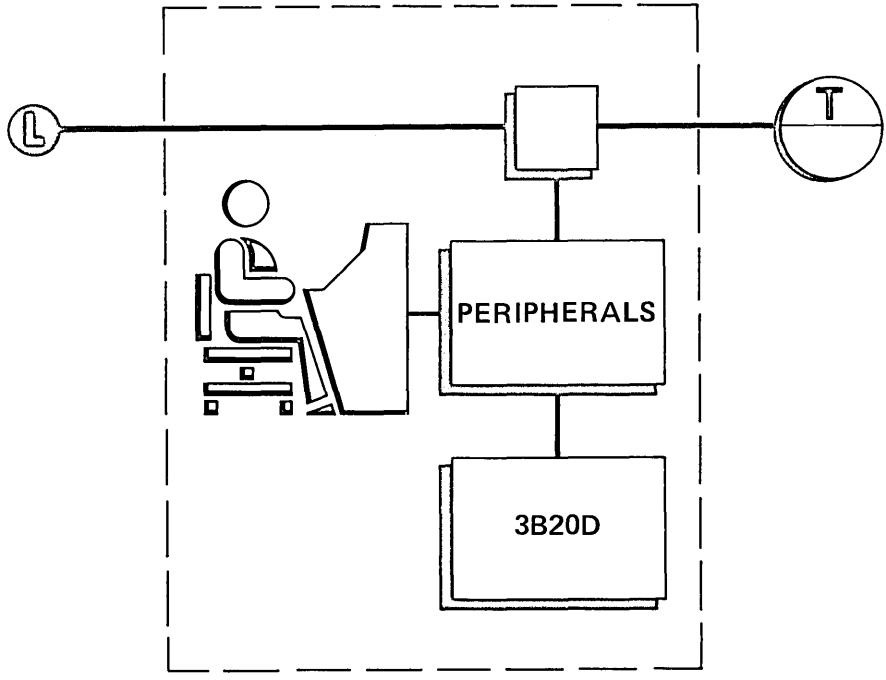
THE BELL SYSTEM TECHNICAL JOURNAL
MARCH 1983
VOL. 62, NO. 3, PART 3

# TSPS No. 1B




## *Expanded Capabilities With the 3B20D Processor*

# THE BELL SYSTEM TECHNICAL JOURNAL




**THE BELL SYSTEM TECHNICAL JOURNAL** (ISSN0005-8580) is published by the American Telephone and Telegraph Company, 195 Broadway, N.Y., N.Y. 10007; C. L. Brown, Chairman and Chief Executive Officer; W. M. Ellinghaus, President; V. A. Dwyer, Vice President and Treasurer; T. O. Davis, Secretary.

The Journal is published in three parts. Part 1, general subjects, is published ten times each year. Part 2, Computing Science and Systems, and Part 3, single-subject issues, are published with Part 1 as the papers become available.

The subscription price includes all three parts. Subscriptions: United States—1 year $35; 2 years $63; 3 years $84; foreign—1 year $45; 2 years $73; 3 years $94. Subscriptions to Part 2 only are $10 ($12 foreign). Single copies of the Journal are available at $5 ($6 foreign). Payment for foreign subscriptions or single copies must be made in United States funds, or by check drawn on a United States bank and made payable to The Bell System Technical Journal and sent to Bell Laboratories, Circulation Dept., Room 1E-335, 101 J. F. Kennedy Parkway, Short Hills, N.J. 07078.




Comments on the technical content of any article or brief are welcome. These and other editorial inquiries should be addressed to the Editor, The Bell System Technical Journal, Bell Laboratories, Room 1J-319, 101 J. F. Kennedy Parkway, Short Hills, N.J. 07078. Comments and inquiries, whether or not published, shall not be regarded as confidential or otherwise restricted in use and will become the property of the American Telephone and Telegraph Company. Comments selected for publication may be edited for brevity, subject to author approval.


Printed in U.S.A. Second-class postage paid at Short Hills, N.J. 07078 and additional mailing offices. Postmaster: Send address changes to The Bell System Technical Journal, Room 1E-335, 101 J. F. Kennedy Parkway, Short Hills, N.J. 07078.

# THE BELL SYSTEM TECHNICAL JOURNAL

DEVOTED TO THE SCIENTIFIC AND ENGINEERING ASPECTS OF ELECTRICAL COMMUNICATION

Volume 62 March 1983 Number 3, Part 3



## TRAFFIC SERVICE POSITION SYSTEM NO. 1B

***W. S. Hayward, Jr.,*** *Guest Editor*

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Overview and Objectives

By R. E. STAEHLER and J. I. COCHRANE

(Manuscript received June 30, 1982)

*This paper presents an overview and introduction to the detailed technical papers that describe the Traffic Service Position System No. 1B. The objectives and design philosophy are discussed and the overall organization of the system is described.*

## I. INTRODUCTION

### *1.1 Background*

In January 1969 the Bell System's first stored program controlled operator services system was introduced into the field. This system, named the Traffic Service Position System (TSPS) No. 1, employed a hardware/software architecture designed to permit the addition of new features to further automate operator functions as those features became technologically and economically viable.

#### *1.1.1 Initial capabilities*

The initial design[1] of TSPS No. 1 was developed to be used in conjunction with most local and toll switching systems in the Bell System. The system enabled customers to dial a number of calls heretofore only possible with operator assistance and thereby relieved the operators of many tedious operations demanded by cord switchboards. It automated routine operator functions for coin and noncoin calls, such as call timing and recording, recording of originating directory number, and recording and transmission of customer-dialed numbers.

Customers benefited from the advantages of the higher speed and increased accuracy of a stored program controlled system. The auto-

mation of operator functions was a great improvement over manual methods of number recording, timing, charge calculations, and billing.

Operating companies benefited because the automation and administrative features enabled operators to handle calls more efficiently and effectively, thereby reducing costs. The operator groups could be remotely located at convenient sites, and the entire work force could be managed more effectively.

Operators benefited from the more attractive working environment, the automatic equitable distribution of calls to their positions, and the satisfaction of improving their services to the customers.

#### *1.1.2 Technological advances*

The decade following the introduction of TSPS No. 1 brought major advances in technology in the areas of larger scales of integration in circuits, semiconductor memories, automated machine speech response techniques, microprocessors and miniprocessors, and advanced operating systems. A large number of these innovations have been incorporated into the operator services system as need and economic viability have been demonstrated.

#### *1.1.3 Additions to initial system capability*

Additional firmware and software[2-5] were continuously developed and introduced into the system (with new technology when applicable) to provide new and improved features. These include automation of hotel/motel charge quotation; partial automation of international call handling; time and charge quotations for noncoin, nonhotel calls; recording of charges for directory assistance calls; provision of service to remote locations; and automated charge quotation and coin collection for coin toll calls.

Two of the newest features introduced in 1979 and 1980, respectively, are automated verification of busy lines (with ensured privacy) and Automated Calling Card Service, which allows credit card calling without operator assistance from telephones using dual-tone multifrequency signaling.

#### *1.1.4 Rapid nationwide deployment*

Since its introduction in 1969, TSPS No. 1 has been rapidly deployed throughout the Bell System's nationwide telecommunications network to the point where there are more than 157 systems installed. Over 95 percent of Bell System customers and a large number of customers served by other companies are served by TSPS No. 1, providing almost universal availability of stored program controlled toll and assistance operator services.

### *1.1.5 TSPS No. 1 capability*

The continuing growth of operator services system traffic, plus the continuing addition of new features, have steadily reduced the remaining real-time capacity of the TSPS No. 1 processor, the Stored Program Control No. 1A (SPC 1A) Processor, in sites in the field. Since real-time processor capacity is a function both of the hardware configuration and of the call mix at an individual installation, a program called TSPS Real-Time Capacity Program (TSPSCAP) was developed to run on an off-line computer and permit the operating companies to verify the remaining capacity at each specific site.

Site-by-site surveys of the TSPSCAP results conducted in the mid-1970's indicated that some of the TSPS No. 1 sites were at their real-time capacity. Furthermore, a large number of additional sites would soon reach their real-time capacity.

Finally, this same growth of traffic and addition of features has exhausted the memory capacity at many sites because of the increased size of the software programs and data tables required.

### *1.1.6 Contemplated new network services*

Plans to utilize TSPS in the future[5] as an action control point in the emerging stored program controlled network dictated a need for new processor peripherals, such as a mass memory disk that cannot be provided by the current SPC 1A Processor.

## II. TSPS NO. 1B DEVELOPMENT

For the reasons given in the previous section, it was projected in the mid-1970's that a major evolution of TSPS No. 1 was necessary to provide continuing operator services for the 1980's and beyond. This major evolution was designated TSPS No. 1B.

### *2.1 TSPS No. 1B capability objectives*

The basic objective in the TSPS No. 1B development was to increase capability compared to TSPS No. 1.

Studies of network growth and economics led to the conclusion that an appropriate call capacity objective for TSPS No. 1B would be a design with an initial objective of 1.6 times that of the TSPS No. 1. Studies also indicated that to provide adequate office data and program storage, a reasonable objective for the physical memory of TSPS No. 1B would be four times that provided by TSPS No. 1.

The TSPS No. 1B also had to eliminate those functional restrictions in TSPS No. 1 that limited its evolution as an action control point in the stored program controlled network. This could be accomplished

by the provision of processor peripherals such as random access bulk storage and general-purpose data links.

### 2.2 TSPS No. 1B architecture objectives

The new features demanded by the utilization of TSPS in the stored program controlled network also required that TSPS No. 1B provide a flexible architecture for future developments.

The TSPS No. 1B design had to incorporate a high-level language and an advanced operating system not only to expedite introduction of new features, but also to maximize the productivity of the software developers.

Since there are over 150 TSPS No. 1 sites in the field, the size of the investment in TSPS No. 1 software programs and peripheral hardware is very large. To preserve the majority of this large investment, the existing software and peripheral hardware must be retained. At the same time, new high-level software and additional peripheral hardware must be added to the system for new features. Thus, the TSPS No. 1B design must permit the use of the new high-level language in parallel with the continued availability of the existing assembly language development environment.

An important objective was the ability to upgrade the TSPS No. 1 sites in service that were limited by capability exhaust without interrupting call processing.

### 2.3 Other objectives

Besides additional capabilities and a flexible architecture, other basic objectives were also set for the TSPS No. 1B development.

The TSPS No. 1B must provide dependable operator service 24 hours a day. This requirement was converted to a specific design objective of reducing service interruptions that result in a total loss of service to an average of less than 3 minutes per year per system.

To be cost effective, the TSPS No. 1B should take advantage of the newest technology and reduce energy consumption and floor space usage.

### 2.4 TSPS No. 1B design approach

To meet these objectives for the TSPS No. 1B, the basic design approach implemented was to replace the existing processor with a new processor while retaining both peripheral hardware and existing software. The most efficient way to retain the software was for the new processor to provide emulation capability. Further, peripheral interface hardware was needed in order for the new processor to execute existing software and control the existing peripheral hardware.

Where required, software could be written in the new high-level language.

## III. 3B20D PROCESSOR

The 3B20D Processor[6,7] was under development at the time the TSPS No. 1B development was initiated. It is one of a family of general-purpose processors designed not only to meet the real-time demands and dependability requirements of switching systems, but also to be versatile enough for a broad spectrum of present and future telecommunications applications.

A task force formed in 1976 to evaluate different processors concluded that the 3B20D Processor would meet all of the objectives stated in Section II and recommended that the 3B20D be utilized in the TSPS No. 1B design.

In brief, the 3B20D Processor is significantly faster than the SPC 1A Processor and has four times the physical memory capacity of the TSPS No. 1 processor. It provides additional processor peripherals such as random access bulk storage disks and general-purpose data links. The 3B20D Processor is designed with a flexible architecture and an advanced operating system, the Duplex Multi-Environment Real-Time (DMERT) operating system, and supports the high-level C programming language and a robust software environment. Finally, it provides switching system dependability, and is designed with large-scale integrated circuit technology, thus reducing energy consumption and size.

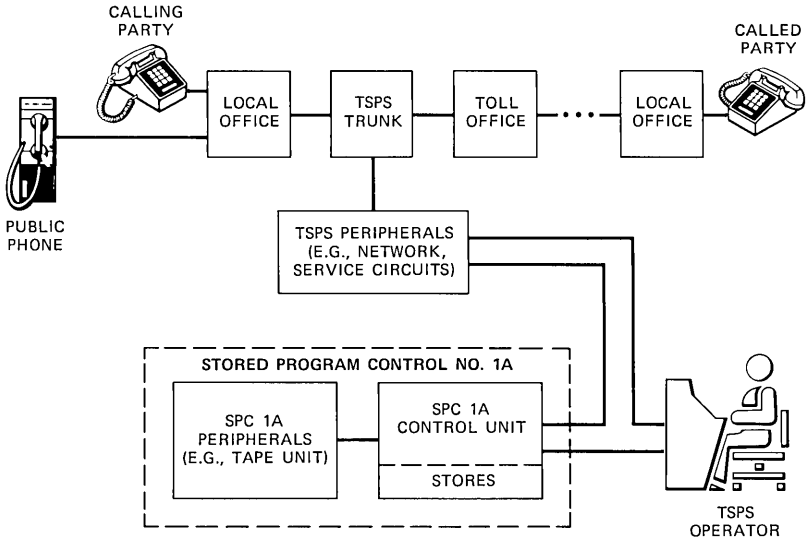
## IV. ORGANIZATION OF TSPS

### *4.1 TSPS No. 1 organization*

As we see in Fig. 1, TSPS No. 1 bridges an operator or service circuits onto a trunk connecting local and toll offices. TSPS No. 1 has dedicated specialized trunk circuits and peripherals (including the link network, service circuits, signal distributors, and scanners), a Stored Program Control No. 1A (SPC 1A) Processor, and operator consoles. Once the connection between customers is established, connections to operators or service circuits are released. Such connections are reestablished if additional services are required.

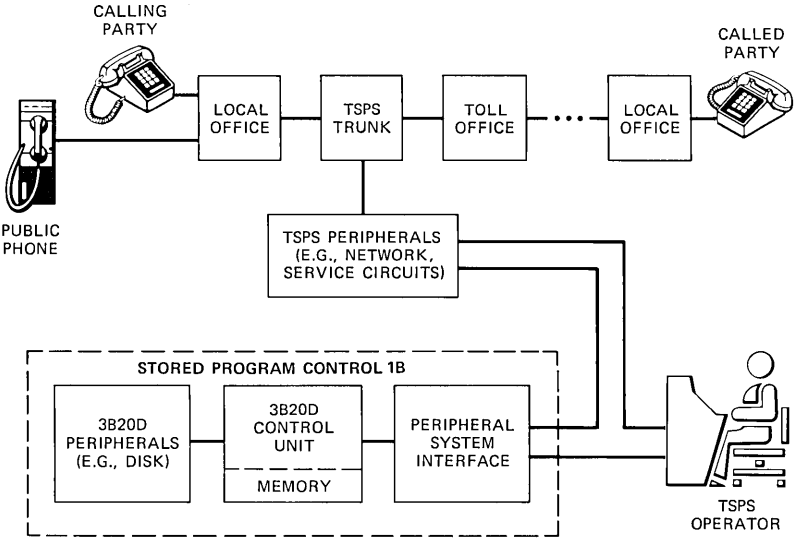
### *4.2 TSPS No. 1B organization*

As we see in Fig. 2, the general architecture of TSPS No. 1B is the same as TSPS No. 1, except that the SPC 1A is replaced by the Stored Program Control 1B (SPC 1B). The SPC 1B includes a 3B20D Processor, a Peripheral System Interface (PSI) to interface the existing TSPS peripherals to the 3B20D Processor, and various 3B20D Proc-




SPC – STORED PROGRAM CONTROL
TSPS – TRAFFIC SERVICE POSITION SYSTEM

Fig. 1—TSPS No. 1 organization.




TSPS – TRAFFIC SERVICE POSITION SYSTEM

Fig. 2—TSPS No. 1B organization.

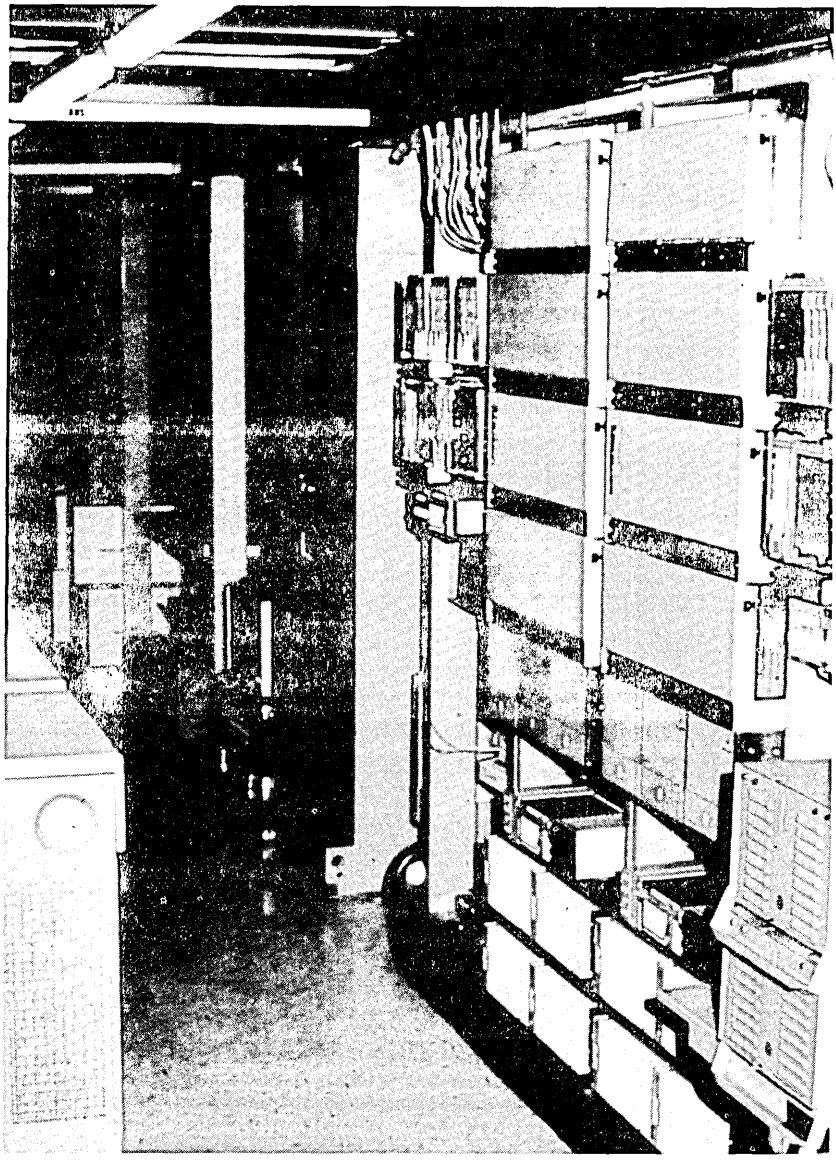


Fig. 3—3B20D Processor in the new TSPS No. 1B service installation at Fresno, California.

essor microcode and native-mode software to emulate the TSPS No. 1 environment.

The articles in this issue covering the TSPS No. 1B[8-16] include detailed descriptions of the system, the software architecture and

hardware configuration, the software development system, integration and testing of the TSPS No. 1B, retrofit procedures, capacity and reliability evaluation, Switching Control Center System interface, and long-range planning for TSPS No. 1B installation and growth.

## V. TSPS NO. 1B STATUS

### *5.1 New service installations (cordboard replacements)*

The first TSPS No. 1B was placed in service in Fresno, California, in November 1981. The 3B20D Processor in the TSPS No. 1B in Fresno is shown in Fig. 3. The Operator Services Center at Fresno is shown in Fig. 4.

The second TSPS No. 1B was placed in service in San Antonio, Texas, in January 1982.

### *5.2 Retrofit service installations (processor replacement in an in-service system)*

The first TSPS No. 1B retrofit or processor replacement in an in-service TSPS No. 1 took place in Redwood City, California, in March 1982.

### *5.3 Continuing deployment*

As of December 1982, 37 TSPS No. 1B's were in service. All but two of the TSPS No. 1B installations were retrofit installations to provide urgently needed expansion of either call capacity or memory capacity. Continuing retrofit deployment is scheduled for 1983 and beyond.

## VI. PERFORMANCE

Performance data from all TSPS No. 1B sites indicate that all design objectives have been achieved. More details are covered in later articles.

## VII. FUTURE TRENDS

The increased capacity of TSPS No. 1B will accommodate anticipated growth in operator services traffic for a number of years. In addition, an advanced operating system and high-level programming language will allow TSPS No. 1B to evolve readily to: (1) support the evolution of the stored controlled network,[5] (2) continue to improve operator efficiency, and (3) respond to changes caused by the evolving restructure of the telecommunications industry.

## VIII. SUMMARY

This paper has presented a general background for TSPS No. 1B as an introduction to the technical papers that follow. While all design

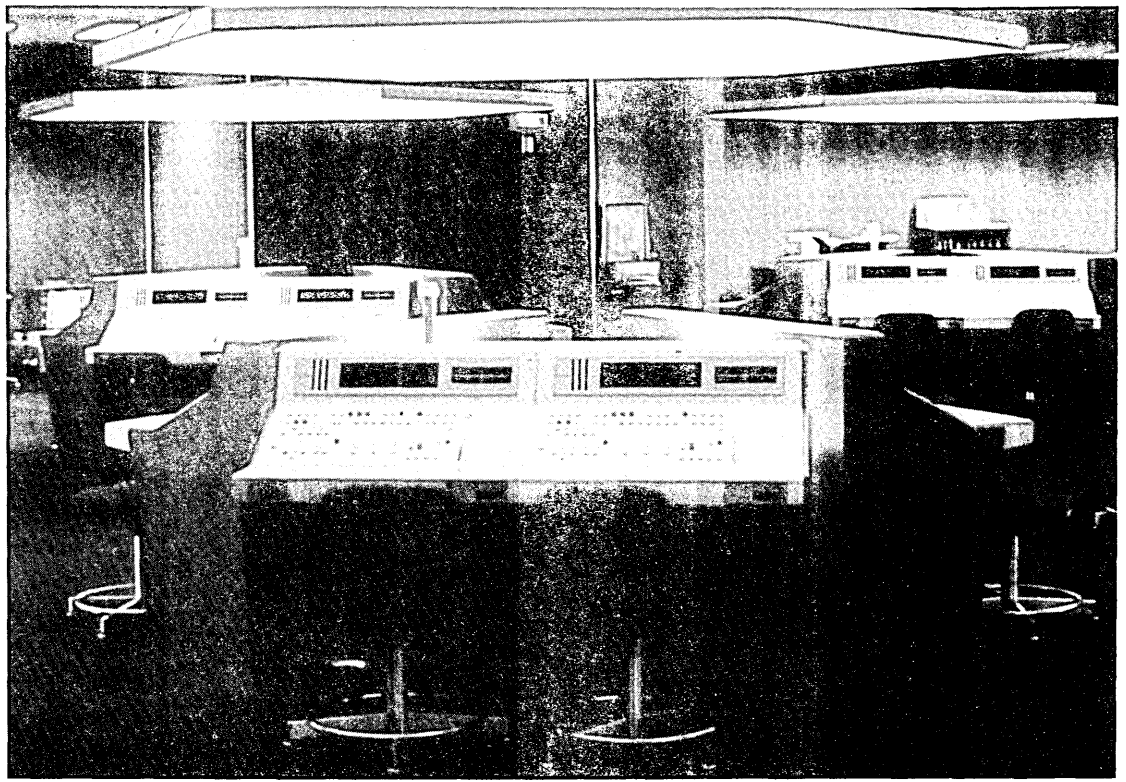


Fig. 4—Operator Service Center in Fresno, California.

details could not possibly be included, the papers in this issue provide a comprehensive overview of TSPS No. 1B.

## IX. ACKNOWLEDGMENTS

The development of TSPS No. 1B required the participation of hundreds of people in many organizations in Bell Laboratories, Western Electric, AT&T, and the operating companies. All of the authors in this issue are indebted to these organizations for their cooperation and the team effort that culminated in the successful completion of the TSPS No. 1B project. The authors of this paper also wish to acknowledge the contributions of all the team members whose work is summarized here, as well as the support of D. J. Leonard, K. E. Martersteck, and C. M. Rubald and L. C. Stecher, the coordinating editors.

## REFERENCES

1. R. J. Jaeger and A. E. Joel, Jr., "TSPS No. 1—System Organization and Objectives," B.S.T.J., special issue on TSPS No. 1, *49* No. 10 (December 1970), pp. 2417–43.
2. R. E. Staehler and W. S. Hayward, Jr., "TSPS No. 1 Recent Developments: An Overview," B.S.T.J., special issue on TSPS No. 1, *58*, No. 6 (July 1979), pp. 1109–18.
3. E. M. Prell, V. L. Ransom, and R. E. Staehler, "The Changing Role of the Operator," 9th Int. Switching Symp. Rec. (May 1979), pp. 697–703.
4. J. C. Kylin, E. M. Prell, and R. P. Weber, "Benefits of Integrating Data Bases into the SPC Network," Internal Conf. on Commun. Rec. (June 1979), pp. 3.4.1–4.
5. S. Horing, J. Z. Menard, R. E. Staehler, and B. J. Yokelson, "SPC Network: Overview," B.S.T.J., special issue on SPC Network, *61*, No. 7, Part 3 (September 1982), pp. 1579–88.
6. N. X. DeLessio, J. R. Kane, M. W. Rolund, J. M. Scanlon, and R. E. Staehler, "The 3B Processor System and Its Application to TSPS," 10th Int. Switching Symp. Rec. *3* (September 1981), Session 33A, Paper 4, pp. 1–9.
7. J. M. Scanlon, et al., "3B20D Processor and DMERT Operating System: Prologue," B.S.T.J., special issue on 3B20D Processor, *62*, No. 1, Part 2 (January 1983), pp. 167–9.
8. N. X. DeLessio and N. A. Martellotto, "Traffic Service Position System No. 1B: System Description," B.S.T.J., this issue.
9. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.
10. H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
11. T. G. Hack, T. Huang, and L. C. Stecher, "Traffic Service Position System No. 1B: Software Development System," B.S.T.J., this issue.
12. R. Ahmari, R. S. DiPietro, S. C. Reed, and J. R. Williams, "Traffic Service Position System No. 1B: Integration and System Testing," B.S.T.J., this issue.
13. J. C. Dalby, D. Van Haften, and L. A. Weber, "Traffic Service Position System No. 1B: Retrofitting the Processor," B.S.T.J., this issue.
14. B. A. Crane and D. Suk, "Traffic Service Position System No. 1B: Capacity and Reliability Evaluation," B.S.T.J., this issue.
15. J. J. Bodnar, J. R. Daino, and K. A. VanderMeulen, "Traffic Service Position System No. 1B: Switching Control Center System Interface," B.S.T.J., this issue.
16. P. L. Bastien and B. Wycherley, "Traffic Service Position System No. 1B: TSPS No. 1/1B Long-Range Planning Tools," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# System Description

By N. X. DeLESSIO and N. A. MARTELLOTTO

(Manuscript received June 30, 1982)

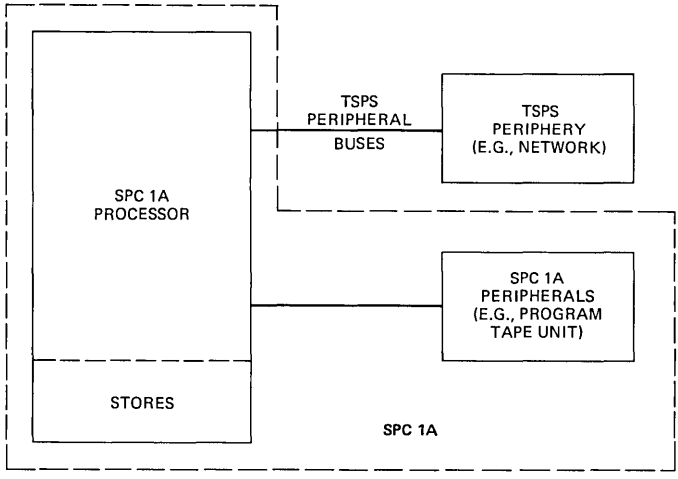
*The Traffic Service Position System No. 1B (TSPS No. 1B) is the first field application of the Bell System's new 3B20 Duplex Processor (3B20D) in the emulation mode. The 3B20D Processor replaces the existing Stored Program Control No. 1A (SPC 1A), while retaining the existing TSPS periphery and software. A key factor is the ability to switch between the emulated software and the 3B20D native software within a single process with a single instruction. This allows the flexibility of adding software in either environment as appropriate.*

### I. INTRODUCTION

The 3B20 Duplex Processor (3B20D), with its associated Duplex Multi-Environment Real-Time (DMERT) operating system, meets the objectives of Traffic Service Position System No. 1B (TSPS No. 1B).[1-3] The 3B20D Processor is significantly faster than the Stored Program Control No. 1A (SPC 1A) and provides the required increase in processor capability. The ability of the 3B20D Processor to emulate allows the retention of most of the existing TSPS No. 1 software and peripheral hardware. In addition, the 3B20D Processor consumes much less energy and is significantly smaller in physical size than the SPC 1A.

### II. SYSTEM STRUCTURE

The 3B20D Processor replaces the existing SPC 1A (Fig. 1) while retaining the existing TSPS No. 1 periphery and—through emulation—preserving the existing TSPS software. This software preservation is accomplished by defining (through microcode) one of the four




SPC – STORED PROGRAM CONTROL
TSPS – TRAFFIC SERVICE POSITION SYSTEM

Fig. 1—Existing TSPS system structure.

3B20D instruction sets to be that of the SPC 1A, thus emulating the old processor and allowing existing TSPS software to be transported to the 3B20D Processor almost intact. The ability exists to switch between the emulated instruction set and the 3B20D native instruction set within a single process with a single instruction. This allows the flexibility of adding new software into either environment, as appropriate. For example, some new TSPS No. 1B software, generated to take advantage of the new disk capability and to provide a new interface to maintenance personnel, is written in the C programming language,[4] which compiles into 3B20D native-mode assembly language. Both emulated and native-mode software are run under the DMERT operating system, allowing operating system services to be available to both forms of software. The emulated SPC 1A assembly language code is structured as a single process executing under DMERT.

The existing TSPS periphery is retained and interfaced with the 3B20D Processor through the use of a Peripheral System Interface (PSI) circuit. This unit is designed to interface the TSPS peripheral buses with the 3B20D Central Control Input/Output (CCIO) bus via an Application Channel Interface (ACHI). The PSI duplicates the signals, timing, and error checking of the SPC 1A. This enables TSPS peripheral units to remain unchanged. Future hardware can be added to the existing TSPS peripheral buses or can utilize the I/O processor of the 3B20D Processor to off-load the main processor and to provide fast block data transfers through direct memory access.

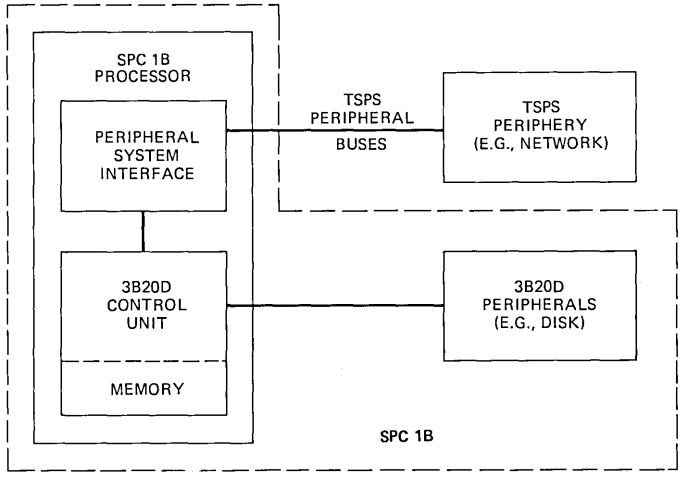
The combination of the 3B20D, PSI, and microcode required to

emulate the SPC 1A is called the SPC 1B (Fig. 2) and is the entity that replaces the SPC 1A. Because the processor replacement is economically attractive both for new installations and for retrofits, techniques have been developed to replace an SPC 1A with an SPC 1B in an in-service office. The resultant system provides significant increases in call processing and main memory capacities, allows the preservation and future growth of existing software and peripheral hardware, and adds modern software and hardware architectures to facilitate future feature introduction.

## III. PROCESSOR HARDWARE DESCRIPTION

The 3B20D Processor has been developed as the first member of a family of processors designed for a broad range of Bell System applications. The DMERT operating system provides a comprehensive set of functions associated with management of system resources such as the real-time, memory, input/output, and software processes.

The control unit of the 3B20D (Fig. 3) uses a 32-bit architecture throughout, including the memory buses to main store and an 8K-byte cache. Extensive self-checking logic is employed to ensure immediate detection of errors, thus supporting quick and graceful recovery measures. Hamming correction of all single-bit errors and detection of all double-bit errors are performed by the main memory controller. In addition, data parity is checked on every refresh operation required for the dynamic random access memories (RAMs) used in main




SPC – STORED PROGRAM CONTROL
TSPS – TRAFFIC SERVICE POSITION SYSTEM

Fig. 2—TSPS No. 1B system structure.

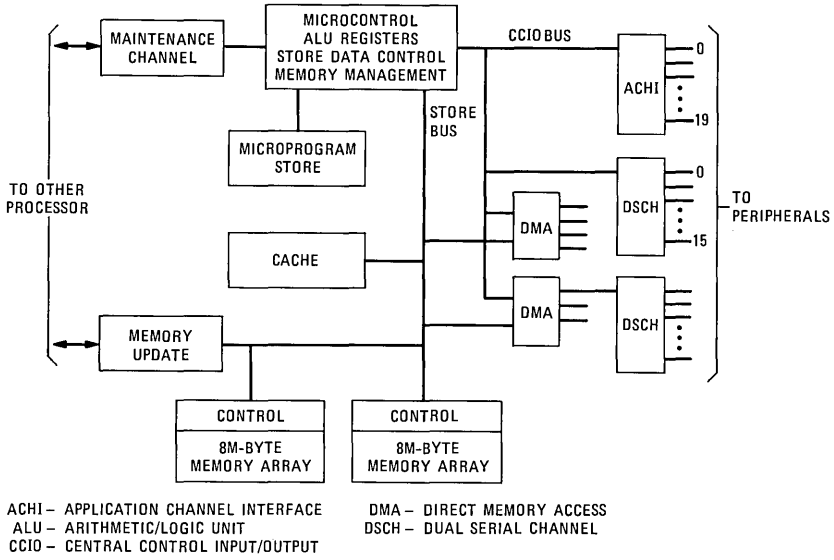



Fig. 3—3B Processor block diagram.

memory, thus ensuring that even infrequently used addresses are periodically checked for integrity. The 3B20D Processor uses a 24-bit virtual address, which is converted to a 24-bit physical address using a paged segmentation scheme. The 16M-byte address space is divided into 128 segments, each having up to 64 pages of 2K bytes each. Memory protection can be provided on either a segment or a page basis. Physical memory is growable in 512K-byte increments, to a maximum of 16M bytes. The main-memory access time is 525 nanoseconds, while the cache access time is 250 nanoseconds. Memory communication provides a byte-addressing capability with byte, half-word, full-word (32-bit), and move block options as part of the instruction repertoire. A memory management unit performs virtual-to-physical address mapping and main-store access protection. A high-speed, two-way, set-associative memory called the Address Translation Buffer (ATB) is provided to reduce the overhead associated with the address translation function. The ATB is divided into eight sections that are assigned to processes by software.

The Central Control (CC) is microprogrammable with the capability of executing a variety of instruction sets. Up to four instruction sets can be selected dynamically. The microstore uses a 64-bit word length with up to 16K words of high-speed, bipolar programmable read-only memory (PROM) or RAM available. A variable microcycle ranging from 150 to 300 nanoseconds is employed to optimize execution times. The native instruction set of the 3B20D Processor was designed to be

compatible with the C programming language. It optimizes the execution and memory-space utilization of the language while including instruction-level support for all C-language data types and control structures.

Figure 4 is a general block diagram of the 3B20D Processor. Two basic connections exist between the duplicated Control Units (CUs). One is an update connection that serves to keep the off-line CU's memory completely up to date. The second connection is a maintenance channel over which diagnostics of the off-line CU are performed. The Central Control and memory are duplicated and grouped as a switchable entity. The I/O and disk systems can be accessed by either CU through duplex intelligent controllers. The Disk File Controllers (DFCs) are normally both active in order to keep the data on the disks identical. Thus, under trouble conditions, either disk can support system operation. Unlike the SPC 1A Processors, the CUs are not run in a synchronous matching mode. Instead, both stores (on-line and standby) are kept up to date by the memory-update hardware concurrent with instruction execution. This is achieved by having the on-line memory-update circuit write into both memories simultaneously when memory data are written by the CC. Under trouble conditions, when control is switched to the standby CU, its memory will contain up-to-

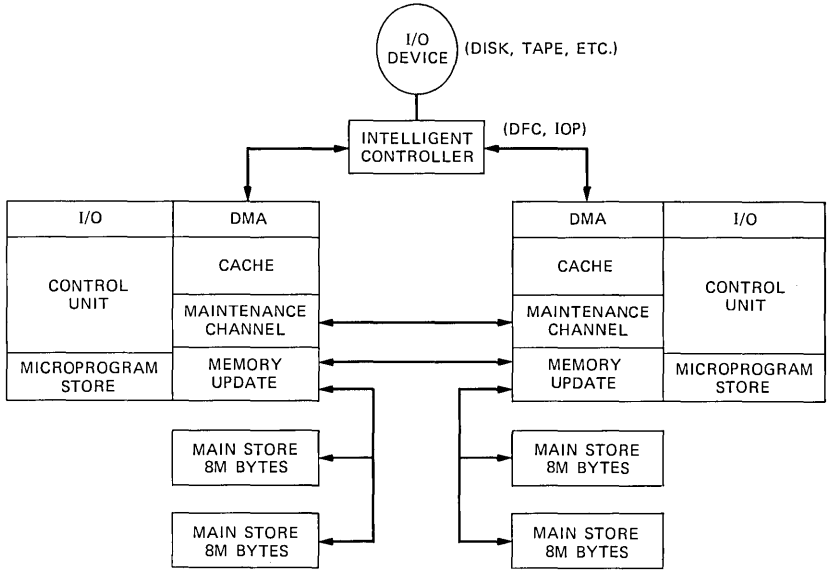



Fig. 4—General block diagram of the 3B20 Duplex Processor.

date information without performing a complete transfer from one CU to another.

3B20D Processor peripheral units are connected to the CC via the Direct Memory Access (DMA) unit. The DMA does not interface directly with peripheral units, but rather communicates with two intelligent subsystems, a Disk File Controller, and an Input/Output Processor (IOP). The CC builds job blocks for a peripheral unit that in turn notifies the CC upon job completion. The parallelism afforded by the autonomous processing capabilities of the DFC and IOP frees the CC for other work. Communication to both the DFC and the IOP is via Dual Serial Channels (DSCH) that allow any peripheral to operate with either CC of a duplex pair. Each DFC is capable of supporting 16 movable-head disk drives of 300M-byte capacity. Each IOP is capable of supporting a wide variety of peripherals, such as nine-track tape units, printers, synchronous and asynchronous data links, maintenance terminals, scanner/signal distributors, and custom network interfaces. Peripherals also may be connected to the Central Control Input/Output (CCIO) bus via an interface such as the Application Channel Interface (ACHI) used to communicate with the TSPS No. 1 periphery that was retained.

## IV. OPERATING SYSTEM DESCRIPTION

DMERT is the real-time operating system for the 3B20D Processor. This operating system was developed concurrently with the hardware and uses a multiple-environment approach where time-critical code coexists with time-shared software.[5] That is, one environment supports real-time response while another environment provides a time-shared interface similar to that of the *UNIX** operating system.[6] The architecture of the operating system is process oriented; this allows applications to write software at the level most productive for each task. A process is an instance of a program executing on the processor and is characterized by a separate virtual address space. The multiple environments are implemented via a kernel that supports three levels of processes, described below. The DMERT kernel provides the most primitive virtual machine as it handles hardware interrupts, timer interrupts, and operating system traps. In all cases, the kernel saves the state of the interrupted process, provides whatever service is requested, and then restores the state of the interrupted process. The first level of process, known as a kernel process, is offered limited services by the DMERT kernel and is dispatched because of real-time events such as interrupts. The second level of process, known as

* Trademark of Bell Laboratories.

supervisor, is offered more services by the DMERT kernel in the area of I/O and dynamic memory allocation. Supervisor processes support a third level of process called a user-level process. An example of this is a *UNIX* operating system process controlled by a kernel-level process. In order for processes to cooperate in accomplishing their tasks, DMERT provides a set of interprocess communication and synchronization mechanisms including messages, events, process ports, interprocess traps, and shared memory. These interprocess communication primitives are fundamental to the DMERT structure.

