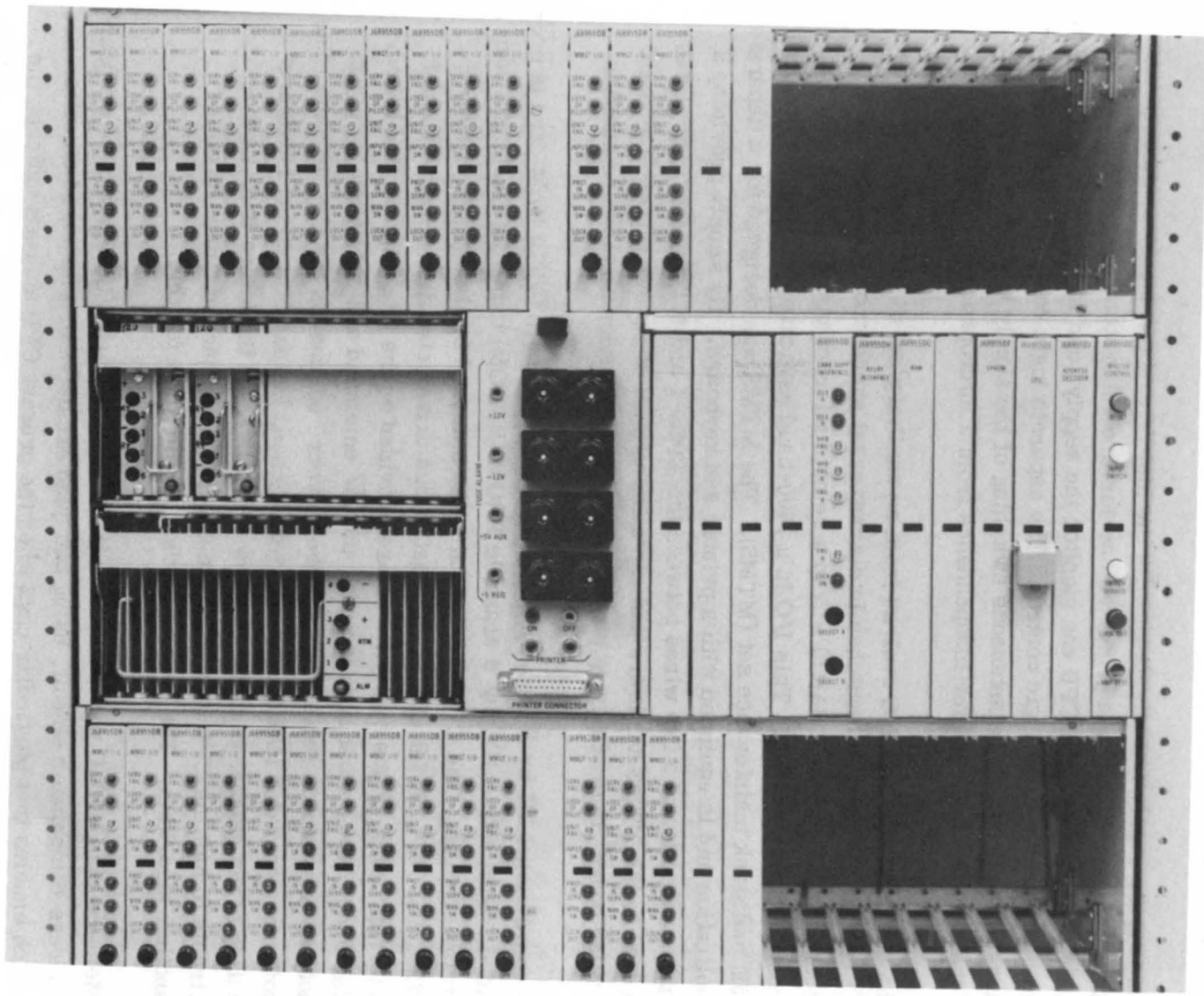


Figure 19-1 - TPSS-1 Controller

The relay I/O card simply provides 8-bit input and output ports which connect the office relay alarm system.

The carrier supply I/O interfaces with the MMGT carrier supply. Although the MMGT carrier supply is a protected one-for-one subsystem, with its own hard-wired logic control, the CPU can monitor the supply, and in the event of a failure, can override the dumb logic control. The separate logic control was provided because of the need for instantaneous switching of the supply in the event of failure of the working side, and for independence from a microprocessor which is down.

The diagnostic display I/O evolved from an auxiliary set of light-emitting diodes (LEDs) and thumbwheel switches into a universal synchronous/asynchronous receiver-transmitter (USART) link to a remote microcomputer that is used for system troubleshooting. This I/O is a 2400-baud asynchronous interface to the L5E multitask maintenance set (MTMS). The MTMS was designed for a separate application and is equipped with a printer and keyboard. By simply providing a cable connection of a few wires between the TPSS-1 and MTMS, the TPSS-1 has access to the MTMS hardware and can eliminate all of its own on the diagnostic display board, thus achieving simultaneous cost reduction and feature addition.

A conventional 2K by 8 static transistor-transistor logic (TTL) RAM card is provided, along with a 12K by 8 erasable PROM (EPROM) using the Intel 8708 integrated circuits. The CPU card contains the standard Intel chip set of 8080A, 8228, and 8224, and it also holds a set of bus driver circuits. Eight levels of interrupt and a 25-ms sanity timer are provided on the interrupt control board. The sanity timer must be reset with an $\overline{\text{IOW}}$ command at least every 25 ms; if not, an interrupt is generated, and the software commands the CPU to halt. The software generated HALT command was chosen, instead of a direct hardware gating of the clocks, in order to allow flexibility. In the future it may be desirable to halt the CPU only after the second occurrence of timer runout or another algorithm. The present method should allow for this.

Memory-mapped I/O was chosen for the TPSS-1. Each I/O card has an on-board address comparator circuit, using two quad exclusive or integrated circuits, which enables that particular card when the present CPU address matches the wire-wrapped connector address. All positions are wired at the factory so that MMGT I/O cards can be added to the controller with no on-board switch options necessary. As a fail-safe feature, a separate address decoder card also

provides a unique enable lead to each I/O position. Thus both the on-board address comparator and the separate enable lead must be simultaneously active to operate the I/O card. Parity generation and checking is also done to further ensure proper operations.

The conventional means of remotely controlling a piece of transmission equipment is through a bank of relay contacts, which can be quite costly; the LPSS-3 has over 500 relay contacts. While such remote control and telemetry is not provided for now, TPSS-1 could be equipped with this capability quite easily and economically. New E2APR equipment has been designed to control the office relays; this equipment too has a microprocessor. It is possible to provide a direct, serial interface between the E2APR and the TPSS-1 microprocessors. From the TPSS-1 standpoint, such interface would reduce the hardware requirement from 500 relays (at a cost of over $10,000) to one board (at a cost of under $100), and in addition, it would allow much greater alarm interface capability and flexibility.

The controller design is a flexible unit that is only half the cost of a hard-wired logic design and yet allows a variety of additional features (see Figure 19-2).

SOFTWARE

The TPSS-1 program was written entirely in assembly language with the use of a macro assembler. A higher level language, such as PL/M, was not used because the program is I/O-intensive, and very little number crunching or data manipulation is involved. Within the limits of assembly language the program is structured through the free use of macros and subroutines. The program was conceptualized in a top-down manner, and written in a top-down and "do what's needed now, first" fashion. The program length is under 4K bytes.

EXPERIENCES

The remark was made that "Engineers can't program." The corollary is "Programmers can't engineer." Assuming one person cannot do the entire job, who should do the work on a large microprocessor system? In TPSS-1 an engineer was assigned overall responsibility, while an intern programmer was assigned to write the program. Assigning overall responsibility to the

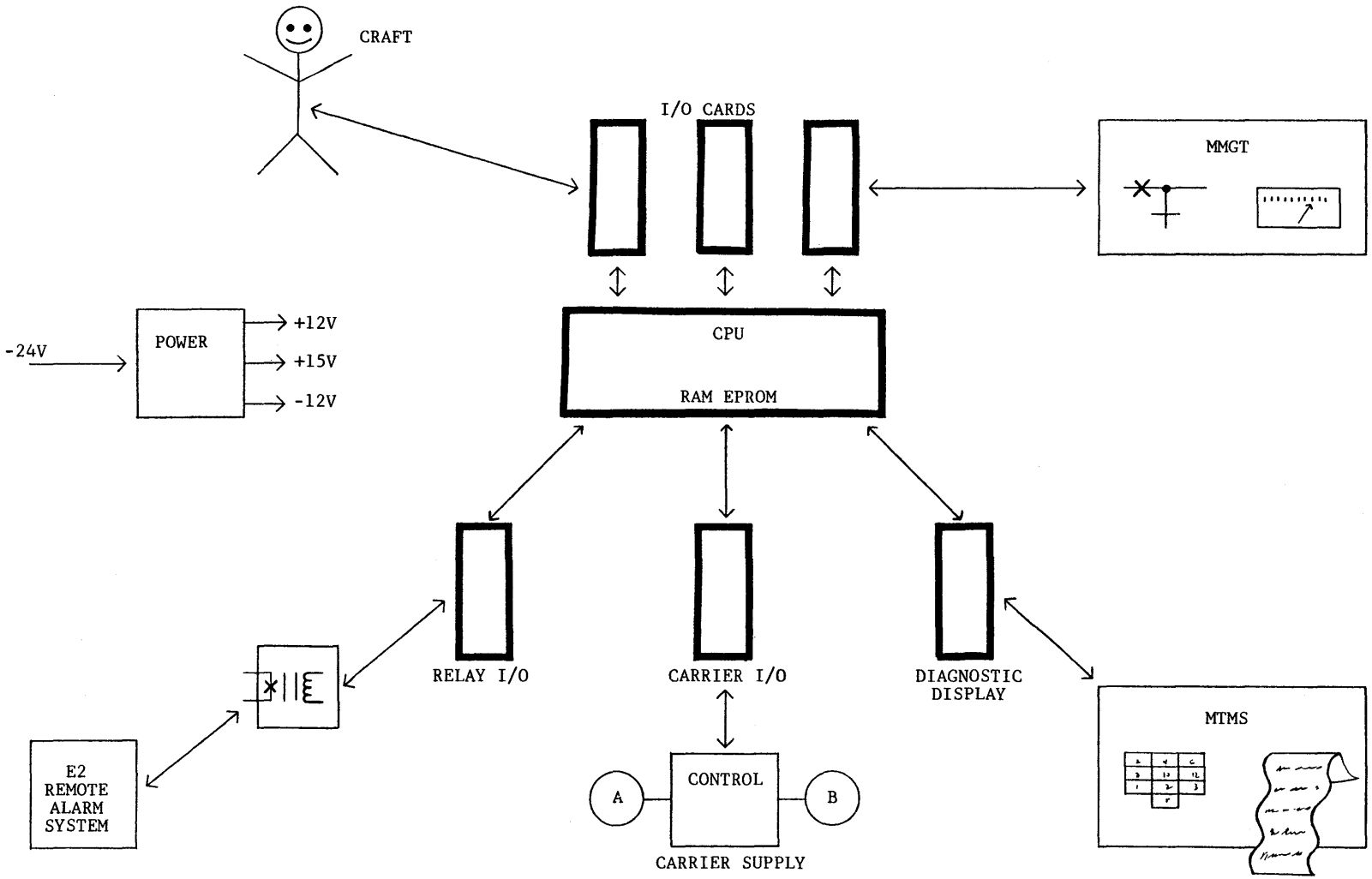


Figure 19-2 - TPSS-1 System

engineer implied that the engineer should be able to read and write programs and make comments; it also meant that the software was tailored to the hardware. This arrangement seems to have worked well. Because TPSS-1 is a piece of hardware and has definite manufacturing dates, the rule of hardware over software allowed the hardware to remain fairly fixed but forced the program to be flexible and nonoptimum. This hierarchy would not work as well with a system that has very simple hardware but involved programming.

The TPSS-1 development system consists of an MDS-800 with PROM programmer, cathode ray tube (CRT) terminal, diskette, acoustic coupler to the BTL Control Data Corporation computer, and an in-circuit emulator. Is this extravagant? We found that every penny spent on support hardware was justified. If you wait 45 minutes for a paper tape to load, you not only waste time but lose patience and sometimes even forget why you wanted to load the tape in the first place. A "luxury" system might cost $15K, but if you save months of debugging time or if you meet your Bell Laboratories design information (LDI) date, it is justified.

Microprocessor design is all that it is touted to be. The worker finds changes easy to incorporate, new features easy to add, and the finished product an all-around better design. After striving hard to convince management of the microprocessor's advantages, the designer's propaganda sometimes backfires. The chip cannot do everything, and the designer must resist the pressure from others to toss tasks into the processor's lap. The "since you're so smart, here, you do it!" attitude can force the processor beyond its capabilities. The engineer must admit that the processor is limited in its capabilities and keep the design empire at a manageable size.

While there is a new set of pitfalls and problems with every microprocessor design, many advantages and pleasurable new experiences await today's logic designer turned programmer. The TPSS-1 project was completed on time, with a low ulcer quotient, and is a better system than could have been produced with conventional logic circuitry.

APPLICATIONS II

THE 32A COMMUNICATION SYSTEM

W. N. Johnson, J. J. Shanley, J. H. Van Ornum,
T. L. Wang, BTL Dept 3231, HO, NJ

ABSTRACT

The 32A Communication System (32A CS) is a new electronic key telephone system for installations requiring up to 79 telephones. The system utilizes the new multibutton electronic telephones (METs), the Intel 8080A microprocessor, and a PNPN solid-state, space-division switching network. This paper highlights the hardware, software, maintenance, and administration aspects of the 32A CS.

INTRODUCTION

The 32A CS is a new electronic business communications system for installations requiring up to 79 telephones. It is a self-contained system using stored program control and, primarily, METs. TOUCH-TONE telephones are used entirely, with TOUCH-TONE-to-dial pulse conversion provided for rotary central offices (COs). A large complement of button-oriented features is provided beyond those available with existing key systems; most of these features are found in COM KEY,* PBX, centrex, and DIMENSION custom telephone services in varying degrees.

Processor-enhanced maintenance and administration techniques and a 4-pair wiring plan facilitate low labor involvement. An economical implementation is achieved by retaining the system software in read-only memories (ROMs).

As mentioned previously, many unique features are possible when METs, software-determined feature operation, and a solid-state, space-division network

* Trademark of AT&T.

are combined. For example, the new 5- and 10-button electronic telephones will accommodate multiline users through single-button access to outgoing facilities (e.g., CO/PBX, WATS, FX lines). Light-emitting diodes (LEDs) associated with these buttons indicate the busy conditions of the facilities, thus retaining the visual monitoring advantage of key service. Five-way unassisted conference calling with selective deletion of parties is a standard system feature. The conference is established with essentially no transmission loss. In addition, improved transmission level performance is provided on CO connections. Optional call coverage allows multibutton pickup service within a group on a single button per group per telephone basis. This feature can be optionally enhanced with busy, no-answer, and all-calls conditions redirected to a designated station.[1]

HARDWARE

The hardware of the 32A CS is composed of five basic parts:

- Processor.
- Memory complex.
- Switching network and associated control.
- MET input/output (I/O) interface.
- Individual network ports.

The "heart" of the system is the processor, which controls the 32A hardware in response to stimuli from the peripheral network. The processor fetches instructions and data from the memory complex, translates them into system commands, and transmits these commands to the network control unit and MET I/O interface. These two modules interpret the commands to provide switching and control functions for the various network ports.

The "brain" of the 32A processor is the Intel 8080A microprocessor. Most support functions for the microprocessor are handled by medium scale integration (MSI) devices from the Intel 8080 family in an effort to reduce the amount of integrated circuits. A memory-referenced I/O strategy is employed to save real time and memory space. Separate data buses are used for memory and I/O communication to solve loading problems and increase noise isolation. The priority-encoded interrupt scheme utilizes restart vectors to provide inputs for the real-time clock (10- and 25-ms interrupts for call processing) and the error logic that detects and reports memory failures. Independent hardware is added

to monitor the sanity of the processor-memory complex. A cycle timer checks that the 8080A instruction cycle is of the proper duration, while the software sanity timer verifies that the scheduled software routines are completed within a reasonable time interval. A failure in either of these sanity tests will cause the system to take corrective action.

The 32A memory complex contains the program and data segments that govern system functions. The program is stored in 16K-bit ROMs (although 8K programmable ROMs [PROMs] are being substituted for initial system introduction), while translation and status information is being stored in 1K-bit complementary metal oxide semiconductor random-access memories (CMOS RAMs). The RAM power supply is backed up by a battery to ensure nonvolatility of the translation information. The error detection and protection schemes used to enhance system reliability include even-parity checks, translation data write protection, memory-access boundary tests, and functional error checks (such as attempts to write into ROM or execute from RAM).

The solid-state, space-division network of the 32A CS consists of a matrix of PNPN crosspoint switches, with a maximum of 120 ports sharing 24 communication links. These links are used to establish connections between ports. Most control and maintenance functions are centralized on a common board that interfaces with the processor. The PNPN array, however, is distributed over the port boards, with each board containing its own connections to the 24 common links. This configuration allows the addition or deletion of ports according to network requirements.

The MET I/O hardware interfaces the parallel data stream of the processor with the serial data stream used to communicate with the METs. Hardware processing of the I/O data is used to relieve the processor of demands on its real time.

The network ports vary in type and function, yet all possess a universal interface with the network and the network control logic. Present port types include loop-start line circuits, MET station circuits, standard (2500 type) station circuits, special-purpose trunk circuits (FX, tie, etc.), auxiliary circuits for paging, and music-on-hold and tone plant TOUCH-TONE receivers.

SOFTWARE

The 32A software has been designed in structured program blocks. A structured macro assembly language (SMAL2)[2] supports a standardized program module format, permitting greater readability. This format includes:

- A prologue summarizing the module, including entry and exit parameters.
- Block comments in preference to line comments.
- A statement of entry and exit parameters at each call to a subroutine.

In addition, individual module listings have been limited (ideally to 2 to 3 pages) to facilitate understanding of the module. The average 32A object module contains 46 bytes (there are 1 to 3 bytes per 8080A instruction). Less than 100 bytes of program storage are required by 90 percent of the object modules. The average source module that produces these object modules contains 102 lines of comments and SMAL2 code.

Program structure for the 32A CS is similar to other electronic switching systems, such as the No. 2 ESS[3] and the DIMENSION PBX.[4, 5] An interrupt processor, which is activated every 10 ms, collects digits from the TOUCH-TONE receivers and controls rotary outpulsing. A second interrupt processor, activated every 25 ms, resets the task dispensing supervisor and sets a relinquish flag. This flag requests the return of control to the task dispenser when the interrupted process reaches a convenient termination point. The task dispenser sequentially passes control to the peripheral line and station scan routines, the line and station change handling routines, and, finally, to the base-level maintenance routines, if the relinquish flag has not been set in the meantime.

Time-critical events are detected via polling performed in the scan routines. These events are recorded in the appropriate change buffers. The change handling routines can then, at a more leisurely rate, poll and process any changes found in the change buffers. It is anticipated that the peripheral change rate will be significantly less than 1 change per 25 ms and that the change handling time will be about 10 ms per change.

MAINTENANCE AND ADMINISTRATION

The maintenance and administration objectives of the 32A CS include dependability, maintainability, ease of operation, low first-costs, and low annual maintenance costs. First-cost considerations lead to system constraints such as no redundancy, no resident off-line diagnostic programs, and a limited program allocation for maintenance and administration. Particular attention has been given to keeping the system operational during trouble periods or during change of customer services, and, in addition, to providing built-in fault locating procedures that decrease repair time.

Maintenance

Trouble detection is performed via high-level progress checks (software sanity checks), hardware tests, hardware-software consistency checks, audits, and error interrupts (hardware sanity checks). The progress checks are used to verify that the system is not stuck in a loop. Some of the hardware tests used for fault detection (network connection tests) are made during call processing; however, most of them are regularly scheduled during base-level time. The hardware-software consistency checks are used to update the hardware states to agree with the software records when transient faults occur. The software audits check for inconsistencies within the software records, and when transient errors occur, the associated call is aborted. It is left to the customer to place the call again.

Trouble reporting is accomplished via error counters, fault records, LED indicators, and alarms. The error counters contain peg counts of the various system errors that have been detected. The fault records contain a listing of the current hard faults and the previous five transient faults. The LEDs are processor controlled and identify the faulty circuit pack. A local alarm (LED) is displayed at the central answering position when any fault has been detected. The system provides for remoting alarms, and, in this case, all the information in the fault records and counters can be transmitted and displayed at a remote center.

Trouble locating is executed using the components of trouble detection and reporting, as well as by customer trouble reports and test calls. The test calls are used to reserve and set up facilities for manual testing in conjunction with trouble

verification and repair procedures. Trouble location is the circuit pack level in most instances, using a LED per circuit pack to identify the defective pack. This is possible because a complete function is contained on a single plug-in unit. For the cases where the trouble is not identified to a single circuit pack, fault records point to a craft procedure that locates the defective unit without resorting to circuit pack swapping.