I/O is accomplished by a kernel process, known as a driver. A unique driver is provided according to the type of I/O device, such as disk or IOP. Drivers receive I/O requests in the form of messages. When the I/O is completed, the message is returned to the requesting process. An intermediate supervisor process known as the file manager stands between disk users and the disk driver. This process implements a logical file system on the disk for those processes that care to use it. Files can be created, opened, closed, grown, and deallocated through the file manager. Hierarchical directories of files such as those found in the *UNIX* operating system are supported.

A set of processes and a special IOP peripheral controller provide a modern craft interface for the 3B20D and the TSPS No. 1B. A split-screen cathode ray tube (CRT) and a printer interface are utilized. The top portion of the screen is the status and display portion. Various status and display pages can be called upon demand. A special page that provides basic machine-control features such as initialization requests is provided by peripheral firmware. The lower portion of the screen is utilized for terminal I/O messages.

The continuous operation aspects of the 3B20D Processor are supported by a number of processes that are an integral part of the DMERT operating system. These processes handle error interrupts, control processor switches, and provide I/O to common peripherals such as disks; they also run equipment diagnostics and audit key data structures for consistency. Reference 7 contains a detailed description of both the 3B20D Processor and DMERT.

## V. SOFTWARE STRUCTURE

The software structure of the TSPS No. 1B system was governed by three major design goals. First, the software architecture had to maximize the call-handling capacity. Second, steps were taken to maintain the SPC 1A programming environment as closely as possible in order to allow maximum utilization of existing software support utilities. Third, the existing TSPS software was to be emulated with a minimum of modifications. This guideline precluded unnecessary redesigns or restructures of the current field-proven software.

The emulation of the SPC 1A at the instruction level by the microcoding of its instruction set and at the system level by the PSI has allowed most TSPS programs to be executed on the 3B20D Processor with minimal modifications. The TSPS emulated code has been incorporated into a single, large, high-priority kernel process under DMERT.

The single process structure was dictated primarily by the existing tightly coupled nature of the TSPS software. As on the SPC 1A, all TSPS data and programs share and have access to the entire address space and communicate through data structures that reside in memory. Retaining these structures maintained the goal of exact emulation and avoided interprocess communication overhead detrimental to achieving the required performance gain.

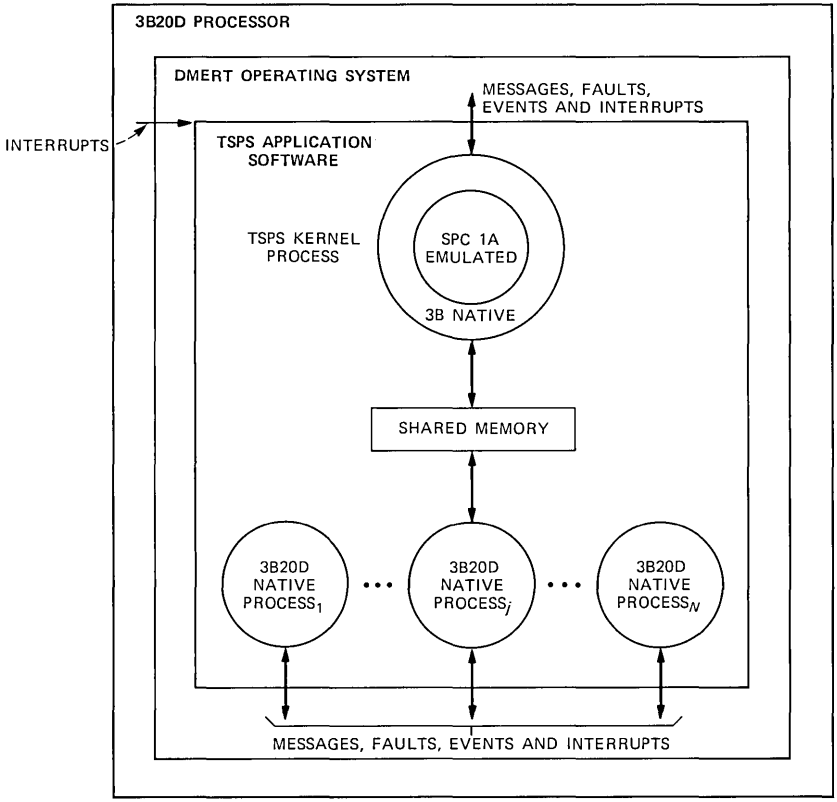
In addition to the emulated code, the TSPS kernel process contains native-mode (C-language) code that provides several capabilities. First, it provides a standard C-language data structure interface to the operating system and to other processes. Native code resident in the TSPS kernel process also works in conjunction with emulation microcode to implement system-level emulation of the SPC 1A interrupt structure within the actual interrupt structure defined by the 3B20D and DMERT. In addition, native code resident in the TSPS kernel process has eliminated the need to introduce new instructions, such as system calls, not available in the emulated instruction set.

All entries into the TSPS kernel process are through native code, which then calls the emulated code as a subroutine with a single instruction. Certain functions exist in the current TSPS software that would have required extensive modifications to emulate because of machine dependencies. In these cases, new replacement native code was written as subroutines that are called with a single instruction from the emulated code. In cases where completely new functions were implemented, native-mode processes separate from the TSPS kernel process were generated. An example of this is the TSPS File System Interface, which provides disk file system access as a replacement for functions previously implemented using the program tape unit of the SPC 1A.

The control structure of the TSPS call processing and peripheral maintenance software, being emulated, is almost identical to what exists on the SPC 1A. The major difference is that rather than running continuously as on the SPC 1A, the high-priority TSPS kernel process voluntarily must give up control of the machine periodically to allow lower priority processes to run. This is done by requesting periodic time-outs from DMERT. After the specified time has passed, the TSPS kernel process is reentered. The native code responds to the event and causes the emulated code to resume where it had left off.

The real-time breaks are not noticeable to the emulated software. The combined DMERT and TSPS priority structure is such that the TSPS kernel process dominates control of real time.

The process structure of TSPS No. 1B is summarized in Fig. 5. The DMERT operating system running on the 3B20D Processor provides a high-level, multiprocess environment for TSPS application processes. In this environment, the TSPS kernel process appears to the DMERT operating system to be identical to other native-mode processes that utilize the facilities of messages, faults, events, and interrupts in communicating with the operating system and—through the operating system—with other processes. The TSPS kernel process, in some instances, also communicates with other TSPS application processes through shared memory. An example, noted above, is the TSPS File System Interface. Within the TSPS kernel process, the combination




SPC – STORED PROGRAM CONTROL
TSPS – TRAFFIC SERVICE POSITION SYSTEM

Fig. 5—TSPS No. 1B process structure.

of emulation microcode, PSI, and native code creates an SPC 1A environment, thereby shielding the emulated code from the details of the specific machine and operating system on which it is running.

## VI. SUMMARY

The processor capability was increased by replacing the existing SPC 1A with the 3B20D Processor while retaining the existing TSPS periphery and using emulation to preserve the existing TSPS software. A key aspect in the software architecture is the ability to execute either the emulated instruction set or the 3B20D native instruction set, and to switch between the two within a single process with a single instruction. Thus, it is possible to add new software to either environment and thereby increase the future flexibility of the system. Because the processor replacement is economically attractive both for new installations and for retrofits, techniques have been developed to replace an SPC 1A in an in-service office.

## VII. ACKNOWLEDGMENT

The design of the 3B20D Processor and TSPS No. 1B required the cooperative efforts of a large number of people in Bell Laboratories, Western Electric, and AT&T. The authors wish to acknowledge the contribution of all of the team members whose work is summarized here.

## REFERENCES

1. R. E. Staehler, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
2. N. X. DeLessio, J. R. Kane, M. W. Rolund, J. M. Scanlon, and R. E. Staehler, "The 3B Duplex Processor System and Its Application to TSPS," 10th Int. Switching Symp. Proc., September 21–25, 1981.
3. N. A. Martellotto, "An Operating System For Reliable Real-Time Telecommunications Control," Fourth Int. Conf. on Software Eng. for Telecommun. Switching Systems Proc., University of Warwick, England, July 20–24, 1981.
4. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
5. H. Lycklama and D. L. Bayer, "*UNIX* Time-Sharing System: The MERT Operating System," B.S.T.J., *57*, No. 6, Part 2 (July 1978), pp. 2049–86.
6. D. M. Ritchie and K. Thompson, "The *UNIX* Time-Sharing System," B.S.T.J., *57*, No. 6, Part 2 (July 1978), pp. 1905–29.
7. J. M. Scanlon, "3B20D Processor & DMERT Operating System: Prologue," B.S.T.J., *62*, No. 1, Part 2 (January 1982), pp. 167–9.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Real-Time Architecture Utilizing the DMERT Operating System

By R. J. GILL, G. J. KUJAWINSKI, and E. H. STREDDE

(Manuscript received June 30, 1982)

*The Traffic Service Position System No. 1B (TSPS No. 1B) architecture was conceived to increase performance significantly for future features and traffic growth. The design preserves the TSPS No. 1 software with minimal changes. At the same time, the 3B20 Duplex Processor (3B20D) used in TSPS No. 1B provides additional processor peripherals and a modern programming environment with a real-time operating system. This paper describes how the TSPS No. 1 software, initially designed to run on the Stored Program Control No. 1A (SPC 1A), executes on the SPC 1B of the TSPS No. 1B. The SPC 1B is a 3B20D tailored for the TSPS application and provides an SPC 1A environment by directly emulating its instruction set. The paper also presents major TSPS application processes and their interaction with the emulated TSPS process and the Duplex Multi-Environment Real-Time (DMERT) operating system of the SPC 1B. In addition, the paper describes the integration of TSPS maintenance software into the DMERT maintenance structure.*

## I. INTRODUCTION

The Traffic Service Position System No. 1B (TSPS No. 1B) real-time architecture was designed to meet the project goals discussed in Ref. 1. The implementation of this architecture entailed four major developments:

(*i*) Replacement of the Stored Program Control No. 1A (SPC 1A) of TSPS No. 1 with the 3B20D Processor, the TSPS Peripheral System Interface (PSI), and microcode to execute the SPC 1A instruction set,

which together comprise the Stored Program Control No. 1B (SPC 1B)

(*ii*) Emulation of most existing TSPS No. 1 software structured as a process under the Duplex Multi-Environment Real-Time (DMERT) operating system

(*iii*) Development of additional processes to support the emulation

(*iv*) Integration of the PSI and TSPS peripheral maintenance into the overall DMERT maintenance structure.

Before discussing the TSPS No. 1B real-time architecture, this paper presents two sections of background information. Section II presents an overview of how TSPS operates using the SPC 1A. Section III reviews the fundamentals of DMERT, and Section IV describes the SPC 1B and the TSPS No. 1B software architectures. These sections enable the reader to understand the real-time architecture of the TSPS No. 1B. More detailed information can be obtained by reading the references.

## II. TSPS NO. 1 REAL-TIME ARCHITECTURE

This section presents a brief overview of the TSPS No. 1 operation on the SPC 1A and provides a base for understanding how the TSPS No. 1 was emulated on the TSPS No. 1B. A complete description of TSPS No. 1 operation on the SPC 1A can be found in Refs. 2 and 3.

### *2.1 SPC 1A programming environment*

The SPC 1A uses 20-bit addresses to reference approximately one million 20-bit words of main memory. The address spectrum consists of up to 30 store name codes with each name code containing 32K 20-bit words. Store name code 0 is not used since low memory addresses are mapped into the SPC 1A's buffer bus (see Section 2.2). Store name code 31 (037) is not implemented. Hence, the maximum physical memory size for the SPC 1A is 960K 20-bit words. The SPC 1A does not support memory management. Hence, there is no virtual addressing; all addresses are physical addresses. All of memory is equally accessible (shared) by all programs.

Write protection can be set on a 2K word boundary within a name code. Protected areas within the name code, however, must be contiguous, and there can be only one protection change boundary within a name code. Memory is unprotected from the high end of the address spectrum within each name code. For example, if one fourth of a name code is to be unprotected (read/write), then the first three quarters would be read only, and the last quarter would be unprotected. Protected areas can be unlocked to change programs or fixed data by having the processor execute a special unlocking sequence. Typically, protected areas are used for office data, read-only tables, and program

logic. Unprotected areas are used for volatile data (e.g., call information).

SPC 1A instructions are 40 bits wide (two 20-bit words). The odd-addressed word contains the operation code, while addresses or data, when present, are placed in the even-addressed word. There are relatively few instructions, but a single instruction can have many options and perform several operations. Each instruction takes from one to three 6.3-microsecond machine cycles to execute. The execution time of each instruction is fixed regardless of the number of options specified. Hence, the time a segment of code will execute can be precisely determined.

### 2.2 Buffer bus

The SPC 1A buffer bus is an internal collection of processor and peripheral status and control registers. The buffer bus registers are used for such things as peripheral and processor configuration control and status indications, interrupt sources, and interrupt masks. The buffer bus consists of 24 registers with 20 to 24 bits each. Each register has a low-memory address (e.g., 600 or 2200) associated with it. All buffer bus registers can be read with Memory to Register (MR) instructions. Some registers can be written, set, or reset using Register(s) to Memory (RM or RRM) instructions or Constant to Buffer Bus (CBB) instructions. Other registers are read-only.

### 2.3 Interrupt structure

#### 2.3.1 Interrupt levels

There are nine interrupt levels in the SPC 1A. These interrupt levels are A (highest priority), B, C, E, F, G, H, J, and K. A tenth level, L (commonly referred to as base level), runs continuously in the absence of any interrupt. Normally, base level is only interrupted every 5 ms by J-level. Although, base level is the lowest-priority processing level, the bulk of TSPS software (e.g., call processing, diagnostics, audits) executes in base level.

Each interrupt level can only interrupt lower-priority levels with the sole exception being that A- and B-level can interrupt each other. An A-level interrupt is caused either by a manual action at the Maintenance Control Center (MCC) Control and Display (CD) frame or by the execution of an ANOP instruction (used to fill unused instruction space). B-level is entered as a result of a processor switch or for emergency actions required as a result of system-sanity-check failures. C- and K-level interrupts generally occur as a result of processor errors. C-level handles SPC 1A Processor fault recovery. K-level interrupts are only enabled during processor diagnostics for error-data recording. SPC 1A store errors will generate E- and G-level interrupts.

E-levels also result from software errors such as invalid addresses or protection violations. E-levels provide data for immediate problem analysis and necessary store reconfiguration. G-level is used for collecting store-error data for intermittent failures. Peripheral-unit errors will generate an F-level interrupt. F-level software is dedicated to peripheral fault recovery. J- and H-levels perform the system I/O.

#### 2.3.2 J- and H-level interactions

The J-level interrupt is generated every 5 milliseconds by a clock pulse. Because of its periodic nature, J-level provides the main time reference for all application processing. Within J-level there are two classes of jobs: high priority and low priority. The high-priority jobs are executed first. While they are running, H-level is inhibited. In making the transition to low-priority jobs, H-level is enabled. H-level programs consist of the high-priority, J-level jobs. An H-level is caused whenever J-level executes longer than 5 ms and low-priority jobs are being run (see Fig. 1*). This ensures that the high-priority jobs are executed every 5 ms, unless H-level runs longer than 5 ms. In this case the high-priority jobs would miss an execution, as H-level cannot interrupt itself.

J- and H-levels on the SPC 1A have two separate interrupt sources that are driven by the same clock pulse. The H-level interrupt source is only enabled when low-priority J-level is executing. When a return from interrupt is performed from low-priority J-level, the H-level inhibit bit is set.

#### 2.3.3 Interrupt handling

When an interrupt occurs under normal conditions on the SPC 1A, the processor saves the address of the interrupted program and passes control to the first instruction of the interrupt program for that level. The address of the interrupt program and the save area (bin) for the interrupted program address are known (i.e., hard-wired) by the processor. The interrupt-handling program for each interrupt, except J-level, first saves the contents of the seven general-purpose registers in memory and then determines what actions to take. Because of the frequency of the J-level interrupt (every 5 ms), general-register contents are copied to an auxiliary set of registers by the hardware. The overhead of saving and restoring the registers every 5 ms is greatly reduced by performing this function in hardware.

The return from interrupt is normally performed by the Execute Go Back To Normal (EGBN) instruction. The one exception is J-level. J-

* Acronyms and abbreviations used in the figures and text are defined in the Glossary.

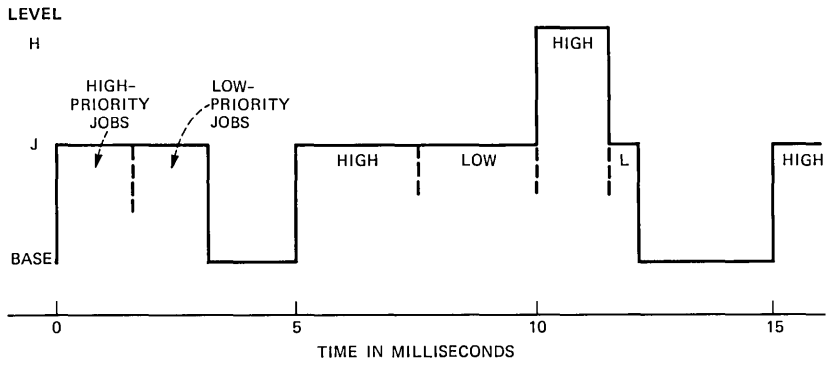



Fig. 1—5-ms input/output processing.

level uses the Go Back To Normal H- or J-level (GBNHJ) instruction. In returning, the interrupt handler (except for J-level) first restores the general-purpose register contents from memory (if it were going to return to the point of interrupt) and then executes the EGBN. The EGBN instruction determines what interrupt level is being returned from by looking at the interrupt-level activity flags in the buffer bus. It restores the address of the interrupted program from the save area for that interrupt and clears the highest interrupt bit in the buffer bus interrupt-level activity word, thus dropping to the next highest, previously active level. If an intermediate-level interrupt is pending, it would be serviced at this time. The GBNHJ is slightly different in that it causes the general-purpose registers to be restored by hardware from the auxiliary set of registers. It also disables the H-level interrupt in addition to performing the functions of the EGBN.

Any interrupt servicing routine can change its point of return by overwriting the saved address in the appropriate interrupt bin in memory. Also, an interrupt is effectively returned from by merely clearing its activity bit in the buffer bus, which essentially erases the history of the interrupt. This immediately puts the program into the next lower, previously active level. This re-enables that interrupt level as well as any intervening levels.

#### *2.3.4 Interrupt inhibiting*

Interrupts are normally only masked when the program is handling a higher-priority interrupt. The interrupt activity bits in the buffer bus inhibit lower-priority interrupts from occurring. E-, F-, H-, and J-level interrupts can also be individually inhibited by setting the specific inhibit bits in the buffer bus. The H-level inhibit should always be set except when in low-priority J-level. The J-level inhibit can be set for brief periods when executing "critical region" code in base level, which

should not be interrupted by J-level activity because of potential interference. Similarly, the H-level inhibit could be set in low-priority J-level to prevent H-level interference. Some instructions have an "inhibit I/O interrupt" option that inhibits H- and J-level interrupts until the completion of the following instruction. For some instructions this option is implicit (not an option, but always on). The E- and F-level inhibits can be set manually as well as by software. C-level interrupts can be inhibited by software, and A- and B-level interrupts can be manually inhibited from the MCC.

### 2.4 Program structure

J-level (and H-level) programs execute under control of the Executive Control for I/O (ECIO) program. For the most part ECIO passes control to a predetermined and fixed set of programs every 5 ms. The list of jobs varies from execution to execution, but is cyclical over a 300-ms interval for high-priority work and over a 200-ms interval for low-priority work. Thus, J-level jobs execute with frequencies that are multiples of 5 ms, ranging up to 200 or 300 ms. Some program executions are permanent (always active) while others are run only on demand.

Similar to J-level, base-level programs are run under control of the Executive Control for the Main Program (ECMP) routines. Base-level programs execute in one of six priority classes. These priority classes are interject (highest priority), A, B, C, D, and E. ECMP passes control to jobs in priority classes A through E with a relative frequency of 15:8:4:2:1, respectively (see Fig. 2). Jobs within each priority class can run every execution of the priority class or only as needed on demand. Each priority class has a set of task dispensers that pass control to individual programs (tasks) when work is to be performed.

Interject work is requested when certain immediate actions are required (e.g., to immediately unload a full input hopper) or at a fixed frequency on demand (i.e., J-level-initiated sanity check of interject operation). A check for interject requested work is performed at least once in every base-level loop (one E-priority class to E-priority class cycle) and after every task. Interject should be serviced quickly. Hence, all base-level tasks are designed to run in short segments. Figure 3 illustrates the base-level priority-class execution with the fixed-interject check and another representative interject job executing in the middle of a priority class.

### 2.5 System integrity

The objective of system integrity is to provide an uninterrupted call-processing environment. The reliability goal for TSPS No. 1 is to achieve less than three minutes per year of total system outage

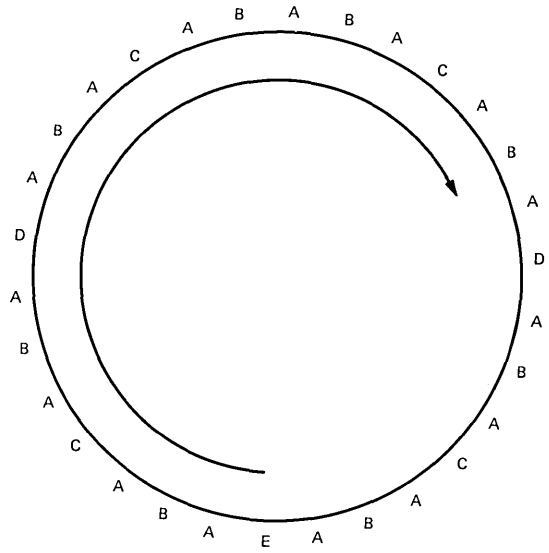



Fig. 2—TSPS No. 1 base-level loop.

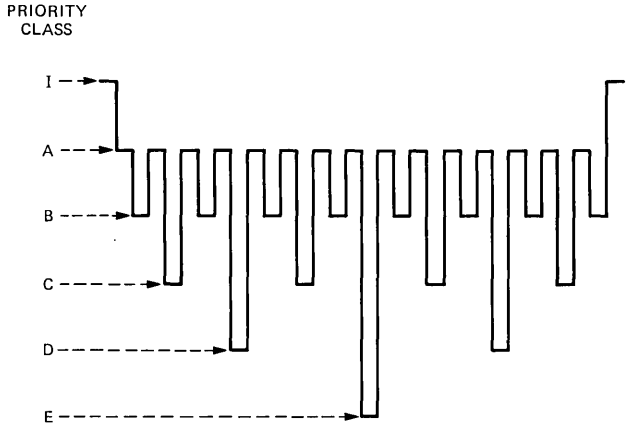



Fig. 3—TSPS No. 1 base-level, priority-class frequency.

averaged over all systems. This goal covers outages attributable to any possible cause. Hence, the term "system integrity" is used in this and other sections of this paper to encompass software stability as well as the traditional hardware reliability.

### *2.5.1 Hardware integrity*

The maintenance strategy for TSPS hardware is based on the duplication of all critical units. This hardware redundancy allows

faulty units to be switched out of service with the load being carried by the remaining good unit. Maintenance programs are organized on priority levels such that the faulty unit can be removed from service as soon as possible, and then later, while the system is processing calls, the faulty unit is diagnosed in order to isolate the failing circuit pack. There are three main types of maintenance programs for the TSPS peripherals. These are, in order of decreasing priority, fault-recognition programs, diagnostic programs, and exercise programs.

Fault-recognition programs run when the presence of a fault is detected. They determine which of the duplicated units is in error, and reconfigure the system around the problem. Before returning to call processing, the fault-recognition program initiates a diagnostic request on the unit suspected of malfunctioning. The purpose of the diagnostic program is to provide resolution of the fault by indicating to the craft personnel the smallest replaceable unit (e.g., a circuit pack). This is accomplished by running a series of tests on the suspected hardware unit and then comparing the actual test results with a set of expected values. Another method of detecting faults employs the use of exercises. These programs are similar to diagnostics in that they run selected tests on the hardware. They differ in that they are intended to find faults in circuits not exercised by normal system operation (e.g., by call processing). Unlike diagnostics that are initiated by fault-recognition programs upon detection of a fault, the exercise programs are scheduled periodically.

### *2.5.2 Software integrity*

***2.5.2.1 Initialization and recovery.*** Whenever the state of the software is such that normal processing cannot continue, call-processing recovery actions are taken in an attempt to restore the system's sanity. The least severe actions are taken first. If these fail, the recovery attempt is escalated to the next highest level. The five TSPS recovery phases are Minor Audits, Major Audits, Selected Audits (miniphases), System Initialization A (SIA) and System Initialization B (SIB). Except for Selected Audit phases, these recovery phases can be manually requested or automatically generated by the software. Selected Audits phases can only be initiated by software, but they can be manually inhibited.

All audits that run during a recovery phase are "stitched" together. This means that they are run consecutively. Meanwhile, the normal base-level priority-class execution and all call processing is temporarily suspended. Minor Audit phases are short and, as a result, they have the least effect on call-processing activity. Major Audit phases are more extensive, and, hence, have a more disturbing effect. Selected Audit phases are run as the result of problems detected by software

sanity checks. A specific set of audits are stitched together, depending on the error found. SIAs and SIBs run a complete set of audits. An SIA also zeroes most unprotected memory and initializes the hardware. An SIB is more extensive in that it performs a more thorough hardware initialization. If an SIB fails to restore system sanity, another SIB will automatically be taken with a different hardware configuration. Looping SIBs with hardware reconfigurations will continue indefinitely until the system is recovered or manual intervention takes place. For the Minor, Major, and SIA phases, recovery actions are identical whether the phase is software generated or manually requested. On the other hand, a manually requested SIB will zero all unprotected memory, but it will not cause a hardware reconfiguration.

***2.5.2.2 Reference returns.*** Under certain conditions, maintenance-interrupt routines (levels A through K) must return to H-level, J-level, or base-level at a reference point in the job administration stream when a return to the interrupted point is not warranted. This safe transfer of control is called a reference return. The base-level reference return will result in the base-level cycle restarting at the end of E priority. The H- and J-level reference returns result in the cancellation of the job being run at the time of the interrupt, but the remaining scheduled jobs are executed.

***2.5.2.3 Sanity.*** Sane program execution is monitored via a hierarchical scheme. Base level checks itself every E-priority class by determining that the various priority classes have been run the proper relative number of times (15:8:4:2:1). Base-level sanity is checked in J-level by requesting an interject job every 500 ms and monitoring its execution. Failure to execute interject suggests a base-level loop. High-priority J-level (and, hence, H-level) monitors low-priority J-level by monitoring the execution of a fixed 100-ms job. High-priority J-level (and H-level) has the responsibility to reset a hardware sanity timer every 500 ms. Failure to reset the timer within 640 ms or resetting it too soon (less than 320 ms) will cause the timer to generate a B-level interrupt. Low-priority J-level insanity will result in a minor audit call-processing phase. Continued interject response failures or base-level priority class execution insanity will also trigger a minor phase. High-priority J-level also monitors system sanity by checking the average time of the last three E-E cycles. If this time exceeds a threshold a minor phase is taken. Lower threshold crossings will trigger overload recovery actions.

***2.5.2.4 Overload strategy.*** If at any point the elapsed time to run the last three E-E cycles exceeds a minimum threshold, phase 1 overload actions are taken. These actions consist of gradually busying trunks back to the local offices and reducing the rate at which processing of new calls is allowed to begin. If during phase 1 overload the elapsed E-E times exceed another, higher threshold, phase 2 actions are taken.

These actions busy all trunks to the local offices except for a minimum number and inhibit the processing of any new calls in the system. As E-E times return to acceptable levels, the system returns to phase 1 actions and eventually to normal. The return to normal operation (unbusying trunks and increasing the new call-processing rate) is done gradually as the overload subsides.

## III. DMERT OPERATING SYSTEM

The DMERT operating system[4] evolved from the MERT[5] operating system. While MERT was designed to operate on a simplex minicomputer, DMERT has incorporated maintenance software to control the duplex hardware in order to provide Electronic Switching System (ESS) reliability on the 3B20D Processor.

### *3.1 Processes*

An executable entity under DMERT is called a process. A process is a collection of programs and data with a distinct virtual address space (see Section 3.2) that is executed as a unit to perform a single (or set of related) function(s). Once in execution, a process controls the scheduling of its internal routines, barring any external stimulus such as an interrupt or fault. While the process is executing it appears to have an entire (virtual) machine to itself, although it may be interrupted by higher-priority processes or even swapped out to disk while waiting for some event to occur (e.g., the completion of an I/O request). The process address space may be completely protected from access by other processes or it may be shared at will. One process can communicate with another via several mechanisms supported by DMERT, as described in Section 3.6.

### *3.2 Memory management*

The basic unit of memory handled by the 3B20D's memory management system is a page: 2K 8-bit bytes (five hundred twelve 32-bit words). A segment consists of 1 to 64 pages that need not be contiguous in physical memory. A process in DMERT consists of at least three segments, where one segment contains the process stack, another contains the process control block (PCB), and at least one segment contains executable code. Each process executes in its own logical (or virtual) address space, which may be as large as 16M bytes (4M words). Memory management swaps processes between memory and disk, enabling many processes to coexist even though the sum of their memory requirements exceeds the physical memory of the processor. It also provides protection from misuse (i.e., writing into read-only memory) and unauthorized access by other processes.

The information required to perform virtual address to physical

address translation is maintained by DMERT in segment and page tables in memory. Accessing these tables for every address translation would take a considerable amount of time (6.8 microseconds each). To speed up this process a high-speed associative memory called an address translation buffer (ATB) is used to keep the most recently used translation data. There are eight 64- × 2-word-set associative memories. Four of the ATBs can be dedicated to individual processes. One other is dedicated to the kernel. The other three are shared dynamically by all other processes. There is one each for kernel, supervisor, and user processes. These processes are described in Section 3.3. Address translation via the ATB is performed in 150 nanoseconds.

### *3.3 Abstract machines*

DMERT supports four levels of software. Each level provides a different abstract view of the machine to the software. These abstract machine levels are:

(*i*) The kernel
(*ii*) Kernel processes
(*iii*) Supervisor processes
(*iv*) User processes.

The kernel is the lowest abstract software level under DMERT and the core of the operating system. It provides the basic services of the operating system, such as interrupt control, process dispatching, scheduling, and timing. It essentially extends the set of operations for kernel and supervisor processes.

The kernel-process level is used for those processes that have stringent timing constraints and must respond rapidly to real-time stimuli such as interrupts. Also, processes that must directly interact with hardware devices (such as a peripheral-unit driver) are coded as kernel processes. Similar to the kernel, kernel processes can have direct hardware access. Kernel processes also share some system data (e.g., the kernel stack and message buffers) with the kernel. Other system data are accessed via kernel services. Because of their performance requirements, kernel processes are totally memory resident and cannot be swapped out to disk. Some DMERT kernel processes (e.g., the process manager and memory manager) are referred to as special processes and share the kernel address space.

The next highest abstract machine level is for supervisor processes. This level is for those processes that neither have stringent timing constraints nor require direct access to the hardware or system data. Access to hardware (i.e., for I/O) and to system data is provided to supervisor processes by the kernel and kernel processes. The hardware and system data are completely shielded from supervisor processes.

Because of this layered software structure, errors in supervisor processes are much less likely to have catastrophic system effects. The price for this protection, however, is slower response time and performance. To improve response time supervisor processes have the option of being memory resident and not being swapped out to disk.

Both kernel and supervisor processes have the option of being nonkillable. A nonkillable process must perform its own internal fault recovery. A killable process can be terminated and recreated under certain error conditions.

The highest and most abstract software level under DMERT is the user-process level. User processes exist only in conjunction with a supervisor process, and in effect are just a unique state of that supervisor. The user portion of the process, however, does have a separate virtual address space distinct from the supervisor portion. A supervisor process can gain access to its user address space, but the reverse is not true. As a result, user processes are totally removed from the details of the actual machine and operating system under which they execute. Hence, the user level is the easiest programming level. However, user processes have poorer performance than supervisor processes.

### *3.4 Interrupt structure*

Interrupts are detected between the execution of two instructions and change the sequence of execution. More specifically, an interrupt results in the interruption of the current executing process and a transfer of control to a specific interrupt-handling process. The state of the interrupted process is saved on the interrupt stack so that it can be restored at the completion of the interrupt processing and the interrupted process can resume execution. There are 32 maskable, hardware interrupt sources contained in the interrupt source (IS) register. The 3B20D also has four unmaskable interrupt sources. Interrupts can be generated by hardware (i.e., clocks and peripheral devices), microcode, and software. Corresponding to the IS is an interrupt mask (IM) register. The IM and IS registers are "anded" together to determine which interrupts are allowed to occur between any two instructions.

Table I shows the layout of the IS as used in TSPS No. 1B. Bit 0 is the highest-priority interrupt source. That is, if more than one interrupt source is set and unmasked, the lowest-order bit position will be serviced first. Of particular interest are the Program Interrupt Request (PIR) sources (bits 17 through 31). There is one PIR per DMERT execution level 1 through 15 (see below). A PIR is set in response to a process request to send an event or fault (discussed later). The PIR corresponding to the execution level of the receiving process is set.

Table I—DMERT interrupt source register

| Bit | Use |
|---|---|
| 0–1 | Hardware errors |
| 2 | Software errors |
| 3 | Unused |
| 4 | PSI errors |
| 5 | Timer (10 ms) |
| 6 | TSPS (5 ms) |
| 7–9 | Unused |
| 10–13 | Direct Memory Access Input/Output (DMA I/O) |
| 14–16 | Non-DMA I/O |
| 17–31 | Program Interrupt Requests (PIRs) |

Since supervisor and user processes always receive their events at level 1, PIR 0 is not required. For user processes, the associated supervisor process receives events and faults and deals with them appropriately on behalf of the user process.

Currently, the only use of the unmasked interrupt sources is for Operating System Traps (OSTs). The execution of an OST by a process causes this interrupt to be set, which results in the kernel gaining control of the machine (the kernel services this interrupt source) and providing the requested service. In some instances the service is provided by a kernel process. *UNIX** operating system user-level services are provided by a DMERT supervisor process. In these latter two cases the kernel passes control to the appropriate process via its OST entry.

### *3.5 Execution levels and priorities*

DMERT prioritizes processes into sixteen execution levels (see Fig. 4). Execution level 15 is highest in priority. Kernel processes run at execution levels 2 through 15. Application kernel processes can only use execution levels 3 through 14. Level 2 is reserved for DMERT special processes, and level 15 is used by the DMERT Timer and Error Interrupt Handler. Supervisor and user processes run at execution level 0, and supervisor processes in a critical region run at level 1. Running at level 1 prevents any other supervisor process from interrupting the supervisor process while in its critical region. Within execution level 0, all supervisor and user processes are further prioritized within a 256-level priority structure. Supervisor and user processes are scheduled on a highest-priority basis by the DMERT scheduler. Processes of equal priority are scheduled among themselves using a round-robin scheme.

Kernel processes are dispatched as a result of an interrupt. This

* Trademark of Bell Laboratories.

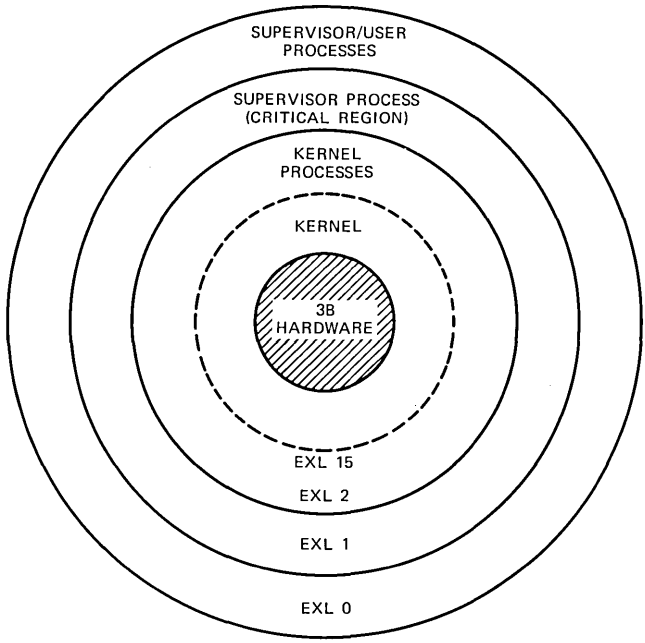



Fig. 4—Hierarchical organization of DMERT.

interrupt may be from a peripheral device or simply a PIR indicating the reception of an event or fault. Each execution level has a unique IM associated with it. The IM for each execution level inhibits all interrupts that are handled at the same or lower execution level. Since supervisor and user processes run at execution levels 0 and 1, they are always preempted by kernel processes.

### *3.6 Interprocess communication*

DMERT provides several mechanisms for interprocess communication. These include events, messages, faults, shared memory, and shared files.

#### *3.6.1 Events*

An event is a single bit of information having a predefined and agreed upon meaning between cooperating processes. When an event is sent from one process to another, DMERT will set the PIR interrupt for the level at which the receiving process executes. When that PIR is unmasked (the current execution level specifies an IM that has that PIR unmasked), DMERT will handle the PIR interrupt and dispatch the process for which the event was intended at its event entry. The type of event(s) sent is passed as a parameter.

#### 3.6.2 Messages

A message is a mechanism for transmitting multiple words of data between cooperating processes. The content of the message must be predefined and understood by sender and receiver alike. The reception of a message is indicated to the receiving process by a message event. Except for kernel processes, the contents of a message buffer (the data being passed) is copied from the sender's address space into the message buffer and then to the receiver's address space by various DMERT operations. All kernel processes include the system message buffer segments in their virtual address space. Hence, transmission of messages between kernel processes is much more efficient as the data is actually passed in shared memory.

#### 3.6.3 Faults

Faults are another mechanism for interprocess communication that are usually used to indicate an error, system initialization, or other emergency-type of communication. A fault consists of an 8-bit (byte) fault code that has a predefined meaning between sender and receiver. The reception of a fault causes a process to be dispatched at its fault entry.

#### 3.6.4 Shared memory

Processes can share memory on a segment basis. The entire segment (up to 32K words) would be mapped into the virtual address space of each process. This is the most efficient means of interprocess communication, but it may result in tight coupling between the sharing processes.

#### 3.6.5 Shared Files

Sharing a file is very similar to sharing memory except the storage is done on a secondary storage device (specifically a disk). This mechanism is obviously not as efficient as memory, but provides a media for sharing larger amounts of data.

### 3.7 DMERT system integrity

#### 3.7.1 Structure and strategy

The DMERT integrity package is based on the duplex, self-checking, nonmatching philosophy of the 3B20D Processor, and the hierarchical organization of the DMERT operating system. It was designed to provide tolerance to both hardware and software faults, so that the 3B20D Processor running under DMERT meets its reliability requirements. The functions provided by the integrity package running under DMERT appear in all abstract machine levels. Their placement in the hierarchical structure depends upon their complexity, the desired real-time response, and the services that they require.

The nondeferrable functions are activated when a hardware or software fault has been detected or a maintenance request has been made via a TTY message. They may initialize one or more system components, reconfigure the system, or just generate status reports. Some nondeferrable functions (e.g., processor initialization) cannot assume operating system sanity and require a fast response time. They are placed in the kernel and are initiated by the hardware self-checking circuits of the processor. Other functions like the recovery from a fault in an on-line peripheral can assume operating system sanity, but they must be performed in the minimum amount of time. Consequently, they are implemented as kernel processes.

Some functions such as diagnostics require services provided by lower abstract machines and their execution can be defferred. These execute under a DMERT supervisor process that provides a *UNIX* operating system environment. The deferrable integrity functions include the initiation and control of the diagnostics, administration of diagnostic requests, and requests to remove or restore a unit to service.

### *3.7.2 Software integrity*

***3.7.2.1 Overload and sanity.*** The focal point of the software integrity package of DMERT is the System Integrity Monitor (SIM). SIM is a kernel process that is responsible for, among other things, system overload control, sanity monitoring, and coordination of initialization actions. SIM coordinates its actions with an application process called the Application Integrity Monitor (AIM). The combined action of these two processes defines the overall software integrity strategy. Further discussion on overload and sanity is presented below as part of the TSPS No. 1B system integrity, which includes a description of the AIM process.

***3.7.2.2 Initialization levels.*** DMERT provides six levels of initialization (levels 0–5). The application can specify sublevels for DMERT levels 0–4. Level 0 is for application initialization only. That is, DMERT does not perform any initialization; DMERT merely notifies the application to initialize. Level 1 results in DMERT initializing and then notifying the application to initialize. In this case, all processing in the machine comes to a halt, the kernel and interrupt stacks are cleared, interrupt sources are disabled, and all kernel processes as well as the currently running supervisor process are faulted. Level 2 is a reboot of the system. At this level, however, certain protected segments of memory specified by the application are not lost, nor is the system Equipment Configuration Data (ECD). A Level-3 bootstrap reloads the ECD from disk, while a Level-4 bootstrap reinitializes all of memory including the protected application segments. Level 4 can only be requested manually. Level 5 is also manually initiated. It

reloads the system disk from tape. A manual bootstrap is then required to initialize the system.

## IV. THE SPC 1B

The SPC 1B Processor is the functional entity that replaces the SPC 1A in TSPS No. 1B. In reality, the SPC 1B is an SPC-like environment created on the 3B20D by several components. The Peripheral System Interface (PSI), emulation microcode, 3B20D hardware, native-mode software in the TSPS process, and the DMERT operating system all play a role in realizing the SPC 1B image. This section will describe the basic characteristics of the SPC 1B, briefly discuss its components, and concentrate on those capabilities required to emulate the SPC 1A at the instruction level. Other aspects of the machine and the cooperation of the components are developed in subsequent sections of this paper.

### *4.1 SPC 1B components*

The PSI provides the necessary interface between the 3B20D Central Control (CC) and the TSPS peripheral buses. Emulated programs communicate with the TSPS peripherals without hardware modifications to the peripherals themselves. The microprogrammed control and the flexibility of the 3B20D architecture make it feasible to emulate a machine of vastly different characteristics. Emulated instructions are implemented by a microprogram that co-exists with the microprogram for the native instruction set. An off-line object code post-processor complements the emulation microcode by creating instruction formats optimized for execution on the 3B20D hardware (see Section 4.5).

Finally, some functions must be emulated at the system level. Functions such as unlocking the write protection on an emulated store or requesting a processor switch are handled by native-mode software in the TSPS process. It provides services to emulated software not easily performed in the SPC 1A environment. In many cases, TSPS native-mode code interacts with other processes and the operating system to realize other services. Section 5.1.1.1 discusses TSPS native code in depth.

### *4.2 Basic characteristics*

The characteristics of the SPC 1B are quite different from those of the physical 3B20D. In the 3B20D Processor, all data paths, memory locations, and registers are 32 bits wide. Memory addressing is byte-oriented and 24 bits wide, providing a 16-million-byte capability. Invisible to both software and firmware is the memory management hardware, which provides virtual addressing. A high-speed buffer cache

is also included, which shortens the effective memory access time. The Arithmetic/Logic Unit (ALU) provides ones or twos complement arithmetic by allowing microprogram control of the carry in bit. A Rotate-Mask Unit (RMU) provides right rotates in any amount up to 31. The rotate amount also selects a mask from 16 mask classes to implement shifts and item extraction. In addition to the AND operation with a mask, the OR with the complement of a mask is possible to facilitate operations such as sign extension. There are 16 general-purpose registers in the 3B20D, although three are reserved for stack maintenance.

On the other hand, the SPC 1B, like its predecessor SPC 1A, is a 20-bit word-addressed machine. The arithmetic is done in ones complement and emulates a subtractor circuit. The significance of a subtractor is that the minus zero result is avoided in almost all cases. Seven general registers can be used for indexing, data manipulation, return addresses, and peripheral communication. A null (N) register can also be specified as a source of a zero operand. Rotates and shifts are allowed in both right and left directions. Contiguous bit masks of most sizes from 1 to 20 bits can be used for item manipulation. An insertion masking operation is also available to allow user-specified, non-contiguous bit masks.

### *4.3 Mapping the SPC 1A image*

Since assembly-language programs are designed with an intimate knowledge of the machine they are written for, the 20-bit structure of the SPC 1A is embedded deep into the TSPS software. As a result, operations such as rotation and arithmetic represent a potential source of emulation errors. Because of this and the basic goal of emulating with minimal changes, the 20-bit architecture of the SPC 1A has been retained. Because of the difference in word size between the 3B20D and the SPC 1A and other differences described in the previous section, an image of the SPC 1B must be mapped onto the physical 3B20D hardware.

A 20-bit SPC word is contained right adjusted in a 32-bit 3B20D word. The most significant 12 bits are maintained as zeroes in both registers and memory locations. Twenty-bit word addresses are converted into 24-bit byte addresses by multiplying each address by four and forcing the uppermost two bits to be zeroes. The emulation of 20-bit addressing retains the limitation of 1 million word addressability as on the SPC 1A. Further, the one-million-word emulation address space must start at virtual address zero in the four-million-word TSPS process address space.

Registers zero through seven have been chosen as the SPC 1B registers. This assignment is identical to the numerical encoding on

the SPC 1A, thus simplifying the post-processing of SPC object code. Although the N register is assigned to be register zero, it requires special handling. When used as a source or argument operand, register zero must first be cleared. This guarantees a source of zero in the event that the N register was modified by specifying it as a destination in a previous instruction. The SPC 1B also contains an imaginary ones register, which may be used in register-to-memory and register-to-buffer bus instructions. The 'ones' register of the 3B20D is used for this purpose.

### *4.4 Instruction implementation*

The instruction set of the SPC 1B is very similar, but not identical to, that of the SPC 1A. Maintenance instructions related to the SPC 1A Processor were deleted for the SPC 1B. Other instructions that dealt closely with the SPC 1A hardware have required slight modifications. In spite of these changes, an SPC program or emulation mode programmer cannot and need not distinguish between the SPC 1A and 1B in most cases. Architectural differences described in previous sections are handled by microcode and are invisible to the programmer.

#### *4.4.1 Related 3B20D processor hardware*

To provide a framework for the description of instruction implementation, that portion of the 3B20D architecture directly affecting the emulation will be presented. A complete description of the 3B20D hardware can be found in Refs. 4 and 6.

As shown in Fig. 5, the CC is structured around a source and destination bus. The Data Manipulation Unit (DMU) accepts data from the source bus, and gates results to either an internal register or to an external register via the destination bus. The DMU contains the ALU, RMU, general registers, and parity circuits. Another circuit between the buses is the Find Low Zero (FLZ) unit. This circuit accepts 32 bits as input and yields the binary value of the bit position of the least significant zero. Finally, a direct path between the buses exists for fast data transfers between the external registers.

The store interface consists of the Store Address Register (SAR), Store Control Register (SCR), Store Data Register (SDR), and Store Instruction Register (SIR). These registers, along with a separate store operation field in each microinstruction, allow store operations to be done in parallel with 3BCC operations. To facilitate instruction fetching, hardware is dedicated to increment the Present Address (PA) register. A Program Status Word (PSW) bit specifies the PA increment amount to be 2 or 4, depending upon whether halfword or fullword mode is desired. The output of the incrementer is loaded into the SAR and back into the PA when a fetch is initiated.

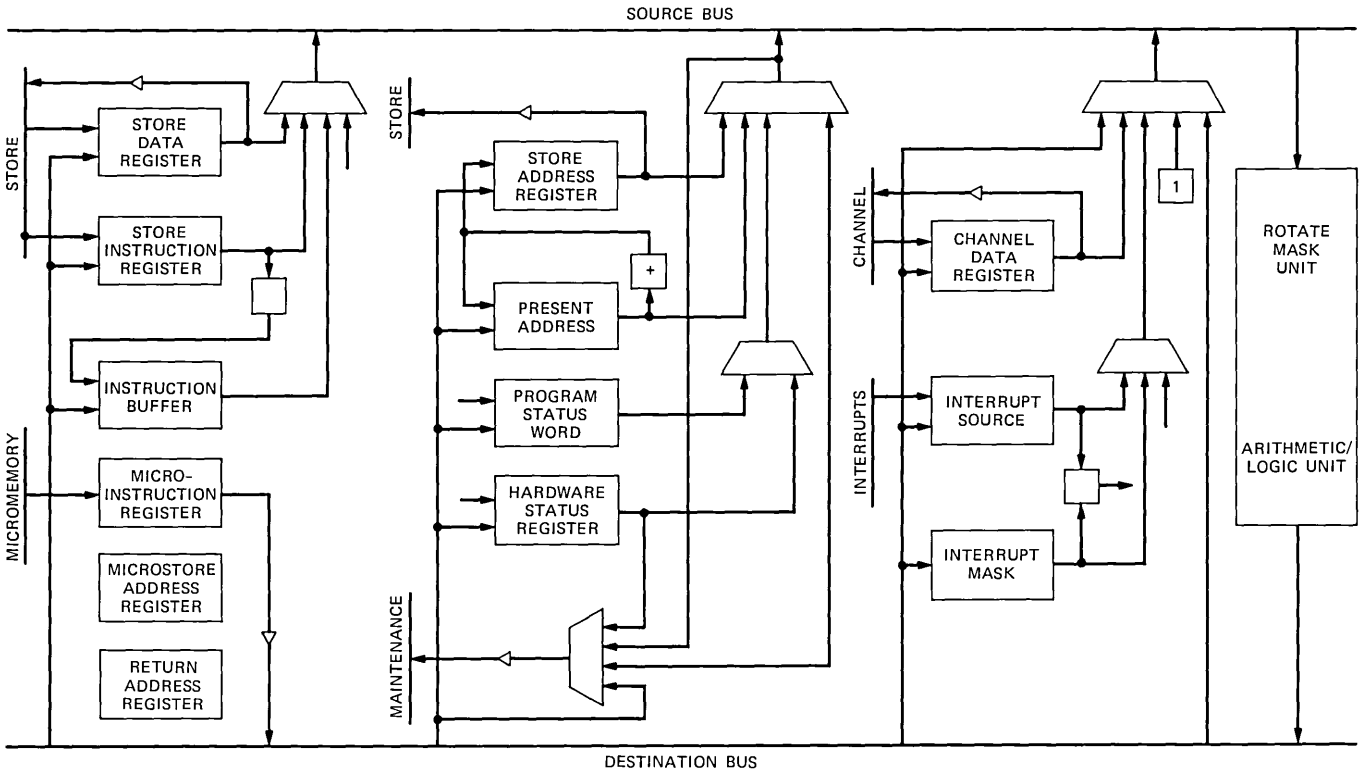



Fig. 5—Processor control architecture.

The basic I/O registers are the Channel Address Register (CAR), and the Channel Data Register (CDR). The Hardware Status Register (HSR) holds status and response information relating to channel operations. Channel operations are performed by microcode via the Pulse Point Register (PPR). Bits in the PPR are set and reset by microcode to form control pulses on the CCIO bus.

#### 4.4.2 Instruction fetching and decoding

An opcode on the 3B20D is eight bits wide. Multiple virtual machines are implemented by providing four complete sets of 256 opcodes. The instruction-initiation operation is a store-field function that operates as follows. Fetched instructions are loaded by the store into the SIR. When a new instruction is to be started, the contents of the SIR are transferred into the Instruction Buffer (IB). The opcode portion and two emulation-mode bits in the PSW are used to form a microstore address, which is the entry point into the microprogram for that instruction. Therefore, the mode bits effectively partition the microstore into four segments and, hence, four instruction sets.

#### 4.4.3 Instruction staging

As mentioned above, the instruction currently being executed is located in the IB. General registers and indices for the 16-way branch microinstruction can be indirectly specified by four-bit fields (nibbles) in the IB. For single-bit testing, the high-order bit of each nibble is available as a condition for the conditional jump microinstruction.

Similarly, there are three fields defined in the IB for rotate and mask operations. For a rotate and mask operation, a microinstruction must specify a mask class, a rotate amount, and masking operation. The rotate amount also selects which mask in the named class is to be used. The three five-bit fields in the IB can be used to specify the rotate amount and corresponding mask to be used.

### 4.5 Basic operation

The SPC 1B instruction set differs from other sets in that most basic operations can be specified as options on many instructions. Hence, the instruction set is small in number but rich in data-handling provisions. As an example, there is no explicit ADD instruction. Addition can be specified as an option on most instructions. This section will describe the implementation of the basic operations common to many instructions.

#### 4.5.1 Instruction formats

Instructions on the SPC 1A were encoded in a 40-bit double word with a four-bit basic opcode. Opcodes were extended by other bits in

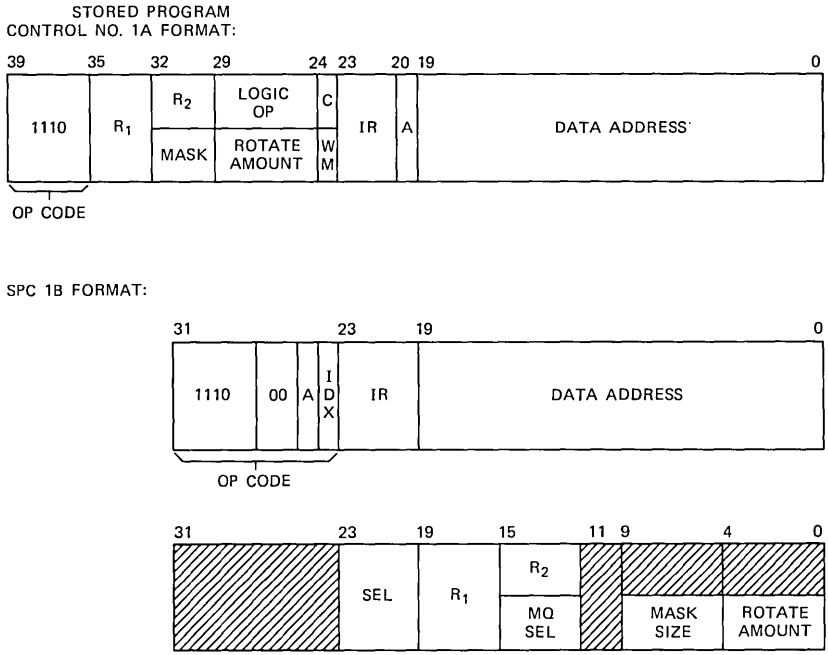



Fig. 6—Memory-to-register instruction.

some instructions, usually in a non-adjacent field. Register fields are three bits. Mask size was specified in a four-bit field that was not strictly binary encoded. The rotate amount is contained in a five-bit field. Options are encoded in multi-function fields within the instruction. In general, these fields were not aligned with the IB fields mentioned above for instruction staging. A bit-for-bit emulation would have resulted in a very inefficient result. Since one of the fundamental goals was an increase in real-time capacity, new formats were designed to provide the most efficient encodings. As an example, Fig. 6 shows the SPC 1A and SPC 1B formats for the Memory-to-Register (MR) instruction.

A post-processor is used to realign the 40-bit SPC 1A object code word into two 32-bit words suitable to execute on the 3B20D. In addition to simple bit shuffling, the post-processor adds information to aid the emulation microcode by performing those computations that can be done off-line. In the same fashion, bits have been included with the basic 4-bit opcode to take full advantage of 8-bit opcode encoding. For example, bits specifying an option can be included in the opcode to eliminate execution time required to decode that option. The unique entry point for each opcode would specify not only the instruction, but

also the occurrence of the option. The post-processor is described in more detail in Ref. 7.

#### 4.5.2 Execution protocol

As mentioned previously, SPC 1B instructions require two 32-bit words to hold the 40 bits of information contained in the SPC 1A instructions. Even in those cases where all relevant information can be held in one word, the spacing of two addresses between instructions must be maintained. As an example, indexed transfers into transfer tables, a common structure in TSPS software, would have had to be recoded had this spacing not been retained. Unlike the SPC 1A, which has a 40-bit memory bus, two separate fetches are required for each instruction. The placement of argument fields in the instruction words has to correspond to the logical execution flow for the instruction. SPC 1B formats are designed to match the flow of execution. Since instructions are a multiple of 32-bit fullwords, the mode bit in the PSW is set for a PA increment amount of 4. In contrast, the 3B20D native instruction set operates on 16-bit halfwords, with a PA increment amount of 2.

The basic fetch-execute protocol is for each instruction to fetch both words of the next instruction. When an instruction is started, the first word is in the IB and the second word is being fetched into the SIR. In this way, the processing of information in the first word can be effectively overlapped with the fetch of the second word. When the first word is no longer needed, the second word is gated from the SIR into either the IB or a scratch register as needed. The SIR is now free to accept the fetch of the first word of the next instruction. The last microinstruction, in addition to requesting that the next opcode be decoded, also starts the fetch of the second word of the next instruction.

#### 4.5.3 Arithmetic

As described previously, the SPC 1B represents negative numbers in ones complement form. A difficulty arises in the end-around carry for a 20-bit word on a 32-bit arithmetic unit. The algorithm used is to insert ones in the upper 12 bits of one of the operands to propagate the carry from bit 19 to the carry-out bit. In addition, a carry-in of one is initially assumed. If the carry-out is a one, then the carry-in assumption was correct and the operation is complete. If a carry was propagated from bit 19 to the carry-out, the upper 12 bits of the result are automatically cleared, as desired. Also, this assumption properly avoids the minus zero result that would have occurred if the carry-in was not made. This is the only case where the carry-in itself forces the carry-out. If the carry-out is a zero, the operation is repeated without the carry-in and with zeroes in the upper 12 bits of both operands. Since

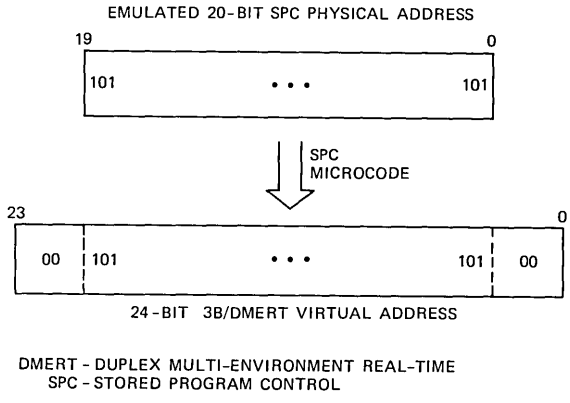



Fig. 7—SPC 1B address translation.

no carry-out of bit 19 will result, the upper 12 bits of the result will be zero.

#### 4.5.4 *Effective address generation*

For those instructions that access memory, indexing of the address is performed by adding the address and the contents of the specified index register. Any of the SPC 1B general registers, including the N register, can be used as an index register. Since negative indices are possible, the indexing operation is done in the same manner as ones complement arithmetic described above. The resulting 20-bit word address is then rotated left two bits (actually left 30 bits) and masked to clear bits 23 to 22 and 1 to 0 to form a 24-bit byte address, as shown in Fig. 7. Since indexing is costly in terms of execution time, the post-processor detects the specification of N as the index register, and sets an indexing bit to indicate that indexing is not required. The indexing bit is usually contained in the opcode to provide free decoding. The indexing bit, non-existent in the SPC 1A, is an example of information provided by the post-processor to facilitate efficient on-line execution.

#### 4.5.5 *Shifts and rotates*

In the SPC 1B, shifts and rotates are performed over 20-bit data words, even though these words physically reside in 32-bit 3B20D words. Since the 3B20D RMU only accepts right-rotate amounts, all left-direction amounts must be converted by taking their complements with respect to word size (i.e., word size minus left amount equals right amount). For shifts, the proper mask is applied to inject zeroes into vacated bit positions.

Implementation of the wrap-around from bits 0 and 19 for rotates is more difficult. Values less than 12 can be handled by copying the

upper or lower 12 bits of the 20-bit data word into the upper 12 bits of the 32-bit word. For left rotates, the upper 12 bits are used, while the lower 12 bits are copied for right rotates. Rotates greater than 12 can be converted to their equivalent value in the opposite direction before applying the above algorithm. For example, a left rotate of 15 can be effectively realized by a right rotate of 5. It should be noted that rotate and shift amounts specified in the instruction can be converted by the post-processor and encoded as an adjusted value. Rotates and shifts based on the contents of a register must be adjusted during run-time by the microcode.

#### *4.5.6 Data handling*

The SPC 1B provides the capability of reading (unpacking) and writing (packing) contiguous data items of almost arbitrary size and position within a 20-bit word. A data item is defined by its size, M, and its position, Q. The unpacking operation involves reading a data word, rotating it right by Q, and applying a mask of size M. The packing operation involves applying a mask and then rotating it into position by the amount Q. This value is then inserted, using a read-modify-write sequence, into the target location.

Since the MQ option is used heavily in TSPS programs, implementation of the rotation in the manner described previously would be very costly. Luckily, in most cases much of this overhead is avoided. When the total of the rotate amount plus the mask size is less than or equal to 20, no wrap-around actually occurs. The more efficient shift operation can be used instead. In TSPS, this is true nearly every time an MQ is specified.

#### *4.5.7 Conditional transfers*

The SPC 1B, like the SPC 1A, has no explicit condition flags. However, the results of data manipulations can be tested by conditional transfer instructions. Conditional transfers provide the capability of testing either the entire contents or a specific bit of any general register. The 3B20D condition codes, carry, overflow, zero, and negative, are used for these tests. It should be noted that like the SPC 1A, the minus zero value, coded as all ones, is not detected as zero in these conditionals.

A special conditional transfer instruction is the Detect Right Most One (DRMO) instruction. This instruction transfers if no ones are present in the test register. If a bit is set, the bit position in binary of the least significant one is loaded into a result register. The DRMO and its variation, the DZRMO, which zeroes the rightmost one, are used, for example, by task dispensers to initiate clients based on a job-activity word. The 3B20D FLZ is used to implement these instructions.

#### 4.5.8 Alternate entry points

There are two functions in the SPC 1B instruction set that are unique in that their operation extends across instruction boundaries. They are the Inhibit Interrupt (I) option and the Execute (EXC) instruction. The I option inhibits the J- and H-level interrupts until after the next instruction is completed. Although hardware exists in the 3B20D to unconditionally start the next instruction, blocking all interrupts if error interrupts are pending is unacceptable. In this case, the I option is implemented by raising the execution level to block J-level and H-level, starting the next instruction, and restoring the execution level during the instruction following the I-option instruction. The EXC instruction calls for the execution of an instruction at a target address, followed by an automatic return to the next sequential instruction after the EXC. The responsibility of the target instruction is to restore the PA to resume sequential execution.

Since any instruction can follow an I-option instruction or be the target of an EXC, every instruction must determine if a special operation has preceded it. Explicit tests by microcode at the start of each instruction would cause a prohibitive amount of overhead even when these operations are inactive. Instead, an alternate instruction set is used to force a different entry for the same opcode. The I option and EXC microcode forces the next instruction to be entered at its alternate entry by setting the appropriate emulation-mode bits in the PSW. The alternate entry typically determines which operation to unwind, restores the mode bits to their normal value, and transfers to the normal entry point to execute the instruction. Since the SPC 1B does not have condition codes, the PSW condition flags are available to specify whether the EXC or I option is in effect.

#### 4.5.9 New SPC 1B instructions

There are four new instructions in the SPC 1B instruction set. The Register to Buffer Bus (RBB), Buffer Bus to Register (BBR), and Ones to Buffer Bus (OBB) instructions have been added to reference the buffer bus system. These instructions, along with the buffer bus, are described in Section 5.1.4. The remaining new instruction is the Switch Mode and Transfer (SMT).

The SMT instruction allows emulated programs to transfer to the native mode and begin executing its instruction set. It is patterned after the native-mode CALL instruction to provide a calling sequence consistent with the C language. The SMT puts a stack frame, including two register arguments, on the stack, switches the mode bits in the PSW to specify native mode, and transfers to the address specified in the instruction. Two instructions in the native language are also used to support mode switching. They are the Call to Emulation (CALE),

and Return to Emulation (RETE) instructions. Except for the mode switch, these instructions are identical to the standard native CALL and RETN instructions.

The ability to transfer between native and emulated code allows C language to be incorporated into the TSPS process. In fact, the TSPS process has standard C-coded entry points, as described in Section 5.1.1.1. Mode switching eliminates the need for new instructions to perform operations not possible with the emulated instruction set. Examples are Operating System Traps (OSTs), 32-bit data manipulations, and stack accesses. Finally, designers of new features can evaluate on an individual basis which language would be most suitable. Trade-offs can be made between the necessary degree of coupling with existing emulated code and the ease of programming in C.

### *4.6 Peripheral orders*

As mentioned in Section 4.1, the PSI provides the necessary timing and electrical interface to the TSPS periphery. The function of the microcode for peripheral instructions is simply to pass necessary information for the PSI to execute the order. At the end of the sequence, the PSI passes back an answer word and an indication of whether the periphery returned the correct reply signals. A failure indication is used to generate the emulated F-level interrupt, as discussed in Section 5.1.5.1.

The CC communicates with its channels over the Central Control Input/Output (CCIO) bus. An Application Channel Interface (ACHI) is provided with the 3B20D processor to allow the PSI limited access to the CCIO bus. In essence then, the PSI appears as a main channel to the 3B20D processor. Commands and data are sent and data received in a parallel fashion to/from the ACHI-PSI via the CDR CAR, and PPR. Status information, returned by the PSI into the HSR, is used to indicate a failed peripheral order.

The sequencer in the PSI performs four basic functions. In addition to the peripheral order, there are sequences to read a register and write a register. These functions are needed not only for buffer bus references, but also for diagnostic access. The remaining sequence is a pulse sequence, which is used to send special maintenance pulses to the periphery.

All SPC 1A peripheral orders are retained in the SPC 1B instruction set. In addition, a set of PSI instructions have been provided in the native instruction set to execute the four PSI operations. The native-mode PSI instructions were originally designed for the PSI diagnostic, discussed in Section 5.3, which is written in C and native assembly language. They are also used by the Application Integrity Monitor (AIM) process for PSI fault recovery and initialization (Section 5.2.2.1.).

One additional capability provided by the emulation microcode is a set of microcode routines to execute peripheral operations on the off-line processor. These routines accept data from the 3B20D Maintenance Channel (MCH), temporarily release the inhibit on the off-line PSI, and execute the order without making any main-store accesses. By not interfering with the main-store update mechanism, these routines can be executed while the off-line processor remains in the standby mode.

Off-line sequences for pulse and peripheral orders are used by TSPS peripheral fault recognition to retry failed orders and provide better fault resolution. An off-line PSI initialization routine is used by AIM just prior to a soft switch. It is also used periodically by AIM as a hardware audit of the off-line PSI. Should this routine indicate a failure in initializing the PSI, a diagnostic will be requested by AIM. The benefit of this audit is to reduce the latency time for detecting faults in the standby PSI.

### *4.7 Emulation-dependent software*

Although the majority of SPC 1A instructions have been emulated on the SPC 1B, there are functions that do not work in the same manner as on the SPC 1A. In general, the functions requiring modification are in the maintenance programs, which are by necessity more processor-dependent.

The most obvious source of change is the need to recode or eliminate those routines that contain instructions not carried over on the SPC 1B. The most common source of change is recoding buffer bus references to use the new instructions. Unmodified buffer bus references would simply reference a memory location in segment zero since the microcode does not trap the address as a buffer bus address in normal memory-access instructions. A third source of change is due to the new object-code formats and the difference in size between instruction and data words. Since the emulated code can only access 20 bits, instructions cannot be moved or created via data operations. Similarly, data cannot be executed as an instruction because the uppermost 12 bits of data words are zeros. On the SPC 1B, the zero opcode is interpreted as the ANOP instruction. This protective measure would trap a wild transfer into a data area by generating an A-level interrupt. Finally, usage of part of an instruction as data may not work because of the shifting of fields in the new object-code formats. An example of this in TSPS No. 1 code is to read the address field of a transfer instruction and store it to be used as an address later.

A more subtle difference is the execution time of instructions on the SPC 1B. The SPC 1A is a fixed-cycle machine, and instructions are made up of a number of these basic cycles. The execution time of an

instruction could be determined exactly, independent of the options specified. A common dependence on the execution time of an instruction is in timing loops. A precise time delay is created by executing a loop the correct number of times.

Since the 3B20D executes emulated instructions several times faster than the SPC 1A,[8] timing constants coded for the SPC 1A produce delays proportionately smaller on the SPC 1B. The modification of these constants is not quite as simple as multiplying by a speedup factor for the following reasons. First, the SPC 1B instruction set is microprogrammed, and thus every option must be accounted for when calculating the execution time for an instruction. Second, since the 3B20D employs virtual addressing, memory management delays owing to translation are incurred as a function of program flow. The cache memory also plays a role by shortening the memory access time when the access is contained in the cache. Since the cache is shared by all processes in the system, its effect is even less predictable. Finally, Direct Memory Access (DMA) activity steals memory cycles from the processor and represents an invisible form of interference.

As a result, timing loops with tight window tolerances cannot be guaranteed. In several cases, routines that performed window timing required recoding. On the other hand, minimum timing poses no problem. Best-case estimates can be made to determine the minimum execution time for a program segment by assuming no translation or DMA delays and all cache hits. Any delays actually incurred during execution serve only to lengthen the program segment time, which is acceptable for minimum timing.

## V. TSPS NO. 1B SOFTWARE

This section describes how the TSPS software was ported to run under the DMERT operating system. In addition, it describes the system integrity software and other processes required to complete the emulation.

### *5.1 The TSPS kernel process*

The emulated TSPS software has been incorporated into a single, large, nonkillable kernel process under DMERT. This is due largely to the real-time constraints of TSPS software operation and its existing structure on the SPC 1A. All TSPS code on the SPC 1A shares a single physical address space. All data and programs are equally accessible from any other program. The entire software structure is very tightly coupled. Subdividing the software into several processes would have required a restructure and redesign of the TSPS software. Also, the operational timing constraints of TSPS software (e.g., 5-ms I/O processing, interject responsiveness and base-level E-E times) require

TSPS to be memory-resident and interrupt-driven at the kernel-process level.

### *5.1.1 Emulation environment*

Emulation of existing TSPS software on the 3B20D Processor required the creation of an SPC 1A environment. That is, in order to work properly the TSPS software had to be shielded from both the actual physical machine (3B20D) it was running on and the DMERT operating system it runs under. This environment has been established with a combination of hardware (PSI), firmware (the microcoded SPC 1A instruction set), and software (native-mode code within the TSPS process).

***5.1.1.1 Native-mode code.*** Except for the emulated code within the TSPS process, all software in the TSPS No. 1B system is coded in 3B20D native mode. Most of the software is coded with the C language. Hence, the 20-bit emulated code exists in a 32-bit universe.

Figure 8 illustrates the structure of the TSPS process. The TSPS process is dispatched (given control of the machine by DMERT for execution) at either its interrupt, event, or fault entry. These entry points are coded in 3B20D native mode. Thus, the TSPS process always begins execution in 3B20D native-mode code. These entry points determine what type of processing is to be performed and transfer, while simultaneously changing instruction sets, to appropriate routines within emulated software.

In addition to the native-mode entry points, the TSPS process also contains native-mode routines for communication with DMERT and other processes and for performing certain functions which the emulated code is incapable of doing. As a result, the native-mode code within the TSPS process completely isolates the emulated code from the 32-bit universe surrounding it. Hence, to the rest of the system the TSPS process appears to be a typical (albeit very large) native-mode kernel process.

***5.1.1.1.1 Interrupt entry.*** The only hardware interrupt source to which the TSPS process is attached is a clock-driven 5-ms interrupt for I/O processing. This single interrupt source is used for both J- and H-level processing. This same 5-ms clock pulse is transmitted to the PSI to synchronize its clock with the processor. The PSI clock transmits clock pulses to TSPS peripherals. When the interrupt source fires, the TSPS process is dispatched at its interrupt entry point. The interrupt entry is coded in 3B20D native assembly language because of its simplicity and need for efficiency: it runs every 5 ms. Its only functions are to determine whether this execution is a J- or an H-level, set the appropriate bit in the interrupt-level activity flag buffer bus word, and transfer to the appropriate emulated routine.

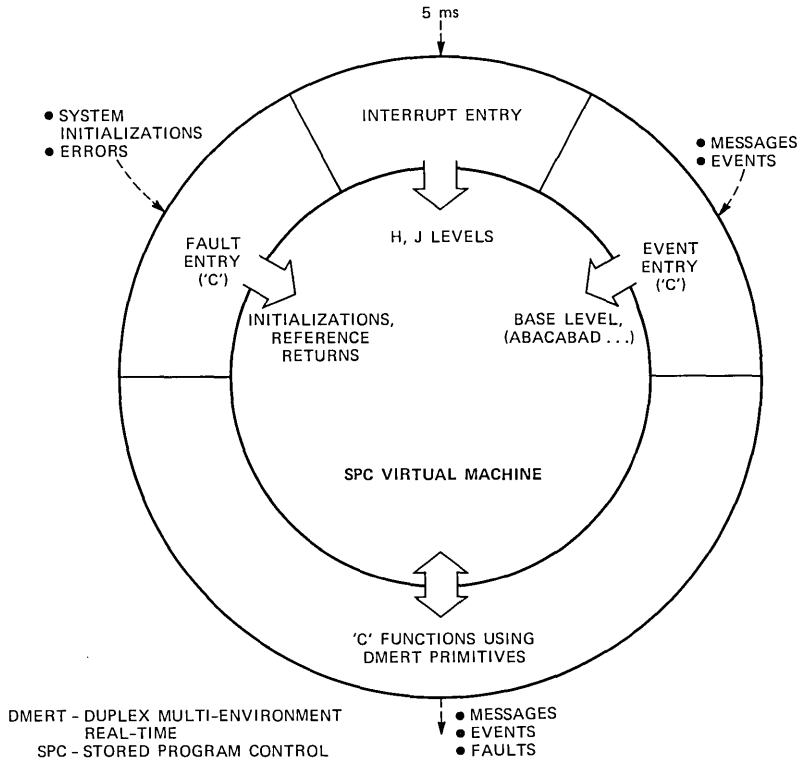



Fig. 8—TSPS Process.

Handling both J- and H-level interrupts with a single interrupt source is accomplished by using two execution levels for J-level processing. When TSPS is dispatched at the interrupt entry to handle a J-level interrupt, it runs at execution level 12. While at level 12 the interrupt source is masked off by the level-12 interrupt mask. Hence, the interrupt entry cannot be re-entered. When high-priority J-level work is completed, the H-level inhibit bit in the buffer bus is cleared. In doing this the microcode for the buffer bus write operation drops to execution level 11 for low-priority J-level work. This is possible since no other processes in the system run at execution levels 11 or 12. At level 11, the 5-ms interrupt source is unmasked and will allow the TSPS interrupt entry to be re-entered if J-level does not complete within 5 ms. The interrupt entry reads the buffer bus to determine whether a particular execution is a J- or H-level. If neither activity bit is set, then it is a J-level and the J-level activity bit is set. If the J-level activity bit is already set, then an H-level has occurred and the H-level bit is set. In the latter case low-priority J-level work has been interrupted and its state has been saved on the interrupt stack.

Although the interrupt handling begins in the native-mode interrupt entry, the return from interrupt is always performed from emulated code. At the end of H-level, the last job executes an emulated EGBN instruction. The EGBN clears the H-level activity flag in the buffer bus and performs a return from interrupt sequence, which causes the saved J-level state to be popped from the interrupt stack. J-level then resumes execution at the interrupted point. At the completion of low-priority J-level an emulated GBNHJ instruction is executed. The GBNHJ clears the J-level activity flag, sets the H-level inhibit bit, and does a return from interrupt. Hence, the process that was executing when J-level began is resumed at the point of interrupt. This interrupted process itself may be the TSPS process performing base-level work.

***5.1.1.1.2 Event entry.*** The TSPS process event entry is coded with the C language and runs at execution level 5. An event sent to the TSPS process causes the process to be dispatched and begin execution at its event entry. The primary events sent to the TSPS process are for initialization upon creation, interprocess message reception, and time-outs requested for real-time breaks in TSPS-base level processing.

***5.1.1.1.2.1. Initialization event.*** An initialization event is sent to a process upon its creation. As TSPS is a nonkillable kernel process, this will occur only after a system bootstrap. The native-mode code initializes the process (i.e., attaches to the 5-ms interrupt source, connects to a system port for message reception, etc.) and prepares to take an SIA or SIB call-processing recovery phase.

An important function that is performed during the initialization is setting the proper page protection over the emulated-code address space. DMERT creates processes with protection set on segment boundaries. The TSPS process, however, emulated the SPC 1A protection mechanism. Hence, some segments may have both protected and unprotected areas. The protection within these segments must be modified by changing the protection on the appropriate pages. TSPS uses a DMERT OST for this run-time page-protection change. This same OST is used by the emulated recent change programs when applying a modification to protected memory.

***5.1.1.1.2.2 Time-out event.*** The TSPS process is only one of many processes time-sharing the SPC 1B. The TSPS process must voluntarily take frequent real-time breaks to allow other processes at execution level 5 and below to execute. (Base-level processing is done at execution level 5, which is the lowest execution level at which the TSPS process runs. This is described in more detail below.) These real-time breaks are performed by issuing a time-out request to the DMERT timer* and then relinquishing control of the machine. (The DMERT timer is

part of the kernel that executes at execution level 15. It runs every 10 ms as a result of clock-driven interrupt source. The Timer maintains the DMERT system clock and provides general-purpose timing for all DMERT processes.) At the end of the time-out period the timer will send TSPS a time-out event. When TSPS takes the break, it remembers where in emulated code it has left off. Upon receipt of the time-out event it will resume execution within emulated code at the same place where it had left off. A related event is one sent by the scheduler when the system is idle to wake TSPS up early, before the time-out event has fired. This is described further in Section. 5.1.6.

Currently there are five areas where the TSPS process takes real-time breaks. These are during memory zeroing in an SIA or SIB, during initialization timing loops that are longer than 10 ms, during audit stitching, at the end of an SIA or SIB while waiting for the J-level portion of the phase to complete, and during the base-level E-E cycle.

***5.1.1.1.2.3 Message event.*** The TSPS process receives interprocess messages from other processes. TSPS is notified of the message reception with a message event. When TSPS fields the event, it will process all queued messages and take whatever actions are appropriate. Some uses of interprocess messages by the TSPS process are discussed in the later section describing other TSPS application processes.

***5.1.1.1.3 Fault entry.*** The TSPS process fault entry is also coded in the C language. The TSPS process is dispatched at its fault entry as a result of a processing error (i.e., a protection violation or an invalid address), a system initialization, or a fault sent by another process as a means of interprocess communication. The TSPS fault entry runs at execution level 12 so that recovery actions can be taken without interference from J-level processing.

***5.1.1.1.4 Additional routines.*** Besides the standard process entry points there are other special-purpose native-mode routines included within the TSPS process. These routines are used to interface with DMERT (e.g., to execute an OST), to use interprocess communication mechanisms such as events and messages in order to communicate with other processes, and to implement some functions that cannot be performed in emulated code. In addition, some new code added to the TSPS process was coded using the C language for development convenience.

### *5.1.2 Address space*

TSPS is currently designed as a DMERT "small" process. A small process can have up to 64 segments, and, hence, a virtual address space as large as two million words. Figure 9 shows the TSPS process virtual address space as a small process under DMERT.

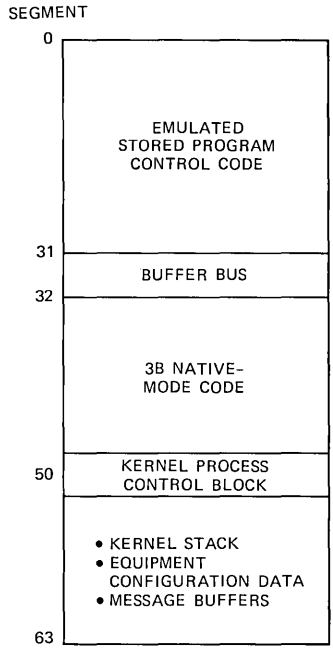



Fig. 9—Virtual address space in the TSPS process.