Trouble recovery is achieved by aborting troubled calls and busying out the faulty facilities. The busied-out facilities are reexamined periodically. When errors are extensive, a cold-start initialization (which drops all calls) returns the system to normal operation. Finally, if all attempts fail, the system switches to power-failure-backup service.

Repair procedures, consisting of step-by-step details for repair and verification of system operation, are contained in two fault directories, one for alarmed troubles and the other for customer reported troubles that are not alarmed. Procedures are simple, compatible with today's craft procedures whenever possible, and minimize disturbing effects during the repair process.

Administration

The system is administered primarily through a portable service access unit (SAU) under control of software resident in the 32A. Its purpose is to read and write translation memory. Data are written either via a keyboard or tape. A 4-bit numeric display is provided for readout.

Translation changes can be made while the system is operational without affecting the other stations. These changes are performed locally by the customer or the craftsperson.

CONCLUSIONS

The first 32A CS will be placed in service at the end of 1976. It is the latest, and the smallest, member of the Bell System electronic switching systems. No. 1 ESS, the oldest member, is controlled by a maxicomputer; the DIMENSION PBX, the next to the youngest member, is controlled by a minicomputer. Now, in the 32 CS, the progress of electronic technology has allowed the use of the microcomputer as the controller.

ACKNOWLEDGMENTS

The Bell Laboratories 32A CS project has engaged more than 40 people. It is difficult to properly acknowledge the individual contributions of everyone involved; thus, acknowledgments are summarized by recognizing the following supervisory groups:

- M. R. Dungan and group - EMI measurements.
- C. E. Nahabedian and group - project control and support tools.
- C. R. Lindemulder and group - physical design.
- J. A. Miller and group - administration hardware and maintenance philosophy.
- D. C. Trimble and group - system hardware.
- C. D. Weiss and group - system software.

REFERENCES

1. 32A Communication System, DL Special Systems No. 682, Development Letter to AT&T, June 28, 1976.

2. D. H. Copp, User's Guide to the SMAL2 Language for the Intel 8080 Microprocessor, Memorandum for File, August 2, 1976.

3. R. J. Andrews, J. J. Driscoll, J. A. Herndon, P. C. Richards, and L. R. Roberts, "Service Features and Call Processing Plan," The Bell System Technical Journal, Vol 48, No. 8, Oct, 1969, pp 2713-2764.

4. Call Processing - Description, BSP 554-101-105.

5. General Software Description, BSP 554-102-100.

APPLICATIONS II

THE BUSINESS SWITCHING SYSTEM

R. M. Smith, BTL Dept 3233, HO, NJ

ABSTRACT

The Business Switching System (BSS) switches a maximum of 16 multibutton electronic telephones (METs) and 8 central office (CO) lines. An Intel 8080 microprocessor controls all BSS functions, including the switching network. It recognizes MET button depressions, detects line ringing, lights MET lamps, controls MET ringers, and implements the BSS feature set. The BSS employs switching circuits and techniques compatible with processor speeds and data accessing methods. As a result, the BSS can be packaged in an overnight-size suitcase for easy installation.

The BSS program is written in a structured macro assembly language (SMAL)[1, 2] for the Intel 8080 microprocessor. The BSS program structure mirrors the logical structure of call processing familiar to key telephone system designers. By calling various combinations of functional modules that control the physical facilities, a designer can implement different feature sets without changing the main body of code. This paper describes the functional hardware layout and the program structure for the BSS.

INTRODUCTION

The BSS furnishes key telephone service for installations requiring from 2 to 16 telephones and from 1 to 8 CO lines. The BSS uses METs, which receive and transmit serial data at high rates. See Figure 21-1. Direct station selection (DSS) is provided, which permits intrasystem calls. The DSS buttons are located on the upper faceplate. Station calls appear on intercom (I/C) buttons. LINE

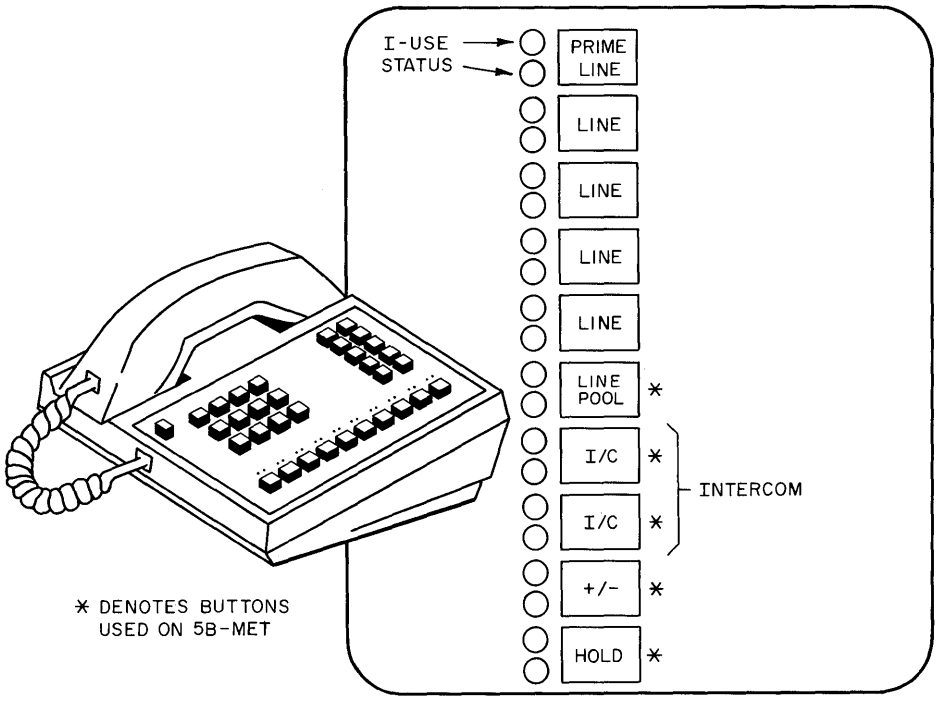


Figure 21-1 - 10-Button MET

buttons pick up CO lines. The +/- button adds or removes lines or stations from conferences.

An Intel 8080A processor controls the BSS in a standard processor configuration. An overnight-size suitcase houses all BSS electronics except the METs. The Intel 8080A microprocessor-based system, power supply, and mother board for station and line circuits form the floor of the suitcase. Figure 21-2 shows the physical arrangement of the printed circuit cards. The processor and its peripheral circuits fill the upper right-hand corner of the suitcase floor. The read-only memory (ROM) board and the random-access memory (RAM) board (located below the ROM board) plug into the processor board. Immediately below the memories, the power supply occupies the lower right-hand corner. The processor plugs into the edge of the backplane, which covers the left two-thirds of the floor space. Also, the station, line, and tone boards plug into the backplane.

The BSS schematic is shown in Figure 21-3. The audio signal enters the system at the tip and ring of the battery feed circuit, which feeds the monobus circuit.[3] The monobus circuit converts the audio signal to a current that flows along the station horizontal to the connected vertical and down the vertical through the ter-

minating resistor. Solid-state crosspoints connect verticals to the station and line horizontals. Idle lines and stations connect to the ground vertical, the quiet bus, to prevent monobus oscillation.

Data controlling the system flow through a 16-bit wide parallel data bus that extends from a latch on the processor board, along the backplane, to all circuit boards. MET serial data travel to all station boards on a 2-conductor bus and from all station boards on another 2-conductor bus. The 16-bit control bus selects the station to receive the serial data. Data from peripheral circuits return to the processor via an 8-bit data bus.

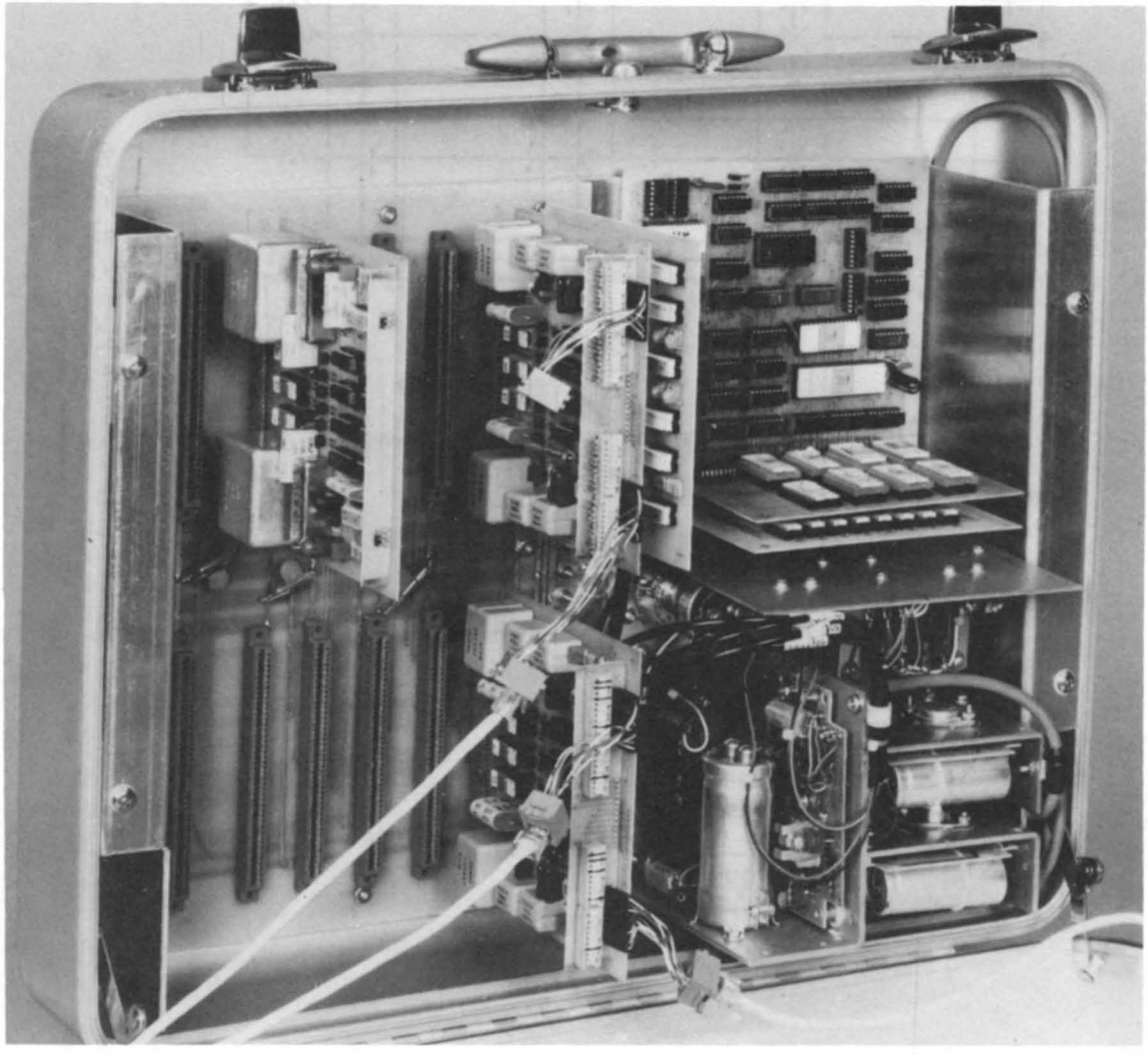


Figure 21-2 - Printed Circuit Card Arrangement

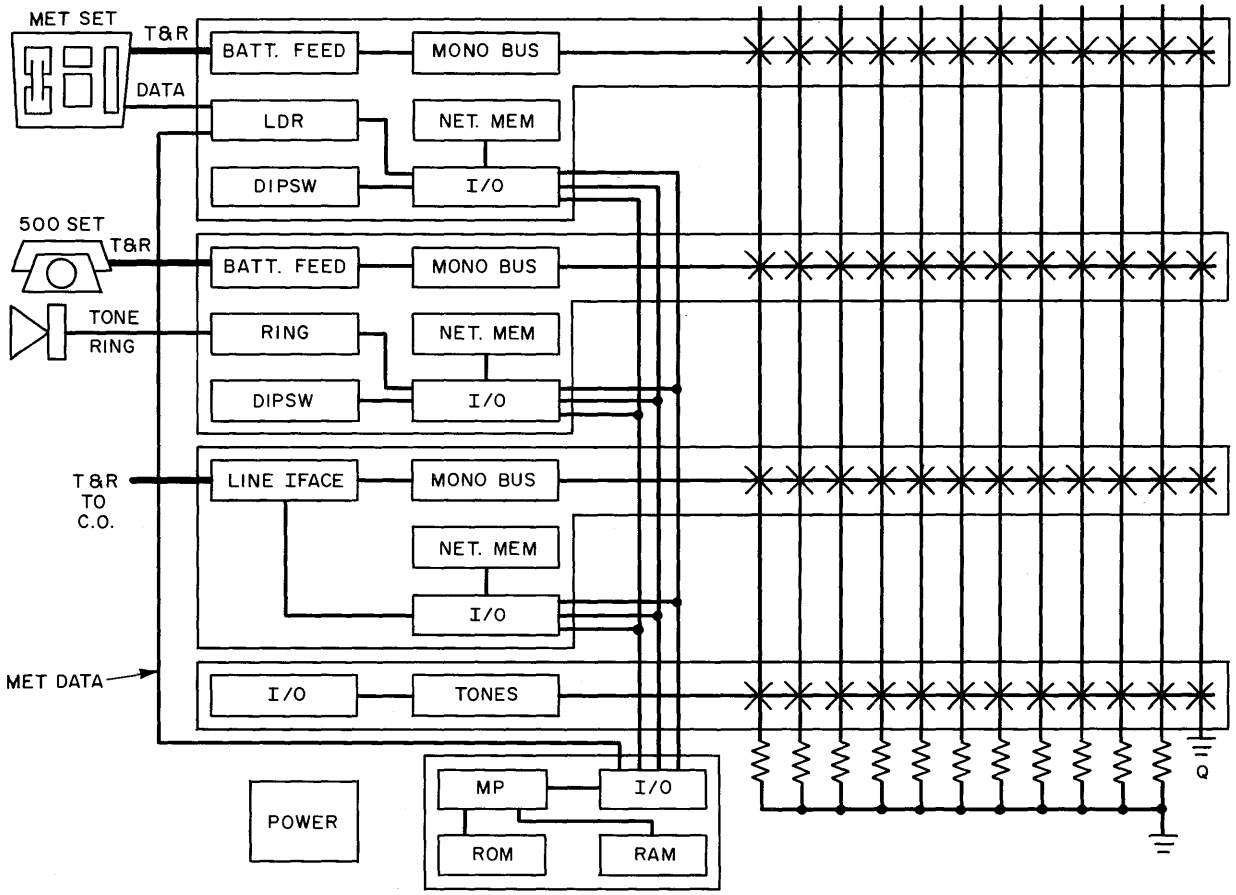


Figure 21-3 - BSS Schematic

One disadvantage of processor-driven customer systems has been the difficulty of entering station features, identities, and restrictions into the system memory. The BSS implements feature programming through switches located on the station boards. See Figure 21-2. The switches for each of two stations can be seen on each of the two station cards. The schematic layout of the switches can be seen in Figure 21-4. The buttons on a MET appear on the left; switch fields appear on the right. Typical switch settings, with the resulting line-button identifications, are represented by the line numbers next to the buttons and ON dots on the

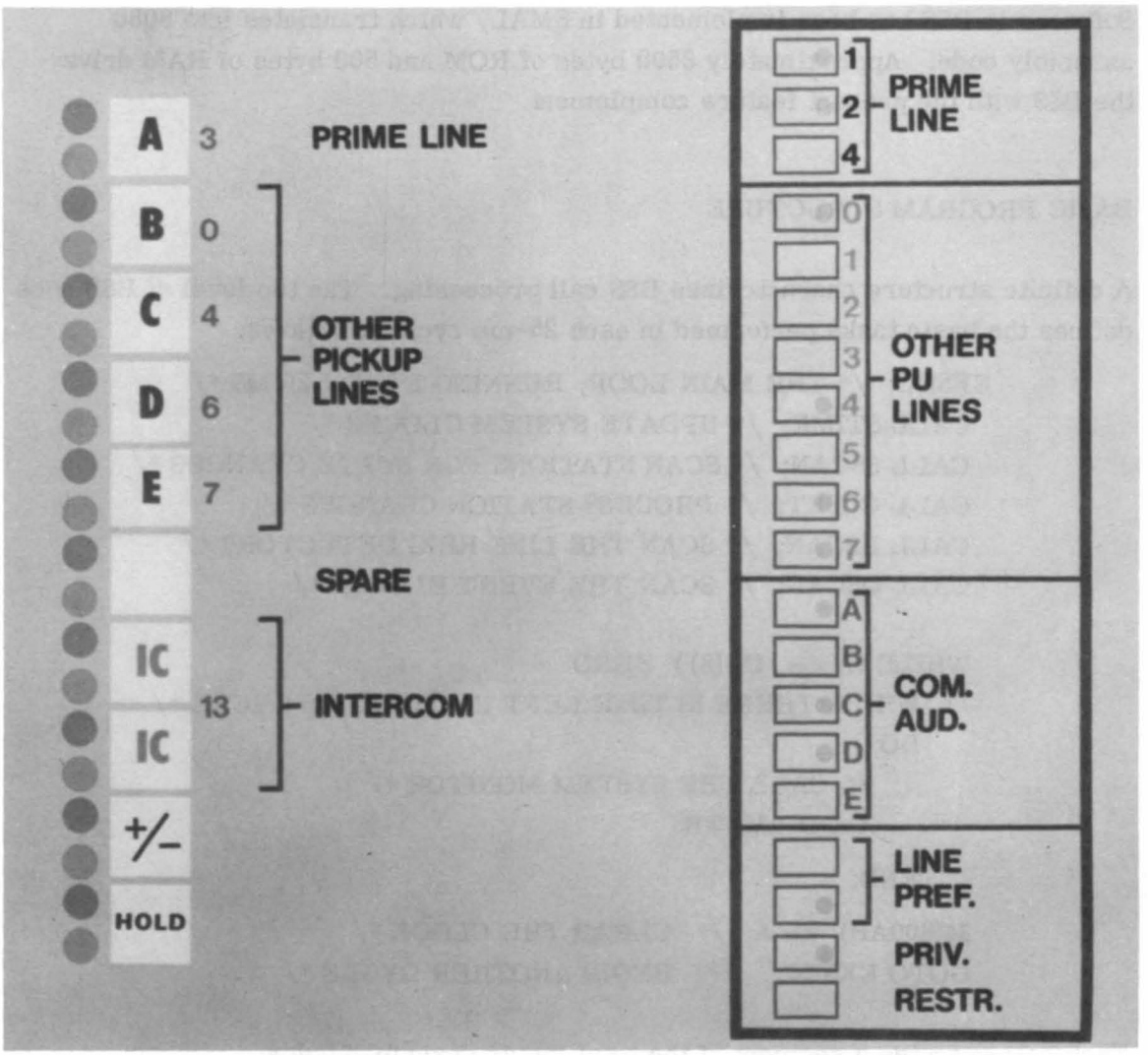


Figure 21-4 - 10-Button MET Translations

switches. Every 6 seconds the program reads these switches and updates tables that define station and line assignments and features. The slot into which a station is plugged defines the access number of that station. The feature switches and station and line board slot addressing provide a convenient way to administer the station and line assignments and features. No other memory-entry interface is needed.

SOFTWARE

Software in BSS has been implemented in SMAL, which translates into 8080 assembly code. Approximately 8500 bytes of ROM and 500 bytes of RAM drive the BSS with the present feature complement.

BASIC PROGRAM STRUCTURE

A definite structure characterizes BSS call processing. The top level of BSS code defines the basic tasks performed in each 25-ms cycle as follows:

```
EKSEC: /* THE MAIN LOOP, RUNNING EVERY 25 MS */
  CALL STIME; /* UPDATE SYSTEM CLOCKS */
  CALL SSCAN; /* SCAN STATIONS FOR STATE CHANGES */
  CALL CGSET; /* PROCESS STATION CHANGES */
  CALL LSCAN; /* SCAN THE LINE RING DETECTORS */
  CALL QSCAN; /* SCAN THE EVENT BLOCKS */

  WHILE (A <= IN (8)) ZERO
  /* WHILE THERE IS TIME LEFT IN THE 25 MS CYCLE */
     DO;
        /* CALL THE SYSTEM MONITOR */
        CALL IMNTR;

     END;
  M(800AH) <= A; /* CLEAR THE CLOCK */
  GOTO EKSEC;   /* BEGIN ANOTHER CYCLE */
```

Each call executes a segment of the total job as described below.

STIME exists as a subroutine. See Figure 21-5. The system variables indicated in the figure change with each 25-ms invocation of STIME.