***5.1.2.1 Emulated code.*** The emulated code resides in the first 32 (0–31) segments of the virtual address space. Thus, the emulated code resides in the first one million words of the virtual address space. This, in fact, is the only part of the virtual address space that is directly addressable by the emulated code. As mentioned previously, all addresses in the emulated code are 20-bit word addresses as in the SPC 1A. A 20-bit address can only address one million words. Hence, the 20-bit physical word addresses on the SPC 1A are now 20-bit virtual addresses in the TSPS process on the SPC 1B.

All 32 segments are fully allocated to their maximum size of 32K words. In effect each segment emulates an SPC 1A store name code. The emulated address space, however, includes segments 0 and 31, whereas store name codes 0 and 31 do not exist on the SPC 1A. Emulated buffer bus references are mapped by the microcode into segment 32. As a result, the emulated code within the TSPS process has a full 1M word address space.

***5.1.2.2 Native-mode code.*** The remaining 31 segments are used for native-mode code within the TSPS process. Not all of these segments are currently used. Those that are used contain TSPS code and data, the process control block, and shared DMERT segments such as the ECD, the system message buffers, and the kernel stack.

### 5.1.3 Native-mode/emulated-code interface

The TSPS process contains both emulated SPC 1A and native-mode code. As all processing begins in native code, all emulation-code routines are called as functions (subroutines) from the process entry points. Similarly, native-mode routines can be called as subroutines from the emulated code.

All processes under DMERT use a program stack to save the state of a function when it calls another and to pass arguments. Kernel processes (including TSPS) share a common kernel stack with the kernel. A standard stack frame must be maintained when calling another function. Similarly, a standard return sequence is followed. To maintain sanity while transferring between emulated and native code, the TSPS process follows the same protocol. The mechanism used by the TSPS process was patterned after the structure used to transfer between native-mode assembly code and C-language code.

### 5.1.4 Buffer bus

As described in Section 2.2, the buffer bus on the SPC 1A is a software-accessible set of hardware control and status registers. Since the microcode emulates the hardware action on the SPC 1B, accessing one of these registers causes the microcode to perform a special action, emulating what the hardware would have done. Hence, firmware must be aware that a buffer bus location is being accessed, and what action is required. To eliminate the overhead required to test each memory address in memory reference instructions, new instructions have been added to explicitly reference the buffer bus.

The Register to Buffer Bus (RBB), Buffer Bus to Register (BBR), and Ones to Buffer Bus (OBB) instructions are the exact images of their SPC 1A counterparts, RM, MR, and OM. Converting an SPC 1A buffer bus reference requires only a change to the instruction mnemonic. The address and option fields are identical. The addition of these instructions has the added benefit of not having to set aside store zero as buffer bus images, thus allowing an additional 32K words of usable memory. Depending on their function, buffer bus registers are mapped into either memory locations outside of the SPC 1B address space, PSI internal registers, or 3B20D hardware registers.

Not all registers have been carried over from the SPC 1A. Locations used only in processor and store maintenance were removed along with their corresponding programs. Other locations have been added to maintain and access the PSI. Still others are partially retained, with individual bits having been removed. The set can be broken down into three components: interrupt system, craft interface, and peripheral system.

Locations related to the interrupt system are the Interrupt-Level

Activity Flags (ILAF), Maintenance Interrupt Sources (MAIS), Interrupt Inhibits (PEST), 3B20D Inhibits (3BPEST), and the Millisecond Clock State (MSEC). With the exception of the MSEC register, these are discussed in Section 5.1.5. MSEC on the SPC 1B is the value of the 3B20D Interrupt Timer. The Interrupt Timer is used as the 5-ms clock source for the TSPS J-level interrupt. Access to this timer allows programs to predict when the next J-level will occur. This function is necessary for clients such as diagnostics, which must synchronize their actions with J-level.

Buffer bus locations related to the craft interface are Maintenance Control Center Data (MCCD), MCC Interrupts (MCCI), and Emergency Recovery Display (ERD). The hardware of the SPC 1A MCC has been replaced by the 3B20D craft interface hardware, and DMERT and TSPS craft software. These locations reside in memory locations accessible by TSPS native-mode craft software. The craft interface is discussed in more detail in Section 5.3.3.1.

The remaining locations are all related to the peripheral system. The PSI hardware directly implements many of these locations. Locations emulated by the PSI hardware require that microcode send the necessary command to the PSI to access data or perform some hardware action.

### *5.1.5 Interrupt structure*

***5.1.5.1 Emulated interrupts.*** The emulated interrupt structure consists of A-, E-, F-, H-, and J-level interrupts. Of these, only the 5-ms H- and J-level interrupts are driven by a 3B20D hardware-interrupt source. A-level interrupts are either emulated by native-mode software or generated by the microcode as the result of executing an illegal opcode or an ANOP instruction. E-levels are emulated by native code as the result of fault codes received from DMERT, and F-levels are generated either by microcode or native-mode code.

The microcode for the ANOP instruction traps to an illegal instruction routine in the native microprogram that generates an error interrupt. The DMERT kernel handles the error interrupt, and immediately faults the TSPS process. The TSPS process fault entry determines that an ANOP instruction was the cause of the fault, sets the A-level activity flag in the buffer bus and transfers control to the emulated A-level interrupt-handling routine.

An A-level interrupt can be simulated by native software through setting the A-level activity bit in the buffer bus, which raises the execution level to 12 (the level at which A-level runs), filling in the interrupt-state bin locations, and transferring to the A-level interrupt routine. This is done, for example, to indicate a manual request for a recovery phase. The TSPS process fault handler will emulate the SPC

1A hardware by "generating" an A-level interrupt in the aforementioned way. When the A-level routine is entered, it appears to the software that a hardware interrupt had occurred.

SPC 1A store-error E-level interrupts are not emulated, as this type of error is not seen by the TSPS process. They are handled completely by DMERT fault recovery. Software E-levels (i.e., protection violations and bad addresses) are fielded by TSPS as faults. When this type of error occurs, the TSPS process is immediately interrupted and entered at its fault entry with the state of the machine at the time of the error passed as parameters. The fault entry sets the E-level activity bit in the buffer bus and transfers to emulated code to handle an E-level interrupt. Again, to the emulated code it appears just as if a hardware interrupt had occurred on the SPC 1A.

During the execution of an emulated I/O instruction or buffer-bus instruction,* any failures will be detected by the microcode. The microcode 'generates' an F-level interrupt by setting the F-level activity bit in the buffer bus, saves the address of the interrupted program in the F-level bin, and transfers directly to the emulated F-level routine.

Most PSI errors are detected by the microcode also. An F-level is also generated in these cases. This is done so that F-level can perform an on-line and, if necessary, an off-line retry before letting AIM take PSI recovery actions. In the few cases where a PSI error interrupts AIM first, AIM will fault the TSPS process. The fault entry will then emulate an F-level in the same way as E-levels are handled. There is another class of errors that generate 3B20D error interrupts. The DMERT Error Interrupt Handler (EIH) will field these and fault the running process. In these cases TSPS passes control (via a fault) to AIM for recovery actions.

The microcode and native code must of course be aware of interrupt priorities. If TSPS is already in A- or E-level processing, then the microcode must only set the F-level interrupt source bit and continue processing. An F-level will be generated by the microcode if the source is still set when an EGBN is performed from A- or E-level. Also, if the F-level inhibit is set in the buffer bus the error must be ignored. Setting the J-level inhibit while in base level causes the microcode to raise the execution level of the process to 12. The interrupt mask for execution level 12 masks off the 5-ms interrupt source. Similarly, the "I" option is implemented via the microcode. The microcode guarantees that both J- and H-level interrupts are inhibited for the instruction with the I option and the next instruction executed.

---

* As some of the buffer-bus registers are contained in the PSI, a failure while trying to access these registers is considered a PSI error.

*5.1.5.2 Returning from interrupts.* The emulated EGBN instruction works the same as it does on the SPC 1A for A-, E-, and F-level interrupts. Because H- and J-level interrupts are driven by a hardware-interrupt source, the EGBN for H-level and the GBNHJ instruction used by J-level operate differently. This difference (described below) is undetectable by the programmer.

When a 5-ms interrupt occurs, the state of the interrupted process is saved on the 3B20D interrupt stack. The return from interrupt (EGBN for H-level or GBNHJ for J-level) must restore the state from the interrupt stack and not from memory-resident bin locations. Hence, these two instructions perform the same function as on the SPC 1A, but the actions taken to restore the interrupted state of the machine are different.

### *5.1.6 Emulated program structure*

The emulated program structure within the TSPS process is almost identical to that which exists on the SPC 1A; the major difference is that processing always begins in native code. The native code sets up the proper environment and passes control to the emulated software for the bulk of the processing.

H- and J-levels still run under control of ECIO. As pointed out earlier, the only difference is that a single interrupt source is used on the SPC 1B. The front-end native code determines whether an H- or J-level has occurred by the state of the buffer bus. Different execution levels are used to unmask the H-level interrupt source when going from high- to low-priority J-level.

Base-level programs still run under control of ECMP. Base level is essentially called as a large subroutine of the event handler. Rather than run continuously, as on the SPC 1A, base level must voluntarily give up control of the machine to allow lower-priority processes (processes that run at execution levels below 5) to run. Base level requests a time-out event from DMERT when at least 10 ms have elapsed, and then exits.

There are two ways for base level to be re-entered in response to its time-out request. If during the real-time break there are no processes ready to run, the DMERT scheduler will enter its idle loop at the lowest system priority. At that point, the scheduler finds that the TSPS process has requested, via an OST during its initialization event entry, that a specific event be sent by the scheduler whenever the idle loop is entered. The event sent by the scheduler causes TSPS to be entered at its event entry, thus waking base level up early. If the idle loop is never entered during the break, base level must wait until the full duration of the time-out request has expired. In this case, receiving

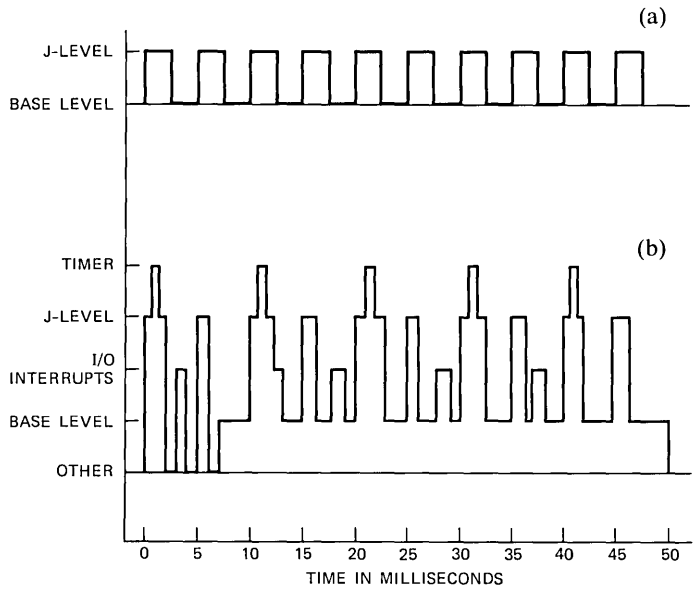



Fig. 10—Interaction between processing levels of (a) SPC 1A and (b) SPC 1B.

the time-out event from the DMERT timer causes base level to resume execution.

This early wakeup mechanism employed by TSPS has a significant impact on the real-time profile of the system. By eliminating idle system real time, base level executes more E-E cycles. This is especially true when the offered call load is light and real time is ample. A higher E-E rate improves the execution of TSPS diagnostics, audits, and other services that are run as a function of E-E rate. The full impact of the early wakeup mechanism is described in Ref. 8.

Base-level code in ECMP was modified to take its real-time breaks between priority classes. There are five breaks per E-E cycle: one after each C and E priority class. Figure 10 contrasts the interaction between various processing levels on the SPC 1A versus the SPC 1B. On the SPC 1A, base level runs continuously, only being interrupted (in the absence of errors or manual actions from the MCC) every 5 ms by J-level. On the SPC 1B, however, there are other processes executing concurrently with TSPS. DMERT I/O processes run at execution levels higher than base level, but lower than J-level. J-level executes every 5 ms whether or not base level is executing or taking a real-time break. Table II shows the execution levels of various DMERT processes, the TSPS process, and AIM.

The real-time breaks are not noticeable to the emulated software. They merely appear as an additional priority class placed in between

Table II—DMERT execution levels

| Level | Processes |
|---|---|
| 15 | Error Interrupt Handler, Timer |
| 14 | Generic utilities |
| 13 | System and Application Integrity Monitors |
| 12 | TSPS (Fault entry, A, E, F, H and high-priority J) |
| 11 | TSPS (low-priority J) |
| 10 | Disk and IOP drivers |
| 9 | — |
| 8 | — |
| 7 | File Manager |
| 6 | — |
| 5 | TSPS (Base level) |
| 4 | — |
| 3 | Plant measurements, Field update |
| 2 | Special processes (Scheduler and Memory Manager) |
| 1 | Supervisor processes (critical region) |
| 0 | Supervisor/User processes |

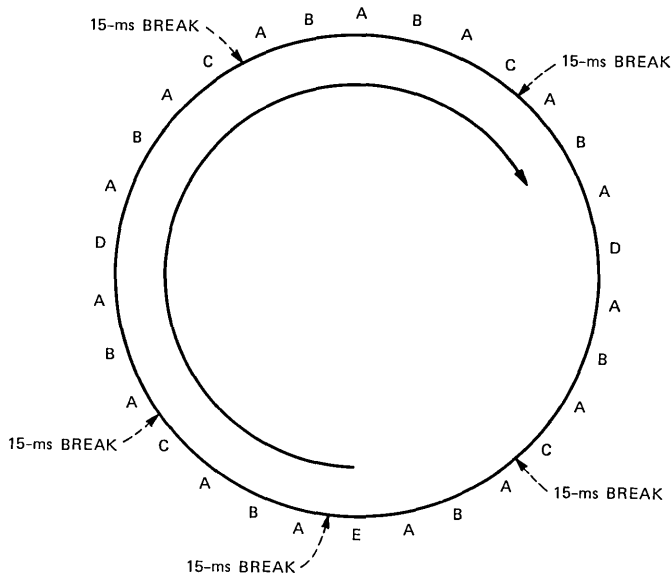



Fig. 11—TSPS No. 1B base-level loop.

the two priority classes separated by the break. Figure 11 shows the modified base-level loop with the five real-time breaks. The insertion of these breaks, though, has required one minor change to interject processing. To guarantee proper interject responsiveness additional interject checks have been inserted. A check for interject work is made after each real-time break. Hence, interject is serviced within 20 ms as a result of a base-level real-time break. Once the interject check is made, base-level processing resumes with the next scheduled priority

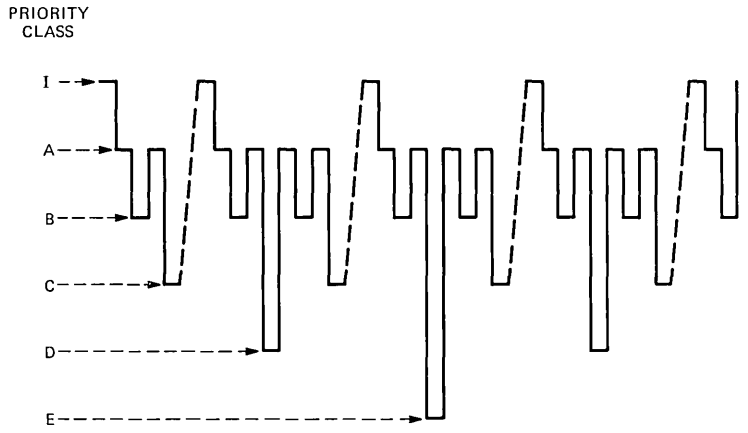



Fig. 12—TSPS No. 1B base-level, priority-class frequency.

class. Figure 12 shows the base-level priority-class execution with the added interject checks.

Another important point is the relationship between the DMERT timer and J-level execution. As the timer runs at execution level 15, it will interrupt J-level. The 10-ms timer interrupt, however, is offset from the 5-ms interrupt by 1 ms. Hence, the timer runs every 10 ms, 1 ms after J-level begins. This guarantees proper operation of the Automatic Message Accounting (AMA) data recording. The AMA data transfer routine must write data to be recorded to the AMA within 1 ms after the 5-ms interrupt occurs. (Recall that the AMA tape unit is synchronized with the 5-ms clock pulse.) Since the timer will not run until this 1-ms window has passed, it does not interfere.

### *5.2 System integrity*

The TSPS No. 1B integrity software consists of three distinct software packages. In addition to the software package provided with DMERT and the emulated TSPS No. 1 integrity package, a new integrity software package was developed. The major functions of this software are:

(*i*) Interface and coordination of integrity-related activities between DMERT and the TSPS application (e.g., initialization, overload, and processor switch)

(*ii*) Sanity and integrity of application processes

(*iii*) PSI maintenance.

These functions have been implemented in the Application Integrity Monitor (AIM), the native-mode portion of the TSPS process, and the PSI diagnostic process.

#### 5.2.1 Hardware integrity

The TSPS No. 1B hardware architecture and, in particular, the design of the PSI, made it possible to retain most of the peripherals used in TSPS No. 1. As a result, the maintenance strategy for these peripherals remained virtually unchanged and the maintenance software implementing it was emulated with only minor changes. The entire maintenance software for the TSPS peripherals resides in the kernel TSPS process and represents a distinct maintenance package.

Unlike the SPC 1A, the SPC 1B does not provide matching between the on-line and off-line processors. Instead, both the 3B20D and the PSI employ extensive self-checking hardware circuits to detect most service-affecting faults. The basic switchable entity is the simplex SPC 1B. When a fault is detected in the on-line processor, a switch to the off-line processor is performed. This switch may be followed by an initialization sequence. The faulty circuit pack is then identified with the aid of a diagnostic. The SPC 1B hardware is designed such that no single fault in either processor can cause a system outage. The TSPS No. 1B hardware architecture, including the PSI, and its maintenance, is described in Ref. 6. TSPS No. 1B reliability is covered in Ref. 8.

#### 5.2.2 Software integrity

*5.2.2.1 Application integrity monitor.* The Application Integrity Monitor (AIM) process is the sole application interface to DMERT for system integrity issues—initialization, sanity, processor switches, and DMERT overload conditions. Primary responsibility for monitoring the sanity of the TSPS application and controlling its recovery resides with AIM. In addition, AIM performs PSI fault recovery and initialization. Reports of DMERT recovery actions and resource overload are made to AIM, which in turn controls the application response to the event.

AIM is a nonkillable kernel process running at the same execution level as the DMERT System Integrity Monitor (SIM), higher than all other application processes. It interfaces with SIM via interprocess messages. AIM executes briefly every 500 ms at execution level 13 or as a result of PSI error interrupt or faults sent from the TSPS process. Since AIM is at a higher execution level than the TSPS process, it runs immediately when faulted by TSPS. This is done, for example, when TSPS requests a phase. TSPS faults AIM with its request, and AIM requests the phase of DMERT and reports back to TSPS about when to start its initialization.

*5.2.2.2 Initialization.* The TSPS call-processing recovery phases were integrated into the DMERT initialization levels to provide a coherent strategy. In the resulting initialization mechanism, between two and four TSPS initialization levels are associated with each DMERT

initialization level. Table III shows the mapping of TSPS recovery phases to DMERT initialization levels. TSPS minor and major phases are not taken after a bootstrap (DMERT initialization levels 2 through 4). The bootstrap recreates the TSPS process in memory, zeroing all its unprotected data areas. Hence, a minor or major phase will not initialize the TSPS process; an SIA or an SIB is required.

The TSPS No. 1B initialization mechanism employs a sequential escalation for most transitions. In addition, TSPS executes four SIB phases, with each using a different peripheral configuration, before escalating to the next higher DMERT level. However, in some cases, such as manual initialization or an interrupted initialization, the sequential escalation may be bypassed. Control of the escalation sequence is distributed between SIM and AIM. If the highest automatic initialization phase [DMERT level 3 and TSPS level 4 (SIB)] is reached and the system still does not recover, the system will loop continuously, rebooting the system and taking SIBs with varying peripheral configurations until the system recovers or manual action is taken.

A few minor changes were made to the initialization code to allow the TSPS process to take real-time breaks during recovery phases. Since ECMP is not cycling through priority classes during phases, it was necessary to add other breaks during the phases. Continuous running of TSPS during a several-minute-long phase would cause several DMERT sanity and overload checks to fail as lower-priority processes would not be run. Thus, during a phase the TSPS process will break during memory zeroing, for timing loops greater than 10 ms in duration, during audit stitching, and while base level waits for J-level to signal the end of initialization. The length of the phase is not affected by these real-time breaks. The length of the phase is determined by the amount of peripheral equipment, and hence the number of 5-ms interrupts (J-levels) that must occur for peripheral equipment initialization. Although each interrupt will execute faster, the total

Table III—DMERT level (EAI command)

| | 0 (Application Only) (50) | 1 (51) | 2, 3, 4 (Bootstrap) (52, 53, 54) |
|---|---|---|---|
| Application parameter | | | |
| 0 | No action | Reference return | Boot without TSPS process |
| 1 | MNA | MNA | Boot + SIA |
| 2 | MJA | MJA | Boot + SIA |
| 3 | SIA | SIA | Boot + SIA |
| 4 | SIB | SIB | Boot + SIB |
| L | Limp mode | Limp mode | Boot + limp mode |
| Other values | No action | Reference return | Boot + SIA |
| No value (null) | No action | Reference return | Boot + SIA |

elapsed time depends on the number of interrupts and not the speed at which they execute.

*5.2.2.3 Reference returns.* Taking reference returns in the TSPS process is more difficult than on the SPC 1A. All function calls, whether they be between native-code routines or between native code and emulated code (or vice versa), maintain a history of the function call on the process stack. (Transfers between emulated routines do not affect the stack. These transfers work just as on the SPC 1A. In fact, emulated code cannot explicitly access the stack and is unaware of its existence. Only the microcode for the SMT instruction manipulates the stack to maintain a proper stack frame when calling a native routine.) A function must make a normal return to the calling routine to properly unwind the stack. Direct transfers between functions would quickly result in an incongruous stack from which the process would have no way of properly returning to the calling routines. Hence, when taking a reference return, the TSPS process must properly unwind the stack.

To further explain reference returns within the TSPS process, consider a fault generated as the result of an invalid address. The TSPS process will be interrupted immediately, entered at its fault entry, and passed the state of the machine at the time the fault occurred. The fault entry then "generates" an E-level interrupt. The state information indicates the instruction in error and the contents of all the general-purpose registers at the time of the error. Because the TSPS process contains both native and emulated code, this error could have occurred in either. In addition the error could have occurred at a point many levels deep in nested function calls. If a direct transfer to a known "safe" reference point is performed, then a later return to the original calling function (i.e., the event or interrupt entry) will fail as the program stack will not have been unwound properly. To accommodate the existence of the stack, slight modifications to the handling of reference returns had to be made.

If the E-level occurs in H- or J-level, an immediate return from interrupt (using an EGBN or GBNHJ) will be executed. The return from interrupt will clear all of the interrupt's function call entries from the stack and allow the interrupted process to resume processing. This has the effect of canceling not only the current J-level job in progress (as on the SPC 1A), but also all jobs scheduled to execute that 5-ms entry.

If the E-level occurs in base level, a similar situation exists with respect to the kernel stack. In this case, however, a simple return from interrupt will not suffice. Base level must stimulate itself to be re-entered at some later point in time. (J-level, of course, is re-entered by the next 5-ms interrupt.) This is normally done with a time-out request.

This is not sufficient here, either, as base level must know that its next entry is for the purpose of taking a reference return rather than continuing its base-level loop. Hence, base level sends a fault to itself and then performs the return from interrupt. As before, the return from interrupt will clear the stack of any function call entries made by base-level processing. When the fault entry is entered, the fault code will indicate that a base-level reference return is to be made. The fault entry drops to execution level 5 and transfers to the base-level reference point.

The base-level reference-return handling is functionally equivalent to that on the SPC 1A. The only difference is that a momentary delay is experienced while base level relinquishes control of the machine to pop its function call entries from the stack and is then re-entered at its fault entry before taking the reference return.

***5.2.2.4 Sanity.*** The integrity of the system is preserved through a sanity detection mechanism implemented as a hierarchy of sanity timers, integrity checks implemented in native and emulated software, and a recovery mechanism implemented as a hierarchy of initializations.

Each application process has a built-in set of checks designed to detect any abnormal behavior that may lead to insanity. In particular, the TSPS process has preserved, through emulation, the hierarchical structure of sanity checks between base-level, low-priority J-level, and high-priority J-level. Also retained are the critical data structure checks and a monitor of the frequency of maintenance interrupts. The only sanity check that could not be emulated was the hardware long timer, previously implemented in the SPC 1A to check high-priority J-level sanity. This has been replaced by a software sanity timer implemented by AIM.

At the next higher level, the AIM process is responsible for monitoring the sanity of the TSPS process. It does so through the software sanity timer implemented as a counter shared between the two processes. This counter is incremented by high-priority J-level and decremented and then checked by AIM every 500 ms. The expected value of the counter is 0. Repeated non-zero values indicate an insane condition.

The sanity of AIM is in turn monitored by the DMERT System Integrity Monitor (SIM), a process responsible for the sanity of the entire software system. SIM monitors AIM, and implicitly the entire application, by an application sanity timer. AIM activates the application sanity timer after successful process creation in a system bootstrap. AIM must continue to indicate normal operation by sending a periodic sanity event to SIM to reset the timer.

The highest sanity check in the hierarchy is the 3B20D hardware

sanity timer. SIM periodically resets the sanity timer. Hence, the sanity of SIM is implied by the failure of the timer to fire. Should the timer fire, a stop-and-switch operation will be performed to force the off-line processor into the active role.

***5.2.2.5 Overload control.*** The main objectives of the overload control strategy in TSPS No. 1B are to:

(*i*) Preserve system sanity

(*ii*) Maintain a high level of call completions regardless of the load applied to the system.

There are two types of resources that may be exhausted and lead to overload: TSPS call-processing resources, used exclusively by the TSPS process (e.g., TSPS peripherals, software resources used for call processing within the TSPS process, TSPS real time, etc.); and DMERT resources, either used exclusively by DMERT or shared with TSPS (e.g., kernel message buffers, nonswappable main memory, disk swap space, etc).

The TSPS No. 1 overload control strategy for TSPS resources has been preserved in TSPS No. 1B, with the software implementing it being emulated within the TSPS process. In this strategy, the E-E cycle time represents the detection parameter, while the number of calls being admitted and the number of active trunks represent the load control parameters. As the load increases, the E-E cycle time also increases in value. There are three thresholds for the E-E cycle time, determining the transitions from normal state, to phase 1 (minor) overload, to phase 11 (major) overload, and system initialization. During overload, the number of calls admitted and the number of active trunks are gradually reduced. As the load decreases, the E-E cycle time also decreases in value and the system is returned to its normal state.

The overload control strategy for DMERT resources shared by DMERT and TSPS is based on overload detection by the process that administers the particular resource, overload monitoring by the SIM process, and overload control by the SIM and AIM processes. Upon determining that an overload condition exists, SIM notifies the craft, attempts to free some resources to alleviate the condition, and then notifies AIM. For some DMERT-detected overloads (e.g., kernel or supervisor and user-level lockout) an initialization is executed to restore resources. Once notified of these overload conditions, AIM requests a DMERT Level 1 and TSPS minor audit initialization (MNA) initialization. Subsequent overload indications result in escalation of initializations.

***5.2.2.6 Processor switch.*** An example of the communication between DMERT and the TSPS application required for effective control of the system is illustrated by the processor switch. The duplex SPC 1B

normally operates with one processor on-line actively processing calls and the other off-line with its memory continuously updated on every write operation. The off-line unit is thus ready to be switched on-line should the need arise. A processor-switch request can be one of three types: recovery, routine, and manual. A recovery processor switch is requested when a fault has been detected in the on-line processor and processing cannot continue on it. The routine processor switches are scheduled periodically, while the manual requests are made by the craft when needed.

Certain time-sensitive activities within the TSPS application, e.g., writing on the AMA tape, benefit from an advanced notification that a processor switch is about to take place. Such a notification is used to complete time-sensitive operations and get ready for the switch. However, only routine and manually requested switches can be postponed until the application has been notified and given its approval. Therefore, the routine and manual processor switch requests are routed through the AIM process. In turn, AIM notifies the TSPS application processes of an imminent processor switch and, after their completion of time-sensitive operations, returns a switch go-ahead to SIM. As a defense mechanism, if the application does not approve the switch within 10 seconds, DMERT will proceed with the switch.

### *5.3 Other TSPS application processes*

In addition to the TSPS process numerous other native-mode processes were developed to support the emulation. These processes provide needed functions for a complete system under DMERT. The major processes are discussed below. Figure 13 illustrates the interaction between these major processes. The AIM process has been described previously, and therefore will not be included here. The PSI Diagnostic Driver and Control processes are detailed in Ref. 6.

#### *5.3.1 Kernel processes*

***5.3.1.1 PSI diagnostic driver.*** The PSI Diagnostic Driver process is a killable kernel process that performs diagnostic tests of the on-line PSI. To perform the on-line tests the driver must synchronize its operation with TSPS J-level operation in order not to interfere with ongoing peripheral I/O. It will execute the tests at execution level 12 when a non-interfering 3-ms window between J-levels is found. Its other processing is performed at lower execution levels. The driver shares memory with AIM and the TSPS process for synchronization of activities and in order to monitor TSPS peripheral equipment equippage and status.

#### *5.3.2 Supervisor processes*

***5.3.2.1 File system interface.*** The TSPS No. 1B application has only

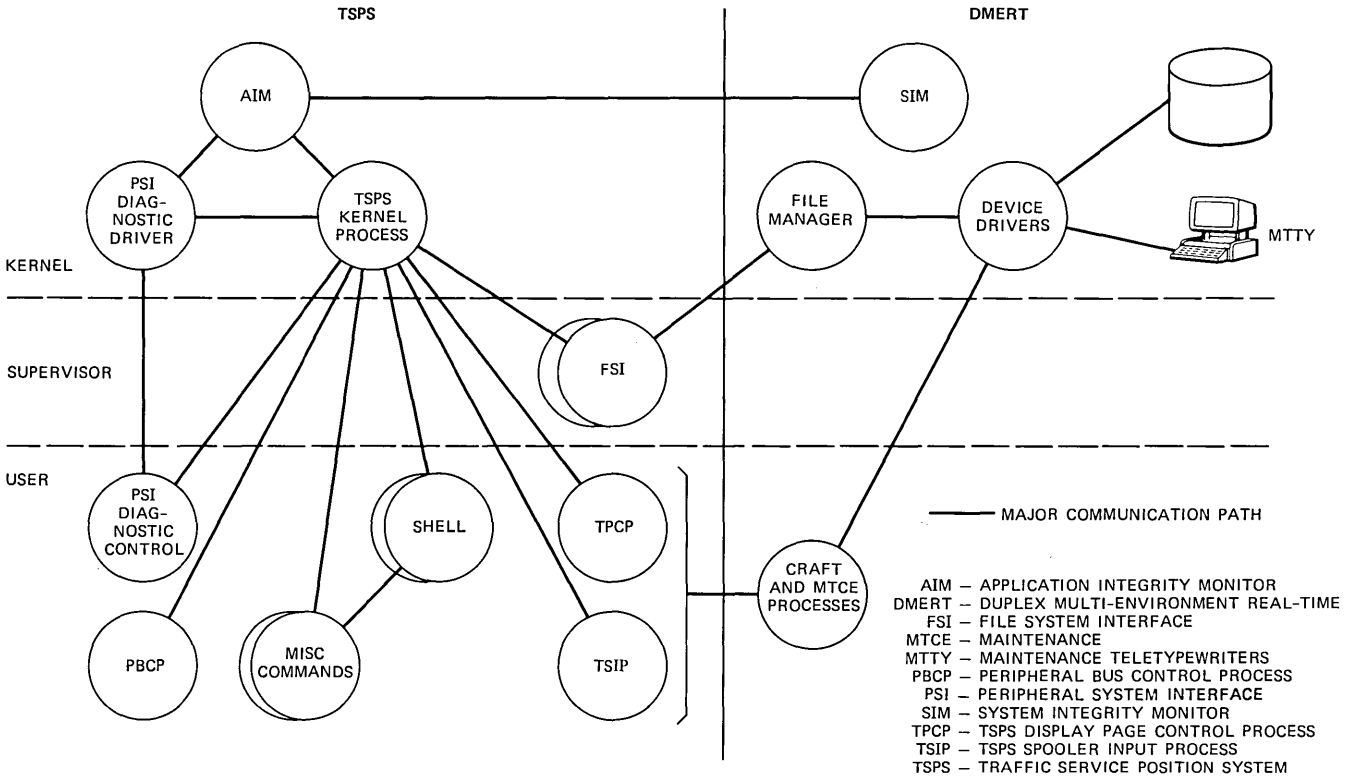



Fig. 13—Interaction between TSPS and DMERT processes in TSPS No. 1B process architecture.

one supervisor process, the File System Interface (FSI). The SPC 1A's program tape unit (PTU) was not retained in TSPS No. 1B. All required operations of the PTU were functionally reproduced with the FSI. Examples of such functions include dumping emulated office data, protected memory, or program to tape. The FSI shares the emulated-code address space and interfaces with the DMERT file manager to transfer data between the emulated address space and the DMERT disk file system. The 3B20D's nine-track tape unit and DMERT I/O facilities are used to move the data between tapes and the disk file system. None of the FSI functions are time-critical. Hence, the FSI is designed as a killable supervisor process that is created upon demand and dies when it has completed its function. Multiple instances of the FSI can simultaneously coexist as long as their respective functions do not interfere with each other.

#### *5.3.3 User processes*

The TSPS application has numerous user-level processes. Most of these perform demand tasks as the result of craft input. The process is created to perform the task and then dies when it is completed. The most significant user-level processes are described below.

***5.3.3.1 Craft interface processes.*** The maintenance center craft interface for the TSPS No. 1B consists of a video display terminal for input, output, and status displays, along with an adjacent printer for a hard copy of all output messages. The terminal's screen is split into four areas, as shown in Fig. 14. The top is for system status information and is always displayed. The variable-sized middle section is for displays which, for example, may show the status of a particular hardware subsystem. The remainder of the screen is for scrolling system output messages, and there is a single, dedicated line for input. This interface is also remoted to the Switching Control Center System (SCCS), described in Ref. 9, for remote maintenance.

Although the actual I/O to the devices is done by the DMERT I/O driver, the data read and written to the devices are formatted and interpreted by a collection of user-level processes. The major DMERT processes include the input Shell, the Controller of Output Spooler Process (CSOP), and the Display Administration Process (DAP).

The application interface to CSOP and DAP is designed to be from other user-level processes. Hence, TSPS has developed two user processes that serve as intermediaries between the TSPS kernel process and CSOP and DAP. The TSPS Spooler Input Process (TSIP) shares a memory buffer with the TSPS process. TSPS queues messages by priority in this buffer. TSIP dequeues these messages and passes them to CSOP via interprocess messages. CSOP then merges the TSPS-generated messages into a single, systemwide, prioritized queue for

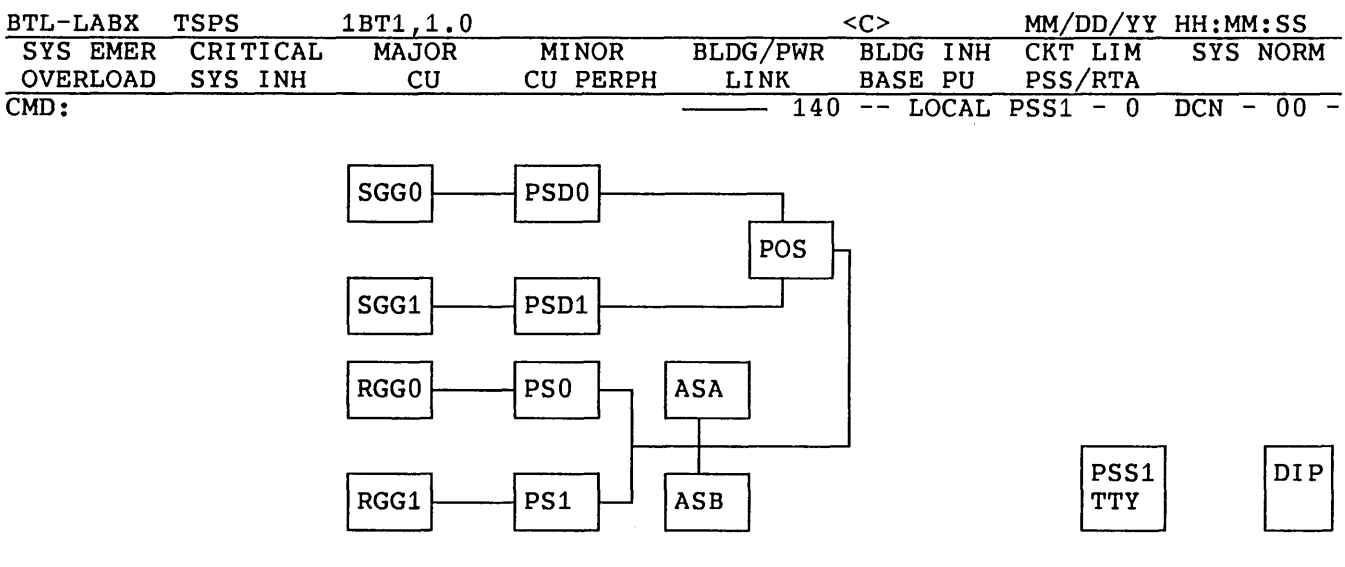



Fig. 14—Maintenance center craft interface for TSPS No. 1B.

printing to the maintenance terminals. The TSPS Display Page Control Process (TPCP) controls the TSPS application displays. TPCP receives change of state information from the TSPS process via interprocess messages. TPCP then translates this into the appropriate display page control information and directs DAP via interprocess messages to change the state of the various display page indicators.

The above-mentioned processes are potentially subject to being killed as the result of a system error or manual action. However, they provide a critical service and must have continuous operation. DMERT provides a Craft Monitor (CMON) process that monitors the operation of craft processes. CMON will immediately recreate any craft process that dies, along with any other process that must be reinitialized. CMON itself is monitored by the User-Level Automatic Restart Process (ULARP). ULARP will automatically restart any user-level process under its surveillance should it die. ULARP execution is monitored by the DMERT System Integrity Monitor process (SIM).

TSPS input message syntax is quite different from that used by DMERT. The processing of these input messages is also performed quite differently. As such, it was necessary to modify the DMERT shell to handle TSPS input as well. When an input message is entered by the craft and read by the combined shell, it first checks for a TSPS syntax message (the most likely to be input). If the input is a TSPS message, it is sent to the TSPS process in an interprocess message. The TSPS process will look up the message in its memory-resident catalog and act on it accordingly. If the input was not a TSPS message, the combined shell handles it in the standard DMERT fashion. That is, a disk directory search is performed to find the appropriate user-level process to execute in response to the command. This process is then created and executed to handle the input accordingly.

***5.3.3.2 PSI diagnostic control process.*** The PSI diagnostic control process is a user-level process that controls the execution of diagnostic tests on the off-line PSI. The PSI diagnostic was designed similar to DMERT common-system diagnostic processes and is woven into the DMERT diagnostic control structure. It is reacted and executed on demand as a result of a manual request, routine exercise or automatic diagnostic request. The PSI diagnostic creates the PSI diagnostic driver and communicates with it through interprocess messages to coordinate on-line and off-line PSI tests and TSPS process F-level interrupt recovery actions.

***5.3.3.3 Peripheral bus control process.*** Power to the PSI peripheral buses is controlled by the 3B20D Processor's scanner/signal distributor (SC/SD). The PSI bus-power control points are duplicated with a set on a SC/SD controller in each I/O processor (IOP). The Peripheral

Bus Control Process (PBCP) monitors and controls the status of the PSI peripheral buses. PBCP interfaces with the DMERT SC/SD administrator and the TSPS process via interprocess messages to coordinate actions. PBCP is a critical process that must always be active. Hence, it runs under the surveillance of ULARP.

## VI. SUMMARY

This article has described how the characteristics of the TSPS No. 1 and the 3B/DMERT systems have been combined in the TSPS No. 1B architecture. Successful emulation of the TSPS No. 1 software has been accomplished by hardware, firmware, and software in the TSPS No. 1B. Emulated code, along with associated native code, has been structured as a single kernel process running under DMERT. This kernel process cooperates with other application processes and DMERT to form integrated maintenance and craft interface packages. The TSPS No. 1B provides a modern, flexible vehicle for the future expansion of TSPS services.