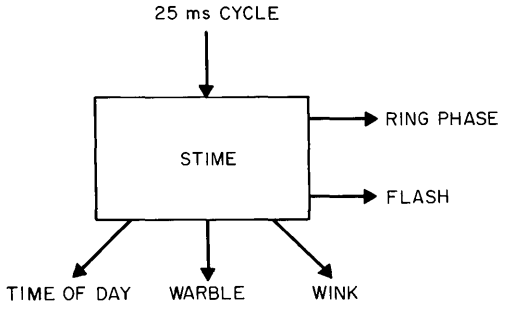


Figure 21-5 - System Timing

Station scanning is illustrated in Figure 21-6. The station scan transmits 25 bits to the station set and receives 15 bits from the station. Transmitted and received bit functions appear in Figure 21-7. The output of the station scan resides in the scan table (SCTAB) last-look words. The last-look words give the station state just before a station state change (for example, a button push, receiver off-hook, etc.) and just after a change. When a state change occurs on two successive scans, the station scan sets an activity bit, which signals an active change to CGSET.

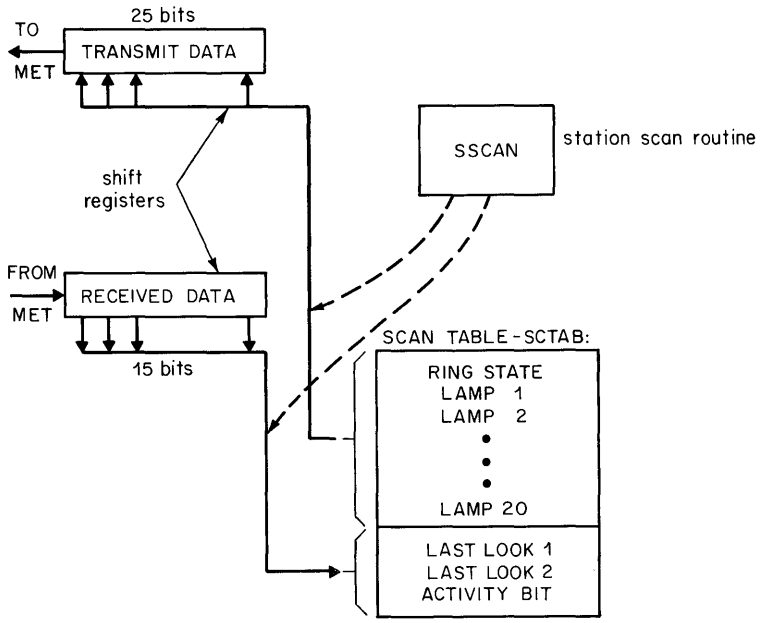


Figure 21-6 - Station Scan

	BIT	DATA TRANSMITTED TO MET SET: CONTROLS	DATA TRANSMITTED TO MET SET: STATES	BIT	DATA RECEIVED FROM MET SET: INDICATES	DATA RECEIVED FROM MET SET: STATES
HEADER	1	RINGER BIT 1	1 = OFF QUIESCENT; 0 = ON; 01 = OFF			
				1	SWITCH HOOK	0-ON HOOK 1-OFF HOOK
	2	RINGER BIT 2	0 = OFF QUIESCENT; 1 = ON; 01 = OFF			
				2	BUTTON DOWN	0-NO BUTTON DOWN 1-BUTTON DOWN
	3	RINGER TONE	0-750Hz 1-1500Hz			
				3	CONTROL (ACTIVITY)	0-NO ACTIVITY 1-ACTIVITY
	4	RINGER VOLUME	0-FULL 1-REDUCED			
				4	DSD OR DSS	0-BUTTON DATA 1-BITS 6-15 DIAL OR DSS INFO
	5	CONTROL (VOICE SIGNAL)	0-OFF 1-ON			
POSITION 1	6	STATUS LAMP 1	LAMPS 0-OFF 1-ON	5	RECALL	0-NO RECALL 1-RECALL
	7	I-USE LAMP 1				
POSITION 2	8	STATUS LAMP 2		6	BUTTON 1	BUTTON ACTIVITY: 0-NO BUTTON PRESSED 1-BUTTON PRESSED SEE NOTE 1
	9	I-USE LAMP 2	SEE NOTES 1 AND 2			
POSITION N	2N+4	STATUS LAMP N		7	BUTTON 2	
				N+5	BUTTON N	
	2N+5	I-USE LAMP N				

NOTES:

1) IF THE SET IS EQUIPPED WITH DSS, AND IF THE DSS BIT (BIT4) IS A ONE, THEN THE BUTTON BIT THAT IS A ONE IS TO BE INTERPRETED AS THE CORRESPONDING DSS BUTTON.

2) BD = 1 ON ALL BUTTON PUSHES (INCL DSS, RECALL)

3) BUTTON DN = 1 WITH DSS OR OTHER BUTTONS

Figure 21-7 - Data Formats

Following the station scan, CGSET scans the SCTAB for set activity bits that indicate a station change. See Figure 21-8. When CGSET finds a set activity bit, it resets the bit, exclusive ORs the new station state word (last look 2) with the old one, and calls the station change routine (SCHNG), which compares this state change word with a table of valid changes. When SCHNG matches the change with a valid change in its station branch table (SBTAB), the entry address to the correct call processor lies in the two bytes following the matched change word. SCHNG then transfers control to the call processing routine entry.

Three call processing modules perform most of the call processing; five call processing modules perform all call processing. If a LINE button is pushed, the line button module (or line processor module) receives the transfer from SCHNG. The DSS module receives intercom calls activated by the DSS buttons. The intercom module receives any intercom button indications. In addition, the off-hook and on-hook modules receive control on the off-hook and on-hook state changes. These two modules usually transfer control to the other modules, except when processing is trivial or not possible with the other modules.

The line button, DSS, and intercom modules have the same structure; therefore, only one, the line button module, will be described. See Figure 21-9. SCHNG

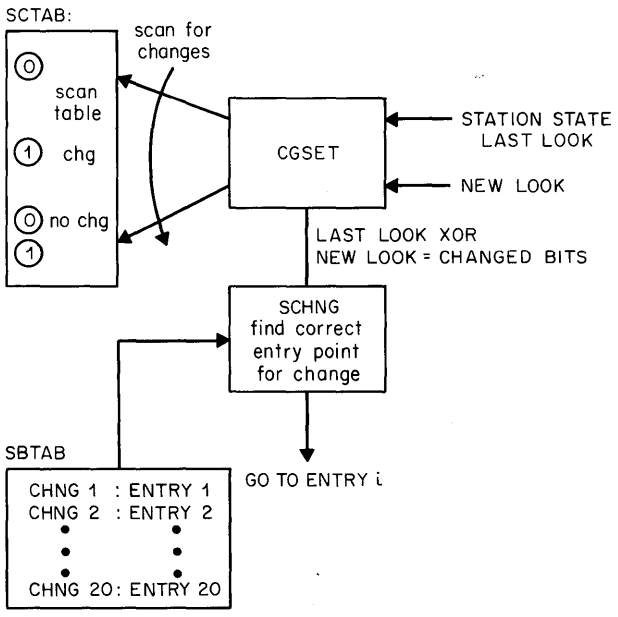


Figure 21-8 - Station Set Change

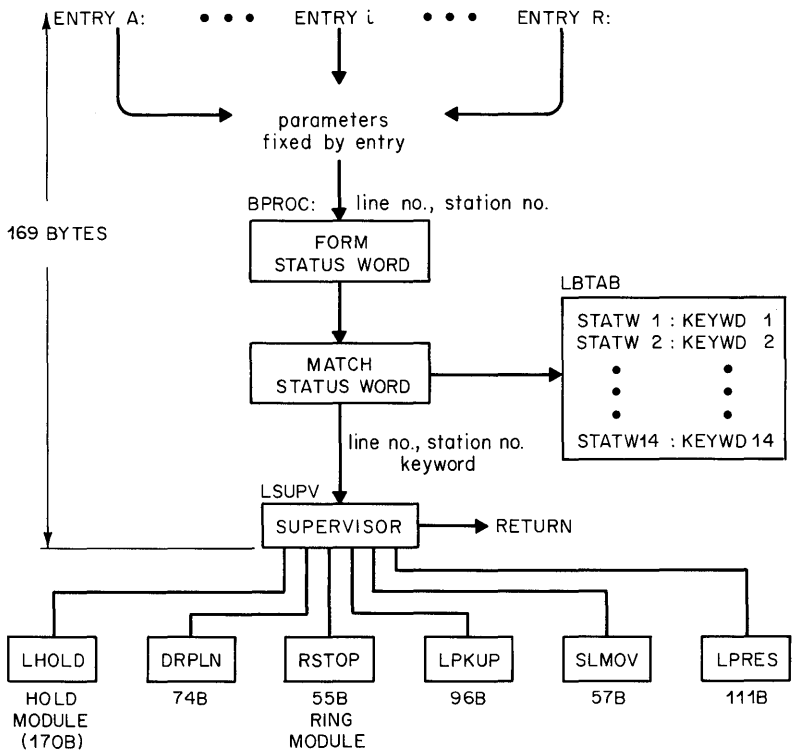


Figure 21-9 - LINE Button Push Schedule

transfers control to an entry (top of the figure). These entries fix the correct parameters in the B register. For example, if the prime line is line 3, the B register contains a 3 after the PRIME LINE button is pushed and the program enters the line button module. The B register contains this line number throughout this call processing sequence. Similarly, the C register contains the station number, the HL register pair contains table indexes, the D register contains subroutine indexes, and the A register passes values into and out of subroutines.

The station number (in register C) and the line number (in register B) allow formation of a status word that describes the station and line states. Figure 21-10 shows the only parameters necessary for call processing of a LINE button. Seven bits encode these states, which are compared to a table of all valid states in the line branch table (LBTAB). When a match of the actual state with an LBTAB state is found, the byte following the LBTAB state (the keyword) indicates the line primitives to be executed. Line primitives perform basic tasks in call processing; for example, DRPLN drops an active or ringing line, while LPKUP picks up an

inactive or held line. The required primitives are executed, and then control is returned to CGSET. CGSET restores all registers saved before calling SCHNG and continues scanning for activity bits.

The line scan routine (LSCAN) reads ring detectors on the line boards. See Figure 21-11. When ringing is detected on two successive scans, an event block is set, which checks the ring detectors in 0.3 second. If the line is still ringing, ringing of all stations on the common audible list of the line is initiated.

QSCAN schedules six events. Figure 21-12 describes these events and shows the flow of control. Basically, an active event block contains a time of day when one of the six routines (also encoded in the block) must be executed. When the time

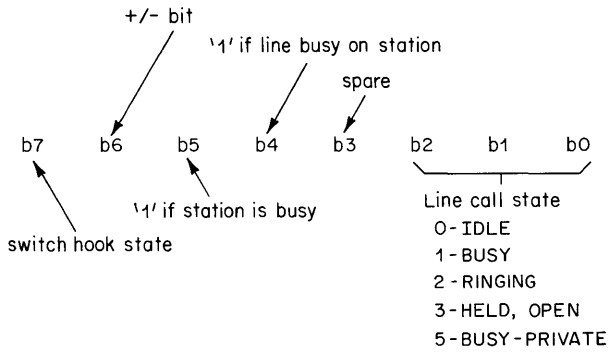


Figure 21-10 - LINE Button Push Status Word

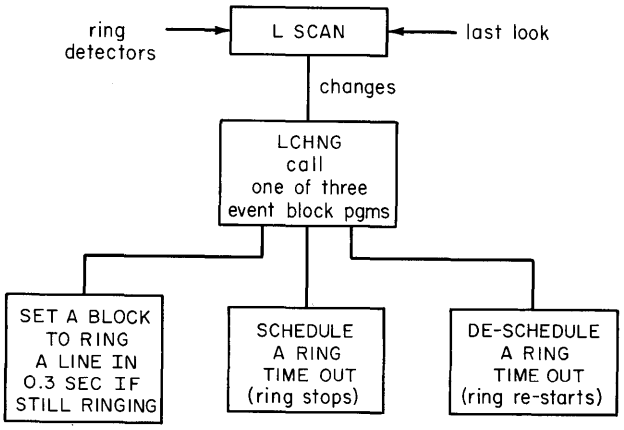


Figure 21-11 - Line Scanning

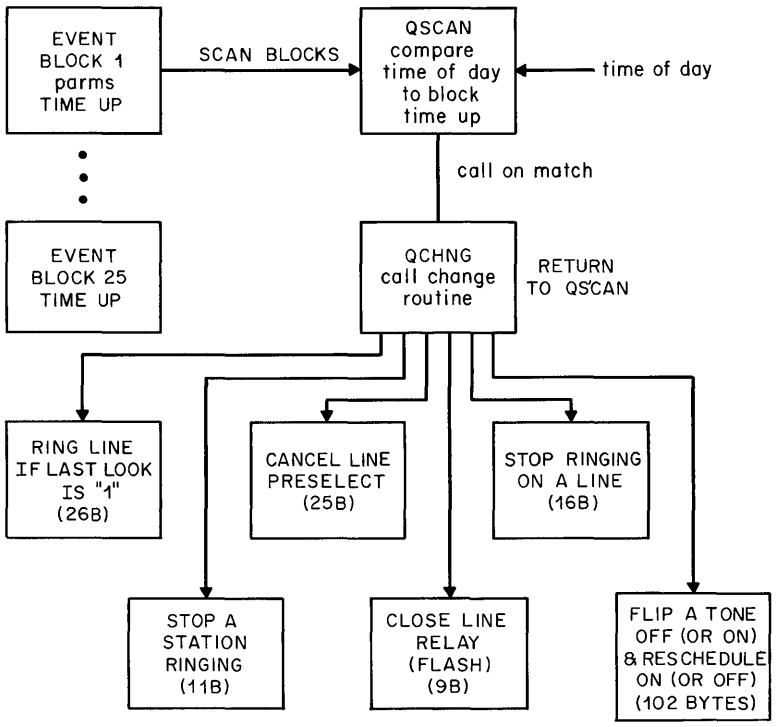


Figure 21-12 - Event Blocks

of day (from STIME) matches the scheduled time, the queue change routine (QCHNG) executes the required program. SKED activates an event block and DSKED deactivates one.

BASIC TABLES

BSS data structures center around the six basic tables shown in the Figure 21-13. All of these tables are written and read by buffering routines called with parameters in the D register, lines (or called stations) in the B register, active stations in the C register, and data written or read in the A register. Beginning at the top of Figure 21-13, LITES and RNGST set all lamps and ringers in the SCTAB. The station scan translates this table, with system clocks, into lamp and ringer station data. Station definition, station type, line assignments, line preference, etc., reside in STDEF. The station definition routine (SDEF) reads this table, and the switches input routine (SWIN) writes this table every 6 seconds. The auxiliary line of definition table (LNDEF) defines the common audible list for each line, and is composed in SWIN from station board switch inputs. The station

data table (SDAT) contains all station dynamic data, such as call state, vertical number, button states, etc. Initially, SDAT contains all 0s. The SDATA routine writes SDAT, and the SINFO routine reads SDAT. The STDEF, LNDEF, and SDAT tables contain the program-station interface.

Line data reside in the line data table (LDAT). The LDATA routine writes LDAT, and the LINFO routine reads LDAT. When LDAT is written, LDATA automatically calls the routine that sets the line relay. LSTAT, which contains the ringer status, is written by LSCAN. The LDAT and LSTAT tables define the line-program interface.

The switching network or vertical information resides in VDAT. The VDATA routine writes VDAT, and the VINFO routine reads VDAT. When VDATA writes VDAT, indicating a horizontal connection (i.e., a line or station to a vertical), the network setting routine is called automatically by VDATA and the network connections are established as dictated by the written data in VDAT.

Call processing programs interface with hardware through table buffers only. Thus, the call processing programmer must keep tables up to date. This practice minimizes programming errors and ensures system consistency. In addition, if the hardware, for example, the switching network, is ever changed, only the interfacing routines need be changed; call processing remains the same.

The BSS supports most desirable key telephone features, including a 30-button attendant station, in approximately 8500 bytes of ROM code. This system has been performing for approximately 8 months as a test vehicle with an in-circuit emulator support system and as a demonstrator with a version of the program in the programmable ROM (PROM). Microprocessor speeds adequately serve the timing requirements.

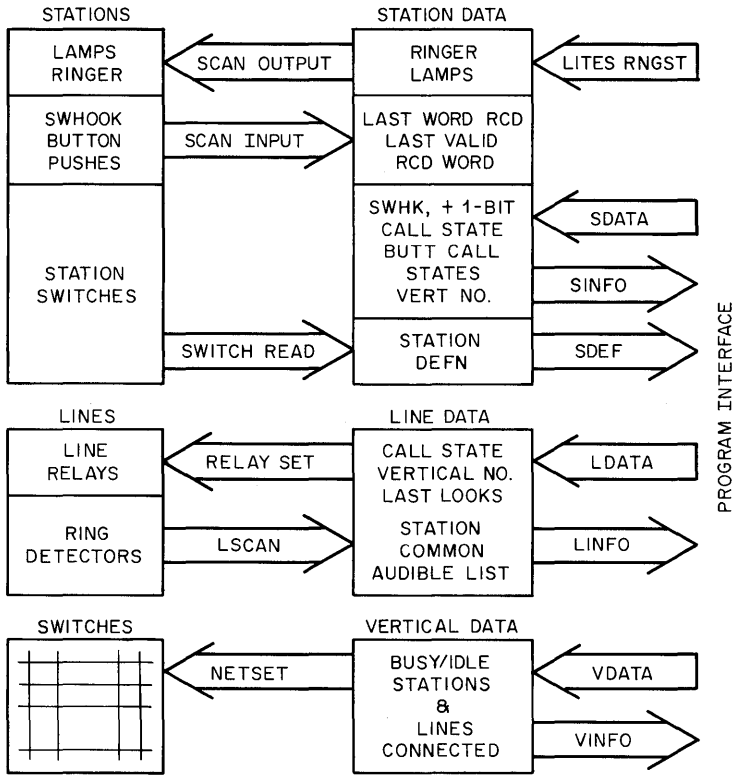


Figure 21-13 - System Data Buffering

REFERENCES

1. C. Popper, SMAL - A Structured Macro-Assembly Language for a Microprocessor, TM-74-3233-3, August 1, 1974.

2. C. Popper, SMAL User's Manual, TM-74-3233-5, October 24, 1974.

3. C. K. Liu, Network Circuit and Logic Control of the EKSII Laboratory Model, Memorandum for File, Case 40128, April 18, 1976

APPLICATIONS II

A PROCON-BASED PERIPHERAL CONTROLLER

J. L. Haase, R. J. Jakubek, S. M. Silverstein, BTL Dept 3234, HO, NJ

ABSTRACT

This paper describes the architectural considerations for the control of customer premises peripheral equipment used in the Phase II offering of the automatic call distributor/electronic switching system (ACD/ESS) service. The implications of connecting a microcomputer, similar to the Western Electric (WE) PROCON, to a No. 1 ESS, via a data link, to control and monitor a variety of complex peripherals are considered. This is a high-throughput, real-time application where reliability and maintenance are of prime importance. The system design is presented with an attempt to highlight those concerns involved in achieving a proper balance between software and hardware implementation alternatives.

INTRODUCTION

The flexibility of stored program controlled central offices makes it possible to provide the business customer with many new features and services.[1] One of the newest to be offered from the No. 1 ESS is ACD/ESS service. The basic function of the ACD/ESS is to distribute incoming calls to groups or splits of agents, with the distribution determined by factors such as call origin, call type, order of arrival, etc. The 60B Customer Premises System (CPS), with No. 1 ESS, provides this service to the business customer.

The 60B CPS contains a subsystem designated the remote data interface (RDI). The subsystem controls console lamps, alphanumeric displays, a cathode ray

tube (CRT), and other peripheral circuits, via a programmable controller (PROCON) based on information transmitted over the centrex data link between the ESS and the customer's premises. The control information received from ESS by PROCON is decoded, processed, and communicated to the peripherals by channels with serial data links.

Translation capability is available within the microprocessor random-access memory (RAM) to allow the definition of agent splits. With this capability, a single data link message can be used to control groups of lamps and displays defined via these translations.

Information gathered by the microprocessor and transmitted to the ESS central office includes console key depressions, CRT keyboard activity, and station equipment status.

SYSTEM ARCHITECTURE

In order to provide the required data multiplexing and demultiplexing between the customer premises peripheral units and the ESS under program control, the architecture shown in Figure 22-1 has been adopted. The ESS data link is connected through a special modulator-demodulator (modem) to a data buffer (DBFR). The data buffer is an independently operating unit that provides message buffering in both directions. It also contains fault indicators and some logic, which, under ESS command, controls the microprocessor (run, reset, and halt operations). An 8K- by 24-bit read-only memory (ROM) program storage unit (PSU) contains all the system programs and is interfaced with the instruction bus. The rest of the microprocessor complex is interfaced with the system bus. This bus contains a 4K- by 16-bit RAM used to hold translation and status information and four input/output (I/O) channels that provide serial-parallel conversion and data link control functions. The channels drive steering units that multiplex the channel data to and from 32 data links that control consoles and lamp displays directly. Also included on the bus is a clock that records real time in 1-μs steps. This clock is used to schedule processor tasks. Finally, there is a display for diagnostic results. Some customer equipment requires special data link interface circuits; for example, the scanner-distribution unit is used to interface with circuits that require parallel data, while an EIA interface card is provided for the CRT terminal.