## VII. ACKNOWLEDGMENTS

The TSPS No. 1B software described in this article was the result of contributions by many people. The authors wish to acknowledge their help in the preparation of this article. In particular, the authors wish to thank J. M. Aiken, P. S. Bogusz, D. L. Brown, D. L. Hofmockel, M. H. Richardson, E. S. Sachs, and M. D. Soneru for their contributions.

## REFERENCES

1. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
2. B.S.T.J., *49,* No. 10 (December 1970).
3. B.S.T.J., *58,* No. 6, Part 1 (July-August 1979).
4. "3B20D Processor & DMERT Operating System", B.S.T.J., *62,* No. 1, Part 2 (January 1982).
5. D. L. Bayer and H. Lycklama, "*UNIX* Time-Sharing System: The MERT Operating System," B.S.T.J., *57,* No. 6, Part 2 (July-August 1978), pp. 2049–86.
6. H. A. Hilsinger, J. H. Tendick, R. A. Weber, and G. T. Clark, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
7. T. Hack, T. Huang, and L. C. Stecher, "Traffic Service Position System No. 1B: Software Development System," B.S.T.J., this issue.
8. B. A. Crane and D. S. Suk, "Traffic Service Position System No. 1B: Capacity and Reliability Evaluation," B.S.T.J., this issue.
9. J. J. Bodnar, J. R. Daino, and K. A. VanderMeulen, "Traffic Service Position System No. 1B: Switching Control Center System Interface," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Hardware Configuration

By G. T. CLARK, H. A. HILSINGER, J. H. TENDICK, and R. A. WEBER

(Manuscript received June 30, 1982)

*The Traffic Service Position System No. 1B (TSPS No. 1B) was developed by replacing the Stored Program Control No. 1A (SPC 1A) with the SPC 1B, consisting of a 3B20 Duplex Processor (3B20D) and a Peripheral System Interface (PSI). The PSI was designed to interface the 3B20D Processor technology with the existing TSPS peripheral system. This article describes the differences in technologies between the SPC 1A and SPC 1B, the hardware design required to overcome these differences, and the fault recovery and diagnostic software development required to integrate the new hardware into the maintenance structure of the TSPS No. 1B.*

## I. INTRODUCTION

The 3B20 Duplex Processor (3B20D) was developed as a general-purpose processor with a set of common system peripherals to support a wide range of applications and an instruction set optimized for a high-level language compiler.[1] To allow the 3B20D Processor to communicate with the Traffic Service Position System (TSPS) peripheral community, a special interface circuit was needed to bridge the difference in speed, timing, and control protocols between the new processor and the existing TSPS peripherals. This circuit is called the Peripheral System Interface (PSI) circuit. To bridge the software technologies without rewriting all existing TSPS operational code, the Stored Program Control No. 1A (SPC 1A) instruction set was emulated via special microcode, and 3B20D native code was used to interface the emulated code with the DMERT operating system. With these changes in hardware and software came the task of developing a

maintenance strategy that could integrate the advantages of the 3B20D Processor and Duplex Multi-Environment Real-Time (DMERT) operating system with the existing TSPS maintenance strategy. Finally, to introduce the 3B20D Processor into in-service TSPS offices, retrofit procedures had to be developed to replace the existing TSPS No. 1 processor with a 3B20D Processor.

The task of incorporating the 3B20D Processor into the TSPS system provided a unique set of challenges. This article describes the hardware and associated maintenance software required to accomplish this task. The emulation microcode, overall software architecture, and the implementation of the processor retrofit are covered in Refs. 2 and 3.

## II. TSPS NO. 1B PROCESSOR AND PERIPHERAL INTERFACE DESCRIPTION

The TSPS No. 1B configuration consists of the existing TSPS peripheral system and a new processor called the SPC 1B. The SPC 1B consists of the 3B20D central control unit, disk file community, input/output processors, and the Peripheral System Interface unit (see Fig. 1*).

The peripheral architecture for TSPS No. 1B remains essentially unchanged from TSPS No. 1. The interface between 3B20D Processor and the TSPS peripheral communication buses is provided by the Peripheral System Interface circuit. In the following sections a short description will be given of the SPC 1A, and the TSPS peripheral system architecture. These sections are included to highlight the significant differences in processor design and to identify integration requirements. For a more detailed description of the SPC 1A see Ref. 4.

### *2.1 Description of the SPC 1A*

The SPC 1A consists of a duplicated pair of central control (CC) units, a duplicated program and data store complex, a program tape unit (PTU), a master scanner, a signal distributor, a central pulse distributor (CPD), a maintenance control center, and a duplicated peripheral communications bus system (see Fig. 2).

The CC has a processor cycle time of 6.3 $\mu$s, with an instruction execution time of from 1 to 3 cycles. Most register operations require a single cycle, while memory writes require three cycles, and most peripheral orders require two cycles. The processor performs logic with a 20-bit combinational logic subtractor, which also has the capability

* Acronyms and abbreviations used in the figures and text of this paper are defined at the back of this Journal.

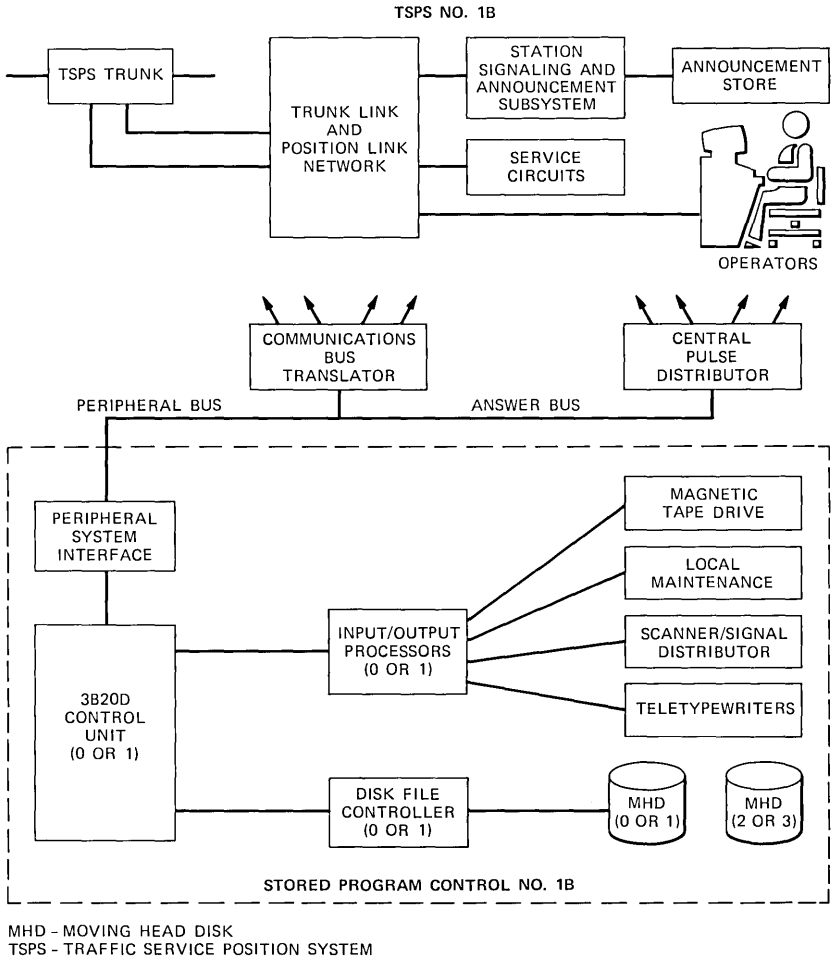



Fig. 1—Traffic Service Position System No. 1B.

of performing the logic operations AND, OR, and EXCLUSIVE-OR. Circuits are also provided to perform shift or rotate functions, both right and left. There are seven general-purpose registers each containing 20 bits, which provide internal storage for information manipulation.

Although the CC arithmetic/logic unit (ALU) operates on 20-bit words, the communication between the CC and the main stores is over a 47-bit data and instruction bus with a 19-bit address bus. Data are physically stored as 47-bit blocks, which consist of either a 40-bit instruction or two 20-bit data words, six Hamming error-correcting code bits, and an overall parity bit. An additional bit is used internal

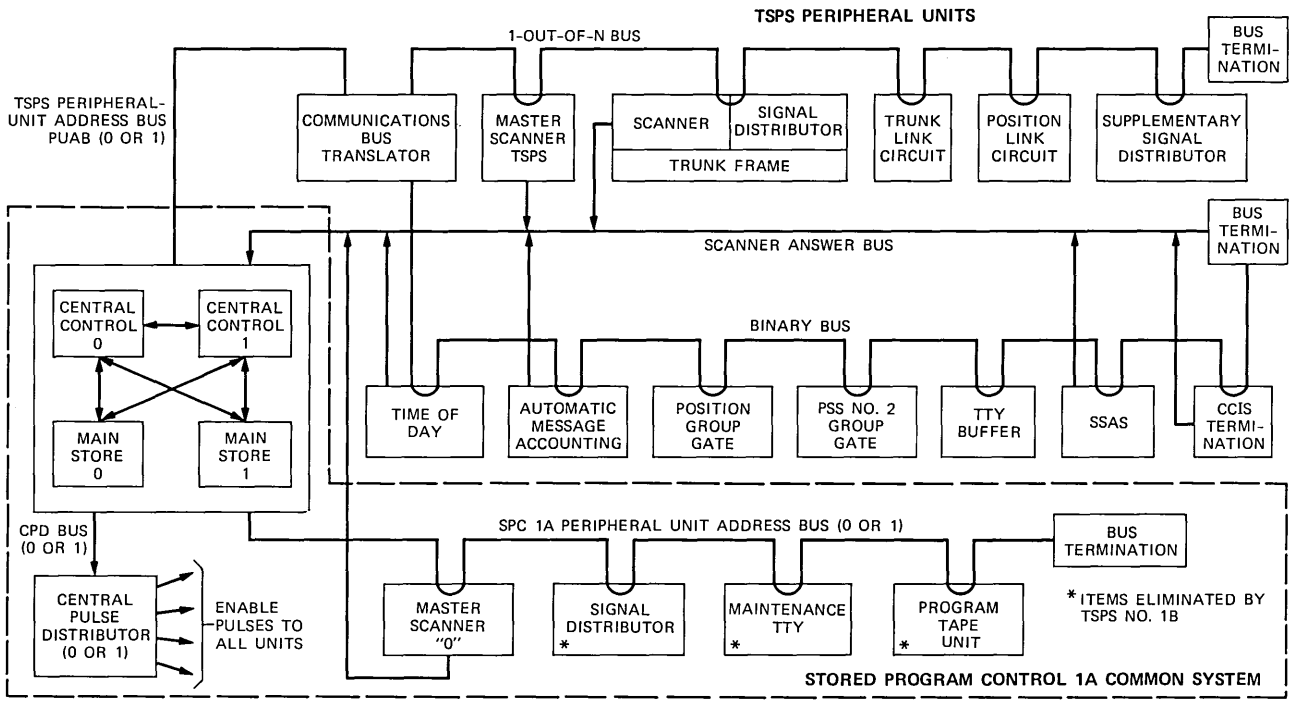



Fig. 2—Traffic Service Position System No. 1.

to the processor to select one of the 20-bit data words for read and write operations. This additional bit provides an effective 20-bit addressing capability.

The SPC 1A program and data store complex consists of duplicated store communities with each community containing up to a maximum of 30 store modules with each module containing 16,384 47-bit words. These stores are used for both program and data storage. Each processor has access to both store communities for increased reliability. Normal nonfault operation is for both store buses to be active with the active processor accessing both stores and comparing results. For processor write operations both store buses are written in order to maintain up-to-date copies in both memories.

The interface of the processor to the peripherals is provided by four AC buses. First is the SPC 1A peripheral-unit address bus, which is dedicated to the units that are common to every SPC 1A installation. The common equipment includes a master scanner, signal distributor, maintenance teletypewriter (MTTY), and a program tape unit (Fig. 2). The second bus is the Peripheral-Unit Address Bus (PUAB), which transmits data to the TSPS peripheral units. The third bus is the Central Pulse Distributor Bus (CPDB), and the fourth is the Scanner Answer Bus (SAB). A more detailed description of this peripheral interface will be given in Section 2.2, since it is this peripheral bus architecture that must be interfaced with the 3B20D Processor.

As mentioned above, a program tape unit, master scanner, signal distributor (SD), central pulse distributor (CPD), and maintenance control center are the basic set of peripherals required for all SPC 1A applications. The program tape unit is a nine-track, 200-bit-per-inch (BPI) tape unit that provides a bootstrap program loader as well as a general-purpose input/output capability. The master scanner is used throughout the TSPS No. 1 for data input from the peripheral equipment. The master scanner contains a matrix of scan points connected in such a manner that they can be interrogated in groups of 16 to determine the state of a particular scan point. The output of the scan points is transmitted to the SPC 1A on the SAB. The signal distributor is accessed via the SPC 1A Peripheral Unit Address Bus and provides a matrix of latching relay contacts that may be opened or closed on command from the processor. The central pulse distributor (CPD) is used to provide unit addressing for data transfers over the shared system buses. The CPD is a translator decoder that takes a binary address from the CPDB and provides a unipolar or bipolar pulse on one of a possible 1024 points. The maintenance control center consists of the MTTY and the Control and Display circuit. The MTTY provides the primary craft interface function. System configuration, status, and system diagnostics may be controlled via MTTY input. In

addition to the MTTY, the Control and Display circuit visually indicates the system status through indicator lamps and allows manual control of the system by means of keys and switches.

### *2.2 Description of the TSPS peripheral interface*

The primary hardware development for the TSPS No. 1B was the PSI, which interfaces the 3B20D to the existing TSPS periphery. The TSPS peripheral interface is described at this time to provide background for description of the PSI.

The SPC 1A Peripheral-Unit Address Bus shown in Fig. 2 has been eliminated for TSPS No. 1B. The units associated with this bus, the master scanner, the signal distributor, the MTTY, and the program tape unit, have either been eliminated or moved to the TSPS Peripheral-Unit Address Bus. The signal distributor was used primarily for SPC 1A maintenance and control and is no longer needed. A few signal distributor points that were not associated with the SPC 1A were reassigned to signal distributor points in the TSPS periphery. The MTTY and the program tape unit were replaced by equivalent functions in the 3B20D Processor, and the master scanner was moved to the PUAB because many of the scan points were used by TSPS peripheral circuits.

The peripheral bus structure retained after replacement of the SPC 1A consists of the Central Pulse Distributor Bus, the Peripheral-Unit Address Bus and the Scanner Answer Bus. These three peripheral communication buses retained for TSPS No. 1B contain four types of leads: data, control, error detection, and diagnostics.

The CPD bus includes 32 enable address bits that select one of the possible 1024 enable pulse outputs from the CPD. Control leads include a bus sync and four execute pulses that activate the CPD decoding logic. Each execute pulse is dedicated to one CPD unit. An additional control lead is the "we-really-mean-it" (WRMI) pulse, which is used to provide additional signaling redundancy, thereby increasing noise immunity. The error-detection leads include parity check bits across the data bus, an execute complete signal, an "all-seems-well" (ASW) response bit, a bit that indicates single or multiple CPD output pulses, and two enable-verify reply signals that indicate reception of an enable pulse by the selected peripheral unit. Diagnostic and maintenance leads include a master reset, and a bit to diagnose error-checking circuits.

The PUAB leads pass through a Communications Bus Translator (CBT) circuit before being transmitted out to the TSPS peripherals. The PUAB has 20 bits of data and three control leads. The control leads include a CBT reset pulse to clear CBT input registers, a latch pulse to load the CBT input registers, and an execute bit that causes

the CBT to send translated data to the peripheral units. Error-checking leads include a parity bit over the data field, and an ASW bit, which is a summary of error checks performed by the CBT.

The SAB is the input bus to the processor; it includes 20 bits of data. Bit twenty is used by some peripheral units as a parity bit; the processor provides a parity check on the scanner answer data. Control functions for the scanner bus are all provided internal to the processor and consist of a window during which peripherals must load answer data onto the bus.

The error-checking leads described above are connected to check circuits in the processor or connected directly to a set of hardware registers called the Peripheral-Unit Maintenance Summary (PUMS) registers. The outputs of the error-checking circuits are also loaded into the PUMS register. The data in the PUMS register is compared with expected data, and if any errors are detected, the processor circuit generates a maintenance interrupt that results in the scheduling of TSPS fault-recovery programs.

Other peripheral interface leads include clock leads and processor-controlled pulse leads. The clock leads include 91-$\mu$s, 0.5-$\mu$s, and 5-ms output pulses, which drive the Automatic Message Accounting tape units, teletypewriter buffers, and other circuits that need them. The pulse leads are direct processor-controlled leads called "P-position" pulse sources. The P-position pulses were provided in the SPC 1A by special processor hardware called Buffer Bus Registers.[4] The purpose of these pulse sources is to provide direct processor control of critical peripheral units for hardware recovery in the event that the normal peripheral communication path is failing. These pulses include: a CPD reset; a network reset for trunk-link and position-link circuits, universal trunk frames, etc; and an auxiliary reset pulse, which is presently used for the TSPS Position Group Gate and the Service-Observing Gate transmitter circuit.

In essentially all cases, the characteristics of the signals transmitted and received on these buses are bipolar AC pulses of 0.5-$\mu$s duration and approximately 6-volt amplitude.

All peripheral orders—both distributes(DIST) and scans (SCAN)—operate with the same basic sequence. An address is loaded onto the CPD bus, which activates one of a possible 1024 enable-pulse points out of the CPD; immediately after the CPD address is pulsed, binary data may be placed onto the PUAB. The enable pulse from the CPD activates receiving circuits in the selected peripheral unit and primes that circuit to receive data over the PUAB. The selected circuit responds based on the data received and the CPD pulse received. For example, if a scanner were being accessed, the binary address would specify a group of 16 scan points to be interrogated, and the state of

the scan points would be gated onto the SAB and returned to the processor. By requiring each circuit in the periphery to read the binary address only after receiving a CPD enable pulse, a high degree of noise immunity and reliability is achieved in peripheral communications.

In summary, there are greater than 200 data and control leads that provide communications between the TSPS peripherals and the processor. They are AC bipolar pulses and are transformer-coupled to the peripheral buses.

## III. GENERAL CHARACTERISTICS OF THE 3B20D PROCESSOR

The central control (CC) of the 3B20D Processor[5] is microprogrammed, extensively self-checked, and handles 32-bit data words in a 24-bit address space. The control unit (CU) consists of the CC, its main memory, cache, Direct Memory Access (DMA), and input/output channels. In the standard duplex processor configuration, two control units operate in a nonmatching mode with one designated as active, while the other serves as standby-ready. A main memory update circuit maintains the consistency between main stores to ensure an up-to-date environment in the event of a processor switch. The SPC 1B Processor and peripheral architecture is shown in Fig. 3.

The 3B20D Processor supports a wide variety of peripheral devices including moving head disks, magnetic tape drives, data links, and terminals. Peripherals are controlled in programmed I/O mode directly by the processor, or with Direct Memory Access (DMA) through appropriate interface circuitry. A Disk File Controller (DFC) unit manages the information flow to and from the moving head disk drives, and an Input/Output Processor (IOP) unit provides interface control for magnetic tape drives, data links, terminals, and other customized peripheral controllers.

The central processing unit of the 3B20D utilizes a 32-bit data architecture throughout, including the main memory buses and an optional 8K-byte cache memory. Extensive self-checking is used to ensure immediate fault detection and possible correction of errors to allow graceful recovery and ensure high system availability. Hamming correction of single-bit errors and detection of all double-bit errors are performed by the main-store controller on each memory read operation. In addition, on every refresh cycle, a memory word is checked such that all words are tested periodically. This capability ensures that memory failures occurring on infrequently used addresses will be detected quickly.

Software processes on the 3B20D use a 24-bit virtual address space, which is converted by memory management hardware to 24-bit physical addresses using a paged-segmentation scheme. The memory man-

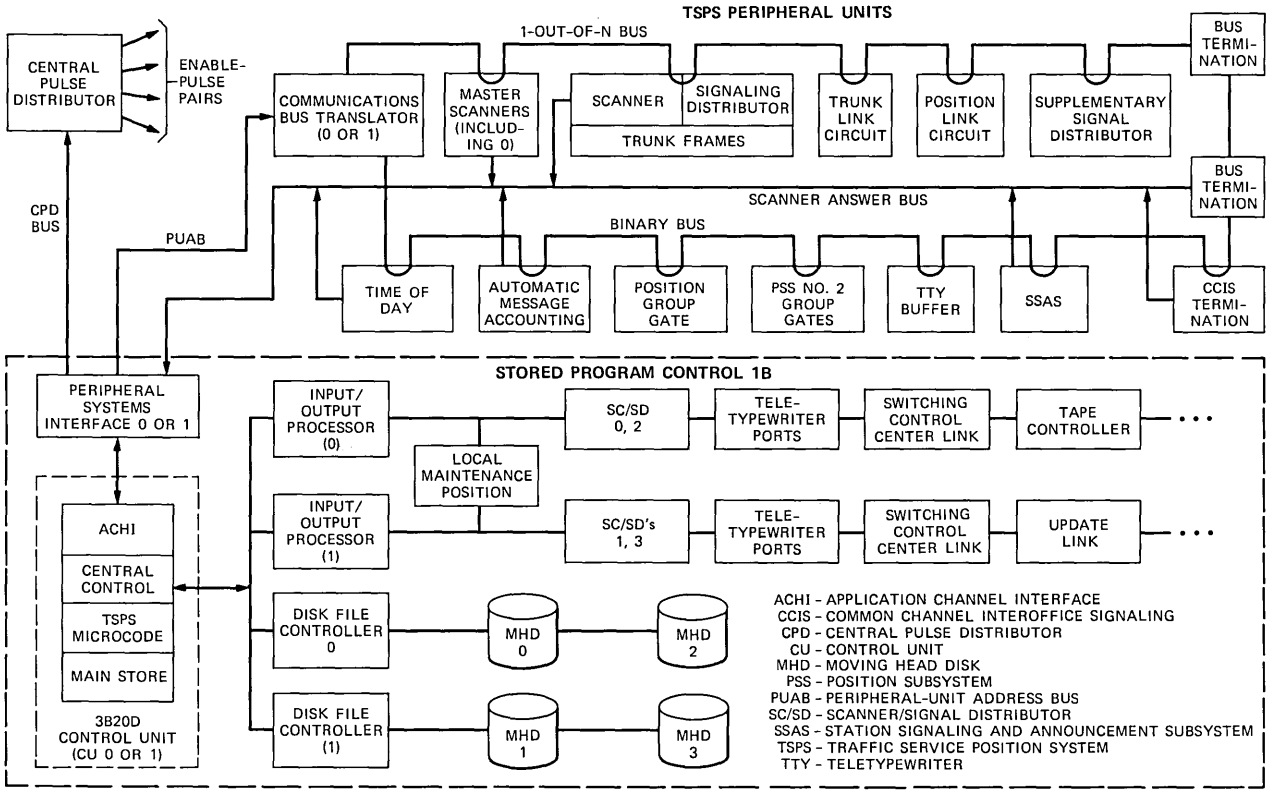


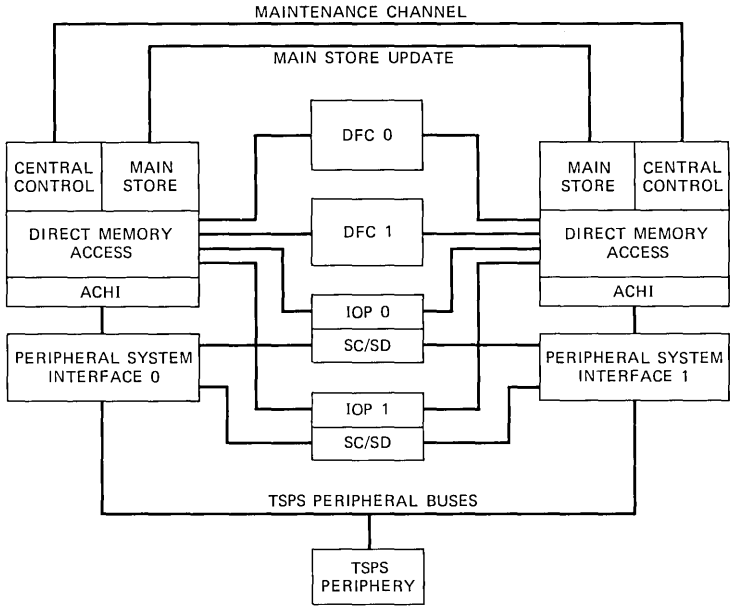

Fig. 3—The SPC 1B Processor and peripheral architecture.

agement circuitry is a high-speed, two-way, set-associative memory called the Address Translation Buffer (ATB). The ATB reduces overhead associated with the address translation function, and also provides main-store access protections on a page or segment basis. Physical memory can grow in half- or full-megabyte increments to a total of 16 megabytes. Memory instructions provide addressing capabilities with byte, half-word, full-word (32 bits), or move-block options.

The 3B20D Processor is a microprogrammed machine where up to four instruction sets can be dynamically selected. A variable microcycle ranging from 150 to 300 nanoseconds is employed to minimize instruction time. The switchable instruction sets allow an application to transport current software, via emulation, to the 3B20D while still providing a modern operating system in a high-level language (i.e., DMERT and C language). The native instruction set is optimized for the C programming language and supports all C data types and control structures. Additional features within the CC include: duplicated Arithmetic/Logic Units (ALUs) for immediate fault detection; extensive bit rotate, mask, and test capability; a real-time clock with one-millisecond resolution; and internal sanity timers.

In its duplex configuration, two basic links exist between duplicated CUs. The first link is driven by the maintenance channel circuit and serves as a communication and control link between the processors. This link is used to execute off-line diagnostics, off-line audits, and information transfers during a processor switch. The second link connects the memory update circuits, which keep both main stores synchronized by sending all writes to cache and main store to the off-line memory controller. This second link also contains a backup maintenance channel in the event of a fault in the regular link.

Peripheral units are connected to the CU via the Direct Memory Access (DMA) unit. The DMA does not interface directly with peripheral units, but rather communicates with two intelligent subsystems; a Disk File Controller (DFC), and an Input/Output Processor (IOP). Communications to both the DFC and IOP are via Dual-Serial Channels (DSCH), which allow any peripheral to operate with either CU of a duplex pair. Each DFC is currently capable of supporting 16 moving-head disk drives of 300-megabyte capacity each. An IOP is capable of supporting a wide variety of peripherals such as nine-track tape units, printers, synchronous and asynchronous data links, maintenance and general-purpose terminals, scanner/signal distributor, and custom network interfaces. Peripherals may also be connected to the processor Central Control Input/Output (CCIO) bus via an interface such as the Application Channel Interface (ACHI). The ACHI is a 32-bit, parallel-interface board that provides I/O capabilities for applications needing high-speed data transfers with minimal overhead.




ACHI - APPLICATION CHANNEL INTERFACE
DFC - DISK FILE CONTROLLER
IOP - INPUT/OUTPUT PROCESSOR
SC/SD - SCANNER/SIGNAL DISTRIBUTOR
TSPS - TRAFFIC SERVICE POSITION SYSTEM

Fig. 4—Duplex configuration of the TSPS No. 1B.

## IV. CONFIGURATION OF THE TSPS NO. 1B

The TSPS No. 1B configuration consists of the duplex SPC 1B, TSPS peripheral units, several printers, terminals, and a duplicated link to the Switching Control Center System (SCCS), which serves as the remote maintenance center.[6] The SPC 1B includes the 3B20D Central Control Unit (CU), disk file community, Input/Output Processor (IOP), TSPS emulation microcode, Application Channel Interface (ACHI), and the Peripheral System Interface. The overall TSPS No. 1B configuration is shown in Fig. 4.

The 3B20D Processor is equipped with nine megabytes of main store for the initial TSPS No. 1B program generic. Hardware added to the basic processor for the TSPS application includes two boards containing three kilobytes of Programmable Read-Only Memory (PROM) for the emulation microcode, and an Application Channel Interface (ACHI) circuit board. The ACHI plugs directly into the CCIO bus and provides a high-speed, 32-bit data transfer interface to the PSI. The PSI was designed as an interface for the different I/O signaling protocols of the 3B20D and the existing TSPS peripheral circuits. A

detailed description of the PSI is given in Section V. The TSPS periphery[4] is preserved in its same basic configuration with only minor differences.[3]

The disk file community for TSPS consists of duplicated DFCs including two Moving Head Disks (MHDs) for each disk file controller (DFC). One disk on each DFC is physically designated as the system disk and contains the DMERT and TSPS programs and data. The other disk drive is a spare that is fully diagnosable and contains a copy of the system disk. If a system disk does fail, the spare drive may be recabled to provide a full-duplex disk configuration once again.

The IOPs are equipped to provide four scanner/signal distributors plus interface boards for a 1600-bpi magnetic tape frame, Craft Interface MTTY, Receive-Only Printer (ROP), an office alarm control circuit, a recent change and verify terminal, an auxiliary printer, a data-linked Field Update interface,[7] and remote SCCS data links. The additional scanner/signal distributor monitors the PSI power switch and also provides duplicated control over +24-volt power to the PSI peripheral bus drivers. The MTTY and ROP are separately connected to one of the two IOPs through an automatically or manually controlled port switch. This 3B20D feature provides a duplicate path between the central processor and the maintenance center equipment without requiring recabling or duplicate terminals. Included with the common 3B20D equipment is a software package to receive and verify software updates at the field site over data links.[7] The SCCS interface is a duplicated 4800-baud link that provides 14 virtual channels through a BX.25 protocol. The channels are used to transmit critical status indicators, alarms, processor recovery messages, and normal maintenance channel messages. In the typical configuration, the SCCS is used to remotely monitor the TSPS sites and has complete capability to control the machine in recovery situations.[6] The local MTTY is a video display console that shows critical indicators, provides several display pages depicting system configuration information, and allows craft personnel to enter system input messages. TSPS equipment configurations have been incorporated into the MTTY displays by providing additional display pages and additional page numbers for the application-only displays. A description of the Craft Interface is given in Ref. 2.

An extensive, non-interfering, field-debugging utility is also available by inserting a Dual Utility Circuit (DUC) into the 3B20D. The DUC has a dedicated slot in the processor that has access to all the important system buses. A Field Test Set (FTS) connects to the DUC via ribbon cable to provide a program debugging environment for both high-level programming languages and TSPS emulated code. Address and data matchers are available, as well as program tracing capabilities. More

information about the debugging environment used in the TSPS No. 1B development is given in Ref. 7.

## V. PERIPHERAL SYSTEM INTERFACE

### 5.1 Basic purpose

The basic purpose of the Peripheral System Interface is to act as a data transfer interface with the TSPS periphery identical to that provided by the SPC 1A. TSPS peripheral units communicate with the SPC 1A via the PUAB, CPDB, and the SAB. The 3B20D Processor communicates with its channels over the CCIO bus. For proper operation, TSPS units require a precisely timed set of handshaking signals in addition to electrical compatibility with the CCIO bus. To avoid changing the peripheral units themselves, the PSI was designed to interface the TSPS periphery with the 3B20D Processor. In addition, an ACHI is provided with the 3B20D Processor to allow PSI indirect access to the CCIO bus. In essence then, the PSI appears as a main I/O channel to the 3B20D Processor. The overall structure incorporating the 3B20D Processor and the PSI into TSPS is shown in Fig. 4; Fig. 5 is a functional block diagram of the PSI circuit.

### 5.2 Overview

The general flow of information between the 3B20D Processor and the TSPS periphery is as follows:

(*i*) A command word along with data information is formatted by the emulation microcode and sent to the ACHI.

(*ii*) The PSI receives the command, decodes the instruction, and starts the appropriate handshaking sequence on the TSPS peripheral buses.

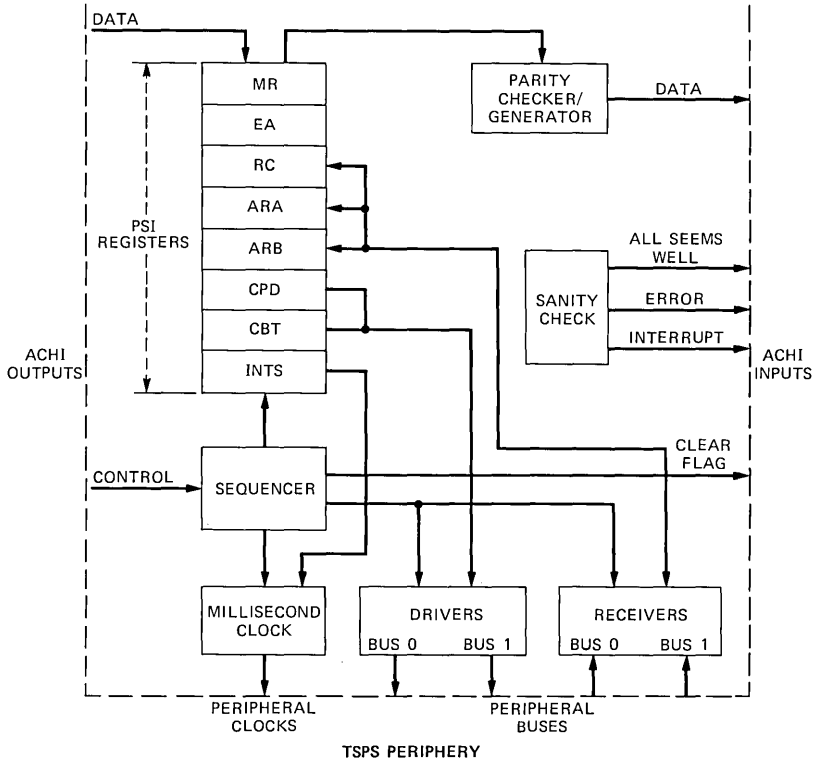
(*iii*) Once the command has begun, the emulation microcode passes a second data word via the ACHI to the PSI. This data is then formatted by the PSI and passed onto the peripheral circuits at the appropriate moment.

(*iv*) Any data sent back from the TSPS periphery along with error check signals are verified and then combined with internal PSI fault-detection information and passed back to the ACHI.

(*v*) The CU receives the data and emulation microcode performs further post analysis and error checking on the data before the conclusion of the peripheral order.

### 5.3 Application channel interface

The ACHI is a 3B20D Processor I/O channel that provides a high-speed, parallel, 32-bit interface between the 3B20D and an application device. Control signals allow the information transfers to be marked as




ACHI - APPLICATION CHANNEL INTERFACE
ARA - ANSWER REGISTER A
ARB - ANSWER REGISTER B
CBT - COMMUNICATIONS BUS TRANSLATOR
CPD - CENTRAL PULSE DISTRIBUTOR
EA - ENABLE ADDRESS
INTS - INTERRUPT SOURCE REGISTER
MR - MODIFICATION REQUEST
PSI - PERIPHERAL SYSTEM INTERFACE
RC - REPLY CHECK

Fig. 5—Block diagram of the Peripheral System Interface circuit.

either data or command information when transmitted over the CCIO bus. When the application passes data to the 3B20D, additional control signals are used to indicate an "all seems well" or error response. Other miscellaneous control signals are also buffered by the ACHI. One such control signal used by the PSI keeps the 3B20D Processor and PSI clocks synchronized, while another is connected to a high-priority processor interrupt to signal when a fatal PSI hardware fault is detected. A third lead is an extension of the CCIO bus I/O inhibit lead. This signal is asserted in the off-line processor to inhibit all channels, thus preventing any off-line channels from interfering with the active processor. The PSI uses the inhibit lead in the same manner; it directly prohibits all PSI peripheral bus drivers from pulsing even though the sequencer may be active.

### 5.4 PSI functional description

The five major functional areas in the PSI are:

(*i*) The ACHI interface
(*ii*) PSI registers
(*iii*) Peripheral bus drivers/receivers
(*iv*) Instruction sequencer and clock circuits
(*v*) Maintenance and fault-recognition circuits.

Each of these areas is described in more detail in this section.

#### 5.4.1 ACHI interface

The ACHI interface uses 5-volt, differential dc signaling on 80 pairs of leads. There are separate input and output buses, with each bus containing 32 data bits, 4 bits of byte parity, and 4 control leads. Data transfers from the ACHI to the PSI are accomplished by making data available on the PSI's input bus and then raising the appropriate command or data flag. This flag starts the PSI sequencer, which gates the data into the appropriate register and resets the control and data flags. The PSI passes data back to the processor by driving its output data bus and then pulsing the normal or maintenance control leads. These leads are used internally in the ACHI to latch the new data and also to indicate that data is present on the next status interrogation by the emulation microcode.

#### 5.4.2 PSI registers

The eight registers in the PSI are used to store data from the ACHI, the periphery, error-check signals accumulated during an order, and to perform maintenance control functions or operations. The CPD and CBT registers are used to store data received from the ACHI for the currently executing peripheral instruction. The outputs of these registers are gated through decoder circuits and then pulsed onto the peripheral buses at the appropriate time. The two answer registers (ARA, ARB) receive data from the duplicated peripheral Scanner Answer Buses. This data is typically matched at the end of the peripheral order sequence and one set is passed to the processor through the ACHI. The Reply Check (RC) register gathers peripheral-error-response data and is used to determine whether to assert the normal or maintenance flag when sending data back to the processor. The Maintenance Register (MR), Enable Address Register (EA), and the Interrupt Source Register (INTS) are all used by diagnostics to provide special maintenance observation and control points during diagnostic exercises.

#### 5.4.3 Peripheral bus drivers/receivers

The PSI must interface to three main TSPS peripheral buses. All peripheral signaling uses 0.5-$\mu$s pulses over AC transformer coupled

buses for maximum isolation and noise immunity. The purpose of each bus is described in Section 2.1. The PSI has access to both halves of each of the duplicated buses.

#### 5.4.4 Instruction sequencer and clock circuits

The PSI sequencer consists of a dual set of timing chains, driven by a 10-MHz oscillator, that can be used to provide pulses from 100 ns to 12 $\mu$s wide. This is accomplished by having the first chain pulse one of its ten output leads every 100 ns, while the other chain pulses one of its twelve output leads every microsecond. These pulses are then logically combined to provide variable-length timing control pulses throughout the execution of the PSI commands.

Four major types of instructions are sequenced by the PSI. The peripheral order is the most common instruction and has three built-in options. These options allow orders to set network points and return response data with or without odd parity, or to set network points only. The pulse order is another instruction that communicates with peripheral equipment and is used to reset network points. Finally, read and write instructions allow direct access to the PSI's registers.

A millisecond clock circuit uses countdown logic, from a 10-MHz source, to produce a 0.5-$\mu$s pulse every 90.9 $\mu$s, 500 $\mu$s, and 5 ms. These pulse chains are used by AMA circuits, TTY buffers, and other miscellaneous peripherals. Through a control lead from the ACHI, the PSI system clocks are aligned with the 3B20D interrupt timers. This feature is necessary to align the AMA data transfers with the clocks to physically write the bytes onto the tape. The alignment process is also used to detect a faulty PSI clock circuit by comparing the states of the 3B/PSI clocks every 5 ms and to generate an interrupt if the PSI clock appears to be fast or slow.

#### 5.4.5 Maintenance and fault-recovery circuits

Since the PSI is considered part of the SPC 1B processor, high reliability and availability plus immediate fault detection is provided. A considerable amount of circuitry is included for special maintenance access for fault resolution and also for immediate error detection during peripheral order sequences. Five separate error sources are combined to generate a processor interrupt if an internal PSI hardware error is detected during the sequencing of an instruction. Command-field parity, data-field parity, multiple commands, data missing, and sequencer sanity checks are made on each order. Sequencer sanity checks are made by comparing a predicted bit stream stored in Programmable Read-Only Memory (PROM) against the actual sequence of pulses. If any error checks fail, an error indication is latched in the PSI INTS register and an interrupt signal is pulsed through the

ACHI and latched into the 3B20D interrupt source register. At the conclusion of the instruction, fault-recovery software is dispatched to take appropriate recovery actions. The clock circuit continuously drives three system clock chains out on the peripheral buses. In addition to possible errors during command sequences, any circuit failures or fast and slow clock errors generate an immediate interrupt, which is passed onto the 3B20D Processor.