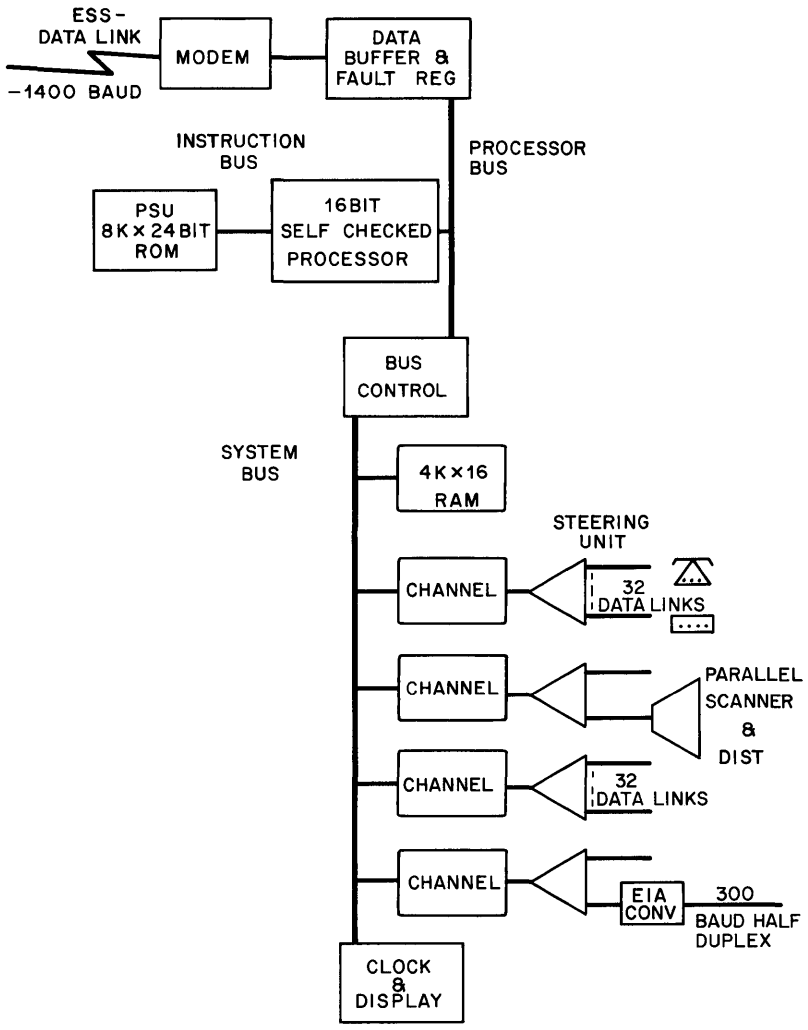


Figure 22-1 - RDI Hardware Configuration

A well thought-out architecture should show consideration of flexibility, reliability, maintainability, throughput, and cost factors, with the right tradeoffs being made to yield a good overall system design. Flexibility is achieved largely through the inherent features of a microprocessor by placing as many functions as possible in the software without adversely affecting the other factors. Maintainability is obtained by separating the processor bus from the system bus to yield more reliable communication between the ESS and PROCON. Here, reliability is important for maintainability, since the PROCON, upon command from the ESS, is responsible for diagnosing and isolating faults for everything located on the system bus. A fault in a unit on the system bus, or of the bus itself, will not prevent the

ESS from communicating with the PROCON. Good fault isolation is obtained by providing a software diagnostic routine for each major unit on the system bus. These include: the system bus, RAM, channels, and real-time clock. The result of each diagnostic is a pass-fail on the unit or circuit pack in question. Thus, the only real cost incurred in accomplishing good maintainability is the added expense to enable the ROM to store the diagnostics.

The choice of PROCON as the processor can be attributed to three considerations: speed, maintainability, and in-house availability. Since the application is one of real-time processing, the speed of PROCON results in increased throughput over other processors. The problem of diagnosing the processor for proper operation is accomplished through the self-checking features of PROCON, resulting in an easily maintained unit.

Perhaps the most important tradeoff affecting cost and throughput is the decision to delegate a function to either hardware or software. In this application, a very time-consuming task is the exchange of button and lamp information between the RAM and the numerous consoles. If this is left completely to the software, the throughput of the system is vastly reduced, resulting in a much smaller number of consoles per RDI, therefore increasing greatly the cost of ACD service. By delegating this job to the semiautonomous channels, throughput increases, allowing more consoles per RDI. Even though additional cost is incurred per RDI by adding the channel hardware, it results in a lower cost per console on a system basis.

SOFTWARE ARCHITECTURE

The software is responsible for providing lamp and alphanumeric control upon ESS command, driving the special interfaces, and detecting and reporting status changes (i.e., button operations) to the ESS. Related to these tasks are such functions as translation management, ESS message processing, data link I/O, timing, and maintenance. The strategy used to implement these functions is periodic data link service under program control. To support this structure, a special-purpose, real-time operating system has been developed for the microprocessor. Figures 22-2 and 22-3 depict a flow diagram of the system. The system can be viewed as a finite state machine. The states are quescient (Q), echo (E), maintenance (M), and active (A).

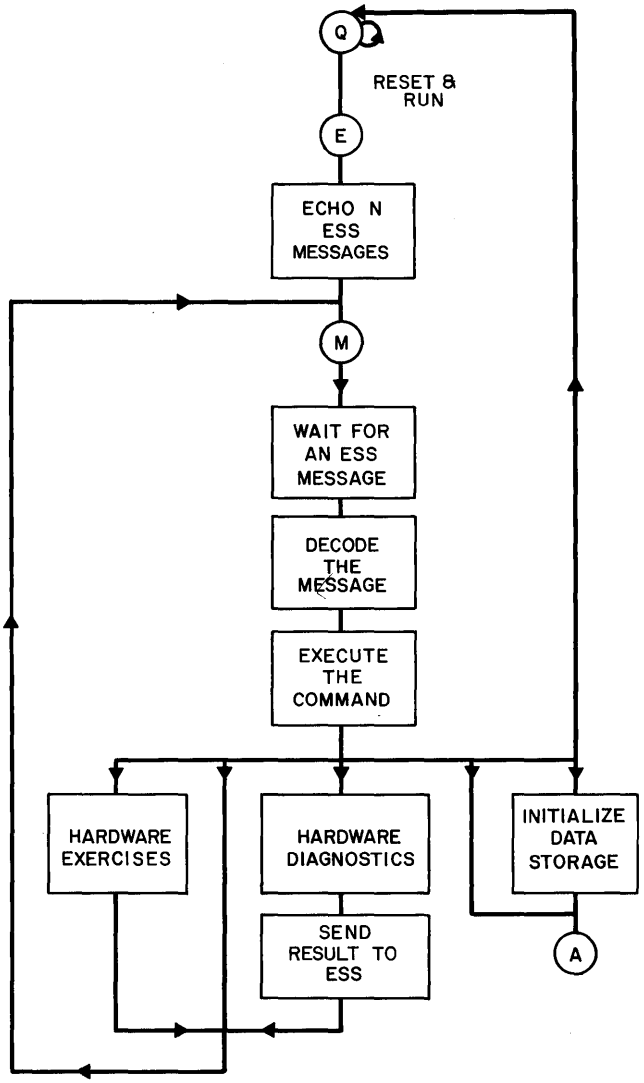


Figure 22-2 - RDI Software Flow Diagram

In the Q state, the processor is idle and can be stopped or reset by ESS command through the DBFR. Upon the reset and run command, the processor goes into the E state. Here, ESS messages are echoed by the software to check the communication link and the sanity of the processor through its self-checking capabilities. After a fixed number of messages, the processor switches into the M state.

In the M state, the processor waits for ESS messages. Upon receipt of a message, the processor decodes it and executes the order, if it is accepted in the M state. These messages include diagnostics (bus, RAM, clock, and I/O channels),

exercises (processor self-checking and system fault detection), and state transition messages (go quiescent, go active, and initialize and go active).

In the A state, all call processing tasks are executed on a cyclic basis. From here the processor can be commanded to reenter the M or the Q state. Thus, typical operation is to bring the processor into the E state by a reset and run order to check the link and the processor. Then, it enters the M state and the surrounding hardware is checked. If the results are reasonable, the ESS issues an initialize and go active command. The processor clears the data base and is

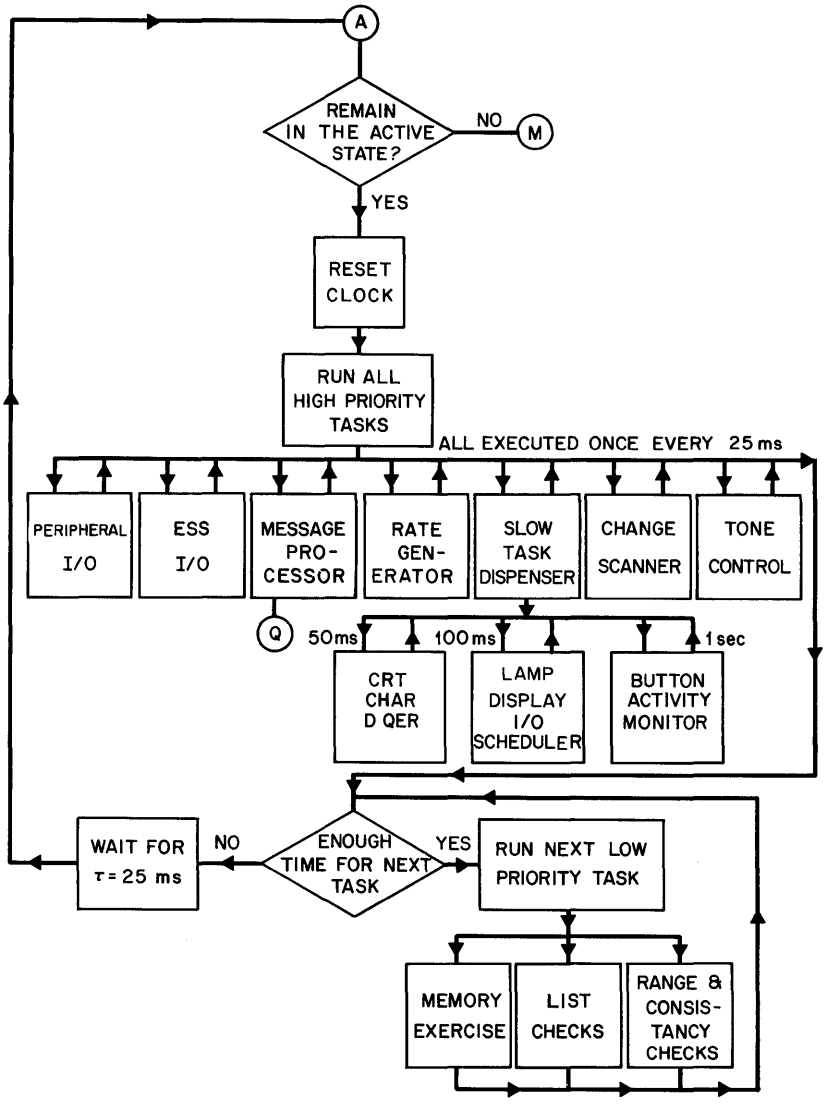


Figure 22-3 - RDI Software Flow Diagram

ready to begin normal call processing. Later, ESS can order the system back into the M state and run diagnostics and exercises to verify operation. After this, the system can be restored to the A state, leaving the translation intact (using the go active order) so call processing can resume.

In the A state, the task dispenser software schedules tasks at two priorities: P1, tasks whose periodic execution is guaranteed, and P0, tasks that are executed on a time-permitting, round-robin basis. The dispatcher schedules six tasks at a 25-ms rate, one at a 50-ms rate, one at a 100-ms rate, and one at a 1-second rate. The tasks run on level P0 are nonessential, such as audits and RAM exercises.

The first task run in an active cycle (peripheral I/O) experiences the least time jitter. It performs data link I/O exchange operations with customer premises peripherals, based upon entries in channel lists. These entries direct the program to control words and a set of data buffers on a per-peripheral basis. The program must know the state of the peripheral so it can select the proper I/O operation. One of the features of this module is that the processor handles multiple channels simultaneously, with I/O overlapped between channels. While some channels are transferring data, the processor services another channel to increase throughput.

A central office data link driver (ESS I/O) is incorporated in the system. It samples the DBFR for messages and buffers any orders for the message processor. It also maintains a 3-level priority queuing structure of messages to be sent to the central office.

The message processor decodes orders from the central office and executes them by updating data entries in the RAM. Messages are of three classes: translation, call processing, and system status.

The rate generator maintains two software clocks and updates lamp bits in output buffers as specified in the system rate lists. The lists are updated by the message processor and the tone control routine.

The change scanner scans the buffers returned from consoles and reports filtered button operations to the ESS via one of the queues. It also provides alphanumeric control and lamp test features. For the EIA data terminal, it provides

adaptive error checking and recovery and maintains character input queues and editing facilities.

The tone control program provides software timers, on a per-console basis, to control the duration of audible signals at the consoles.

The CRT character dequeuer moves characters from a CRT queue to the output buffer for the EIA interface unit at a 50-ms rate. The message processor, upon ESS command, places characters on the queue, while peripheral I/O actually sends the characters to the terminals.

The lamp display I/O scheduler schedules I/O operations for lamp displays at a 100-ms rate by updating the I/O status in the control words used by the peripheral I/O program for lamp displays.

The button activity monitor program prevents console activity from tying up the microprocessor, the data link, and the ESS.

As can be seen from the preceding discussion, structured programming techniques and modular design result in cooperating tasks implementing system features. Also, each task only utilizes a subset of the system resources. For example, the message processor works only with certain data structures, while the peripheral I/O program accesses only I/O buffers and control words in RAM and the channel hardware. These approaches eased the software implementation and debugging stages of the development.

ADDITIONAL MAINTENANCE CONSIDERATIONS

A combination of hardware and software components is used to achieve a 4-faceted maintenance plan. The strategy consists of automatic fault detection, automatic fault isolation, fault indications to the central office and the customer premises, and automatic correction of soft failures.

The ESS communication is checked by data link parity, an all-seems-well signal, and a modem loop-around state. PROCON has multiple parity bits to check the integrity of the PSU. The microprocessor also performs an instruction sequencing check, via other bits in the PSU, and data matching checks via duplicated data manipulation units.

The global behavior of the system is monitored by periodic communication checks from the central office and by the fault detection circuitry of the DBFR (time out, all seems well, link blockage, etc.).

The system also runs auditing programs on its RAM data base to minimize errors. These audits include translation consistency checks, pointer and index range checks, list transversal checks, and traveling block memory exercises. The software automatically recovers its work cycle on most soft failures.

UTILITY SUPPORT

The use of proper support tools that facilitate efficient software development is of paramount importance in the development of a microprocessor-based system. A good support package provides an easy method of source code generation and administration as well as facilitating the assembly and loading of object modules. A mechanism for providing tracing, address halts, register access, and code disassembly facilitates the debugging of programs. A support package called PDT (PROCON development tools), based upon the UNIX operating system and SWAP assembler, has been developed specifically for use with PROCON. This system served as the foundation for the design of software aides to develop the Intel 8080 code for the 32A Communications System project.[2]

CONCLUSIONS

In reviewing this paper, several concepts are worth emphasizing. First, the concept of providing centralized features by multiplexing information over data links is very effective. Second, stored program control and the microprocessor concept form an excellent and flexible basis for providing a wide spectrum of features to a variety of peripherals. Finally, hardware-software tradeoffs are fundamental to the efficient design of microprocessor-based real-time systems.

REFERENCES

1. R. J. Jakubek and S. M. Silverstein, "Microprocessor Control of Customer Premises Telecommunications Equipment," Proceedings of the Annual Conference of the ACM, Houston, Texas, October, 1976.

2. D. J. Hunsberger and J. J. Molinelli, "A Support System for the Intel 8080," paper presented at Microcomputer Symposium, Bell Telephone Laboratories, Holmdel, New Jersey, 1976.

APPLICATIONS II

A MICROCOMPUTER-CONTROLLED SINGLE-LINE TELEPHONE SYSTEM

S. W. Lye, BTL Dept 3313, IN, IN

ABSTRACT

A Texas Instruments TMS-1099 microcomputer has been used as the controller of a single-line key telephone system offering pickup, hold, illumination, tone ringing, intercom, and voice signaling via the 4-conductor wiring common to most residences. The flexibility of the microcomputer was fully exploited by simultaneous system design, program development, and preparation of the hardware for a market trial.

The single-chip microcomputer is an attractive alternative to custom large scale integration (LSI) for small systems having anticipated feature changes or options and relatively low production volumes. Of prime concern in the selection of such a device for production design are the power requirements and the circuits necessary to interface the computer chip to the real world.

INTRODUCTION

In a computer age dominated by news about 8080 and 6800 systems offering increased speed, expanded architectures, and complex resident operating system software, one must seek out application data on the smaller 4-bit microcomputers which actually introduced this new technology. While there has been a reduction in the cost and the complexity of the 8- and 16-bit systems, it is the single-package machine that can now compete with custom logic in control applications within automobiles, appliances, and Bell System station apparatus.

Considerable effort on the part of the semiconductor industry has been directed towards this potential market where efficient, economic, single-package controllers are expected to find large market volumes. Though few in number now, many 4-bit microcomputers will reach the commercial market in 1977.

The first application of the single-chip microcomputer technology in Bell System station apparatus is in the single-line telephone system (SLS). This system evolved as the vehicle for an AT&T market trial of single-line intrapremise communication service for residences and small businesses. The system provides pickup, hold, illumination, tone ringing, intercom, and voice signaling features comparable to those of some business systems. The system, as shown in Figure 23-1, consists of a locally-powered control unit mounted at the protector block and a speaker adjunct associated with each telephone set. Also, special one-piece telephone sets incorporating the speaker and controls within a modern housing have been designed for use with the SLS system. The system is interconnected with standard 4-conductor inside wire arranged in any home-run or daisy-chain scheme.

The challenge in designing a small key telephone system of this type is to make it economical yet compatible with general purpose telephone sets and with the 4-conductor wiring. Economics demand that all functions be performed by the control unit rather than be duplicated in each of the attached telephones or adjuncts. Since two of the four wires are dedicated to the normal speech transmission mode, the additional key button, lamp, ringing, and paging signals must be carefully distributed on the remaining two.

THE SLS SWITCHING SYSTEM

A magnified view of the switching portion of the control unit is shown in Figure 23-2. The system consists of a loop current sensor, a hold bridge, an intercom current source, and an intercom transfer relay. These elements allow the connection of all telephones to either the central office or the intercom line while holding the central office line or monitoring it for an incoming call. The magnitude of the current drawn from the intercom talk battery is monitored for local supervisory purposes, i.e., it is used to select and release the paging system from an intercom connection. Voice and tone signals are selectively mixed by an analog switch before amplification and distribution to the system speakers.

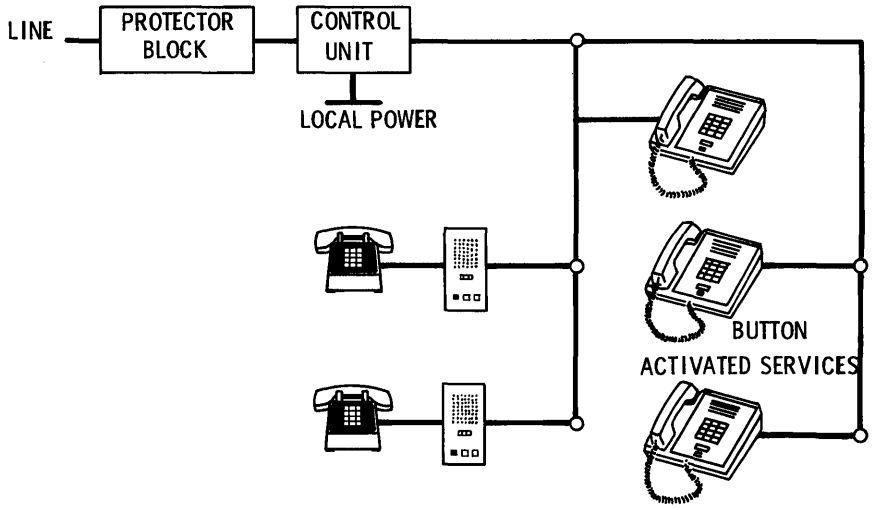


Figure 23-1 - SLS Block Diagram

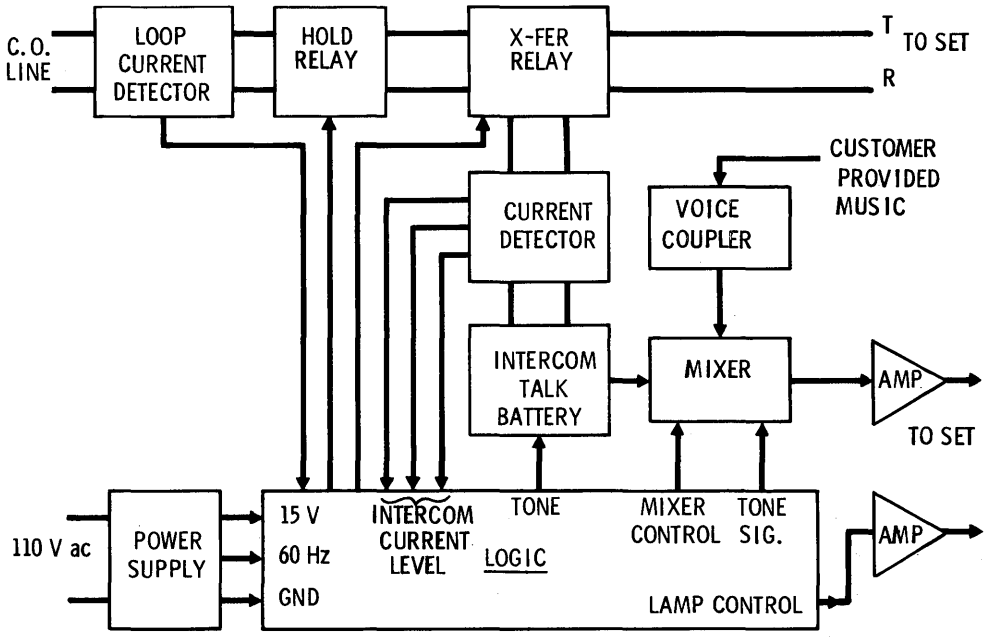


Figure 23-2 - SLS Controller Block Diagram

Also connected to the speaker distribution system is an amplifier identical to the paging amplifier but employed to switch the station light emitting diode (LED) status indicators. The slew rate of the lamp amplifier is controlled to allow the LEDs to flash without generating audible clicks in the speaker system.