Additional maintenance features are included to aid the fault-resolution process and to allow major internal portions of the PSI to be tested before peripheral drivers are enabled out onto the buses. One such feature allows the PSI diagnostic to send data out of the ACHI, through the PSI, and back to the ACHI without any PSI sequencer actions. This allows the ACHI and interface cable faults to be isolated from PSI problems. Another major diagnostic feature allows the preloading of answer and reply check registers before a peripheral order is executed. This allows the diagnostic to check nearly all features of the sequencer and error check hardware before enabling the PSI drivers onto the active peripheral buses. Therefore, this feature helps prevent interference problems between the duplicate processors and allows faults to be distinguished between internal sequencer problems and peripheral interface circuits.

## VI. PHYSICAL DESCRIPTION OF THE SPC 1B

The SPC 1B consists of a number of different frames, units, circuit packs, and interconnections to accommodate the various circuits that comprise the system. These are arranged in configurations that allow for redundancy and growth to achieve a reliable and repairable system.

### *6.1 General*

Most of the wired frames use a newly designed framework developed for the 3B20D Processor.[8] The new framework is 2-ft 2-in. wide, 2-ft deep, and 7-ft high. The greater depth, as compared to existing TSPS frames, which are 1-ft deep, allows for the use of a larger circuit pack and still can accommodate backplane interconnections. Frame uprights are drilled on four surfaces that allow for mounting units of different sizes and dimensions. Some of these holes are used for mounting cable brackets, wire guides, and other hardware for supporting both intraframe and interframe cabling. *BELLPAC** hardware was used in the majority of the unit designs. This consists of apparatus mountings, mounting plates, and other backplane hardware. Units are arranged to accommodate circuit packs of the 8- x 13-inch size. Printed

* Trademark of Western Electric.

wiring backplanes range from two-layer battery and ground to multilayer arrangements for the more complex circuits. One or more units are provided for each frame as required to accommodate the numerous circuits that make up the SPC 1B. The various units on the frame are interconnected with ribbon cable assemblies or similar wiring means.

The circuit packs use dual in-line package devices for most logic functions. Both 200-pin and 300-pin connecters are provided on the circuit packs. Some number of these pins is reserved for battery and ground potentials. Both double-sided-rigid and multilayer printed wiring boards are used. All packs are equipped with face plates and tabs for product-code markings. Circuit packs are mounted on one-inch centers in the apparatus mountings to allow adequate cooling.

The majority of the frames are equipped with their own dc/dc converter units for conversion of central office battery to logic potentials.

The various frames are interconnected with connectorized switchboard cables to form an SPC 1B.

### 6.2 Equipment

#### 6.2.1 3B20D Processor frames

The SPC 1B is composed of two PSI frames, two control unit frames, two peripheral control (PC) frames, four Moving Head Disk (MHD) frames, a tape unit (TU) frame, a power distributing frame, a Local Maintenance Position (LMP), a Recent Change/Verify Position (RCVP), and other craft-machine interface hardware. Figure 6 shows the frames for the processor and peripherals line-up except for the local maintenance position, which is illustrated in Fig. 7. A description of the physical design of the 3B20D hardware is given in Ref. 8; additional frames required for an SPC 1B are described below.

#### 6.2.2 Peripheral System Interface frame

The PSI is a single-bay frame, 2-ft 2-in. wide and 7-ft high, arranged to accommodate the PSI circuitry (see Fig. 8). The frame consists of seven component units. On two of these units up to 20 circuit packs that provide the logic functions and the power control functions can be mounted. Figure 9 shows one such logic unit for the PSI. A third unit at the top of the frame has mountings for transformers, inductors, connecters, and terminations that provide the means for interconnecting the PSI to the existing TSPS peripheral buses. On a fourth unit dc/dc converters can be mounted. The three remaining units are fuse panels. Two of these are for distributing +5V to the logic circuit packs. The remaining fuse panel distributes central office battery to the dc/dc converter input, bus controls, and maintenance functions. A frame filter unit in the frame base filters the central office battery. The logic

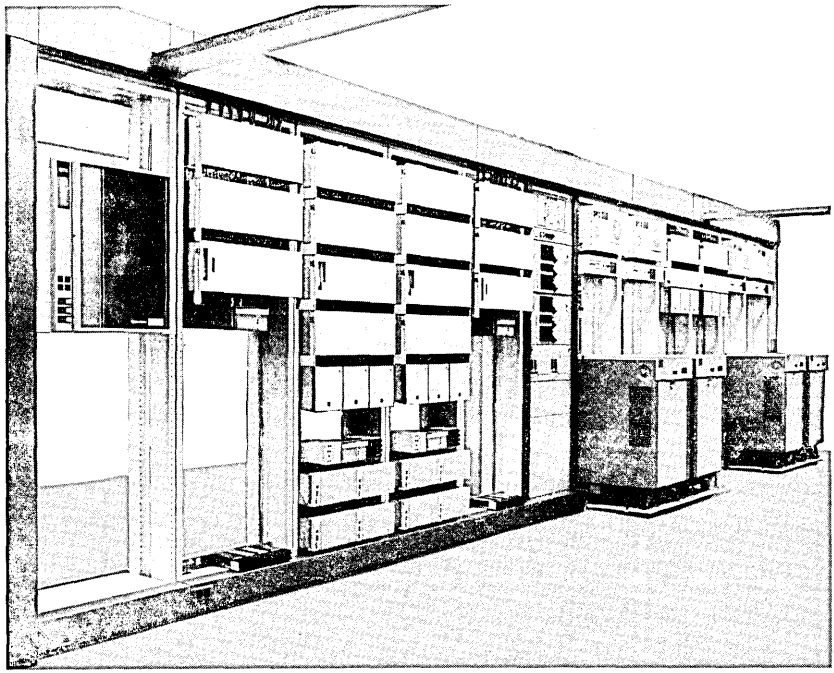


Fig. 6—SPC No. 1B Processor.

units consist of *BELLPAC* mounting hardware. The printed wiring backplane consists of battery and ground layers. Other connections between circuit pack locations are machine wrapped (see Fig. 10). All the circuit packs used in the PSI frame design are of the double-sided, rigid construction equipped with 200-pin connecters (see Fig. 11).

The PSI is duplicated in the SPC 1B. Each frame is associated with and mounted adjacent to a 3B20D Processor control unit frame (see Fig. 6). Connectorized switchboard cables interconnect the PSI frame to the CU frame, the Peripheral Control (PC) frame, and to existing TSPS frames.

#### *6.2.3 Maintenance Control Center*

In the TSPS No. 1B Maintenance Control Center, three bays accommodate the craft-machine interface and related data sets associated with maintenance and administration functions (see Fig. 7).

The Local Maintenance Position (LMP), in two bays, accommodates a shelf-mounted maintenance terminal, a receive-only printer, and data sets for the Switching Control System and the remote field update system[7] interfaces. The Recent Change/Verify position, in one bay, accommodates a shelf-mounted terminal, and data sets associated with

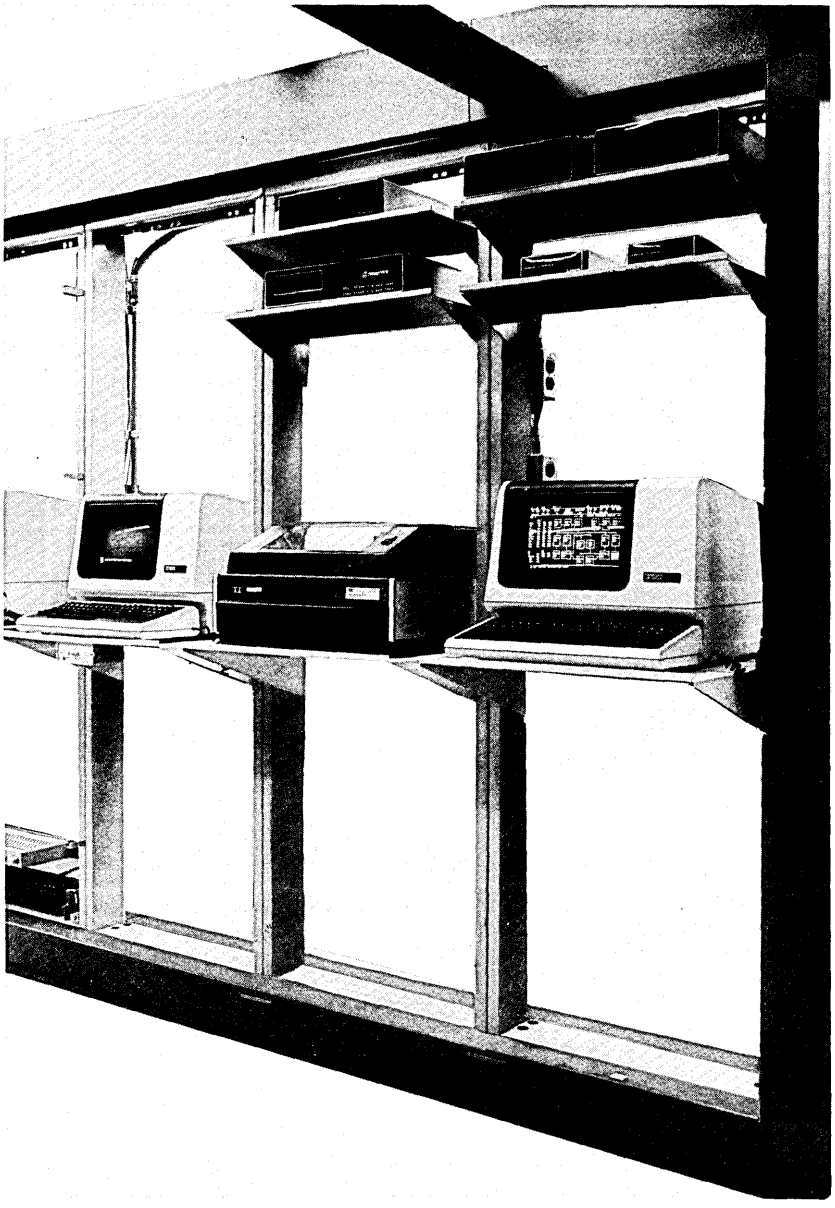


Fig. 7—Local maintenance position.

the remote-maintenance monitor interface and the optional base-unit belt-line maintenance TTY. It also contains a pedestal-mounted auxiliary printer. The LMP bays are mounted adjacent to the existing TSPS No. 1 Control Display and Test frame to facilitate maintenance.

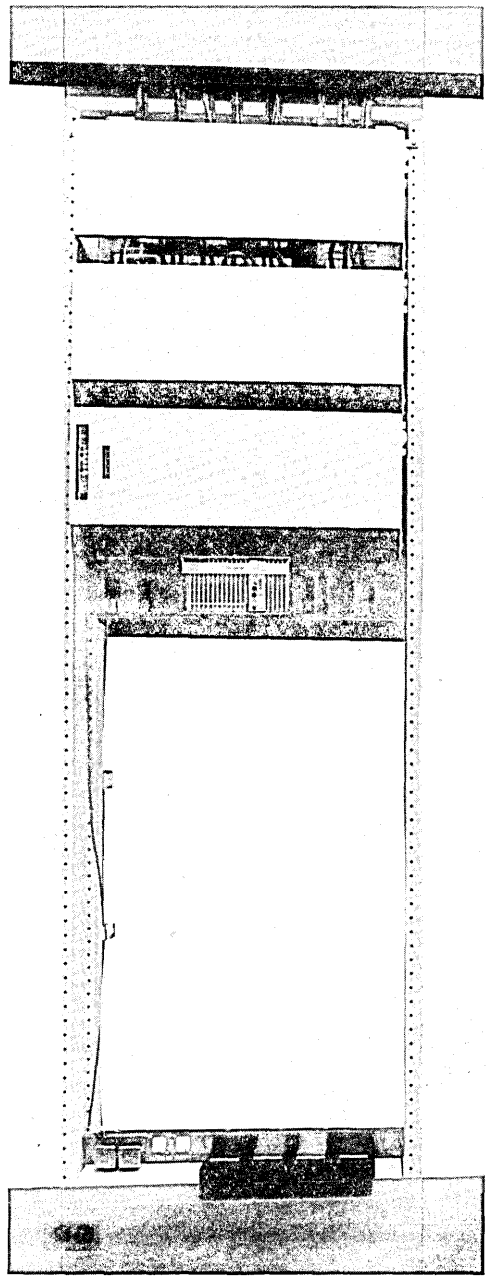


Fig. 8—Peripheral System Interface frame.

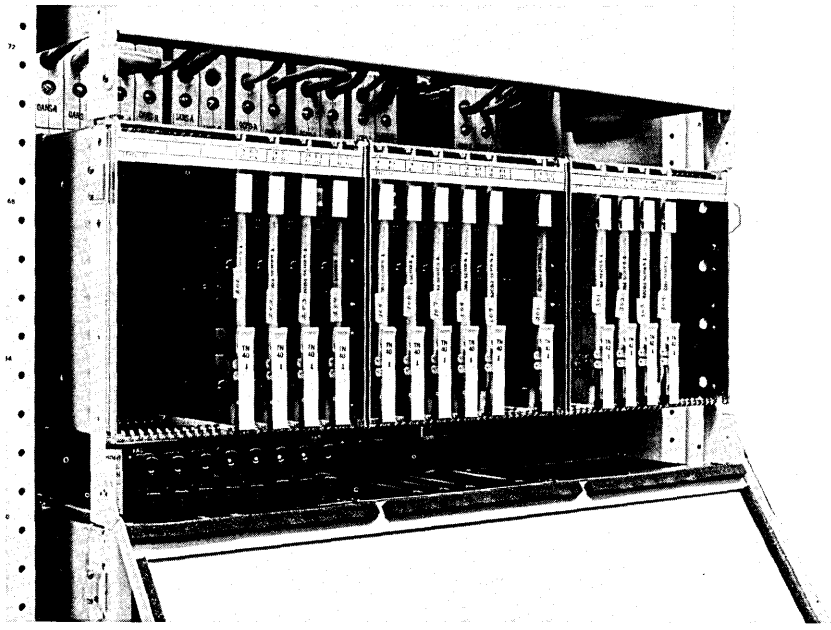


Fig. 9—Peripheral System Interface frame logic unit with circuit packs.

#### 6.2.4 Power

A specially designed power distributing frame distributes central office battery to the various 3B20D Processor frames using one or more pairs of connectorized 10-gauge or 12-gauge feeders. The MHD frame requires a larger pair (4-gauge) for the 30M-byte disk drive. The PSI frames obtain central office battery from the existing TSPS power distributing frames.

The MHD frame also requires 208V single-phase ac for the normal operation of the 300M-byte disk transports. Maintenance terminals, printers, and data sets obtain their protected or essential 110V ac from the existing TSPS ac power distributing unit.

#### 6.2.5 Bus interconnections

The TSPS peripheral buses are duplicated for reliability. Each PSI frame has access to each of the duplicated buses. Connectorized switchboard bus cables are provided between PSI frames and other TSPS frames. As the PSI frame now forms the terminus for these buses, plug-in-terminations are furnished as part of the PSI frame design.

#### 6.2.6 SPC 1B floor plan

The SPC 1B frames are arranged to satisfy the design constraints of the communication links between the individual frames and to comply

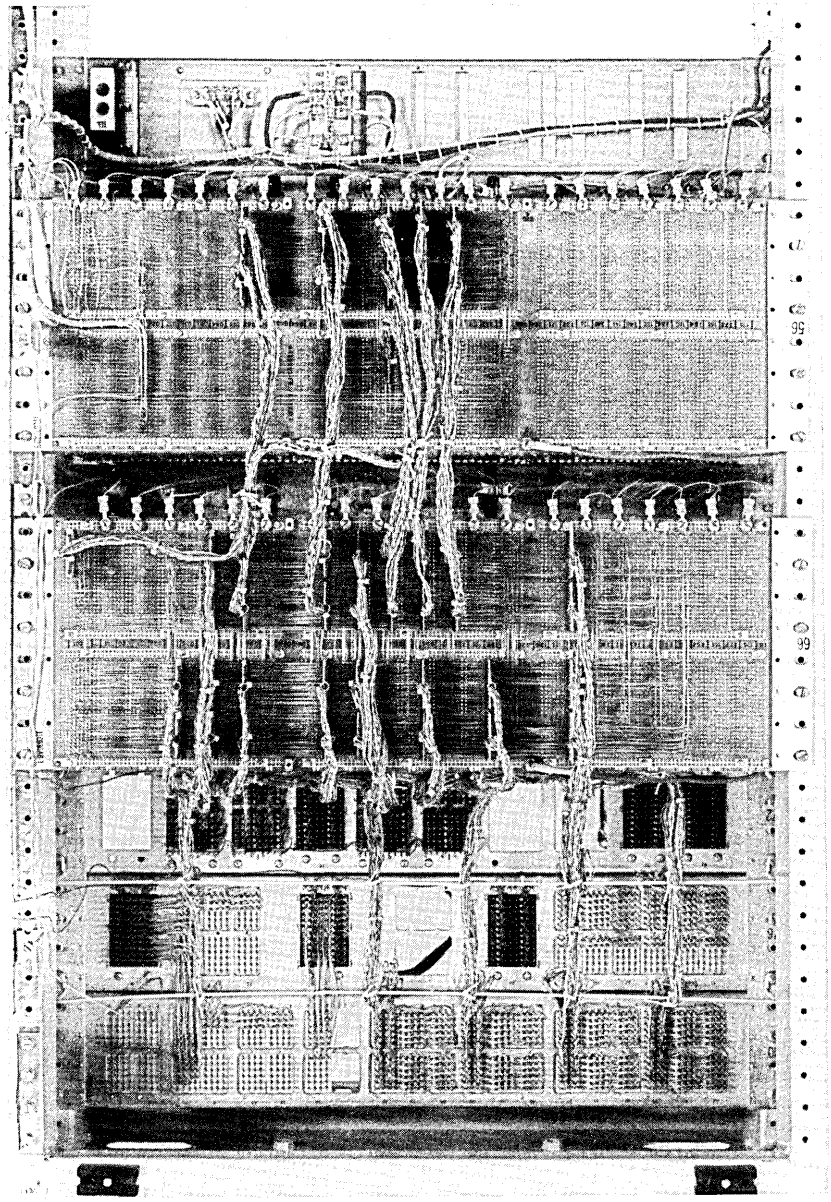


Fig. 10—Peripheral System Interface backplane.

with existing TSPS office equipment arrangements. Figure 12 is an overview of a typical SPC 1B floor plan arrangement designed to meet the constraints described below. The duplicated CU frames must be adjacent to each other and the interconnection from a CU to its

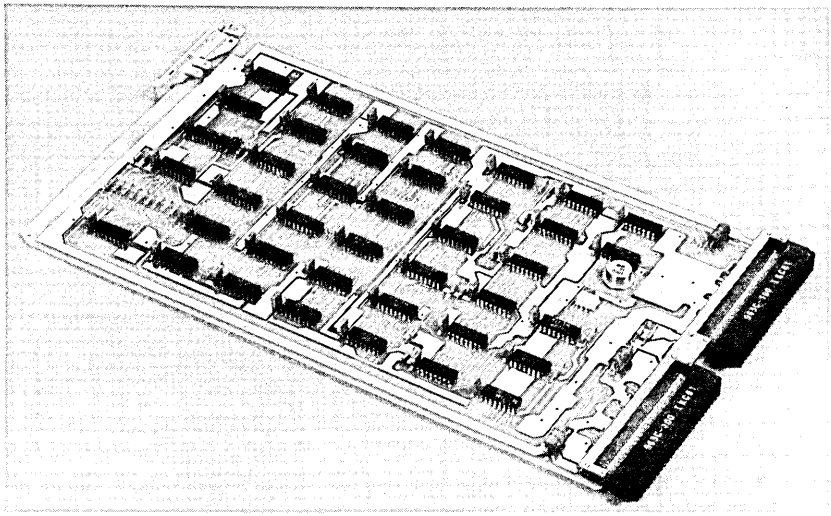


Fig. 11—Typical circuit pack for the Peripheral System Interface frame.

associated PSI frame must be as short as possible. Therefore, the PSIs are located on frames adjacent to the CUs. The MHD frames are located near their associated DFC units, which are mounted on adjacent PC frames. For maintenance of the MHDs, the aisle spacing is increased over the standard TSPS spacing. Location of other units such as the tape drive and IOPs is constrained by cable length between the CU and peripherals driven by the IOP. The 3B power distribution frame is centrally located to minimize length of power feeders to each of the other frames.

## VII. PSI MAINTENANCE SOFTWARE

In addition to the maintenance software provided with the 3B20D Processor and the emulated maintenance software carried over from TSPS No. 1, a maintenance software package was developed for TSPS No. 1B. The major functions of this package are:

(*i*) PSI maintenance

(*ii*) Sanity and integrity of application processes

(*iii*) Interface and coordination of integrity-related activities between DMERT and the TSPS application; i.e., initialization, overload control, and processor switch.

These functions have been implemented in the Application Integrity Monitor (AIM) process, a portion of the native-mode software in the TSPS process, and the PSI diagnostic processes. The following is a description of the PSI fault recovery and PSI diagnostics. Descriptions of items (*ii*) and (*iii*) are provided in Ref. 2.

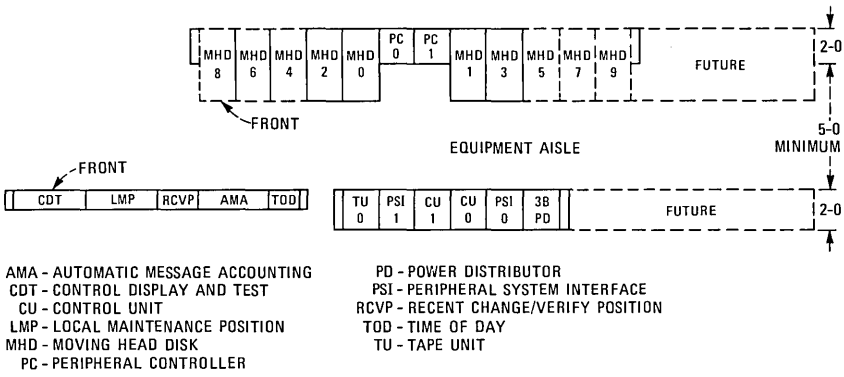



Fig. 12—Typical floor plan for TSPS No. 1B.

The PSI maintenance strategy is based on hardware duplication, fast detection and recovery from service-affecting hardware faults, and the use of deferred diagnostics for the identification of faulty circuit packs. The 3B20D CU and its associated PSI represent a basic switchable entity. This architecture makes the PSI an integral part of the processor. As a result, its maintenance strategy is tightly coupled with the maintenance of the 3B20D Processor.

### *7.1 PSI software data structure integration*

All information about the SPC 1B hardware configuration, status, error counts, and other pertinent data is contained in the Equipment Configuration Database (ECD). The ECD contains a unit control block (UCB) for every equipment entity known or controlled by DMERT. These UCBs are linked data structures that form a hierarchical tree structure representing the physical hardware configuration and the unit interdependencies.[9] This structure links the PSI to the 3B20D hardware architecture and links the PSI to the DMERT maintenance software control structure. The PSI UCB defines the PSI to be a critical CU unit, thus binding the PSI and CU as a single switchable entity. Besides describing the configuration, the UCB also contains error counters and corresponding threshold values for fault recovery, equipment information for service status, and hardware options for diagnostics.

### *7.2 Description of PSI fault detection and recovery*

Detection of PSI faults relies on the 3B20D and PSI self-checking circuits, the TSPS microcode, and the TSPS peripheral-check circuits. Once the fault has been detected, three different fault-recovery programs may be triggered to restore a working system by means of fault resolution and reconfiguration. The fault-recovery program activated

by a fault depends on where the fault is detected. Recovery programs activated may include the TSPS AIM process, the emulated peripheral maintenance program (F-level), or the DMERT Processor Control Process Error-Interrupt Handler (PCPEIH). The following paragraphs will describe the various fault-detection and recovery strategies.

#### *7.2.1 Fault detection*

Most PSI faults are detected by the PSI internal-checking circuits. These errors activate an error-interrupt lead to the 3B20D Processor, which is connected to a processor interrupt source register (ISR) bit. The fault-recovery program triggered by this fault depends on what process is active at the time of the error. A fault that occurs during execution of an emulated PSI peripheral order is detected by the TSPS microcode. The microcode generates a maintenance interrupt (F-level) to the TSPS process. The TSPS F-level program, after establishing whether the fault is transient or hard (e.g., after retry of the failing order), informs AIM of the failure by sending a fault event to the AIM process with a fault code indicating whether the fault is transient or hard.

Faults that occur during execution of native-mode PSI instructions and faults that are detected by the 3B/PSI clock-check circuits, cause an ISR bit to be set and result in the dispatching of AIM at the PSI Error-Interrupt Handler (PSIEIH) entry. For these fault types PSIEIH may interrupt and fault the running process to inform it of the failure of the PSI I/O instruction. The decision to fault the running process is made on the basis of its identity. If the running process is TSPS or the PSI Diagnostic Driver (PSIDGDR), it will be faulted. Since no other processes perform I/O with the PSI, there is no need to fault them. Therefore, they will be allowed to continue execution from the point of interrupt after completion of processing by PSIEIH.

Another class of faults are those that are detected by the PSI peripheral order-checking circuits. In most cases this type of fault is a TSPS peripheral-unit fault, but some may be faults in a PSI driver or receiver that are indistinguishable from faults in the TSPS peripheral unit. For these faults, a retry of the failing order through the standby 3B20D CU/PSI is performed. If the off-line retry succeeds, TSPS informs AIM of a PSI fault through a fault code. If the retry fails, TSPS recovers from the peripheral-unit hardware fault by switching TSPS peripheral units.

Finally, certain types of faults in the PSI or the ACHI will result in parity errors on the processor CCIO bus. This type of error appears as a channel error to the CU by activating an ISR. The ISR activates the DMERT fault-recovery process PCPEIH.[10] PCPEIH will then fault the interrupted process. When such a fault occurs while the TSPS

process or the PSIDGDR process is running, it will send a fault to AIM. AIM will reinitialize the PSI and ACHI, and make a reconfiguration decision.

#### 7.2.2 Fault recovery

Once the fault has been detected and resolved to the PSI, it is the responsibility of the AIM process to perform the required recovery action. If the PSI fault is a transient fault, and the PSI/ACHI reinitialization completes successfully, AIM calls DMERT configuration manager functions, which cause error counters in the PSI UCB to be incremented and compared with threshold values. If either counter exceeds its threshold value, removal of the faulty PSI is requested by means of a processor switch. In the case of a hard error, or if the PSI/ACHI reinitialization fails, no error counting is done, and a processor switch is requested immediately. The processor switch is performed by a DMERT service routine.

#### 7.2.3 Off-line PSI fault-detection and recovery

AIM also periodically exercises the off-line PSI. If the exercise fails, the unit is placed out-of-service and diagnostics are scheduled.

### 7.3 PSI initialization

The PSI must be initialized following a PSI power up, as part of a processor switch, and for certain levels of software initialization in TSPS and DMERT. Initialization of the PSI includes the selection of the Scanner Answer Bus, clearing of the PSI maintenance register, and clearing of the Application Channel Interface (ACHI).

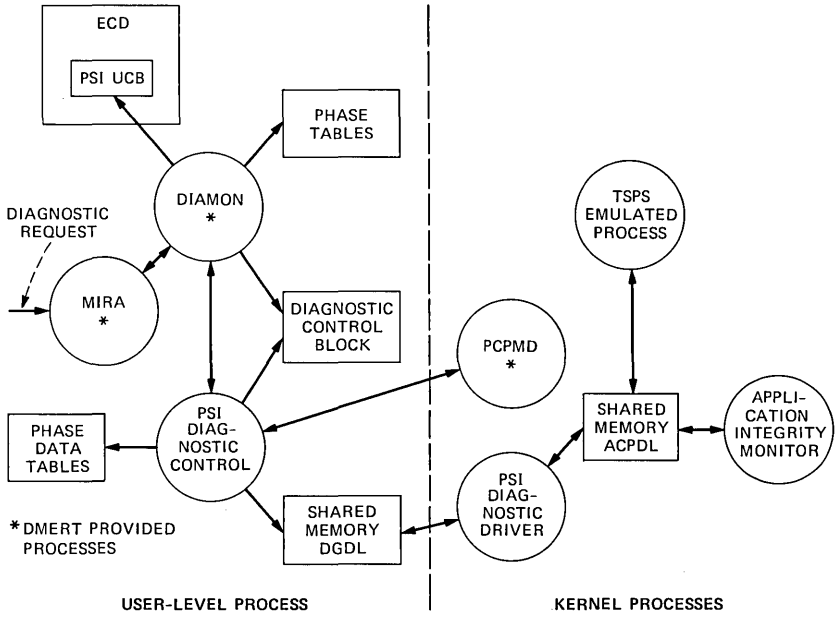
### 7.4 PSI diagnostic

Diagnostics for the 3B20D CU and the TSPS microcode are included in the common DMERT maintenance package. The SPC 1B maintenance package contains the PSI diagnostic.

#### 7.4.1 Operation under DMERT diagnostic control structure

Since the PSI is tightly coupled to the 3B20D Processor and forms a single switchable entity with a 3B20D CU, the PSI diagnostic is designed to operate under the DMERT diagnostic control structure. Diagnostic results are handled by an output message process called the "spooler." A simplified block diagram of this overall software structure is shown in Fig. 13. The PSI diagnostic is initiated and executed under control of the maintenance input request administrator (MIRA) and the diagnostic monitor (DIAMON) process.[11]

Elements of the PSI diagnostic which are unique to the PSI are the Diagnostic Control Process (PSIDC), a Diagnostic Phase Table




ACPDL - APPLICATION CONTROL PROCESS DATA LIBRARY
DGDL - DIAGNOSTIC DATA LIBRARY
DIAMON - DIAGNOSTIC MONITOR
DMERT - DUPLEX MULTI-ENVIRONMENT REAL-TIME
ECD - EQUIPMENT CONFIGURATION DATABASE
MIRA - MAINTENANCE INPUT REQUEST ADMINISTRATOR
PCPMD - PROCESSOR CONTROL PROCESS MAINTENANCE DRIVER
PSI - PERIPHERAL SYSTEM INTERFACE
TSPS - TRAFFIC SERVICE POSITION SYSTEM
UCB - UNIT CONTROL BLOCK

Fig. 13—Maintenance software structure for the Peripheral System Interface.

(DPT), phase data tables (one per phase), task routines, and off-line test routines. Although unique for the PSI application, the implementation of the above diagnostic structure is similar to other 3B20D control unit diagnostics, and a detailed description of the architecture can be found in Ref. 11. Figure 13 provides a high level description of the diagnostic control structure.

### *7.4.2 Off-line execution*

Since the PSI and 3B20D Processor form a single switchable entity, the active processor does not have direct access to the off-line PSI. Therefore, most of the PSI must be diagnosed by test routines which run in the off-line 3B20D Processor.

The off-line tests are divided into two categories of functional tests. Internal PSI functional tests are those which exercise the PSI register logic and sequencer circuits. These tests do not require access to TSPS peripherals and are executed with the I/O inhibit lead activated. The

I/O inhibit lead disables all outpulsing from the PSI to TSPS peripherals. External PSI tests are those that diagnose the drivers and receiver circuits that interface directly with the TSPS periphery. All off-line tests are loaded and executed under the control of the PSI Diagnostic Control (PSIDIAGC) Process with the aid of DMERT library routines that handle communication with the off-line processor. These library routines are provided by the Processor Control Process Maintenance Driver (PCPMD) as described in Refs. 10 and 11.

#### *7.4.3 On-line execution*

Some diagnostic routines must also run on the on-line 3B20D Processor. These routines include the PSI power control tests, the TSPS peripheral bus power tests, and the on-line retries.

The PSI +5V power and the TSPS peripheral bus power are controlled and monitored by a 3B20D common system Scanner and Signal Distributor (SC/SD). The off-line 3B20D Processor cannot communicate with the SC/SD, therefore the on-line processor must perform these tests. The peripheral bus (CPDB and PUAB) power monitoring and control are also connected to the SC/SD. Diagnostic tests of these controls are performed from the on-line processor.

On-line retries of failing off-line tests are performed when it becomes necessary to isolate a fault between the PSI and a TSPS peripheral unit. This situation occurs in the case of a PSI bus driver fault or a fault in a corresponding receiver of a peripheral unit.

#### *7.4.4 PSI diagnostic tests*

The PSI diagnostic test phases are designed to test the hardware in a layered approach such that initial tests start with data bus integrity between the 3B20D and PSI, simple PSI and 3B20D handshaking tests, and then progressing deeper into the PSI hardware until all peripheral driver and receivers have been tested. The test phases of the PSI are divided into internal logic circuit test phases and peripheral driver and receiver test phases. In the following sections the diagnostic test phases are described to show how the layered diagnostic design was implemented.

***7.4.4.1 Internal PSI diagnostic phases.*** The internal diagnostic phases are responsible for detecting faults in the PSI sequencer, registers, and check circuits. These tests are executed with the peripheral bus drivers and receivers disabled to avoid interference with TSPS peripherals. The internal phases can be divided into three sets of tests. The first set of tests relies on maintenance control circuitry to test the basic data paths and ACHI/PSI communication protocols. This phase of testing requires minimal sequencer activity by the PSI, thus providing a high degree of circuit pack resolution by limiting the number of

circuits under test. The second set of internal tests are the tests of the circuit operational functions. All PSI commands are executed to verify proper operation. During these tests error checks are ignored and the commands are tested for proper operation by checking data results. Finally, the remaining test phases of the internal tests are designed to test all internal error detection and matching circuits. Completion of the internal tests provides coverage over most of the PSI logic circuits.

*7.4.4.2 External PSI diagnostic phases.* After the PSI internal circuitry has been tested, the last layer of circuitry, the peripheral bus transmitters and receivers, must be tested.

During these tests the PSI communicates with TSPS peripherals in order to verify operation of drivers and receivers. Therefore, the PSI diagnostic must have access to the equipage and status tables of the TSPS peripherals to check on availability of particular peripheral units. The PSI diagnostic control process provides this access by sharing the memory space of the TSPS emulated process to directly access the office data and status tables for the peripheral units.

Another capability required of the external bus testing is the ability to resolve faults to the PSI or to the peripheral unit used in performing the test. This capability is provided by a special-purpose, on-line diagnostic driver called the PSI Diagnostic Driver (PSIDGDR), which executes the same tests on the on-line processor that were executed in the off-line processor. If the off-line results of the PSI indicated some-test-failed (STF), the on-line test is executed to determine if the fault truly implicated the off-line PSI. The PSI diagnostic control then matches the off-line and on-line results. If the results match, the TSPS peripheral unit is at fault; if the results do not match, the off-line PSI is assumed to be faulty.

### *7.4.5 Isolation from TSPS process*

The structure of the TSPS No. 1B software architecture creates a time-shared environment in which diagnostic execution is performed in time segments that are multiplexed by the DMERT operating system with the TSPS kernel process and other *UNIX** processes. This creates a potential for interference between segments and also creates the possibility of simultaneous execution of an off-line diagnostic routine and on-line TSPS process peripheral orders. Potential interference in these areas requires that the diagnostic control be designed to eliminate interaction. Therefore, a number of capabilities have been developed to provide isolation between the PSI diagnostic and the TSPS emulated process.

---

* Trademark of Bell Laboratories.

*7.4.5.1 I/O inhibit control.* The 3B20D Processor provides an input/output inhibit control lead on the CCIO bus. This lead is active on the off-line processor and is used by CU I/O channels to block transmission of control signals to the peripheral units. In the case of the ACHI/PSI, the I/O inhibit lead is used to inhibit outpulsing by the PSI drivers. The purpose of the lead is to protect the TSPS peripherals from a faulty PSI sequencer and to allow the diagnostic to execute peripheral orders in the PSI without interfering with the TSPS peripherals.

*7.4.5.2 PSI diagnostic-TSPS process synchronization.* The external PSI tests must utilize the TSPS periphery to test the PSI interface drivers and receivers. Therefore, to ensure that the diagnostic peripheral tests do not interfere with on-line TSPS activity, the PSI diagnostic will be synchronized with the TSPS process. No off-line test routines run while the TSPS emulated process is running. This capability is provided by DMERT library routines, which execute programs in the off-line processor. The library routine checks the TSPS J-level timer (which is generated by the 3B20D clock) and will only initiate execution in the off-line processor when a minimum of 2.5 ms remains prior to the next J-level interrupt. In addition, the routine will retain control of the 3B20D Processor and inhibit all I/O interrupts, timed interrupts, and DMA activity. The 3B20D Processor will not return to TSPS base level during this interval.

## VIII. SUMMARY

A special hardware unit, the Peripheral System Interface, was developed to interface the existing TSPS periphery with the 3B20D Processor. The addition of the PSI required a corresponding development of new maintenance software to integrate the 3B20D maintenance strategy with the TSPS No. 1B maintenance strategy. This article has described the constraints imposed by the differences in technology, and the hardware and corresponding maintenance software required to integrate the 3B20D processor into the TSPS system.

## IX. ACKNOWLEDGMENTS

Many individuals have contributed in a significant way to the design and implementation of the PSI and its maintenance software. In particular, the authors wish to acknowledge the contributions of D. A. Peterson and D. J. Kloc for their part in developing the PSI hardware, and W. J. Proetta for his contributions to the PSI physical design. The following people contributed to the development of the PSI maintenance software: D. L. Brown, K. R. Kulhanek, and J. J. Labut for the PSI diagnostic, E. S. Sachs and M. D. Soneru for the PSI maintenance software.

## REFERENCES


1. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
2. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.
3. J. C. Dalby, D. Van Haften, and L. A. Weber, "Traffic Service Position System No. 1B: Retrofitting to TSPS No. 1B," B.S.T.J., this issue.
4. G. R. Durney, et al, "Stored Program Control No. 1A," B.S.T.J., *49,* No. 10 (December 1970), pp. 2445–2507.
5. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 191–206.
6. J. J. Bodnar, J. R. Daino, and K. A. VanderMeulen, "Traffic Service Position System No. 1B: Switching Control Center System Interface," B.S.T.J., this issue.
7. T. G. Hack, T. Huang, and L. C. Stecher, "Traffic Service Position System No. 1B: Software Development System," B.S.T.J., this issue.
8. S. H. Kulpa, J. M. Brown, and A. W. Fulton, "The 3B20D Processor & DMERT Operating System: 3B20D Packaging and Technology," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 221–34.
9. R. H. Yacobellis, J. H. Miller, B. G. Niedfeldt, and S. S. Weber, "The 3B20D Processor & DMERT Operating System: Field Administration Subsystems" B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 323–39.
10. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "The 3B20D Processor & DMERT Operating System: Fault Detection and Recovery," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 349–65.
11. J. L. Quinn, R. L. Engram and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostic Tests and Control Software," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 367–81.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Software Development System

By T. G. HACK, T. HUANG, and L. C. STECHER

(Manuscript received June 30, 1982)

*This article describes the Software Development System for the Traffic Service Position System No. 1B (TSPS No. 1B). It discusses the modern, multicomputer software generation and test facilities that were provided to concurrently support both C-language and emulated, assembly-level software development. The computing environment, software generation and test tools, and standard development process that were developed for the TSPS No. 1B provide a rich, robust programming environment for future network operator services.*

## I. INTRODUCTION

The development and testing of the software for the Traffic Service Position System No. 1B (TSPS No. 1B) was a complex undertaking. In addition to emulating the existing TSPS No. 1 assembly-level program, 3B20 Duplex Processor (3B20D) native-mode (C-language) software was developed to interface to the Duplex Multi-Environment Real Time (DMERT) operating system and to provide the necessary system integrity functions for the TSPS No. 1B.[1,2]

To support software development and testing in this mixed emulation/native mode, it was recognized early in the project that a robust Software Development System (SDS) and a rigorous set of standard software development procedures (or methodology) were necessary. Thus, basic requirements for tools, documentation standards, and control procedures were established for the software development environment for the TSPS No. 1B. These requirements specified that:

(*i*) A simple, interactive user interface to the SDS must be developed. Where possible, commands and procedures for emulation or

native-mode software development should be the same to minimize the complexity of the programming task.

*(ii)* The SDS for emulation-mode software development must be upwardly compatible from the existing TSPS No. 1 SDS. In parallel with the development of TSPS No. 1B, new generic features were being developed and deployed for TSPS No. 1 (such as Automated Calling Card Service).[3] To offer universal service, these features would also have to be concurrently emulated on the TSPS No. 1B.