An adjunct circuit is shown in Figure 23-3. The circuit contains a diode-coupled LED, a speaker with a bipolar blocking capacitor, a volume control and muting contact, and momentary customer input switches. The hold and intercom switches break and short the tip and ring conductors of the talking circuit, thus superimposing the system control signals on top of the central office supervisory currents.

When idle, the switching system connects the station sets to the intercom battery and the ac line termination to the central office line. The controller then monitors the central office line for ringing voltage and the station sets for either an off-hook current (30 mA) or an intercom selection current (80 mA). An intercom call is originated by a momentary operation of the intercom key. The control system responds by lighting the LEDs for a 3-second period. Lifting the handset during the 3-second period causes the controller to connect a 0.5-second signaling tone (750 Hz) to the paging amplifier. A paging message may be initiated after this alerting tone. The paging amplifier is disconnected from the speakers if a second station answers the paging call, or it is connected to a 20-Hz modulated tone if ringing is detected on the central office line.

The central office line hold function is somewhat more critical than the intercom sequence described above. With a single off-hook phone connected through the loop current sensor to the central office, the controller has only one active input bit - the presence or absence of current in the central office loop. The hold button (like the rotary dial pulsing contact, the switchhook, and the central office switching equipment) generates an interruption in loop current when operated. It is the task of the controller to measure the duration of all central office line open intervals and determine the source of those exceeding 100 ms. This is accomplished by:

- Breaking the loop into two parts at the control unit.
- Testing for current in the central office and the station set circuits.

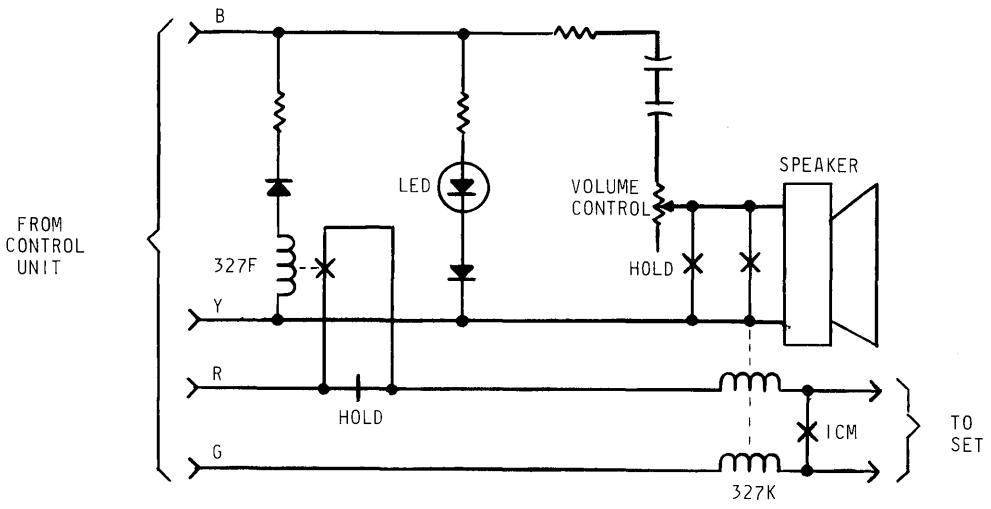


Figure 23-3 - SLS Adjunct Schematic

- Attempting to reestablish station current by reclosing the hold button circuit through a relay operated by a momentary reversal of the lamp signal polarity.

The return of station current within a selected interval following this sequence is sufficient to assure a valid hold request. While the validity check is being performed, the line remains terminated by the hold bridge, and the controller returns a steady signal tone to the handset of the user generating the hold request. When assured that a hold condition exists, the controller monitors the intercom current for a hangup on hold, an intercom-paging request, or a reconnect request from a station set.

When all the combinations of ringing, winking, flashing, holding, and paging are considered, there are as many as seven simultaneous or overlapping events which must be monitored and timed by the control logic.

THE MICROCOMPUTER

In August 1975, with only a transistor-transistor logic (TTL) breadboard to verify the feasibility of the rather unusual control methods just described, a request for model shop design information for a trial system was received. Many questions regarding final features, human factors, requirements, and packaging of the system had to be answered quickly in order to meet the January Laboratories design information (LDI) deadline.

Texas Instruments' introduction of the TMS 1000 microcomputer provided an attractive alternative to the 38 TTL packages of the existing SLS system. The use of the programmable read-only memory (PROM) driven computer controller allowed the physical design to proceed without the final answers to some rather basic operational questions. With only the interface circuits to define, drafting of the printed circuit boards of the control unit was nearly complete before any TMS 1000 code was generated.

The TMS 1000 series is a family of p-channel metal oxide semiconductor (MOS) 4-bit microcomputers with a read-only memory (ROM), a random-access memory (RAM), an arithmetic logic unit (ALU), a clock, and input/output (I/O) logic on a single semiconductor chip. The TMS 1099 used in the SLS control system is identical to the single chip system when it is connected to the PROM instruction memory. The logic diagram of Figure 23-4 shows the architecture of the TMS 1000 system. Note the direct-only ROM addressing, the single-level subroutine stack, the two full-length general purpose registers, and the x-y RAM addressing system.

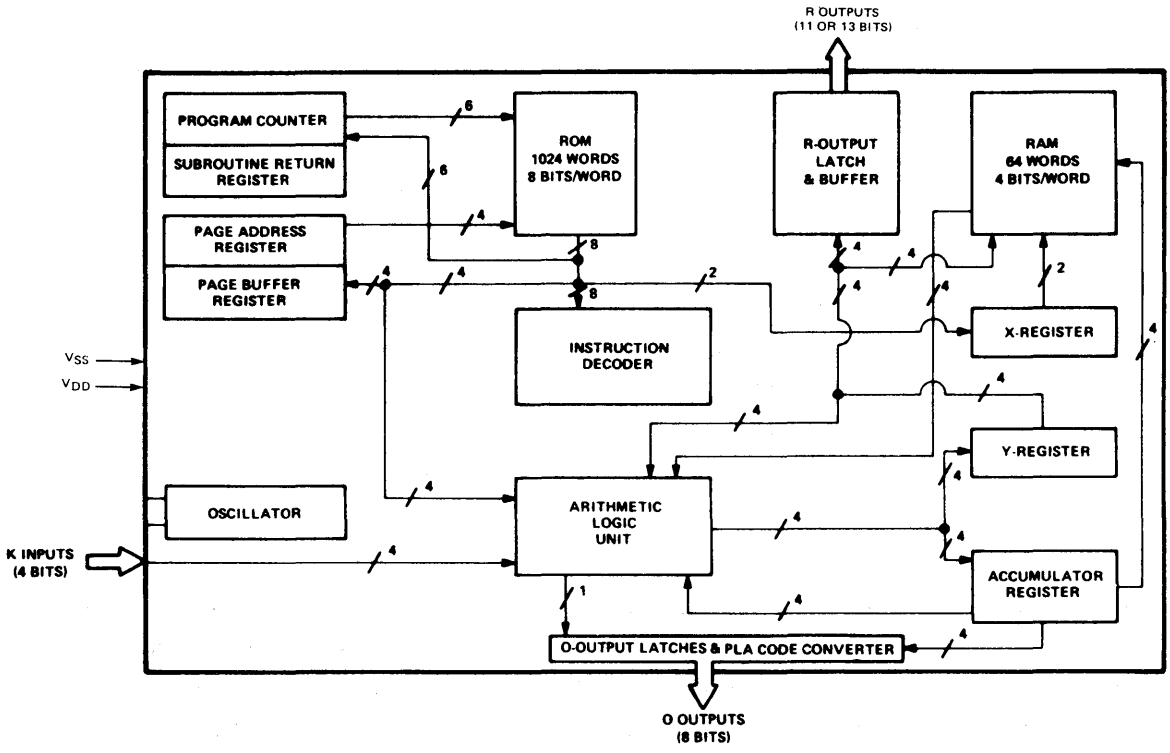


Figure 23-4 - TMS 1000 Computer Block Diagram

Also of interest are the single parallel input port, the single output port (5 bits, expandable to 8-bit codes), and the 11 individually addressable R-output bit latches. Examination of the 43 instructions in the standard TMS 1000 instructions set shows an abundance of decimal arithmetic functions with adequate logic and bit manipulation functions. The effectiveness of the computer in calculator applications is demonstrated by the addition of two 15-digit BCD numbers with a subroutine of 18 instructions.

THE SLS COMPUTER CONTROLLER

The inputs to the TMS 1000 microcomputer in the SLS system are a digital and an analog current signal as described in the discussion of the switching system. Interfacing the contact closure of the loop current sensor is straightforward. Determination of the magnitude of the intercom current is accomplished by a complementary metal-oxide semiconductor (CMOS) Schmitt input inverter package. The averaged voltage is scaled by resistor dividers to fit the positive threshold level of the inverter inputs. Three separate inverters respond to 20-, 40-, and 80-mA inputs. Decoding of the actual level from multiple bit inputs and elimination of the hysteresis effects of the Schmitt inputs are accomplished by the program logic.

The CMOS signals of the input logic must be level shifted for compatibility with the inputs of the microcomputer. This is hampered by on-chip load devices provided for keyboard interface applications.

The p-channel outputs of the microcomputer are used to drive NPN relay drivers, the CMOS analog switch of the paging mixer, and an n-channel buffer which modulates the CMOS tone oscillator. In several cases, the absence of load devices on the TMS 1099 necessitates extra resistors to sink the output leakage current.

SOFTWARE

The SLS software flowcharts were modeled closely after the hardware logic they were to replace. Special care was taken to avoid latch-up conditions which might be generated by single-bit errors or improper reentry after a power interruption.

The absence of any interrupt capability requires that the inputs be continually polled for active data. This is accomplished by program loops which continually check system flag bits, set the output status, and count valid intervals of active data in six 3-byte counters. Only 81 of the 256 bits of RAM are used.

The program consists of 468 instructions stored in two Intel 1702A PROMs. The program was edited and assembled on the Murray Hill time sharing network. The CODEGEN assembler was used to generate a paper tape ROM list which was loaded with a Pro-Log programmer. The present version of the software, now six months old, followed seven earlier editions which represent a total of five weeks programming effort.

PRODUCT CONSIDERATIONS

While one has little trouble justifying a $75 microcomputer with PROM program storage for a small volume trial vehicle, the question of a moderate volume product design must be examined separately. If the results of the trial forecast sufficient sales, custom logic must be considered. A custom chip, with tailored inputs and outputs, is most likely less expensive than a mask programmed microcomputer. However, the TMS 1000, with a present cost of less than $5, is certainly competitive at lower volumes. When all the costs of system development, manufacturing, and current engineering are evaluated, a microcomputer may be justifiable on the basis of development costs alone. The microcomputer ability to accomplish new tasks, features, or options within the same logic package will ensure its competitiveness and extend its economic life.

The success of the TMS 1000 in this application does not imply that this particular microcomputer is the best possible for the application. While the single power supply and low power consumption are valuable assets, the MOS logic levels require interface circuits which cost almost as much as the computer itself. A more flexible design, such as the MAC-4 proposed for manufacture by Western Electric, may be especially profitable if it can successfully reduce this interface cost.

PERIPHERAL TECHNOLOGY

SOME POWER CONSIDERATIONS FOR MICROPROCESSOR SYSTEMS

P. D. Fisher, BTL Dept 2424, WH, NJ

ABSTRACT

The cost of power conditioning for microprocessors is a significant part of total system cost. Precision regulation, special current-limiting features, overvoltage protection circuitry, and other requirements contribute to the high cost of power. This paper describes ac-powered rectifiers, small dc-to-dc converters, and switching regulators suitable for on-card power conditioning and discusses their economic impact on the microprocessor system. A low-power switching regulator (<1W) in a thick-film, hybrid dual in-line package (DIP) configuration that can be used as an on-card power conditioner is also discussed.

INTRODUCTION

Microprocessors and their associated memory and input/output (I/O) peripherals pose a varied and complex power problem to the system designer. The display and peripheral devices often dictate the power requirements of the system, since they consume the bulk of the necessary power. Figure 24-1 is a block diagram of a typical microprocessor system, showing some of the components to which the conditioner must supply power. Microprocessor systems with all of their possible applications are still too new to permit a clear-cut statement of the power-partitioning problem at the present time. As new systems and applications evolve, so new power designs will be offered. Our purpose in this paper is to examine the primary power considerations for present systems and review the economic aspects of various power options.

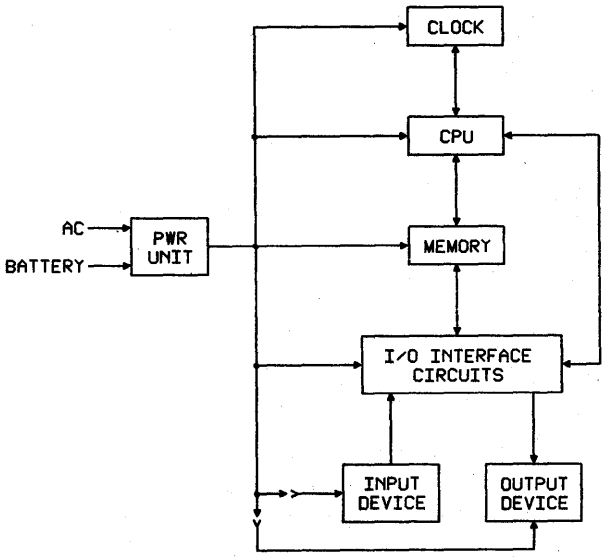


Figure 24-1 - Typical Microprocessor System

SMALL POWER CONDITIONERS

Today's microprocessor systems typically require several watts of both +12 and +5V power and several tenths of a watt of a negative polarity voltage. The complexity of the memory and I/O peripherals in microprocessor systems places total power requirements in the 1 to 50W range.*

AC-Operated Rectifier Power Supply

The ac-operated rectifier power supply usually offers the least expensive power-conditioning option to the microprocessor system designer. The block diagram in Figure 24-2 indicates the simplicity of this approach. The main disadvantage of the line-operated rectifier is its dependence upon the commercial power supply. Such dependence could prove unacceptable if volatile memory systems are used or continuous operation is critical to the application. An uninterruptable power source or a battery reserve system is necessary for these applica-

*Complementary-symmetry metal oxide semiconductor (MOS) or complementary MOS structures can lower overall power in the memory, I/O, and processor areas significantly.

tions. Commercial power lines are also subject to dips, surges, and electromagnetic interference (EMI); the rectifier must be able to condition the power under these abnormal conditions. In addition to its low cost, the line-operated rectifier is generally free from the spike noise of high-frequency switching. This type of noise is associated with switching regulators and dc-to-dc converters, but it is controllable within acceptable limits. The ac-powered supply uses dissipative linear regulators for output voltage control. These regulators provide an inexpensive way to obtain the mop-up regulation and ensure individual limiting of current for the output voltages. A typical 2-output, 10W ac-operated rectifier without any form of reserve system costs about $30.

Central Office or Private Branch Exchange (PBX) Battery

The central office or PBX battery system offers the microprocessor system designer a source of reliable dc power. The battery voltage, however, must be stepped down and regulated and, in some cases, have its polarity changed.

Switching Regulator

When only one voltage is desired, a switching regulator, shown in Figure 24-3, is the most effective technique for power conditioning. A characteristic of its design is that it has one power rail in common with the input power. The inability to isolate input and load grounds can cause system noise problems because of high-circulating currents in the common power path. A series of low-power switching regulators is available from Western Electric - the 225, 226, and 227 types deliver 2, 5, and 10W of power, respectively. These units are assembled on a printed circuit board (PCB) in a 2- by 3- by 1-inch high module with pin arrangements suitable for mounting on the user circuit packs. The regulators contain no alarms, fuses, or on/off switches, but some codes have overvoltage protection. The estimated bulletin cost of these devices is in the range of $15 to $20.

The switching regulator is also applicable when a grounded dc voltage, such as 5V transistor-transistor logic (TTL) power, is available from an existing converter. In this situation, the switching regulator can supply a higher or an opposite polarity voltage.

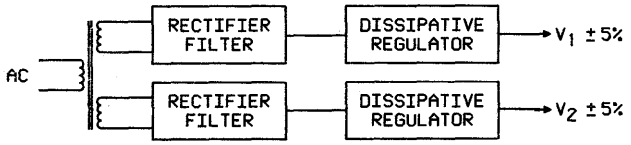


Figure 24-2 - AC Line-Operated Rectifier Block Diagram

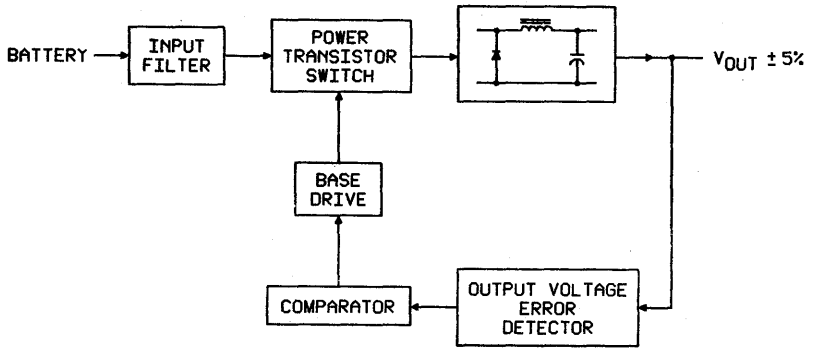


Figure 24-3 - Switching Regulator Block Diagram

Thick-Film Hybrid DIP

A thick-film hybrid DIP which contains a low-power (<1W) on-card switching regulator is under study. This unit requires no external components to operate. A prototype, shown in Figure 24-4, produces 70 mA of +12V power from a +5V input. This unit does not have overvoltage or short-circuit protection. Since the transistor power switch is in parallel with the load, a fuse would afford adequate short-circuit protection. In most cases, failure of the control circuit would cause the output voltage to go to +5 or 0V. This device can readily be applied to the PCB of a microprocessor system. The estimated cost of this film regulator is about $8.25.

DC-to-DC Converter

In situations where multiple output voltages from a central office battery are needed, along with input-to-output isolation, a dc-to-dc converter is desirable. Figure 24-5 shows a block diagram of this type of low-power converter. Transformer isolation and optically coupled feedback control signals ensure input-to-

output isolation. The Western Electric 201, 202, and 203 types supply 2, 5, and 10W of regulated output power from a 24 or 48V battery plant. Packaged on PCBs, the modules are suitable for mounting on the user circuit board by pin connectors, the same mounting method used for the 225 series switching regulators. Figure 24-6 is a photograph of the 201A 2W converter. The 201 and 202 series are mounted on a PCB in a 3.3- by 3.5- by 0.9-inch high module, while the 203 10W converters are mounted on a PCB 3.3 by 6.5 by 1.25 inches. These converters are output-current-limited and act as protectors against excessive output voltage. As in the case of switching regulator modules, the converters contain no fuses or switches, but have in-rush current-limiting to protect the contacts of plug-in circuit packages. They do not require heat sinks if used within their rated temperature, voltage, and current ranges. Up to three output voltages are available from these converters, but only one output is closely regulated to ±5 percent end of life (EOL) tolerance.* The remaining output voltages are normally unregulated with tolerances of ±15 percent. If load current variations are held within ±25 percent, however, ±8 percent voltage regulation for these nonregulated windings can be achieved. For tighter regulation and additional cost, linear dissipative regulators applied to the noncontrolled outputs can achieve a ±5 percent tolerance. The bulletin cost of the 201A 2W 3-output converter is about $24. A typical 202 series 5W converter is estimated at $30, and a 10W 203 type at $40.