*(iii)* Complete software change procedures and tools must be established to support the parallel development of TSPS No. 1 and TSPS No. 1B. These procedures and tools would have to support the early development phase of the project when programmers were initially developing code, as well as later phases of the project when system testing is converging to a certified, production software release.

*(iv)* Finally, a set of test facilities was needed to support the integration and system testing of the software. Special tools would be required to test the emulated, assembly programs, as well as the high-level, native-mode software executing under the DMERT operating system.

The following sections of this article describe how the TSPS No. 1B Software Development System was implemented to meet these requirements.

## II. COMPUTING ENVIRONMENT FOR TSPS SOFTWARE DEVELOPMENT

Before discussing the specific software generation tools and test facilities of the TSPS No. 1B SDS, this section describes the computing environment for TSPS development and gives an overview of the development system.

### *2.1 Overview*

TSPS No. 1B software development is supported by a multiprocessor computing system. The system consists of five computers: one remote IBM 3033 processor located at Bell Laboratories in Columbus, Ohio; and four Digital Equipment Corporation (DEC) PDP-11/70 minicomputers co-located with the TSPS development organization at Bell Laboratories in Naperville, Illinois. A variety of data links support interprocessor communications. Figure 1* illustrates the configuration.

The IBM processor and the two PDP-11/70 computers (labeled PSS1 and PSS2) constitute the Programmer Support System (PSS). It is on these systems that source modification, load building, and

* Acronyms and abbreviations used in the figures and text are defined at the back of this Journal.

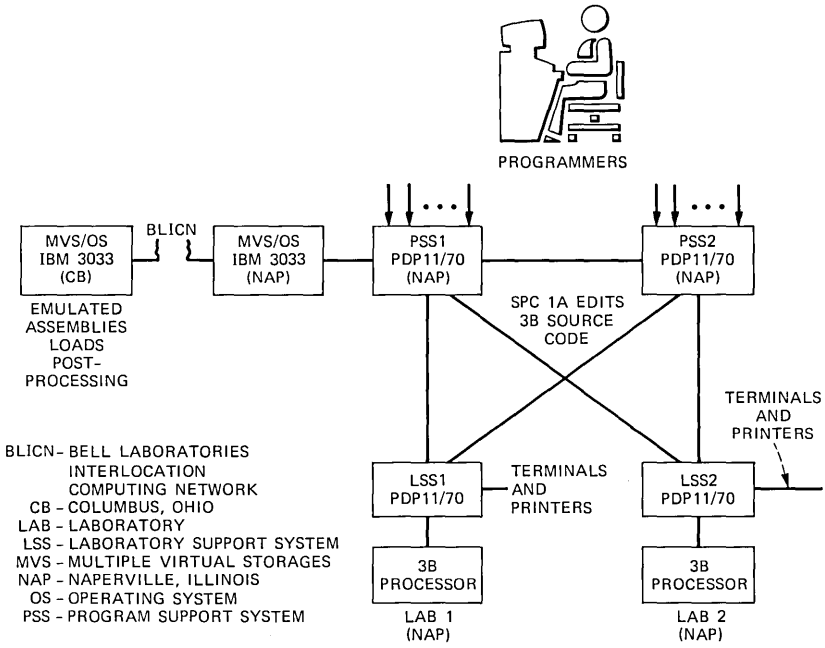



Fig. 1—Software Development System for TSPS No. 1B.

software administration are done. The remaining two PDP-11/70 computers reside in the TSPS system laboratories and are part of the Laboratory Support System (LSS). Each is paired with a 3B20D Processor and is used to support developer, integration, and system testing for 3B20D-based generic programs.

### *2.2 Local interactive development environment*

TSPS programmers use the PSS1 and PSS2 computers to modify source files and create executable versions of software. These computers run the *UNIX** time-sharing operating system, which provides a general-purpose, multi-user, interactive development environment.[4] The *UNIX* operating system has a number of characteristics that make it attractive for software development. First, it is extremely easy to learn and use. The command-line interpreter (the "shell"), the interactive editor, and the hierarchical file system are particularly simple in nature. Second, the shell and C-language programming environment provide powerful and flexible facilities to create and combine software tools to support the generation and administration of software. Many of the tools used in TSPS have been created in this

* Trademark of Bell Laboratories.

environment. Finally, the Programmer's Workbench facility,[5] available with the *UNIX* operating system, offers:

(*i*) A sophisticated document preparation system to prepare documentation supporting the development process

(*ii*) A Remote Job Entry (RJE) subsystem that transmits jobs to other computing systems and returns output to appropriate users

(*iii*) A complete Source Code Control System (SCCS) for controlling and maintaining multiple versions of source code.

The software development is partitioned so that C-language developers use the PSS1 system and emulated, assembly-language developers use the PSS2 system.* System load building makes use of both PSS1 and PSS2. C-language development is solely supported on the PSS1 system. Emulated development, on the other hand, is accomplished through the interaction of the PSS2 system and the remote IBM 3033 processor.

### *2.3 Remote batch environment for emulated development*

TSPS emulated software is developed in a part interactive, part batch-oriented computing environment. Much of the SDS for emulated code has been upgraded and carried over from the previously existing batch-oriented SDS for the TSPS No. 1.[6] The PSS2 system was added to the previously existing SDS for the TSPS No. 1 and provides the front-end user interface to the SDS for emulated programs. When the PSS2 system was added, the programmers gained a modern set of interactive commands that allow them to initiate the various software generation jobs for emulated code. Developers log into PSS2 and modify "edit files" that contain edit statements to be applied against the emulated source files, which are stored on the remote IBM system. The application of edits and all software-generation steps run under the Multiple Virtual Storages (MVS) operating system in a batch mode on the IBM processor.

For example, once a programmer has finished modifying the edit file on the PSS2 system, a single swap command can be issued to construct a Job Control Language (JCL) script designed to invoke, through the *UNIX* RJE facility, a remote emulated assembly on the IBM processor. The job file is sent via the Bell Laboratories Interlocation Computing Network and queued to be executed on the IBM/MVS system at Columbus. When the assembly is complete, the listing file is printed locally. Commands similar to swap are provided for link editing and other functions. These commands are described in more detail in Section III.

---

* Subsequent references to "emulated, assembly" language will be shortened to "emulated" language.

From the programmer's point of view, all activity takes place on the PSS2 system. The commands that create the remote batch jobs look very much like local commands. Since required output data (tapes, files, or listings) are returned to the local users, the use of the IBM/MVS system is transparent to the user.

### 2.4 Local interactive environment for C-Language development

The PSS1 system provides the computing environment for C-language program development. It can support as many as 36 simultaneous interactive users, although prime-time usage is typically about 25 users. For the most part, terminals are connected through a dial-up arrangement. There are, however, a few "hard-wired" terminal connections into the TSPS System Laboratories to guarantee access to programmers using laboratory test facilities.

Software tools that support C-language development on the PSS1 system include: the text editor, the 3B20D Software Generation System (3BSGS), the Change Management System (CMS), and the Source Code Control System. The text editor is used to modify C-language source files. The 3BSGS is a cross-compilation and link-editing system that generates executable files for the 3B20D Processor from source code. CMS and SCCS control and track the development process and provide the mechanisms to maintain multiple versions of source and object files.

From the programmer's perspective, the development scenario is quite simple. A source file is retrieved from SCCS, modified with the editor, compiled to produce an executable file using the 3BSGS, and data-linked to the Laboratory Support System computers for testing in the TSPS System Laboratories. With the exception of testing, all work takes place interactively on the PSS1 system.

### 2.5 Laboratory support processor

The support processor used in the TSPS Laboratory Support System (LSS) is a PDP-11/70 computer running under the *UNIX* operating system. The flexible environment of the *UNIX* operating system can support multiple testers simultaneously. For TSPS development, there are two system laboratories that are used to create the laboratory environment for field support and to enable realistic system testing of TSPS software. Section V describes the LSS in more detail.

### 2.6 Networking facilities

As we can see in Fig. 1, the processors within the TSPS computing environment are interconnected by a variety of data links that create a reliable, secure computer network. Each PSS computer is connected by two separate links to the other PSS computer and the two LSS

computers. One of the connections is a 9.6-kb link designed to be used with the cu command. (The cu command gives a user on one system the appearance of being logged into another.) Commands can be executed on the remote system and files can be transferred in both directions. The second type of connection is a 56-kb link that uses DEC DMC-11 hardware as the primary intercomputer file transfer mechanism. It can support effective file transfer rates of up to 2000 bytes/second. The remaining data link is a component of the Bell Laboratories Interlocation Computing Network connecting the Naperville and Columbus locations. The primary connection is established through 56-kb private lines between two IBM 3033 processors running the MVS operating system, the remote IBM system at Columbus, and a local interface system. The two TSPS PSS computers are connected to the local IBM/MVS system with special-purpose 9.6-kb hardware. Using the RJE subsystem, batch jobs can be sent through the local IBM/MVS system to the IBM/MVS system at Columbus. The capability also exists to return files to the PSS system or generate local output tapes or listings.

## III. SOFTWARE GENERATION TOOLS

### *3.1 Overview*

The primary function of a software generation system is to transform symbolic source language statements into a format executable by the target processor. For TSPS No. 1B, the target processor is the 3B20D Processor. A software generation system is usually a collection of tools that perform this transformation in phases. The compilation (or assembly) phase translates the source module containing symbolic instruction and data statements into machine-readable form, called a "relocatable object file." This file also contains symbol definitions and relocation information to be used in the linking-loading phase. The linking-loading phase enables separately compiled modules (relocatable object files) to be combined into a single executable unit, called a "load file." A final process-loading phase is used to transform the load file to a process-file format.

### *3.2 Emulated software generation*

During the development of the TSPS No. 1B, special-purpose microcode was written for the 3B20D processor to emulate the Stored Program Control 1A (SPC 1A) assembly language used in TSPS No. 1. This allowed a large portion of the existing TSPS software to be transported to the 3B20D Processor without modification. In this approach, the emulated source statements were kept identical to the SPC 1A assembly-language statements. However, the object form (the machine-equivalent form) was modified to take advantage of the

powerful micro-instruction set of the 3B20D. This was done to maximize the real-time processing capability of the TSPS No. 1B system.

To ensure commonality between TSPS No. 1 and TSPS No. 1B assembly-level programs, the existing SPC 1A assembler and loader were retained. The new 3B20D object format for the TSPS No. 1B was produced by two new software generation tools—a "load post-processor" and a "listing post-processor." The load post-processor transforms an SPC 1A load file into a 3B20D load file, and the listing post-processor reformats the assembly listing to reflect the new, emulated instruction formats. In addition, many of the existing SPC 1A SDS tools were retained and significantly upgraded by the addition of the PSS2 system. Figure 2 shows the steps involved in emulated software generation.

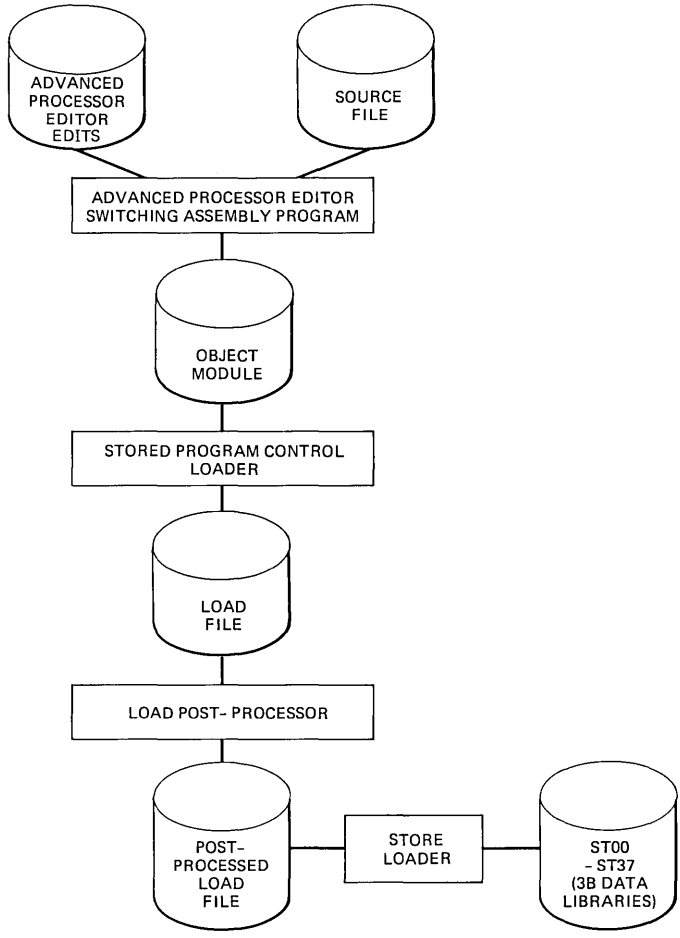



Fig. 2—Basic steps in emulated, assembly-language software generation.

### 3.2.1 The TSPS assembler and loader

The assembly process for TSPS emulated code combines both an editing and an assembly operation. Editor statements are prepared on the PSS2 system and transmitted to the remote IBM/MVS system. These editor statements are then processed and applied to the emulated source file by the Advanced Processor Editor (APE) running under MVS. APE is a simple, line-oriented editor with the basic "insert," "replace," and "delete" functions. It is specifically designed to work with the assembler, passing the edited source as input to the assembly process.

The assembly operation is performed by the powerful SPC-SWAP (Switching Assembly Program) assembler,[7,8] which also runs under MVS. Besides performing the standard source-to-object conversion, it has a sophisticated macro capability and a variety of useful pseudo-operations for controlling listing format and establishing symbol definitions. In addition, there is a mechanism for creating and maintaining special-purpose "library" files. Library files of symbol or macro definitions can be created in one SPC-SWAP run, then later accessed by subsequent source module assemblies for the purpose of symbol or macro resolution. This is not only a convenient mechanism for sharing global symbol definitions between source modules, but also allows a single source file to be assembled in different environments, resulting in different object modules. The latter technique is used in TSPS to develop code for multiple generics from a single source file and is discussed further in Section 4.1.

To perform an assembly, TSPS developers invoke the swap command on the PSS2 system. This command creates a Job Control Language script, which is executed on the remote IBM/MVS system. The primary outputs of SPC-SWAP are an object module, which is retained on the IBM file system, and an assembly listing, which is printed locally.

After the assembly process, the next step in producing an executable file for the 3B20D Processor is to combine the SPC-SWAP object modules for a given generic using the SPC loader. This loader uses a special set of control statements that specify what areas of emulated address space are available for loading object modules, which modules are to be loaded, and, optionally, what their load addresses are. The output of the SPC loader is a file or magnetic tape containing the relocated, fully bound, generic load file. At this point, however, the load file is suitable only for loading into an SPC 1A used in TSPS No. 1. Further post-processing (described in Section 3.2.2) is then done to produce a 3B20D-compatible format.

In addition to a full generic load, the SPC loader also has the capability to produce what is called a "partial-load" file. During the

generation of a full generic load, the SPC loader is directed to create a special file on the IBM/MVS system called a HISTORY. This HISTORY file contains the information necessary to completely characterize the generated load file. This includes the load image itself; the names, starting addresses, and sizes of the object modules; free space; and all external symbol definitions and references. Once a HISTORY file has been created, any TSPS developer on the PSS2 system can reassemble a selected subset of generic source modules and issue the ppload command to build a partial load on the IBM/MVS system. The items input to the ppload command identify the HISTORY file and the reassembled object modules. The partial-load process constructs a new load image from these input files, then compares it to the image contained in the HISTORY file. The final output is a magnetic tape containing the changed instruction or data words. This tape, after post-processing, can be taken into the TSPS system laboratory and overlaid on the full generic load file. The partial-load capability is used extensively in the early stages of TSPS generic development. It is a very flexible and convenient means of creating a developer's private software version for testing.

#### *3.2.2 Post-processing— format conversion from SPC 1A to 3B20D format*

To convert from SPC 1A format to 3B20D format, a load post-processor was developed to transform the output of the existing SPC loader into the new 3B20D encoding. In addition, a listing post-processor was developed to convert program listings to account for the changes in the order encoding and also to provide address and data fields in hexadecimal, which is the native number base for the 3B20D.

The load post-processor accepts as input a HISTORY file produced by the SPC loader. The HISTORY file contains not only the actual load image but also specifies whether an SPC word corresponds to a program or data instruction. Each 40-bit (double-word) SPC program instruction is translated into a 64-bit, 3B20D encoding (two 32-bits words) as required for the emulation.[2] The reformatting that is done depends on the SPC operation code and can completely rearrange the fields in an instruction. The translation algorithm is encoded into a set of common routines that are shared by the load and listing post-processor. This ensures that listing and load translations remain synchronized. Both the load and listing post-processors are run as batch jobs on the IBM/MVS system and are initiated remotely from the PSS2 system.

#### *3.2.3 Creating a DMERT process from an emulated load file*

The TSPS process is a key component of the TSPS No. 1B system since it has primary responsibility for call-processing functions.[2] It is

a DMERT kernel process made up of both C-language and emulated software. The C-language portion is built in the standard fashion using the 3B Software Generation System (see Section 3.3.1). The emulated portion is stored in DMERT data libraries and appears as pure data to the C-language software generation process. The data libraries are stored as separate files on the 3B20D disk and are linked into the TSPS process address space at process creation time. There are 32 data libraries for emulated software, each being one segment (128K bytes) in size.

#### *3.2.4 Overwrite generation*

In the later stages of the development of a generic, rigorous change procedures are introduced to tighten control over changes to the software. This is known as the "frozen" mode of development and begins normally with the start of system testing.[9] In frozen mode, changes in emulated code are applied in overwrite form rather than by full reassembly and linking of source modules. An overwrite consists of just those program and data instructions that are being modified in a program change. Thus, an overwrite tends to be small in size and the impact of the change is local rather than global. Overwrites are usually inserted into a stable program version and tested one at a time in a cumulative fashion. In this way, the generic program evolves in a controlled way with each change being tested before the next is applied.

### *3.3 C-Language software generation*

During the lifetime of the TSPS No. 1 system, all software development was done in assembly language. However, for the introduction of the TSPS No. 1B, as well as for the development of future operator services, significant portions of the new software will be developed in the C programming language. Thus, a new set of software tools was implemented to support the native-mode development process for the TSPS No. 1B.

#### *3.3.1 The 3B Software Generation System*

The 3B Software Generation System (3BSGS) is a collection of software tools available with the 3B20D Processor.[10] These tools transform C-language source code into object files that can be loaded into the 3B20D disk and executed under the DMERT operating system. The main constituents of the 3BSGS are the C compiler (3bcc), the 3B20D assembler (3bas), the 3B20D link editor (3bld), and the 3B20D process loader (3bldp). These programs are designed to execute in the environment of the *UNIX* operating system.

The C compiler accepts C-language source files as input and trans-

forms them into relocatable object files. The 3B link editor, 3bld, can combine several object files into one by relocating program and data instructions and resolving externally defined symbols. It also provides the mechanism to resolve references to library routines. (Libraries are collections of precompiled modules that typically contain commonly used routines that are shared among many modules.) For TSPS No. 1B, the output of 3bld is passed to the 3B20D process loader for final processing.

The 3B20D process loader, 3bldp, takes as input a collection of relocatable object files produced by 3bcc or 3bld. By using a special-purpose specification file, it constructs an output, called a process file, which has a format appropriate for execution under the DMERT operating system.

### *3.3.2 The Make Program*

For TSPS No. 1B, a utility program, Make, is used extensively in conjunction with the 3BSGS during the software generation process. The purpose of Make is to automate the construction of a process file when one or more of the source files on which it depends have been modified. The generation process is controlled by a special-purpose description file, frequently referred to as a "makefile."

The advantages of the Make program are that it:

(*i*) Ensures that a process file is constructed correctly in that no commands are accidentally forgotten or specified incorrectly

(*ii*) Regenerates only those files that have been affected by a change

(*iii*) Minimizes the programmer effort in building executable processes.

For these reasons, Make is used extensively in TSPS No. 1B development. For the initial TSPS No. 1B generic, there are approximately 50 makefiles used in building over 300 3B20D processes.

### *3.3.3 Dependence on DMERT header files and libraries*

The TSPS C-language software runs under the DMERT operating system. The two chief software mechanisms that support communication between the TSPS application and the operating system are header files and libraries. Header files are typically used to define data structures that are shared between processes or shared between source files within a single process. For example, there is a header file that contains data declarations specifying the layout of a DMERT Process Control Block (PCB) for a supervisor or user process. Any source file that references the PCB will include this header file through the "#include" mechanism provided by 3bcc, by which the PCB data declarations are made available to the compilation process. Libraries,

on the other hand, typically contain commonly used functions or routines. References to libraries are resolved by 3bld during the software generation process. Libraries are usually organized on a functional basis. For example, the craft interface library contains functions and routines oriented towards applications involved with the handling of craft input messages or the generation of output messages.

The DMERT operating system contains a large number of header files and libraries to support TSPS development. With each DMERT release, applications receive not only the latest version of DMERT software, but also updated versions of the header files, library files, and the 3BSGS. These are installed on the PSS1 system to support TSPS development.

#### *3.3.4 Linking C and emulated software*

As previously mentioned, the TSPS process contains both C-language and emulated programs. The emulated code is allocated one megaword of the available two-megaword-process address space and contains the bulk of the TSPS call-processing software. Naturally, there is a need to transfer control between emulated and C-language programs. Since these programs require different versions of microcode, a processor "mode switch" is required. Two special assembly-language instructions, cale (call emulated) and smt (switch mode and transfer), were developed to perform this function. Cale is a native-mode instruction that effects a mode switch and a transfer to emulated address space. Smt is an emulated instruction that performs a similar function but in the reverse sense.

From a software-generation perspective, the interface between emulated and native mode is quite simple. Routines called through cale or smt are accessed through transfer vectors. An emulated program, called MADEP, contains a table of transfer vectors, one for each emulated routine called from native mode. To link from a native to an emulated routine, the developer defines a symbol, say etv, equal to the address of the transfer vector in MADEP. (These definitions are kept in a header file dedicated for this purpose.) The emulation routine can then be accessed by an "spcxfr(etv)" function call in a C program. Spcxfr is an IS25 assembly-language routine that executes a cale to effect the transfer.

To transfer from emulated to native mode, the developer defines a global symbol, say ntv, equal to the address of the transfer vector for the desired function. These transfer vectors are forced to specific locations by special control statements in the loader specification file of the TSPS process. Thus, their location is known to the programmer. The transfer is accomplished through the use of the smt instruction.

Although this emulated-native transfer mechanism requires some

manual coordination between emulated and C-language files, this effort is only required when a new routine (to be accessed from the opposite mode) is added. These routines can be modified at will without requiring the referencing programs to be regenerated. Since these special-purpose routines are added infrequently, emulated and C-language development can proceed, for the most part, quite independently.

### *3.4 Overwrite considerations for the C-Language environment*

For C-language software generation, there is no direct analogue to an overwrite for emulated software. C-language frozen development involves full recompilation and link editing of TSPS processes. The effect of this on TSPS development is discussed in Section 4.3.3.

### *3.5 Building a 3B20D disk image*

Thus far, the mechanisms for constructing DMERT processes files have been described. To complete the software generation process, it is necessary to build a 3B20D disk image containing the total collection of process files, special-purpose boot files, and equipment configuration information. To accomplish this, the DMERT operating system utilizes two major data bases: the Equipment Configuration Database (ECD) and the System Generation (SG) database. The ECD contains records describing the characteristics and status of all processor and peripheral hardware. The contents of the ECD specify the current hardware configuration of the system. This hardware status information is placed in a central database, the ECD, to eliminate redundant device information and to provide a unified approach to handling and accessing hardware configuration data. The SG database contains information specifying the structure of the 3B20D disk image as designed by the application. This includes operating system parameters, disk partitioning information, file system names and sizes, names and types of individual files, and the make-up of the operating system boot image.

A DMERT tool, 3bmkdsk, provides the capability to construct a disk image from the ECD and SG databases, DMERT process files, and TSPS process files located on the PSS. This program generates three 1600-bit-per-inch (bpi) magnetic tapes containing 32M bytes of data. The image on tape is used to create a 3B20D disk.

## IV. THE SOFTWARE DEVELOPMENT PROCESS

The development of the TSPS No. 1B with many software tools and literally thousands of files requires a rigorous set of procedures (or methodology) to allow the development of software to proceed in an

orderly and controlled fashion. This section describes the methodology employed for TSPS No. 1B development.

### *4.1 General support environment of TSPS*

At the source-language level, the initial TSPS No. 1B generic program comprises approximately 350 emulated source files and 380 C-language source and header files. In executable form, the emulated software amounts to about 2.3M bytes of program. This excludes scratch data areas and office-dependent data (describing the particular line and trunk arrangements of an office), which also reside in the emulation address space. The C-language source is transformed through software generation tools into over 300 3B20D process and data files.

To control development on this large collection of software modules, official and test versions of both emulated edit files and C-language source files are maintained under the Source Code Control System. SCCS is a collection of programs that can store and retrieve multiple versions of a file in a space-efficient manner. This not only maintains a history of changes but allows the retrieval of various versions of source code that, for example, might represent a recent release, the current official software, software under system test, etc.

In addition to using SCCS for source control, TSPS also makes use of a "featuring" concept. Featuring allows developers to maintain a *single* source file regardless of the number of program generics under development. In featuring, lines added, deleted, or replaced in a source file are bracketed by feature-control directives which, during the assembly of the source file, direct the assembler either to assemble or ignore bracketed source code. The feature-control directives used in TSPS emulated development are:

(*i*) INFOR feature-expression
(*ii*) OUTFOR feature-expression
(*iii*) ENDFOR feature-expression.

Feature-expression is a Boolean combination of feature names. In an assembly, each feature name, and consequently each feature-expression, evaluates to a true or false value. If the feature-expression associated with an INFOR directive is true, the source lines between the INFOR and corresponding ENDFOR are assembled. For OUTFOR, if a feature expression is true, it causes the bracketed source lines not to be assembled. A set of feature expressions is associated with each generic. In this way, a single source file can generate multiple distinct object files, each associated with a particular generic program under development. The featuring syntax used for C software is different, but has the same basic capabilities.

There are two modes of development used in TSPS. The "non-

frozen" mode is the first stage in the development process when most of the new feature software for a generic is written. The intent is to give the programmers as much freedom as possible in writing and testing their code. When most of the new feature development is complete, the "frozen mode" is entered. In this mode, tight control is exercised over changes that are applied to the generic. Each change is documented and tested individually to assure a controlled evolution of the generic software. The sections that follow describe in some detail how development works in these two distinct environments.

### *4.2 Non-frozen-mode methodology*

#### *4.2.1 Emulated development*

***4.2.1.1 Creation and modification of source modules.*** In the non-frozen mode, programmers have a large degree of freedom in modifying existing source files or creating new ones. To create a new source file for emulated program development, the developer simply uses the interactive editor to input emulated statements. The new source file can be assembled, put into executable form by the partial load capability, and tested in the TSPS system laboratories. At the time of the next "base load" (see 4.2.1.3), the source file will be installed on the remote IBM/MVS system where all emulated source files are maintained.

Most non-frozen-mode activity involves modifying APE line edit files under the control and administration of the TSPS Subsystem for Non-Frozen Development (TSPNF).

TSPNF uses a strategy that allows multiple developer teams to work in parallel in a non-interfering way. The hierarchical nature of the file system is used to logically group edit files based on generic name, base-load name, and team name. Thus, several teams may have edit files affecting the same source file. Individual team members can create edit files within the team directories via the TSPNF commands, nfget and nfput. The nfget command is used to retrieve from the TSPNF directory structure, for the purpose of editing, the edit file for a source module corresponding to a particular generic, base load, and team combination. If the TSPNF structure has no edit file for the particular team, nfget retrieves the latest official version of the edit file. The edit file is placed in the user's current directory (work area). Once the file is retrieved, the programmer can modify it as described by interactively adding, deleting, or replacing source lines or feature control directives. Once the edit file has been changed to the programmer's satisfaction, it can be returned to TSPNF for safekeeping by the nfput command.

***4.2.1.2 Generation of private versions for testing.*** The PSS2 interface for emulated program development contains a number of commands

to be used for the generation of IBM Job Control Language (JCL) scripts and for the submission of these scripts to the remote IBM/MVS system for execution. Two of these commands, swap and ppload, play a key role in generating software for testing in the non-frozen mode of development. The swap command is used to generate and submit the JCL for a SPC-SWAP assembly.

The swap command allows the programmer to assemble a new source file that has been created on PSS2 or, in the case of a program whose source is already resident on the IBM/MVS system, to specify the name of an APE edit file to be applied to the program source prior to assembly. Options are provided to retrieve the edit file from the user's current directory, the TSPNF team directory structure, or the official set of line edit files.

Regardless of the source of the input, a *generic-name* parameter is used to establish the proper environment for the assembly. This includes setting up references to the correct SWAP symbol and macro libraries, selecting appropriate feature names, and specifying generic-dependent listing-format options. Typical output files from the swap command are the assembly listing file and an object module. These both reside on the remote IBM file system with the object module placed in a special team partitioned data set (a single data set containing a collection of object modules for a specific development team). The listing file can be printed locally in original or post-processed format.

The ppload command provides the programmer interface to the partial load capability described in Section 3.2.1. This command will generate a partial load run using the HISTORY file from the latest base load (see Section 4.2.1.3) and all of the object modules in a specified set of team data sets. The result is a file on the IBM/MVS system in post-processed format that contains just the program differences introduced by the team object modules. Through the use of an optional parameter, this partial load file can be written to tape. This then provides a convenient mechanism to create a private, team version of software that can be taken into the system laboratory, overlayed on the current official version from the last base load, and tested.

The non-frozen development scenario from the programmer's point of view is quite simple. An edit file for the source file to be modified is retrieved with the nfget command, and changed with the interactive editor. Private developer or team test loads can be built by use of the swap and ppload commands. Once the edits have been tested, the edit file can be returned to the team TSPNF structure with the nfput command. All programmer activity with the exception of the testing itself occurs on the PSS2 system.

*4.2.1.3 Submission and integration of software changes.* Periodically, during the process of developing a new generic, newly developed software is integrated with the latest approved version of the generic program and tested. This process is called the "base load" process because it results in a new, independently certified program load that serves as the base for future development. For a base load, each team leader is responsible for submitting a complete description of the team input. This information includes the list of team edit files to be incorporated into the base load, the features that are completed with this submission, and other appropriate documentation as needed. Using this input, a new software version is constructed and turned over to the TSPS Integration Group for certification in the system laboratory. When certification is complete, the software is made available to the developer community.

### *4.2.2 C-Language development*

Thus far we have described the non-frozen development environment for emulated programs. This section addresses the C-language development environment. The Change Management System (CMS) plays a key role in both frozen and non-frozen C-language development in TSPS. To understand how C-language development proceeds, it is necessary to grasp the basic concepts employed in CMS.

*4.2.2.1 The Change Management System.* CMS is a collection of *UNIX* programs aimed at controlling the activity of programmers, test-teams, and administrators engaged in C-language software development. In CMS, all development activity is tied to the notion of a modification request (MR).

An instance of CMS has three main components:

(*i*) A set of project source files maintained under the Source Code Control System (SCCS), described earlier. This allows multiple versions of a source file to be kept (for example, the current official version, a version undergoing system test, and a developer's private version).

(*ii*) A relational database that maintains the status of MRs and relates MRs to SCCS (version) identifiers. This not only provides the status of MRs but also permits the retrieval of source file versions that correspond to particular features or code changes.

(*iii*) A set of directory structures called *nodes*, each of which is identical in makeup (i.e., reflects the generic program's directory structure) and is capable of holding all source, object, process, and data files that constitute a generic program.

The use of CMS in TSPS C-language development will be the topic of the next several sections.

*4.2.2.2 Creation and modification of source modules.* CMS maintains

an official source repository that consists of one SCCS file for each source file used in a generic development. For TSPS, this includes C-language source files, header files, and makefiles. These SCCS files are stored in a project directory structure, a subtree of the hierarchical file system, organized so that files making up a single process typically are found in a single directory, while the files associated with related processes are grouped in directory structures that are subtrees of the overall project directory structure.

As previously mentioned, the SCCS files can contain multiple versions of the corresponding source file. To modify a source file, the programmer first extracts the latest version of the file to be modified from the SCCS file in the official repository. The programmer then edits the file with the text editor, builds an executable version of the affected process file, and tests the resulting product. When satisfied with the changes to the source file, the programmer returns the new version of the file to SCCS.

All source file change activity is associated with an MR number. Basically, the MR is a name under which a set of software changes are grouped. The programmer is quite free to use MRs as necessary during the non-frozen mode of development. Typically, the programmer will define an MR to represent a feature or subfeature being developed.

***4.2.2.3 Generation of a private version of software for testing.*** The generation of executable process files from C-language source uses the 3BSGS and makefiles, as descibed in Sections 3.3.1 and 3.3.2. However, rather than using the make command, an enhancement of make, called build is employed by TSPS developers. Build is a software tool provided as part of CMS. It provides a very convenient mechanism to construct a private version of a process or set of processes. The programmer executes a single command and only has to deal with those files that have been modified. Other required files are automatically shared from the official directory structure. This sharing not only saves file space, but also eliminates the need for time-consuming, error-prone copying of files.

In TSPS, each programmer maintains a private node for development. All programmers share the official node. The developer scenario is as follows:

(*i*) A command is issued to retrieve the source files to be modified.

(*ii*) The retrieved source files are edited.

(*iii*) Executable versions of the software are constructed using build.

(*iv*) The executable files are transported to the system laboratory for testing.

(*v*) When changes are complete, the source files are returned to SCCS.

There is little chance for error during software construction since build

automatically generates the commands to reprocess any and all files that have been affected by the editing. This eliminates the manual effort to reprocess files, the potential errors caused by forgetting to reprocess files, and the overhead of unnecessarily reprocessing files.

*4.2.2.4 Submission and integration of software changes.* In the non-frozen mode, as developers complete testing of new software associated with an MR, the code is submitted by executing the CMS submit command. This changes the status of the MR in the CMS database from "active" to "submitted." In addition, documentation specifying the feature(s) covered by this submission and the source and process files affected is turned over to the TSPS Integration Group.

As with emulated software, (and usually at the same time), C-language software undergoes a "base load" process. After the load is tested and certified, the official node is updated with all files in the test node and the MRs are marked "approved" and "integrated" in the CMS database. Non-frozen development can then proceed with the newly tested and approved official node.

### *4.3 Frozen-mode methodology*

#### *4.3.1 Overview*

In the non-frozen mode of development, the emphasis is on giving the programmer freedom to make large-scale changes to existing software to facilitate new feature development. As system testing begins, TSPS software is placed under rigorous change control procedures. Problems are documented in the form of trouble reports, which are then carefully monitored. The developers submit correction reports (CRs), which include a description of the fix, as well as the software change itself. Each CR is tested individually against the existing approved software version and must be approved by a Change Review Committee with representatives from all involved project teams before it is officially incorporated in the generic.

The intent of this method is to evolve the software in a rigorously controlled manner. Thus, when operational problems arise, they can be localized more easily to a particular area of software. With this approach, a high degree of confidence in the evolving software product is realized.

#### *4.3.2 Emulated development*

From a programmer's point of view, the scenario for creating and testing a correction in emulation code in frozen mode is similar to developing code in the non-frozen mode; however, the unit of change is substantially different. In non-frozen mode, the unit of change was the partial load. In frozen mode, it is the overwrite described in Section 3.2.4.

Special "patch" directives recognized by SPC-SWAP are placed by programmers in the overwrite source code to ensure that a change does not cause the entire object file to be relocated in memory. Replacement of program or data words is permitted, but only by instruction sequences of equal size. If a change is larger than the instruction sequence being replaced, a transfer to "patch" area is made. (Patch area is spare memory space reserved for this purpose.) In this way, object modules remain the same size and the number of memory locations whose contents change is minimized.

#### *4.3.3 C-Language development*

For C-language development, there is little difference between the frozen and non-frozen modes. The same tools are used and the same basic development scenario is followed. The main difference is that each MR represents an existing system problem or minor enhancement rather than a new feature. Therefore, the amount of software associated with an MR tends to be of a smaller quantity. In addition, as with emulated software, software change reports are generated and submitted to document the change and must be approved through the same process.

## V. LABORATORY SUPPORT SYSTEM

The Laboratory Support System is a software testing system that enables users to test their programs in a TSPS system laboratory. It is generally concerned with the laboratory execution environment of TSPS programs.

### *5.1 System laboratory configuration*

To provide sufficient test capabilities for TSPS development, there are two independent TSPS system laboratories. Generally, both system laboratories support parallel execution environments for TSPS programs. The configuration for a TSPS system laboratory is shown in Fig. 3.

#### *5.1.1 System laboratory hardware configuration*

The TSPS No. 1B is controlled by the SPC 1B, which consists of a 3B20D Processor and a Peripheral System Interface (PSI). The complete description of the SPC 1B is detailed in Ref. 11. In addition to the SPC 1B, a TSPS system laboratory also contains a set of TSPS peripheral units, operator consoles, and a Laboratory Support System.

#### *5.1.2 Laboratory Support System*

The Laboratory Support System is primarily used to support debugging on the SPC 1B, for controlling the operations of the SPC 1B

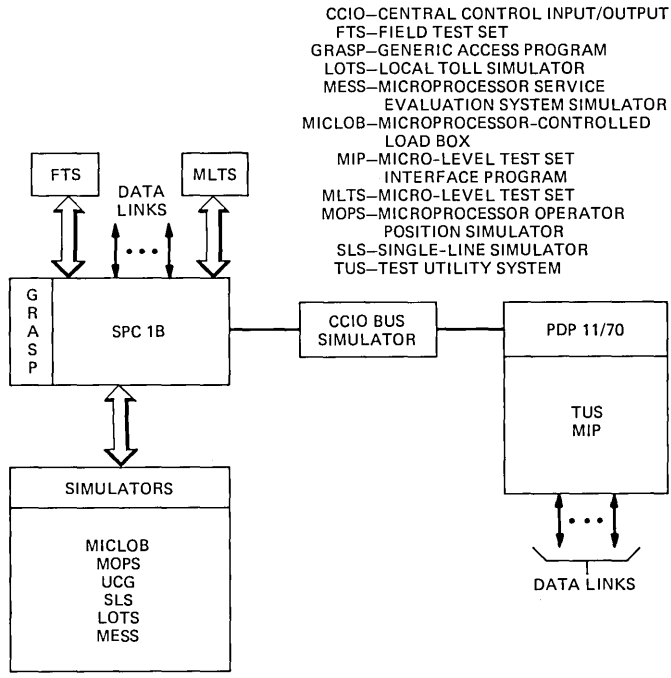



Fig. 3—Laboratory configuration for TSPS No. 1B.

in the test environment, and for loading the SPC 1B memory. The LSS consists of a PDP-11/70 support computer, special laboratory and generic utility systems, and TSPS call simulators, which support program testing. The PDP-11/70 is connected to the SPC 1B by a Central Control Input/Output (CCIO) bus simulator unit.[11]

There are four major debugging tools associated with a TSPS LSS. They are the Test Utility System, the Micro-Level Test Set (MLTS), the Field Test Set (FTS), and the DMERT Generic Access Program (GRASP). Each provides a distinct set of debugging capabilities.

***5.1.2.1 Test Utility System.*** The Test Utility System (TUS) is the principal, high-level, debugging tool for the software developed for the SPC 1B. TUS provides symbolic access to data and a "C"-like utility language. It resides partially on the LSS and partially on the SPC 1B. The TUS support processor subsystem on the LSS performs TUS input and output processing and TUS system control. Users log into TUS on the support processor subsystem; this subsystem converts the user's utility commands into executable command groups that are then sent to the TUS test processor subsystem on the SPC 1B. The test processor subsystem executes under the DMERT operating system. Symbolic references are resolved on the LSS using symbol tables constructed on the PSS as part of the base load process.

As the TUS test processor subsystem on the SPC 1B receives command groups to be executed, requested break points or matchers are set up, and associated utility commands are processed. Any raw data that result from the utility commands are collected and sent back to the TUS support processor subsystem to be processed. Output to the user includes symbolic references and processed data. Commands are available to: read or write any memory location, display or send a message to a process or port, enable or disable a program trace, copy a file from the test processor to the LSS, start or stop a test process, and send an event or fault to a test process with associated data input.