COST OF ADDITIONAL SYSTEM FEATURES

The on-card low-power switching regulator and the dc-to-dc converters described above are all designed with minimal alarm and protection features in order to keep their cost as low as possible. In most cases, self-protection, such as current-limiting and overvoltage, is included. The overvoltage protection for the switching regulator circuits consists of a crowbar circuit mounted on the module and a user-supplied fuse; the voltage limit is usually set at 30 to 40 percent higher than the normal output voltage. A closer tolerance would require a more precise crowbar protection circuit at a premium of $5 to $8.

* EOL tolerance includes variations in line, load, temperature, initial set point, and aging.

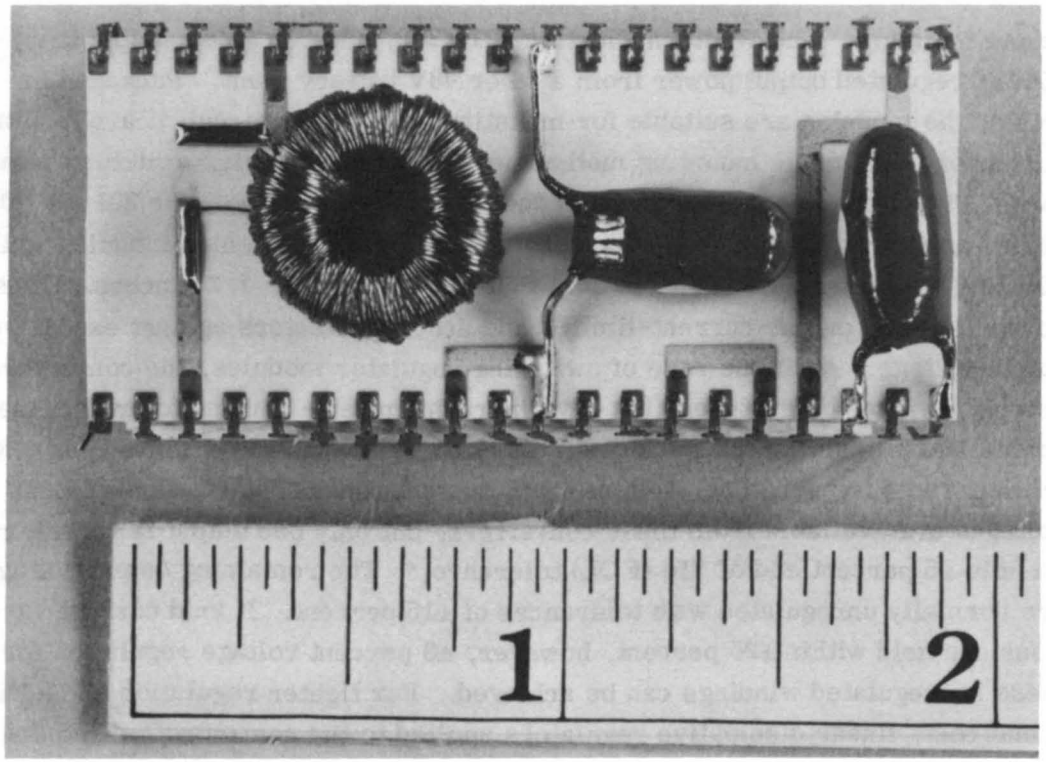


Figure 24-4 - 12V 70 mA DIP-Switching Regulator

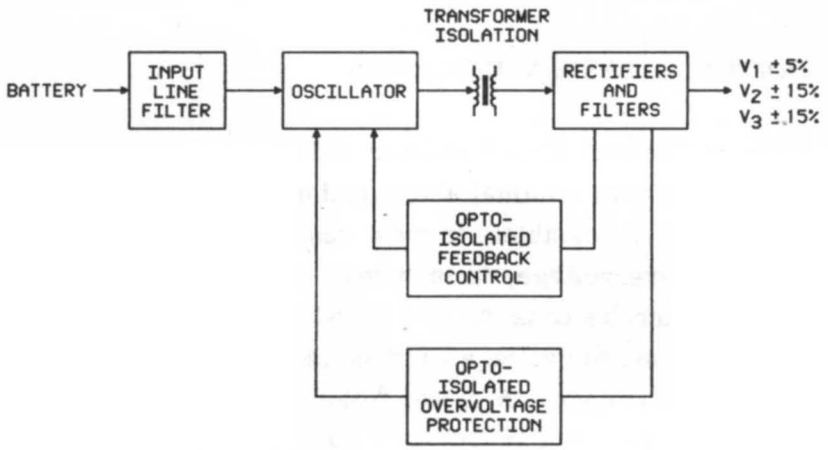


Figure 24-5 - DC-to-DC Converter Block Diagram

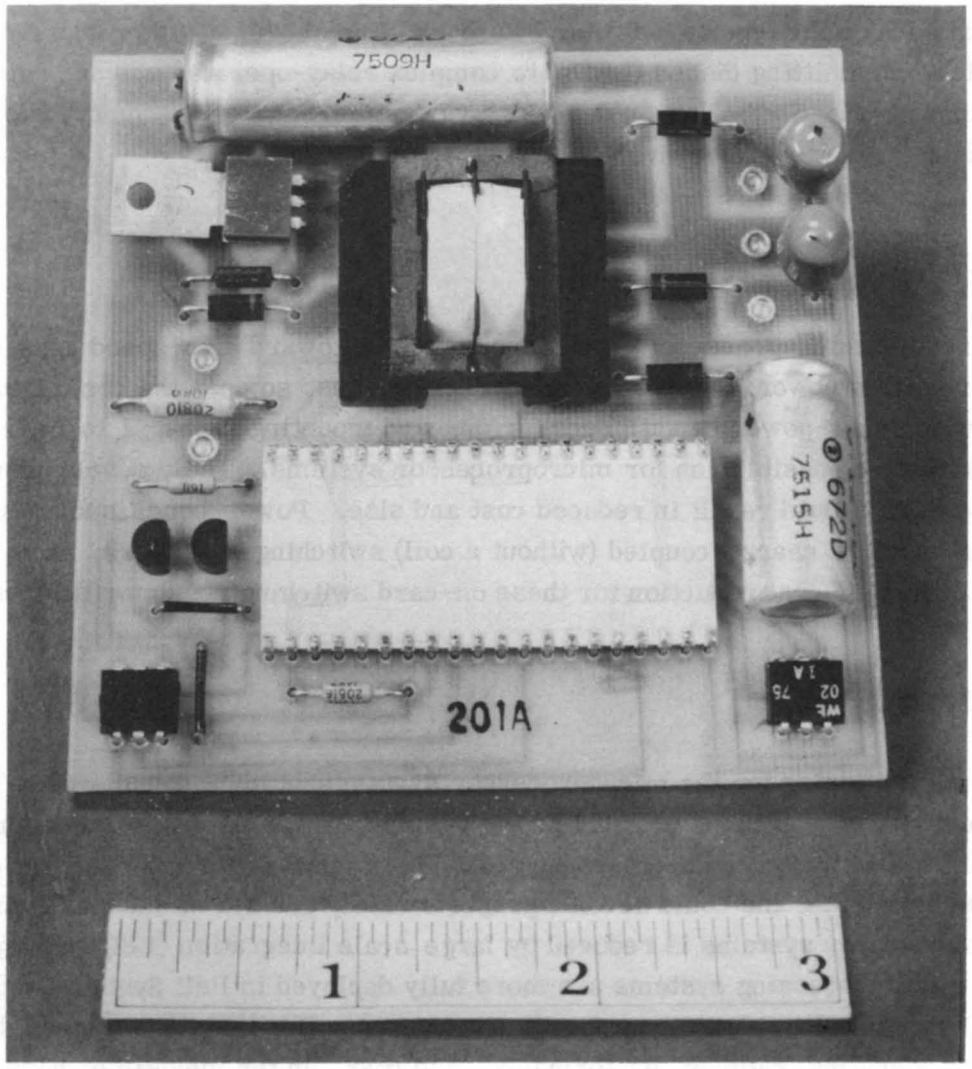


Figure 24-6 - 201A Power Unit

A precision reference and comparator circuit that can be set to within 20 percent of the nominal voltage protects the 201, 202, and 203 converters. However, the usual protection limit is 30 to 40 percent higher than the normal voltage due to specific load requirements. An example of this type of requirement exists in a 5V converter designed to power TTL loads. The output voltage need only be kept below 7V for adequate protection for this type of logic.

Alarm circuits which indicate converter failure, specifically an overvoltage or undervoltage condition, also add to the overall system cost, generally $8 to $10,

depending upon the type of indicator required. They cover a wide range from simple light-emitting diodes (LEDs) to complex relay-operated alarms. A memorandum by S. F. Newton describes various circuits and techniques for alarming the low-power converter series.[1]

FUTURE DEVELOPMENT

The trend in microprocessors today appears to be toward high-speed logic cells requiring less power. As the power level decreases, so does the cost of conditioning it. Low-power conditioners for on-card mounting in the <1 to 5W range are attractive possibilities for microprocessor systems. Their development on hybrid DIPs should result in reduced cost and size. Power-conditioning techniques, such as charge-coupled (without a coil) switching regulators, show promise of further cost reduction for these on-card switching regulator devices.

CONCLUSION

Power conditioning for the microprocessor system should be considered at the earliest stages of design, not only by the system planner but by the microprocessor and memory device designers as well. The impact of these power devices on cost is significant and could become proportionately greater as the outlay for microprocessor systems is reduced by large scale integration (LSI) techniques. Until microprocessing systems are more fully deployed in Bell System equipment, it will be difficult to describe a family of power supplies that have the best balance of operating features, performance, and cost. In the meantime, however, a standard line of power supplies is available to meet many of today's needs. For small processor systems, the modular 201, 202, and 203 series of dc-to-dc converters can provide multiple voltage outputs in the 2, 5, and 10W power range. For the smallest systems, 1W or less, more development is needed, but it is underway, and prototypes of DIP-mounted switching regulators should be available soon.

REFERENCES

1. S. F. Newton, Alarm Circuits for Low Power Converters, Memorandum for File, Case 35763-239, September 3, 1976.

PERIPHERAL TECHNOLOGY

A SURVEY OF SMALL TAPE PERIPHERALS

J. E. Williams, BTL Dept 2452, WH, NJ

ABSTRACT

The broad spectrum of microprocessor applications has generated a variety of requirements for nonvolatile, sequential data storage. Microprocessors can be served by magnetic tape technology in much the same way that larger scale processors are served by more capacious tape peripherals. An overview of small tape systems that are candidates for use as microprocessor peripherals is given here. Devices already on the market and others developed recently are discussed including the following:

- 3M Company DC300 data cartridge.
- 3M Company DC100A mini-cartridge.
- Phillips cassette.
- Information Terminals Corporation mini-data cassette.
- Microvox data wafer.
- Interdyne UNIREEL.

INTRODUCTION

Magnetic tape technology may be able to satisfy a major portion of the growing demand for nonvolatile, electrically alterable, sequential data storage being generated by new microprocessor applications. We have attempted to define the spectrum of small tape systems that could reasonably be used as microprocessor peripherals.

3M COMPANY DC300 DATA CARTRIDGE

With respect to capacity, bit rate, physical size, and medium cost ($18 per cartridge), the upper end of the spectrum is represented by systems employing the 3M Company data cartridge, which is illustrated in Figure 25-1. This device provides an unformatted capacity of 23 by 10^6 bits on 300 feet of 1/4-inch tape. The drive shown in this figure is a member of the KS-21447 minirecorder family developed by Laboratory 245 and currently used in substantial numbers in the DIMENSION private branch exchange (PBX) and 3A central control (CC) based systems at data rates of 48 kb/s.

Although one of the better commercially developed tape media, the data cartridge has required internal modification so that it will perform adequately in Bell System environments. The modifications were the result of detailed analysis and extensive system testing within BTL and illustrate the need for caution in applying off-the-shelf tape technology, regardless of its commercial acceptance and success. The modified device is known as the KS-21439 data cartridge now procured in quantity by WE from two suppliers.

The KS drive and cartridge designs represent a reliable and relatively mature technology. Although generally best assigned to minicomputer tasks, the cartridge capacity could be suitable for microprocessor-controlled data logging-type applications.

3M COMPANY DC100A MINI-CARTRIDGE

The standard data cartridge technology has been extended downward with the recent introduction of the DC100 mini-cartridge (shown in Figure 25-2 with associated drive) offered by the 3M Company. The development of the latter was instigated largely by Hewlett-Packard, which until the recent introduction of the 3M drive, was the only user of the cartridge. At a medium cost of $15 per unit the mini-cartridge provides an unformatted capacity of 2.6 by 10^6 bits on 140 feet of 0.15-inch tape.

Contrary to comments from the manufacturer and from Hewlett-Packard, the initial evaluation of Laboratory 245 indicates adequate tape tension stability over a

wide temperature range at the low end of the speed range (about 5 in/s), permitting operation in low-data rate applications.

Fortunately, the 3M Company has incorporated in the commercial mini-cartridge several of the features originally applied as modifications to the larger DC300A in the KS-21439 program.

Despite a 5-to-1 disadvantage in medium cost compared with possible alternatives, the demonstrated performance of the larger cartridge, the inherently simple drive configuration, the experience gained in the BTL minirecorder program, and the stability of the medium source suggest that this device represents a minimum-risk approach to a medium-capacity microprocessor tape peripheral.

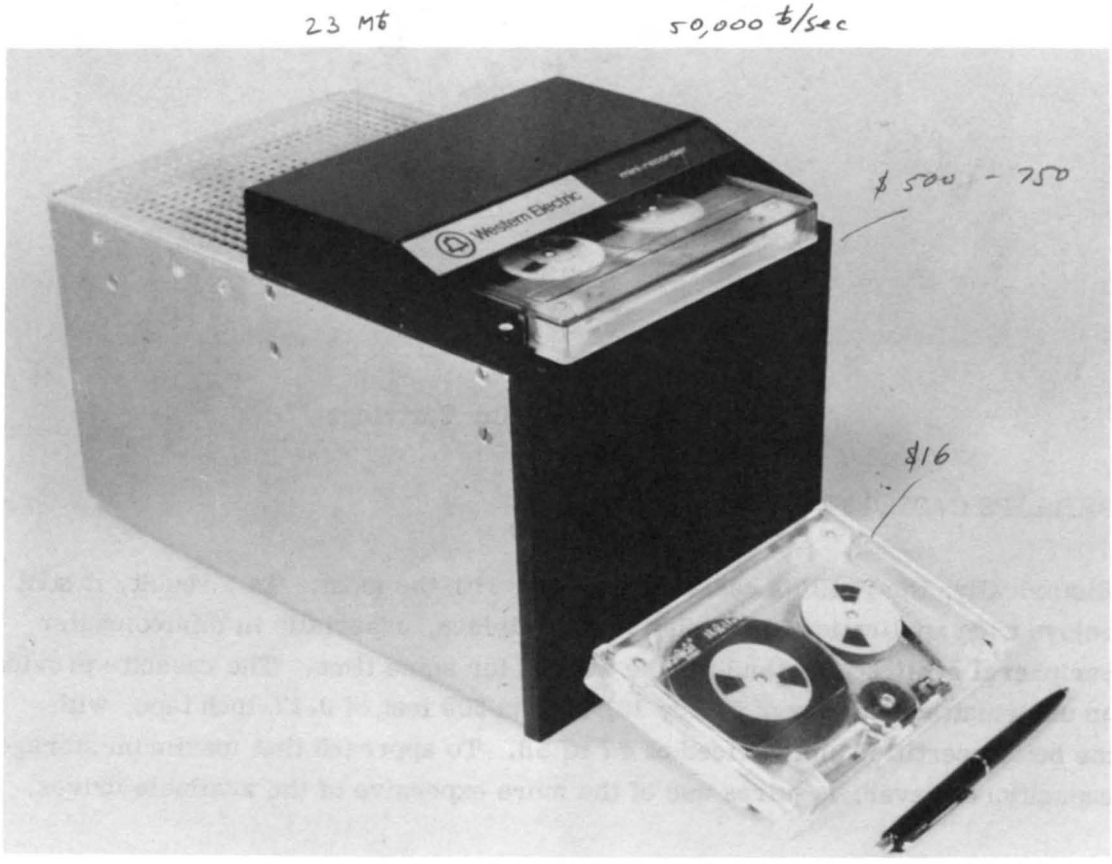


Figure 25-1 - 3M Data Cartridge

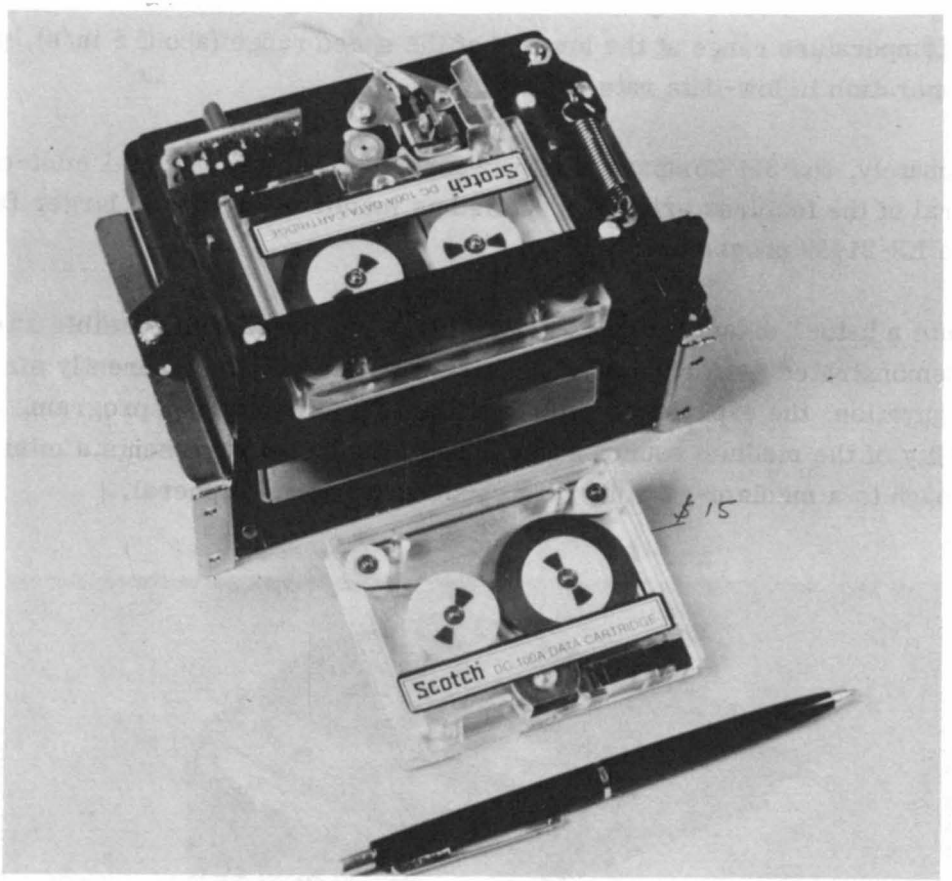


Figure 25-2 - 3M Mini-Cartridge

PHILLIPS CASSETTE

Historically, the Phillips cassette was first with the most. As a result, it still enjoys wide application in the digital marketplace, especially in minicomputer peripheral applications, and will be with us for some time. The cassette provides an unformatted capacity of 5.7 by 10^6 bits on 300 feet of 0.17-inch tape, with the better certified units priced at $7 to $8. To approach that maximum storage capacity, however, requires use of the more expensive of the available drives.

A wide variety of drive designs and recording formats has been executed using this medium with varying degrees of success. Typically, the drives are relatively complex, having up to four motors, movable heads, pinch rollers, etc.

MINI-DATA CASSETTE

A downward extension of the standard Phillips cassette technology is the newly proposed American National Standards Institute (ANSI) mini-data cassette manufactured by Information Terminals Corporation and illustrated in Figure 25-3. The cassette provides an unformatted capacity of 480,000 bits per side on 50 feet of 0.15-inch tape.

The first commercial drive in this technology, shown in Figure 25-3, is the Raymond Engineering mini-raycorder, which is planned for use in initial models of the service access unit (SAU) under development for the 32A Communications System. The unit fills a typical hard-copy backup role, although its use in program paging operations is being considered.

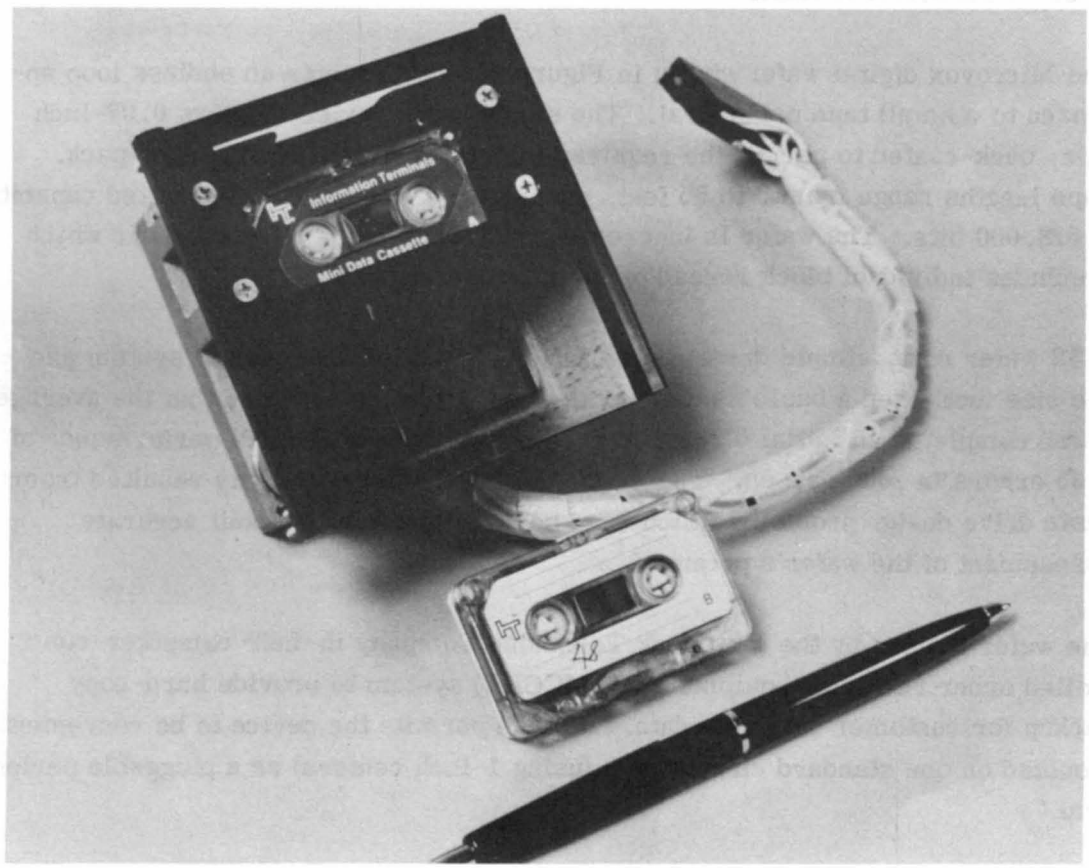


Figure 25-3 - Information Terminals Corporation Mini-Data Cassette

Low medium cost ($5), small package volume, reasonable capacity, and operation from +5V logic power are clearly attractive features for many applications. However, experience to date indicates that a number of problems remain to be solved; they include extended start time, as well as tape drive difficulties after about 100 end-to-end cycles of the medium. The severity of each of these problems depends on the specific application. Hopefully, these difficulties can be cleared. However again they show the need for caution in regard to off-the-commercial-shelf tape technology.

The same forces which cause the general data reliability of 3M cartridge technology to be superior to Phillips cassette technology exist in relation to the mini-cassette and mini-cartridge devices.

MICROVOX DATA WAFER

The Microvox digital wafer shown in Figure 25-4 represents an endless loop approach to a small tape peripheral. The single track device employs 0.07-inch tape, back-coated to permit the required interlayer slip within the tape pack. Tape lengths range from 5 to 55 feet, permitting a maximum unformatted capacity of 528,000 bits. The wafer is inherently unidirectional, a characteristic which precludes individual block reread or rewrite operations.

A $2 wafer cost, simple drive configuration, and minimum overall system package size motivated a basic medium evaluation. Testing revealed that the average wafer completed an initial 625 error-free passes with an overall performance of 0.35 errors in 2000 passes. The major part of the error activity resulted from basic drive design problems which have been eliminated to permit accurate assessment of the wafer's potential.

The wafer is used by the New York Telephone Company in their computer-controlled order receiving equipment (COMCORE) system to provide hard-copy backup for customer alterable data. Its size permits the device to be conveniently mounted on one standard circuit pack (using 1-inch centers) as a pluggable peripheral.

The wafer is relatively new to the digital marketplace and the associated drive has experienced some growing pains. Its commercial future is less certain than that of the 3M Company technology.

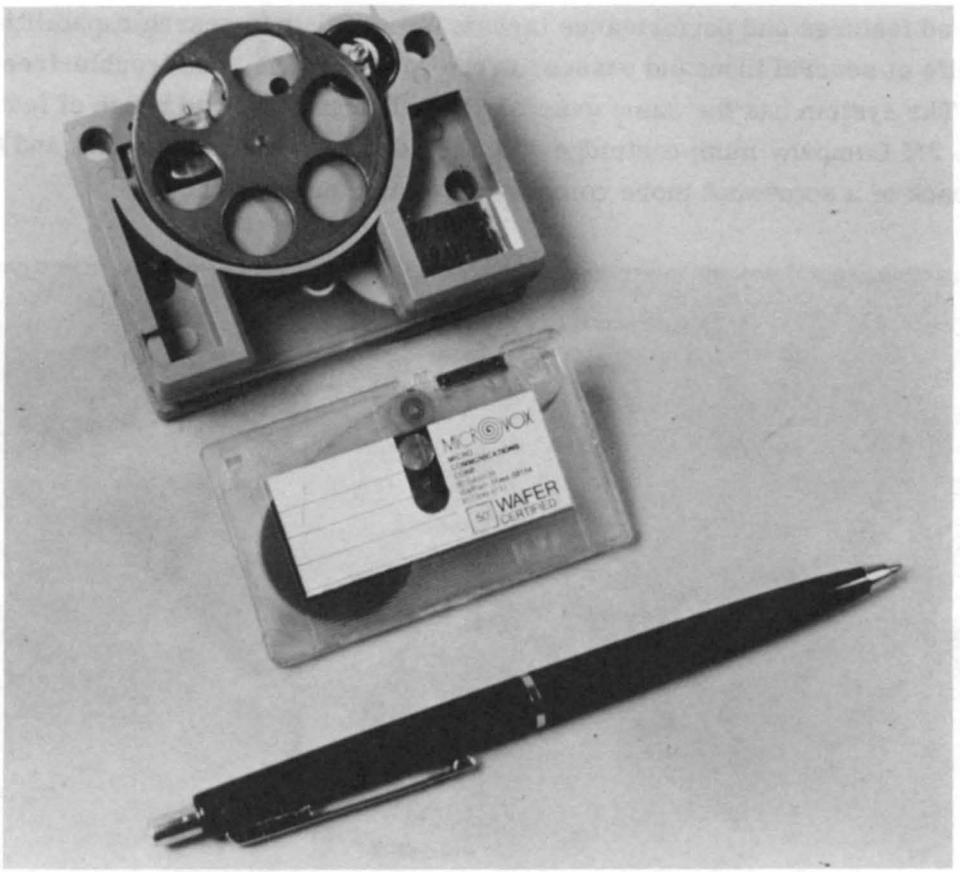


Figure 25-4 - Microvox Wafer

UNIREEL

The Interdyne UNIREEL spool (see Figure 25-5) is a relative newcomer to the digital field and offers a number of interesting features. It has a unit cost of about $2 and contains 150 feet of 0.15-inch tape (2.9 by 10^6 bits, unformatted capacity) spliced to a stiff leader used by the associated drive to thread itself. The last portion of the leader has a wide overlay of compliant material, which seals the UNIREEL during handling and shipping. Once threaded, tape is driven

between the supply (UNIREEL) and takeup reels by a tensioned plastic belt similar to the one found in the 3M data cartridge, although the UNIREEL belt is incorporated within the transport rather than as a part of the separable medium.

Claimed features and performance include 60- to 90-in/s search capabilities, a tape life of several thousand passes, error rates of 10^{-8}, and trouble-free loading. The system has the same general performance level and range of features as the 3M Company mini-cartridge with the advantage of medium cost and the drawback of a somewhat more complex drive and loading scheme.

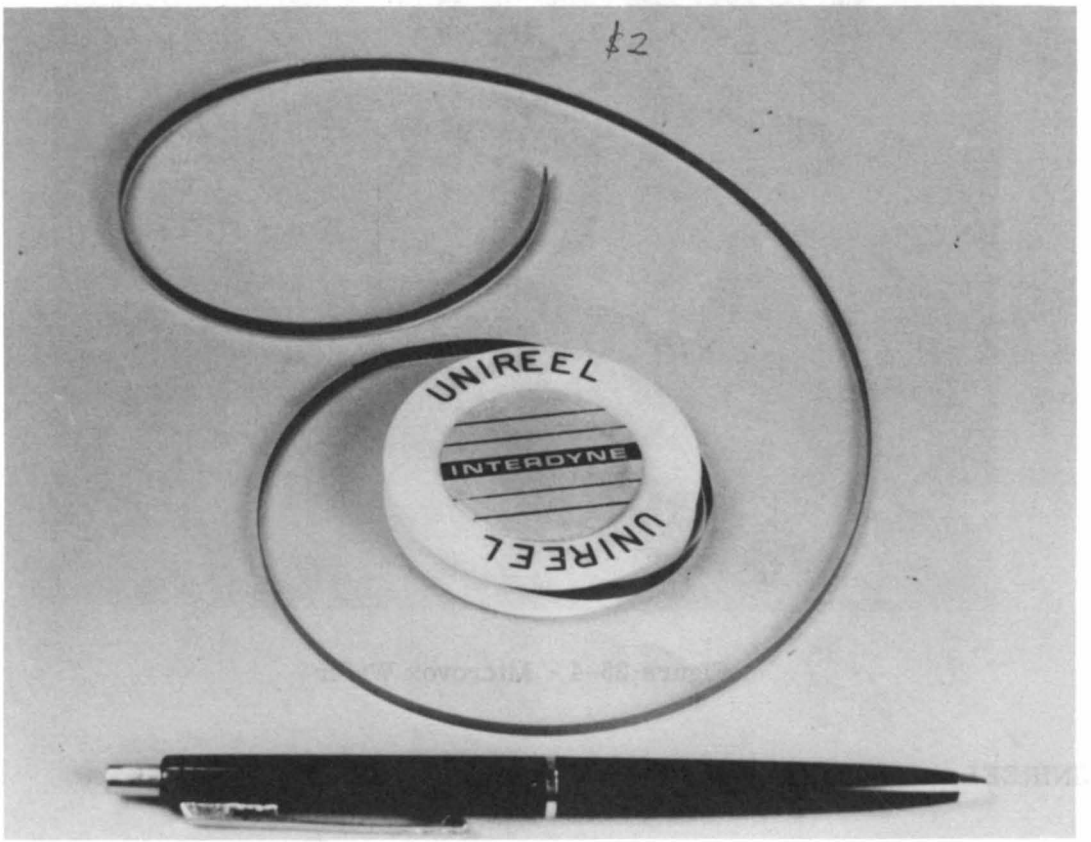


Figure 25-5 - Interdyne UNIREEL

SUMMARY

Of the small tape systems mentioned, only the 3M DC300 data cartridge and the standard Phillips cassette now represent multiple sourced tape media. For the rest, commercial success in an expanding marketplace will ordinarily produce alternate suppliers. With the exception of the standard cassette and UNIREEL, Laboratory 245 is actively engaged in programs involving these media. Its intent, beyond providing for immediate commitments, is to evaluate a family of microprocessor peripherals to service the whole range of microprocessor applications.

PERIPHERAL TECHNOLOGY

SERIAL BUBBLE STORE

R. J. Radner, BTL Dept 2451, WH, NJ

ABSTRACT

The specifications, interface, and operation procedures for a digital memory system based on magnetic bubble technology are presented in this paper.

The serial bubble store (SBS) is a transistor-transistor logic (TTL) compatible, nonvolatile memory with a 0.27-Mb capacity and a 48-kb/s operating data rate. User evaluation units are being supplied by Laboratory 245.

INTRODUCTION

Laboratory 245 is developing memory subsystems based on magnetic bubble technology. One type is a voice announcement system dubbed the 13A, which will attain production status during 1977. Another is a digital memory utilizing the same bubble device (M5021) as the 13A. The latter, the SBS, with its specifications, interface, and operation is the subject of this paper.

SBS is a TTL-compatible, nonvolatile memory containing four serial shift registers that are separately accessible. Data can be written into, or read from, one memory register at a time. See Table 26-1.

Each register contains 68,121 bit locations through which data shift. Except during the writing process, the contents of each register are recirculated. A system clock of 960 kHz produces a serial data rate of 48 kb/s.

TABLE 26-1

SBS SPECIFICATIONS

Item	Description
Capacity	272,484 bits
Access time	1.42 seconds (max)
Number of registers	4, accessible one at a time, by 2-line binary addressing
Capacity of each register	68,121 bits serial
System clock (user-supplied)	960 kHz
Data rate	48 kb/s (max) (submultiples are possible, e.g., 24 kb/s, 4.8 kb/s
Interface signaling	High - False - "0" Low - True - "1" (TTL - compatible)
Operational temperature range	0° to 50°C (ambient)
Storage temperature range (memory retained)	-40° to +80°C
External operational magnetic field maximum	30 oersteds
External nonoperational magnetic field maximum (memory retained)	50 oersteds
Input power	+15V; +8V; -8V; +5V
Power dissipation (while reading or writing)	3W
Physical dimensions	(1) 5.67- by 5.4-inch board (16 staked pins) (2) 5.67- by 7.4-inch board (space for connector)

SBS INTERFACE DEFINITION

This section defines the terminology used in Table 26-2. The timing diagrams shown in Figures 26-1 and 26-2 further define this language. The logic interface is low-power TTL with 1- or 2-gate loading for all inputs.

SYSTEM CLOCK

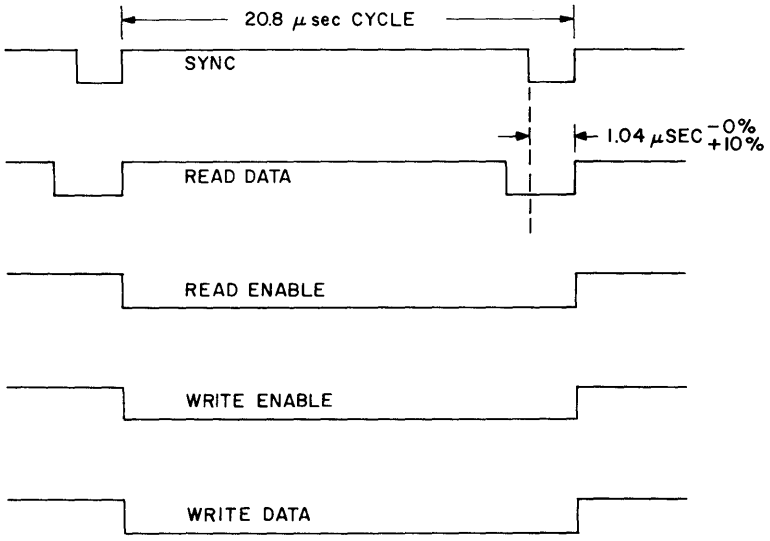
The controlling system must provide a clock signal to the SBS on the SYSTEM CLOCK line. The required clock signal is a 960-kHz ±5 percent square wave with 50 percent duty cycle.

TABLE 26-2

SBS INTERFACE

Pin Number	
Power	
1	+5V ±5%, 150 mA
2	+8V ±5%, 200 mA
3	-8V tracking +8 within 5%, 200 mA
4	+15V ±5%, 40 mA
5	GROUND
Inputs	
6	SYSTEM CLOCK (960 kHz)
7	STORE ENABLE
8	CLEAR
9	REGISTER SELECT 0
10	REGISTER SELECT 1
11	WRITE ENABLE
12	WRITE DATA
13	READ ENABLE
Outputs	
14	SYNC (48 kHz)
15	STORE ON-LINE
16	READ DATA

THIS FIGURE SHOWS TIMING, NOT ACTUAL OPERATION.
READ AND WRITE COMMANDS SHOULD NOT BE SIMULTANEOUS.



TRANSITION TIMES OF READ ENABLE, WRITE ENABLE AND WRITE DATA ARE WITHIN 1μS OF LOW TO HIGH TRANSITION OF SYNC.

Figure 26-1 - SBS Timing

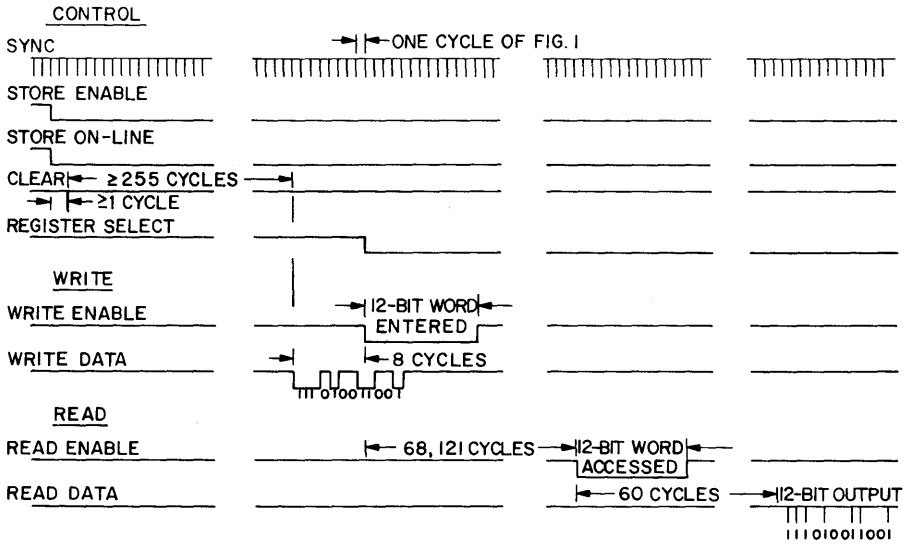


Figure 26-2 - SBS Register Sequencing

SYNC

The clock with which all SBS functions are synchronized appears on this line. The low-to-high transition of this pulse should be used for synchronization of control system functions and WRITE DATA. The high-to-low transition indicates that READ DATA is valid. The SYNC rate is 48 kHz.

STORE ENABLE

A low level on the STORE ENABLE line selects SBS for operation. This level should be brought low within 1 μs after the low-to-high SYNC transition and should be held for one cycle of SYNC before reading or writing is initiated. STORE ENABLE should be held as long as the SBS is operating and should be brought high within 1 μs after the low-to-high SYNC transition. STORE ENABLE must be high while power is being brought up. When STORE ENABLE is low, the SBS power dissipation is 3W. When high, the power used drops to 1.5W.

STORE ON-LINE

When the SBS has been powered, the STORE ON-LINE will come low within one cycle after STORE ENABLE has been provided if the SBS is ready for operation.

CLEAR

In order to ensure that all four registers are initialized, a low is applied to CLEAR. This erases all data in all four memory registers. This line must be held low for at least 1 μs and must be brought high before a subsequent CLEAR can be executed. No functions are valid for at least 255 cycles following the CLEAR command.

REGISTER SELECT (RS0 and RS1)

The REGISTER SELECT lines select the register to be written or read.

Register	RS1	RS0
R0	High	High
R1	High	Low
R2	Low	High
R3	Low	Low

WRITE ENABLE

A low enables SBS to write the data on the WRITE DATA line. There is an 8-cycle delay between the time a bit is placed on the WRITE DATA line and the time at which that bit enters the register. Therefore, WRITE ENABLE should go low eight cycles after the first data bit and must remain at that level eight cycles after the last data bit. WRITE ENABLE must be a valid control between low-to-high transitions of SYNC.

WRITE DATA

The WRITE DATA line is driven by the data to be recorded. Data should be valid within 1 μs of the low-to-high transition of the SYNC and remain valid until the next such transition.

READ ENABLE

A low enables the read circuitry to access the data in the selected register. Data will appear at READ DATA 60 cycles after READ ENABLE is applied.

READ DATA

The serial data stream being read from a register appears on this line. The data are valid for 1 μs after the high-to-low transition of SYNC.

SBS OPERATION

Figure 26-3 shows the logic functional organization of SBS. One of the four registers, R0, has data moving along its 68K-bit shift register which is clocked by CLK. The output of the 68K-bit shift register is returned to the input through two gates and recirculated through the register, except when WRITE is enabled and R0 is selected.

All four 68K-bit shift registers are nonvolatile, as are the 8-bit shift registers on the inputs. The 60-bit shift registers are partly volatile.

SBS provides no indication to the using system of current position within the recirculating shift registers. The using system, therefore, must provide either a nonvolatile counter separate from SBS or must write a distinctive pattern (preamble) into one or more of the SBS shift registers. The distinctive pattern must then be precluded from appearance in data entries.

Although the various shift registers clock data at 48 kHz, the user can read or write at lesser rates. For example, by applying WRITE ENABLE every other cycle, data can be written at 24 kb/s and will fill a 68,121-bit shift register since that register contains an odd number of bits. Similarly, the register can be read at 24 kb/s. The most convenient data rates are those in which the ratio of 48,000 to the data rate is an integer that does not share a factor with $68,121 = 3^4 \cdot 29^2$.

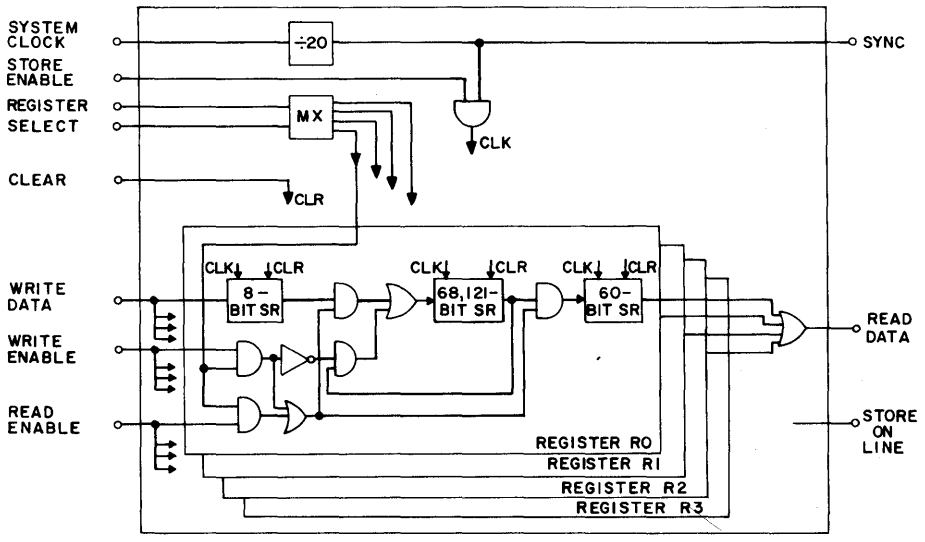


Figure 26-3 - SBS Organization

READ

The output of the 68K-bit shift register R0 will pass through a 2-input AND if that register has been selected and if either WRITE or READ has been enabled. The data then enter a 60-bit shift register that is clocked at the same time as the 68K-bit register. The output of the 60-bit shift register appears at READ DATA.

To read the memory, a low on the READ ENABLE and appropriate signals on REGISTER SELECT lines will result in the appearance of data at the READ DATA line. The high-to-low transition of SYNC indicates the time at which data are valid. The data being read experience a 60-cycle delay, i.e., valid data will appear 60 cycles (counting the cycle in which READ is enabled) after READ ENABLE goes low. During READ the data in a 68K-bit register are recirculated.

In order to read registers R3 and R0 in sequence without interruption of the serial bit stream, the 60-bit shift registers must be taken into account. At a time when the last 60 bits of R3 are in the 60-bit shift register, the register selection should change from R3 to R0 so that the 60-bit shift register associated with R0 can be filled while the 60-bit register associated with R3 is being emptied (READ).

WRITE

In order to write data into a register, a low on the WRITE ENABLE and appropriate signals on REGISTER SELECT lines permit the data on the WRITE DATA line to be recorded into the selected register.

The data appearing on the WRITE DATA line experience a delay of eight cycles (counting the cycle in which data is written) before entering the register. REGISTER SELECT and WRITE ENABLE must remain as set from eight cycles after the first data bit until eight cycles after the last data bit in order to use the contents of the 8-bit shift register that is in series with the WRITE DATA line. During WRITE, the data in a 68K-bit register are not recirculated.

Whenever there is a 1 on the WRITE DATA input, 1s are inserted into all four 8-bit shift registers. REGISTER SELECT will determine in which register

the 1 will be recorded. WRITE DATA must be kept at logic 0, except during writing, because READ ENABLE permits data to be written.

It is recommended that complete rewriting with new data in all four 68K-bit shift registers be preceded by CLEAR (CLR) which sets to 0 the contents of all registers (including the 8- and 60-bit shift registers).

EDIT

Indexing data to be edited must take into account the 8-cycle write delay. With this delay accounted for, the write procedure will enter new information in the desired location, replacing the old. Note that CLEAR cannot be used when editing. A portion of the data in memory may be erased by using the EDIT procedure and entering only 0s on the WRITE DATA lead.

ACKNOWLEDGMENTS

The author wishes to acknowledge the major contributions of T. M. Burford and J. H. Wuorinen to this document and to the SBS configuration and those of G. R. Westerman for his circuit design.

PERIPHERAL TECHNOLOGY

A ROM-RAM-I/O DEVICE FOR THE MAC-8

R. L. Ukeiley, BTL Dept 4391, HO, NJ
W. C. Slemmer, BTL Dept 2261, MH, NJ

ABSTRACT

The ROM-RAM-I/O device is a MAC-8 peripheral which combines read-only memory (ROM), random-access memory (RAM), and 16 bits of peripheral input/output (I/O) circuits on a single chip. Connected with a MAC-8, these devices form an entire microcomputer system. The chip contains 1024 eight-bit words of ROM, 96 eight-bit words of static RAM, and 2 eight-bit programmable peripheral I/O ports. One of these peripheral ports may be used to access the optional on-chip peripheral functions, including an interval timer, read/write strobes, and an interrupt circuit. Use of these options is selected under program control.

INTRODUCTION

Using a microprocessor can substantially reduce the IC package count in a system. Systems that would have previously required hundreds of ICs have been reduced to a central processing unit (CPU) chip with associated memory and peripheral devices. However, there are many small systems whose cost and size are still excessive. For these systems it would be desirable to have a 1- or 2-chip microprocessor system. Some manufacturers have succeeded in realizing a 1-chip system. In order to pack all of this circuitry into a single chip, however, they have had to significantly reduce the capability of the CPU in order to save chip area.

As an alternative, when greater CPU capability is required, a 2-chip system could be considered: the CPU is on one chip and the ROM-RAM-I/O is on a

second chip. As manufacturing technology advances, one can then consider integrating these two chips into a single chip without having to sacrifice CPU capability.

The Bell System MAC-8 CPU is an extremely powerful microprocessor. It would be inadvisable to reduce the power of the MAC-8 to make room for memory and I/O circuitry on the same chip. Hence, a single chip ROM-RAM-I/O peripheral is being designed for the MAC-8. With this device, small systems can be realized with two chips. Larger systems can use more than one ROM-RAM-I/O chip as well as other memory and I/O devices.

THE ROM-RAM-I/O DEVICE

The ROM-RAM-I/O device is a pseudo-CMOS chip containing 1K bytes of ROM, 96 bytes of RAM, and 16 independent programmable peripheral I/O bits. This device provides, in addition to the normal I/O peripheral functions, special features which may be multiplexed onto the peripheral I/O leads. These additional on-chip features include an interval timer, a MAC-8 interrupt control flip-flop, and peripheral strobe generators. The chip will be mounted in a 40-pin dual in-line package (DIP). The allocation of the 40 pins is shown in Table 27-1. As indicated by this table, the pins can be divided into 3 groups: 2 pins for power connection, 16 pins for peripheral I/O devices, and 22 pins for interconnection with the MAC-8 microprocessor. The 22 pins of the third group consist of an 8-bit bidirectional data bus, an 11-bit address bus, and 3 bits for timing and control.

TABLE 27-1
PIN ALLOCATION FOR THE ROM-RAM-I/O DEVICE

No. of Pins	Function
2	Power and Ground
16	Peripheral I/O Pins
8	Bidirectional Data Bus
11	Address Bus
1	Memory Read
1	Memory Write
1	Timing Clock

Figure 27-1 is a block diagram showing the interconnection of a single ROM-RAM-I/O device with a MAC-8 CPU. These two chips form a complete microprocessor system. If a system requires more than one ROM-RAM-I/O device or other peripherals, a mask-programmable chip-enable pin is available.

Each peripheral I/O pin is capable of performing as an input, an output, or a special function under software control. To accomplish this, the ROM-RAM-I/O device includes three internal 8-bit registers which control the function of each I/O pin. These registers are addressable by the microprocessor for loading the desired control word at the beginning of a program, or altering I/O pin functions during program execution. Two of these registers determine the direction of data flow on each of the 16 I/O peripheral pins. When bits within this 16-bit field are programmed high, the output function is selected. Hence, the tristate output drivers will be enabled for the corresponding peripheral I/O pins. The output data for these pins are obtained from the two on-chip 8-bit output registers which may be read or written by the microprocessor. Programming a control bit low disables the tristate output driver. The input function is accomplished by addressing the input ports which transfer the logic state present on the device pins to the data bus. If the I/O pins are used as simple input or output leads, these transistor-transistor logic (TTL) compatible pins are con-

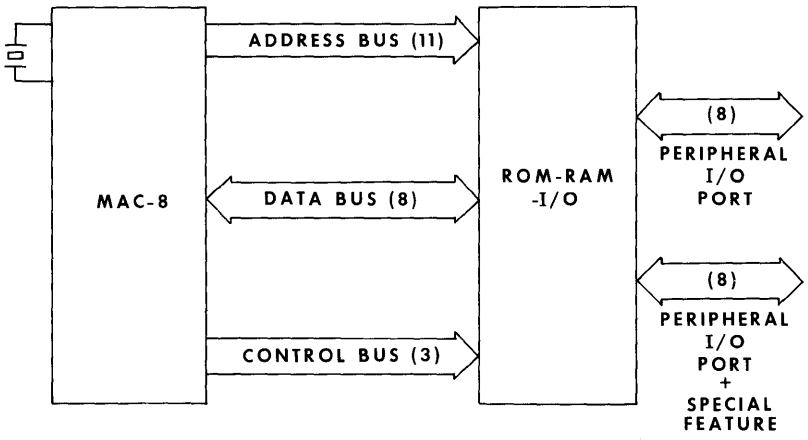


Figure 27-1 - Interconnection of a Single ROM-RAM-I/O Device

nected directly to their appropriate peripherals. However, if the special function is elected, numerous system configurations are available.

The third control register enables the special function features. Three bits within this register determine the operation mode of the interval timer. The first bit selects the source of the timer; the options are the MAC-8 clock or an external clock source. If the latter option is selected, the appropriate I/O pin would have to be connected to the external source. However, if the MAC-8 clock is selected, none of the I/O pins are needed to source the timer. The second special-function bit determines if the count-complete indication is connected to one of the output pins. The user may elect not to have this pin brought out since the timer can be read at any time. Furthermore, the chip has a readable flip-flop which is set when the counter has a full count. Hence, one can use the interval timer without dedicating any I/O pins to it, or have an external source and output indication. Any combination of the above is permissable. The third interval timer special-function bit determines whether or not the divide-by-8 prescaler is used by the counter. If the option is selected, the timer can be programmed to count up to 524, 288 (8 times 2 to the 16th power) counts; without the option, the timer can count up to a maximum of 65, 536 (2 to the 16th power).

The use of the interrupt flip-flop function is determined by a single bit in the special-function register. When this bit is set, two of the I/O pins are automatically connected to an on-chip edge-triggered flip-flop. The external device requesting the interrupt would be connected to the input pin. The output pin could be connected to the MAC-8 interrupt request pin or the MAC-8 could read the status of a second on-chip interrupt latch. When the MAC-8 responds to the interrupt (this event is indicated by a read of the vector at address FFFF), the edge-triggered flip-flop is automatically reset. The second interrupt latch is not reset by a read of FFFF but can be reset under software control. Hence, if more than one device requests the interrupt, the MAC-8 can poll these latches to determine the requester.

The remaining four bits in the special-function register determine which, if any, of the four strobes is activated. Selecting the strobe function allows the user to output a read/write pulse, rather than a level, on the peripheral I/O pins. When a strobe output is addressed the selected output pin will be complemented for the duration of the MAC-8 read or write strobe. Furthermore, when the strobe occurs, the ROM-RAM-I/O bus driver will enter the high-impedance state, allow-

ing communication between the MAC-8 and the peripheral. By use of these strobes the peripheral I/O capability can be expanded from 16 to 44 bits (8 bits/strobe times 4 strobes plus the remaining 12 I/O bits).

Finally, for protyping, the addition of a single gate is all that is required to disable the on-chip ROM. By adding this gate the user can still address the RAM and I/O sections of the chip; however, when ROM is addressed the ROM-RAM-I/O chip will be disabled, allowing external memory to be accessed.

INTERNAL ORGANIZATION

Figure 27-2 is a block diagram showing the preliminary placement of the ROM-RAM-I/O functions on the large scale integrated (LSI) chip. Approximately 75 percent of the chip area is occupied by the 1K bytes of ROM and the 96 bytes of static RAM with their associated access circuits. The ROM is realized in a 128-word by 64-bit array of n-channel ROM cells which are encoded at the thin-oxide mask level. A desired byte is accessed by the application of 11 bits of address from the MAC-8 address-bus. Seven of the bits are used in the 1:128 p-channel word decoder to select the desired word line. Three bits are used by the 8-wide 1:8 n-channel bit decoder to read the desired 8 of the 64 bits on the select word line. The remaining address bit is the ROM enable bit. The RAM is a 24-word by 32-bit array of complimentary static cells. The conventional 6-transistor cell with n-channel access switches is used. In order to conserve chip area, the word lines in the RAM array are slaved from the ROM. In this way the ROM and RAM share a single word decoder.

Figure 27-3 is a block diagram of the internal timer function. This timer consists of a 16-bit down counter with read and load capabilities to and from the MAC-8 data bus, a 3-bit prescaler, a zero detector, and flip-flop with an associated flag bit. The timer is controlled by three bits in the control register. One bit controls multiplexer A which determines if the clock input for the timer is derived from a peripheral I/O pin or, by default, from the system clock. The second bit controls multiplexer B which determines if the zero flip-flop is connected to the timer-output peripheral I/O pin. The third bit controls multiplexer C which determines if the clock input from the peripheral I/O pin is to be routed through the prescaler or sent directly to the counter. Note from Figure 27-3 that the default system clock is always routed through the prescaler.

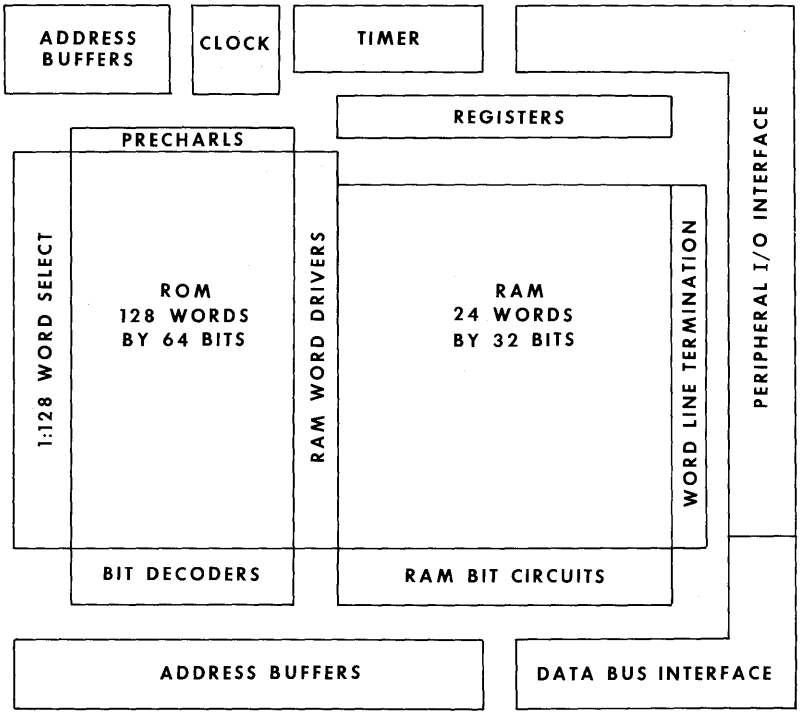


Figure 27-2 - Preliminary Placement of ROM-RAM-I/O Functions on LSI Chip

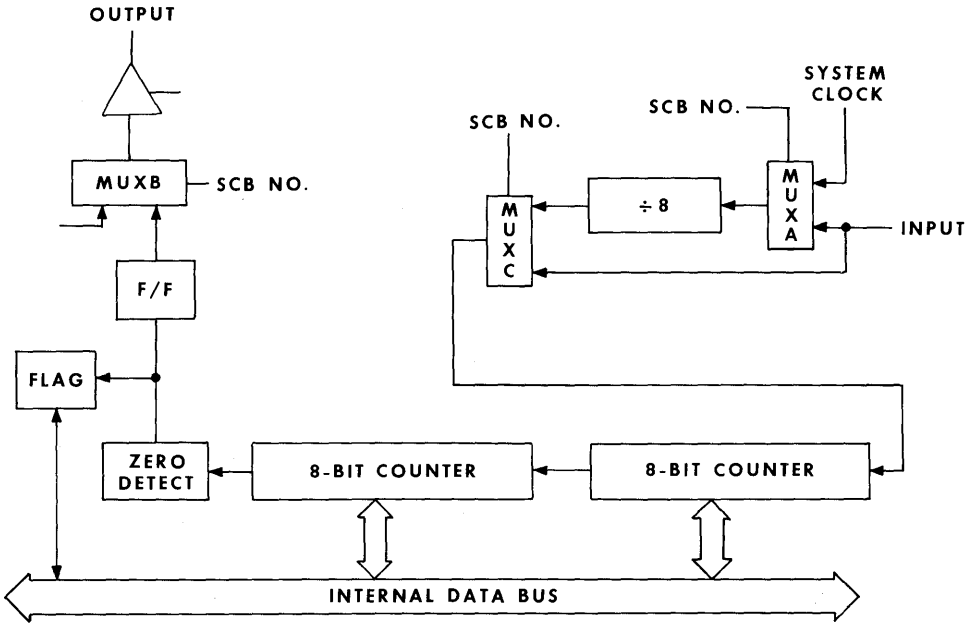


Figure 27-3 - Internal Timer Functions

The timer is loaded by writing two 8-bit timer locations in the ROM-RAM-I/O address space. These locations are arranged so that they can be accessed via a read-16 or write-16 instruction. Alternatively, either the lower or upper byte of the timer can be independently accessed by any MAC-8 instruction which accesses the appropriate ROM-RAM-I/O address space. When the counter is written, the prescaler is preset to all 1s as the starting count for the divide-by-8 countdown. In addition, the clocking of the main counter is inhibited when the counter is read. This prevents ambiguous data transfer resulting from partial propagation of borrow through the counter.

The zero detection, zero flip-flop, and timer function in a manner which is completely analogous to that of the interrupt circuit. The zero detector indicates the all 0s state in the timer and sets the zero flip-flop. This flip-flop provides a low going output level, and is reset during the MAC-8 interrupt sequence. The timer flag bit is also set by the zero crossing and may be read or written via the MAC-8 data bus.

SUMMARY

The ROM-RAM-I/O device described in this paper has been designed to operate in conjunction with a MAC-8 microprocessor. This microprocessor peripheral contains 1K bytes of ROM, 96 bytes of RAM, and 16 independent programmable peripheral I/O bits. This device provides, in addition to the normal I/O peripheral functions, special features which may be multiplexed onto the peripheral I/O leads. These added on-chip features include an interval timer, a MAC-8 interrupt control flip-flop, and peripheral strobe generators.

Connecting the ROM-RAM-I/O device with a MAC-8 forms a complete microprocessor system. In other words, a 2-chip MAC-8 microprocessor system is realizable. Larger systems can be formed by using more than one ROM-RAM-I/O chip as well as other memory and I/O devices.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of D. E. Blahut, W. F. Chow, V. K. L. Huang, and D. C. Stanzione.

PERIPHERAL TECHNOLOGY

A 4K RANDOM-ACCESS MEMORY

R. Pfahl, WE Dept 7112, NIW, LE, ILL
P. Kusulas, BTL Dept 5435, IH, ILL

ABSTRACT

This paper describes the effort required to qualify an external vendor as a supplier for a sophisticated large scale integrated (LSI) circuit - namely the commercial 18-pin 4K NMOS dynamic random-access memory (RAM). The RAM will be used in the new family of ESS processors. The technical team and testing facilities assembled for this project are described. The decisions made and the problems encountered in the development and production implementation of a KS specification (including reliability and parametric testing) are reviewed. It has been observed, for instance, that commercial parts often do not meet even the vendor's own published specifications. Also explained is the role of the source inspector and the vendor's response to a production "bust." The additional problems faced in qualifying second and third sources or new designs from a qualified manufacturer are delineated. Not only must the devices be qualified, but their performance in each unique system environment must also be characterized. A brief review is made of incoming inspection and field data acquired since KS 4K RAMs were first introduced into production in April, 1976. The 1976 and 1977 4K RAM requirement for Switching Equipment Division equipment is presented and the current availability discussed.

Steps are recommended which system designers should take when utilizing commercial LSI parts. These steps would ensure the procurement of reliable functional devices from multiple sources. This report concludes with a brief projection of the future evolution of semiconductor memory devices for ESS processor applications.

Bell Laboratories
Holmdel, New Jersey 07733