Because of its symbolic access to data and its "C"-like utility language, TUS is a powerful debugging tool. In addition, TUS is an integral part of the system overwrite facility. TUS supports a data link that is capable of transferring files at a rate of 56K baud. Using TUS, overwrites are quickly transferred from the LSS to the SPC 1B when requested by the programmer or tester. More details about TUS can be found in Ref. 10.

***5.1.2.2 Micro-level test set.*** The micro-level test set connects to the microbus of the SPC 1B and is used for direct access to 3B20D registers and memory. From a terminal connected to the LSS, simple commands can be sent to the MLTS to insert breakpoints. After a breakpoint fires, the SPC 1B is halted so that the tester can display or change the contents of registers and memory locations. Once this is done, the processor can be restarted. MLTS (unlike TUS) is a stand-alone tool that is completely independent of the operating system on the SPC 1B and can be used to debug at the kernel level. However, it does not provide symbolic testing capabilities and is used only for low-level program debugging.

***5.1.2.3 Field test set.*** The field test set is a portable debugging tool capable of tracing the execution of processes without interfering with normal processor operations (Fig. 4). As such, it provides an effective debugging capability both in the system laboratory and operational field sites. It interfaces directly with the dual-access utility circuit in the 3B20D Processor to record information about the execution of processes in the SPC 1B. The tracing capabilities of the FTS are controlled by simple commands that set up matchers, trigger functions, and trace memory. The FTS can be used to perform a process trace, transfer trace, data history trace, simultaneous data history and transfer trace, function trace, and simultaneous data history and function trace. Like the MLTS, the FTS is completely independent of DMERT.

***5.1.2.4 Generic Access Program.*** The final debugging tool used in the development of the TSPS No. 1B is the Generic Access Program (GRASP), which is a standard DMERT utility system resident on the 3B20D. It provides basic, non-symbolic capabilities to dump registers

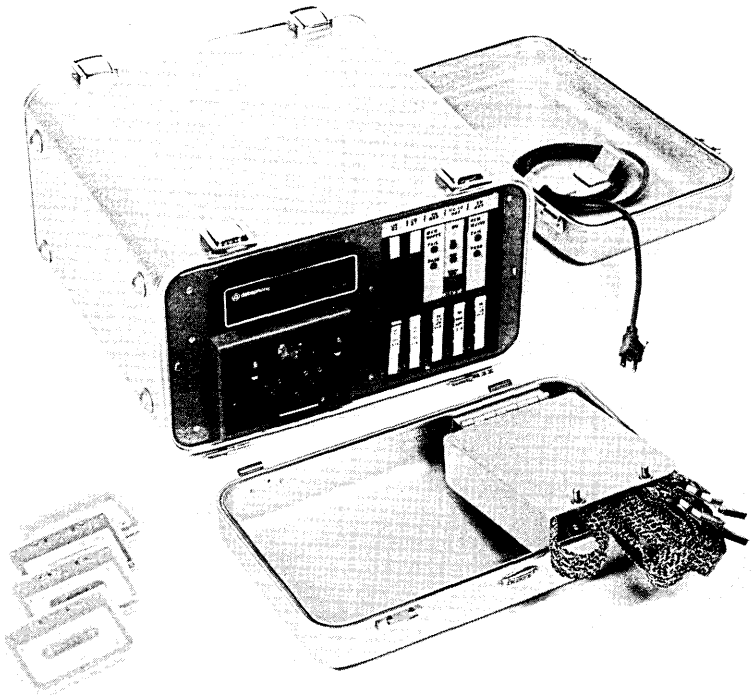


Fig. 4—The field test set.

and memory, and to set up breakpoints to stop program execution at desired points. Using the 3B20D dual-access utility circuit, GRASP also can trace the flow of execution of instructions of a process in a non-interfering fashion. A typical trace might record the "from address" and "to address" of every branch instruction executed. Additional information, such as the contents of the registers, may also be obtained. Because it is running on the same processor as the test processes and requires the support of the DMERT operating system, GRASP will generally interfere (like TUS) with the normal operations of a process and is not effective for kernel debugging.

***5.1.2.5 Choosing the right debugging tool.*** As evident in the previous discussions, the four debugging tools used in the TSPS No. 1B development were designed for specific applications and have overlapping utility capabilities. TUS is a high-level, symbolic debugging tool. The MLTS is a low-level debugging tool. The FTS is a non-interfering debugging tool in the sense that normal instruction sequences and timing are not altered when using it. GRASP is a generic tool available on an operational 3B20D. Within a particular test session, a combi-

nation of these tools may be used. Whenever possible, TUS is used because of its powerful debugging capabilities. The MLTS, FTS and GRASP are used only when needed for a particular problem.

### *5.2 Laboratory software change facility*

When a new TSPS generic is developed, the TSPS system laboratories are used to provide a realistic environment to test the new or changed programs. These new or changed programs are built on the PSS and then down loaded via the LSS to the SPC 1B for laboratory testing. Changes are often required to correct program problems uncovered during the lab session. These changes are in the form of overwrites consisting of program and data instructions being modified.

#### *5.2.1 Emulated program change facility*

To modify emulated programs in the system laboratory, the OVGEN program on the PSS is used to generate an overwrite file. This file is then transferred to the LSS and used as input to an overwrite assembler on the LSS to produce an overwrite object module. Using TUS, this module is then transferred to the SPC 1B disk and a special loader program on the 3B20D is run to overwrite the SPC 1B emulated address space. The overwrite assembler and loader used in this process are described in this section.

*5.2.1.1 The overwrite assembler for emulated code.* NOVA is a utility program on the LSS used to assemble overwrites for SPC 1B emulated code. The input to NOVA is the swap overwrite file produced by OVGEN, and the output is an absolute or relocatable object module and a relocation dictionary for the overwrite. The information contained in the relocation dictionary consists of pointers into the object file and addresses to be assigned by the loader. NOVA also produces overwrite and cross-reference listings to document the change in the laboratory.

*5.2.1.2 The overwrite loader for emulated code.* The loader program (NULOAD) is a relocatable loader for emulation code on the 3B20D. It provides a means by which NOVA-assembled overwrites are loaded into the TSPS process address space in memory. During each lab session, NULOAD automatically monitors the allocation of patch space and assigns patch space as needed by the overwrites. It allows the user to combine various overwrites assembled at different times and by different people without having one patch in one overwrite loaded on top of a patch from another overwrite. NULOAD also is used to load a partial load created on the PSS system.

#### *5.2.2 Native-mode program change facility*

To modify C-language programs in the system laboratory, an entire new process file (pfile) must be built on the PSS. Once this file has

been transferred via the LSS and TUS to the SPC 1B disk, it is installed on the 3B20D disk replacing the previous version of the process file. The next time the process is loaded into memory, the new pfile is used.

### *5.3 Laboratory simulators*

Program testing in the TSPS system laboratory often requires the ability to either place a single call or a substantial traffic load on the TSPS and to automatically handle calls that require operator assistance in a live site. A number of microprocessor-controlled simulators including the Single-Line Simulator (SLS), Local Toll Simulator (LOTS), the Microprocessor-Controlled Load Box (MICLOB), and the Microprocessor Operator Position Simulator (MOPS) have been developed as part of the LSS to provide these capabilities for the TSPS No. 1 system and were also used to test the TSPS No. 1B. These call simulators are described in detail in Ref. 6.

## VI. SUMMARY

The TSPS No. 1B Software Development System provides a complete set of facilities for TSPS generic development on the 3B20D Processor. Software generation tools can compile and link both emulated, assembly-language, and C-language programs. Administrative software and a rigorous development methodology control the evolution of feature development and keep track of the multiple versions of software that exist during the development process. Special-purpose software and hardware in the TSPS system laboratories provide a modern test environment for debugging and certification of the software product. These facilities are an effective and productive software development environment for future network operator services.

## VII. ACKNOWLEDGMENTS

Many individuals have contributed in a significant way to the design and implementation of the TSPS No. 1B Software Development System. In particular, the authors wish to acknowledge the contributions of K. M. Conness, G. F. Gieraltowski, B. E. Holmes, M. S. Koehler, B. T. Rovegno, R. J. Welsch, and S. P. Winker, who were responsible for major components of the Software Development System. In addition, we would like to extend our thanks to D. L. Atkins, B. T. Rovegno, and E. S. Sachs for their many constructive suggestions during the preparation of this article.

## REFERENCES

1. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.

2. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.
3. B.S.T.J., *61*, No. 7, Part 3 (September 1982), special issue on the Stored Program Controlled Network.
4. D. M. Ritchie and K. Thompson, "*UNIX*™ Time-Sharing System: The *UNIX* Time-Sharing System," B.S.T.J., *57*, No. 6, Part 2 (July-August 1978), pp. 1905–30.
5. T. A. Dolotta, R. C. Haight, and J. R. Mashey, "*UNIX*™ Time-Sharing System: The Programmer's Workbench," B.S.T.J., *57*, No. 6, Part 2 (July-August 1978), pp. 2177–2200.
6. J. J. Stanaway, Jr., J. J. Victor, and R. J. Welsch, "Software Development Tools," B.S.T.J., *58*, No. 6, Part 1 (July-August 1979), pp. 1307–34.
7. M. E. Barton, N. M. Haller, and G. W. Ricker, "Service Programs," B.S.T.J., *48*, No. 8 (October 1969), pp. 2866–80.
8. M. E. Barton, "The Macro Assembler, SWAP—A General Purpose Interpretive Processor," Fall Joint Computer Conference, 1970.
9. J. C. Lund, Jr., M. R. Ordun, and R. J. Wojcik, "Implementation of the Calling Card Service Capability-Application of a Software Methodology," Int. Commun. Conf., Denver, Colorado, June 1981.
10. B.S.T.J., *62*, No. 1, Part 2 (January 1983), special issue on the 3B20D Processor & DMERT Operating System.
11. G. T. Clark, H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Integration of the 3B20D Processor into TSPS," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Integration and System Testing

By R. AHMARI, R. S. DiPIETRO, S. C. REED, and J. R. WILLIAMS

(Manuscript received June 30, 1982)

*The integration testing and system testing strategies used in the development of the Traffic Service Position System No. 1B (TSPS No. 1B) are described in this article. It discusses novel methods developed for the integration and testing of the first embedded application of the 3B20D Processor and DMERT operating system. The new testing techniques and project management aides, which were vital to the achievement of system quality for the Bell System service, are described. The first TSPS No. 1B went into service on November 20, 1981, in Fresno, California.*

## I. INTRODUCTION

The first Traffic Service Position System No. 1B (TSPS No. 1B) was placed in service in Fresno, California, on November 20, 1981. This newly installed TSPS brought modern Stored Program Control (SPC) operator services capabilities to telephone customers in Fresno and other San Joaquin Valley communities, and permitted Pacific Telephone to close the largest remaining Bell System Cordboard installation in the United States. Modern operator services[1,2] are bringing convenience to telephone customers and increased efficiency and savings to Pacific Telephone.

Placing the Fresno TSPS No. 1B in service marked the conclusion of a major testing program that was vital to the delivery of a high-quality system, capable of meeting all objectives.[1] This was especially critical because of the planned, rapid conversion of existing TSPS No. 1 sites to TSPS No. 1B. Furthermore, since TSPS No. 1B is the first electronic Stored Program Control system to utilize an embedded

3B20 Duplex (3B20D) Processor and its Duplex Multi-Environment Real-Time (DMERT) operating system,[3] confirmation of its capabilities was essential to the success of the TSPS No. 1B program, as well as to other systems utilizing the 3B20D Processor.

Following the successful introduction of TSPS No. 1B in Fresno, a second new start TSPS No. 1B was placed in service in San Antonio (Southwestern Bell Telephone) on January 30, 1982. Both the San Antonio and Fresno sites were instrumental in the overall test program. The first conversion of an existing in-service TSPS No. 1 office to TSPS No. 1B occurred at Redwood City, California, on March 13, 1982.

The balance of this paper will focus on the overall test program for TSPS No. 1B, specifically on integration and system testing. These systemwide test activities are major components of the methodology used for development of TSPS No. 1B (see Fig. 1*). Integration testing specifically includes: the introduction of various system components (hardware and software) into the development laboratories; the integration of these components to form a stable operating environment; and testing to ensure that a sufficient set of capabilities exists to warrant full, comprehensive, and broad testing. System testing includes the later testing and is done to verify the system, as a complete functional product, before its release to the field.

The integration and system testing of TSPS No. 1B and its components—the 3B20D Processor, the TSPS Peripheral System Interface (PSI), the DMERT operating system, and TSPS application software—was a complex effort. Because TSPS was the first user of the 3B20D as an embedded processor, a substantial cooperative effort with the 3B20D development team was scheduled to identify and solve processor and operating system problems that were encountered during the design, integration, and testing of TSPS No. 1B. A tightly scheduled and controlled series of integration tests were performed by a team at the Bell Laboratories Indian Hill Laboratory in Naperville, Illinois, as new hardware and software capabilities were delivered to the system laboratories. After these new capability packages were integrated and certified as ready for system test, they were delivered to the system test teams at Indian Hill, Fresno, and San Antonio. The use of the two field sites provided settings similar to live TSPS offices for early identification of site-dependent problems and for verification of future in-service retrofit procedures with Western Electric engineers.

The system test plan consisted of over 16,000 unique tests of three general types:

---

* Acronyms and abbreviations used in this paper are defined at the back of this Journal.

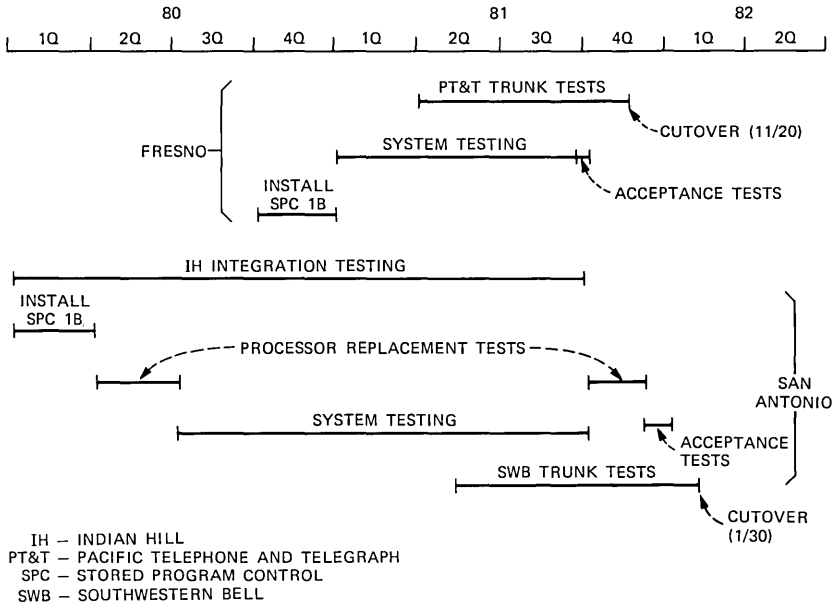



Fig. 1—Test schedule for system integration.

(*i*) Functional tests—to verify that design requirements were met by new features.

(*ii*) Regression tests—to verify continued proper operation of previously delivered features.

(*iii*) Stress tests—to parametrically determine and evaluate system limits, response characteristics, and overall performance under traffic overload conditions and fault conditions.

As depicted in Fig. 2, system test responsibilities were allocated to teams at the three locations with some intended overlap. For example, most of the DMERT feature system testing was scheduled for Fresno, to take advantage of the higher simulated-traffic-load capability at that site. Of these tests, some had been previously run at Indian Hill and at San Antonio, where exposure to different hardware and system configurations was critical. San Antonio was selected as the site to perform exhaustive tests of the existing TSPS periphery and to test external interfaces, such as to the Switching Control Center System (SCCS). In addition, a core of tests was periodically selected for all three system test sites to evaluate key software releases and to regularly assess the performance of a broad spectrum of features. In general, these "Run-for-Record" tests were a subset of the tests included in the system test plan.

Both integration and system testing were pursued for a period of over one year to completely verify the operation of the TSPS No. 1B.

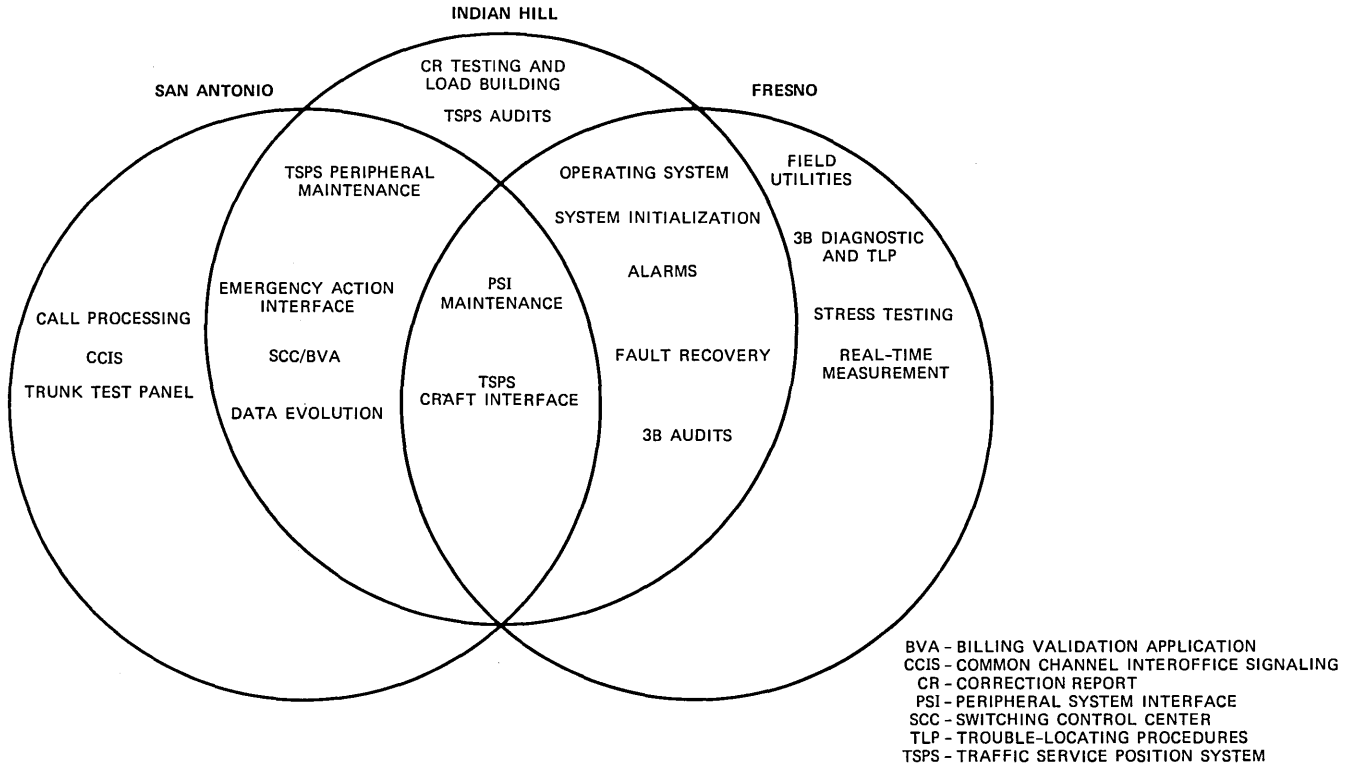



Fig. 2—Test plan overview.

Deliverables were phased to provide a smooth work flow. For this rigorous and demanding test program, tests were conducted ranging from microscopic tests, aimed at verifying operational status of specific elements of system capabilities, to global capacity and performance measurements and evaluations. During the test period, all problems found were tracked and pursued to their resolution. Performance criteria were set, performance was measured to ensure convergence to these criteria, and strong project management techniques were employed to ensure a timely introduction of a high-quality TSPS No. 1B.

The following sections of this article describe how the laboratory integration and site system testing were conducted. The excellent performance of the TSPS No. 1B system at cutover resulted not only from good planning and design but from timely integration and comprehensive testing supported by appropriate documentation, development tools, and change control procedures.

## II. INTEGRATION TESTING

### *2.1 Overview*

A crucial stage of the verification and evaluation process for new 3B20D/DMERT and TSPS capabilities is integration testing. During the early stages of the process, the development environment was considered to be "non-frozen," and designers could freely make large-scale changes to the existing capabilities or introduce new ones. The design and development of various features were accomplished by partitioning them into a series of functional units, which were then tested by the designers to make sure that each unit met the design requirements. Upon the completion of unit testing by the designers, a set of integration tests were performed by an independent team to ensure that each functional unit performed reliably in the total system environment. Upon successful integration of functional units, they were considered "frozen," and changes were made only through formal procedures. The frozen functional units were then ready for system testing and evaluation, the last stage of the verification process.[4]

Certain unique characteristics of the TSPS No. 1B development environment required special attention during the integration process. These characteristics can be summarized as follows:

(*i*) TSPS No. 1B was developed in a multilanguage environment. DMERT software was primarily developed in the high-level C language, while TSPS software was developed in both assembly language (TSPS emulated software) and the high-level C language (TSPS native software).

(*ii*) DMERT software changes were concurrently developed by the common system organization and had to be integrated with the TSPS

software changes developed by the application organization. In some cases a DMERT change required a coordinated change from TSPS or vice versa.

(*iii*) 3B20D firmware/hardware changes had to be similarly integrated with the TSPS firmware, hardware, and software changes, such as those associated with the PSI.

(*iv*) A new generation of TSPS No. 1B support tools was developed, resulting in new load generation, installation, and integration procedures.

The above characteristics were taken into consideration to establish an efficient integration and test strategy, which ensured the quality of the functional units integrated into the system.

### *2.2 Integration testing methodology*

The integration testing philosophy adopted for TSPS No. 1B established a rigorous methodology with systematic checks to independently evaluate all functional units supplied by designers in a single environment. To meet this objective, it was decided that the integration group should be independent of the development groups. Independence provided an unbiased and fresh viewpoint in assessing the functional units and interpreting the test results.

The integration test plan started early in the development cycle, subsequent to the availability of feature requirements. The plan took into consideration the availability sequence of various features and functional units within each feature. A strategy was established that guaranteed efficient handling of all units ready for integration testing without interrupting the other development activities. Special efforts were required to coordinate and synchronize the integration testing of TSPS functional units with 3B20D/DMERT features and capabilities. This was especially crucial in interface areas where direct interaction between the two environments existed. Another major consideration in the plan was that the software, firmware, and hardware environments evolve in a manner compatible with the system test plan, thus allowing the system testing activities to proceed without any interruption.

The TSPS No. 1B integration tests were generated independently, following a thorough analysis and review process, which is summarized as follows:

(*i*) Feature requirement and design specification documents were reviewed to identify functional units within each feature.

(*ii*) Functions performed by each unit, along with its input/output characteristics, were identified.

(*iii*) Software interfaces with firmware and hardware were examined and reviewed.

(*iv*) 3B20D/DMERT interfaces with the TSPS application were thoroughly analyzed, and expected input/output responses of the interface units were identified.

(*v*) The control flow among various units and their time sequence description was carefully inspected.

(*vi*) A general layout and description of data used by each unit were identified.

Based on these examinations, a set of integration tests were developed to exercise each functional unit within the total system environment and to stress 3B20D/TSPS interfaces. These tests were later augmented by the information supplied by the designer(s), upon completion of unit testing, to start the integration testing process. During this process it was first verified that each unit correctly performed all intended functions, including its interaction with other units in a single environment. The second, more subtle, objective was to ensure that the unit did not perform any function that, either singly or in combination with others, would degrade the performance of the system. Third, each unit was examined to make certain that it met the standards set for design structure, documentation, and coding.

Problems identified during the process of integration testing were classified according to their impact on the overall system. Problems were prioritized and fed back to the designer(s) for corrective actions. Problems related to the TSPS software were primarily tracked by Trouble Reports (TRs), while Modification Requests (MRs) were used to track 3B20D/DMERT problems. Specific corrective action by a designer was required to close a TR or an MR.[5] The list of all open TRs and MRs was carefully monitored to evaluate the total impact and to establish a plan for closing each individual TR or MR. The primary objective of this plan was to ensure that the evolution of the TSPS No. 1B software, firmware, and hardware could proceed on schedule without any interruption.

#### *2.2.1 DMERT integration testing*

The development of DMERT software was done in parallel with the development of TSPS software. At well-defined points of the development, a full DMERT release was generated by the common system organization and delivered to the various applications after being tested on a stand-alone basis.[6]

Upon delivery, each DMERT release was first installed in the Program Support System (PSS). Subsequently, steps were taken to incorporate the new DMERT release into the TSPS No. 1B environment (see Fig. 3). These steps can be summarized as follows:

(*i*) The Equipment Configuration Database and System Generation Database (ECD/SG) were updated to reflect the latest DMERT

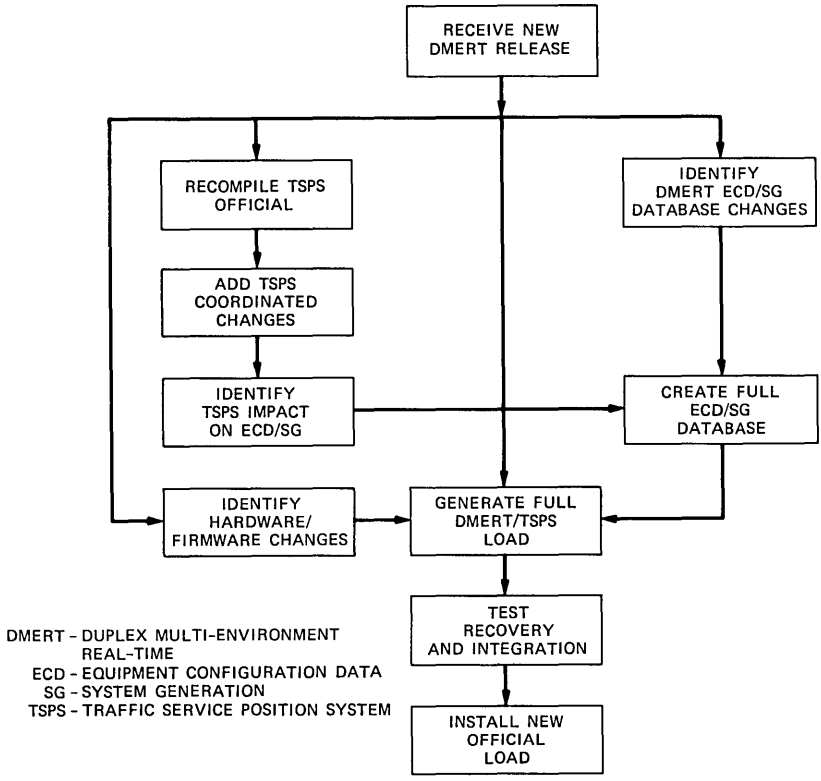



Fig. 3—DMERT integration testing.

changes. These databases contain data structures necessary to generate and run the TSPS No. 1B software system.

(*ii*) Those areas of TSPS software that were directly affected by the new DMERT release were identified. The corresponding TSPS software changes were then developed on the PSS under the control of the Change Management System (CMS). More information about the software generation process may be found in Ref. 5.

(*iii*) The TSPS official native-mode software was recompiled using the new DMERT release to update the TSPS programs that were affected by the DMERT release (such as a library that is shared by both DMERT and TSPS software and is changed by the new DMERT release). All TSPS programs that were changed by the recompilation process were audited for potential impact on the overall system. If necessary, appropriate TSPS software changes were generated [see Step (*ii*)] to compensate for the impact.

(*iv*) Those areas of DMERT that required changes unique to the

TSPS application were identified. These changes were also reflected in the PSS.

Subsequent to these steps, a full TSPS No. 1B load was generated on the PSS and was installed in the system laboratory for load recovery and integration testing. This load contained the latest DMERT release, along with changes unique to the TSPS application; a recompiled version of the TSPS official load; ECD/SG database changes; and the TSPS software changes required by the new DMERT release.

Any problems in the new load were quickly identified using the debugging tools associated with the TSPS No. 1B,[5] and immediate steps were taken to obtain the necessary changes from the appropriate designers (DMERT or TSPS). Once the load was cycling, a set of integration tests were conducted to ensure that each functional unit performed as expected in the total system environment. These tests were based upon the ones independently generated by the TSPS application organization, augmented by those supplied by the 3B20D common system organization. Special emphasis was given to DMERT/TSPS interface modules, TSPS areas directly impacted by the new DMERT release, DMERT changes unique to the TSPS application, and ECD/SG database changes. Problems identified during the process of integration testing were tracked using the strategy described in Section 2.2.

#### *2.2.2 TSPS integration testing*

Modifications to the TSPS software were introduced into the TSPS No. 1B environment via base loads.[5] Introduction of a new base load began with identification of all source file changes that had occurred since the prior base load. Then a new test load was created, containing additional software changes and associated ECD/SG database changes. New software changes were usually grouped in one or more test packages. The packaging strategy for the TSPS changes took into consideration many factors including:

(*i*) Keeping a high degree of resolution in testing of various changes that have an impact on the same programs.

(*ii*) Optimizing the amount of time used in the system laboratory.

(*iii*) Keeping the PSS effort for compiling various changes to a reasonable amount.

(*iv*) Optimizing the system capacity for processing software changes.

Keeping these objectives in mind, all software changes were first mapped into a series of test packages. Program dependencies played a key role in determining the number and content of these test packages. The load of the test packages, along with coordinated hardware and firmware changes, were then installed in the system

laboratory separate from the TSPS official load before the integration testing could proceed.

Upon installation of the load, a predefined set of integration tests were used to examine each test package. These tests were aimed at quickly identifying problems and attributing them to individual changes. Problems encountered during the testing were classified by their impact on the normal operation of the system and tracked as described in Section 2.2. This process was then repeated for all test packages until the necessary resolution was obtained, and it was ensured that all changes could coexist and perform as expected in a single environment. All changes that successfully passed the integration testing were then installed in the system laboratory as the new official base load (see Fig. 4).

Upon completion of the integration testing, functional units were considered "frozen" and ready for system testing. In the frozen environment a set of stringent change control procedures were used so that the TSPS software evolved in a rigorously controlled manner, leading to a high-quality production release. All problems were documented and tracked by appropriate trouble reports, which were carefully monitored. To close a trouble report, a designer was required to submit a Correction Report (CR), which contained the functional description of the change, all necessary information relevant to the software change, and a description of the unit tests used by the designer. Each change was considered an independent entity and was individually tested in the total system environment before its approval. All changes had to be approved by the Change Review Committee (CRC) before they were incorporated in the official load. The CRC comprised representatives from each hardware/software design and test organization on the project.

### *2.3 Testing of 3B20D and TSPS firmware and hardware changes*

Firmware and hardware changes associated with the 3B20D were also tested in the system laboratory environment by the TSPS integration group to uncover problems unique to the TSPS application. These changes, along with coordinated software modifications, were first installed in the system laboratory environment. A series of integration tests were then performed to stress and exercise the firmware-hardware-software interfaces. Problems encountered during this stage of testing were tracked in a fashion similar to that described in Section 2.2.

In cases where 3B20D microcode changes had an impact on the writable portion of the microstore,[7] no firmware change was necessary, and the microcode change was released in a fashion similar to a software change described in earlier sections.

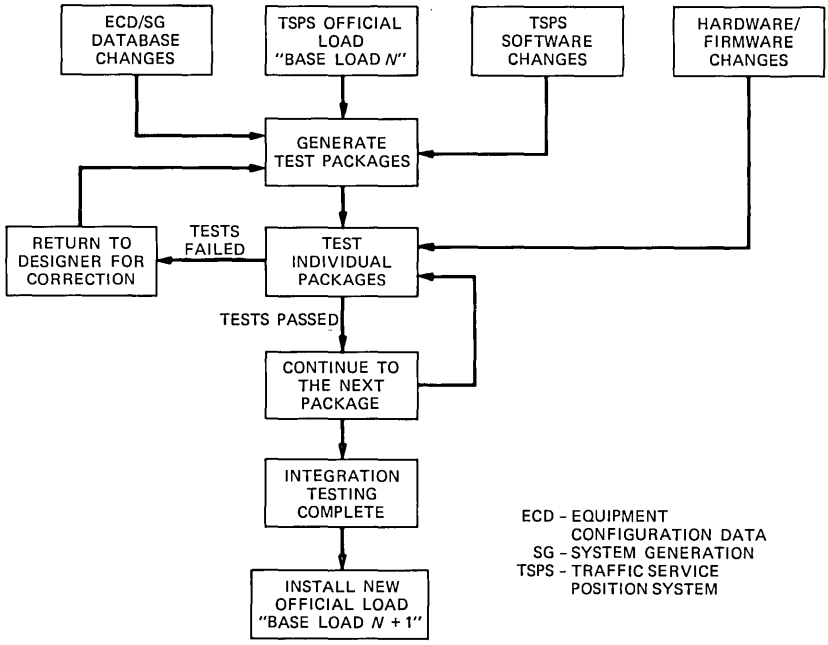



Fig. 4—TSPS integration testing.

Changes to the TSPS microcode were administered under control of CMS in a similar fashion to the TSPS software changes. A change was first developed on the PSS and was then submitted for integration testing. The change was transferred to the system laboratory in one or more files and was installed in the writable microstore unique to the system labs. A series of integration tests were then performed to verify its integrity in the total system environment. Similarly, all hardware changes associated with TSPS were tested in the system laboratory environment using prototype circuits prior to their official availability. Once it was verified that the hardware/firmware changes operated as expected, they were transmitted to Western Electric along with manufacturing and factory test information, and were subsequently installed at the field sites.

### *2.4 Distribution of individual software changes*

Once a stable TSPS No. 1B release was installed in a field site, the subsequent individual software changes were supplied as individual packages and were installed using the field update commands.[5] A package typically consisted of the software change, additional files containing field update commands, and information files required for

installation, testing, soaking, and distribution of the change to the field.

For a change in the 3B20D/DMERT software, the 3B20D common system organization generated a software package that was distributed to each application organization. Before the release of the package to the various applications, it was first tested and certified in the common system environment by the common system organization. Once the change was received by TSPS, it was examined for compatibility with the TSPS environment. In some cases a coordinated TSPS software change was required before actual installation and testing could proceed. A series of tests were then conducted to ensure that the change performed as expected within the total system environment. These tests included those supplied by the common system organization, as well as an independent set generated by TSPS personnel.

Once a software package was successfully tested and soaked in the TSPS system laboratory environment, necessary changes were made to its information files prior to field delivery. These changes were made to delete the test information unique to the system laboratory environment. This information was used by application organizations to verify the modification and was not applicable to the field environment.

The TSPS application software changes required for distribution to the field sites were individually tested and approved using the "frozen environment" methodology described in Section 2.2.2. These changes were then packaged using the standard format described earlier in this section. Necessary considerations were given in the packaging strategy to balancing and optimizing various factors such as the field installation time and the number of system initializations required to install the changes. Also, if necessary, these changes were retested to ensure that no problems were introduced in the final packaging.

## III. SYSTEM TESTING AND EVALUATION

### *3.1 Overview*

System testing also represented a crucial and essential part of the overall verification process. During this stage a complete set of functional tests were written and executed to independently evaluate all TSPS and DMERT operational and maintenance capabilities from a system viewpoint. These tests were systematically performed with the objective of ensuring that the requirements for each feature were met. Special emphasis was given to evaluating the interaction between the new features and the existing features that were emulated from the TSPS No. 1 environment. Furthermore, regression tests were performed to ensure that previous tested capabilities were not being adversely affected by the introduction of new ones. All problems

identified during the system testing interval were documented by appropriate trouble reports.

The system testing effort was divided into several areas with different objectives. They are listed below with the primary objective for each area and are further discussed in Sections 3.2 to 3.5.

(*i*) Call-processing objective—Verification of all emulated call-processing features.

(*ii*) TSPS maintenance software objective—Verification of fault recognition, routine exercises, and diagnostics.

(*iii*) External interfaces objective—Verification of external interfaces such as the Service Evaluation System (SES), Switching Control Center System (SCCS), and Billing Validation Application (BVA), etc. (see Refs. 1, 8, 9, and 10).

(*iv*) DMERT and 3B20D system testing objective—Verification of DMERT and 3B20D features, DMERT and 3B20D/TSPS interfaces, and sample faulting of 3B20D and its periphery in the TSPS environment.

(*v*) Regression testing objective—Determination of impact of new TSPS and DMERT/3B20D releases on previously tested capabilities.

(*vi*) System evaluation runs objective—Analysis of overall system stability and quality at various loads.

System tests were performed both in the system laboratory and the field sites. The field test plan consisted of functional tests, environmental tests, and an overall acceptance test, and was used as a final check that the system met its functional objectives at specified environmental limits. Over 16,000 individual functional system tests were written and executed during the system test interval at the San Antonio and Fresno test sites. The first execution of all system tests was scheduled and accomplished by early in the third quarter of 1981 (see Fig. 5). This timely execution enabled identification of problems, management, resolution, and closure by turnover (see Fig. 6). All significant system tests passed prior to the turnover of the Fresno site to the operating telephone company. The environmental tests verified system performance at the limits of high temperature and low voltage. The functional-site testing effort validated system performance in a fully equipped TSPS office that has been engineered by an operating telephone company with hardware supplied and installed by Western Electric.

### *3.2 Call-processing testing*

The call-processing software in TSPS consists of a set of programs that provide the logic and control for processing telephone calls. These programs supervise the state of the call, transmit and receive signals

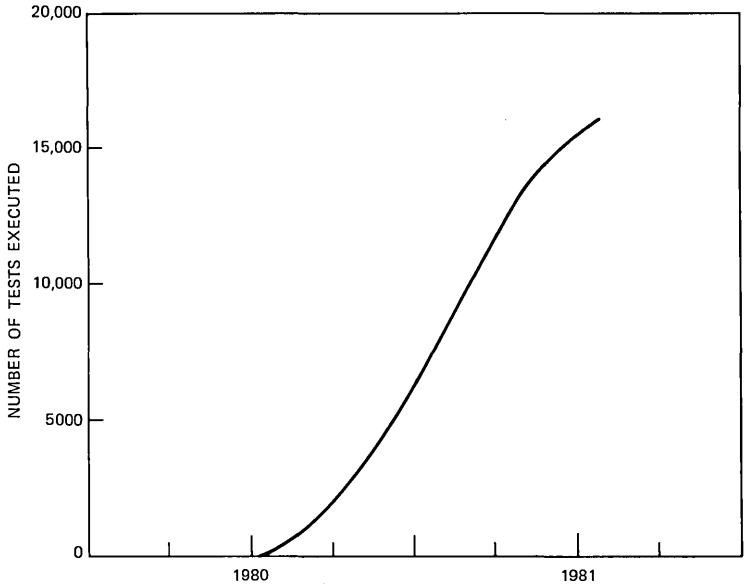



Fig. 5—TSPS No. 1B test schedule.

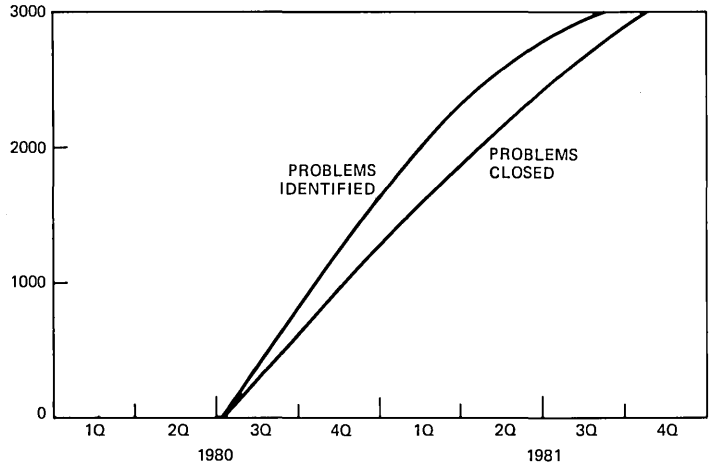



Fig. 6—TSPS No. 1B problem closure.

to and from other switching systems, send information to and receive signals from operators, and record billing details on the calls.

In most cases the call-processing programs were directly emulated from the TSPS No. 1 environment without any structural or code changes. Because the call-processing software was preserved in the TSPS No. 1B, the strategy for testing was mainly to verify the proper

emulation of the existing capabilities. This objective was accomplished by using system test facilities to generate simulated call inputs to exercise various call-processing capabilities, including those involving customers and operators. These capabilities were tested both on a single-call basis and under call loads simulating the normal traffic environment of the system.

### 3.3 TSPS peripheral fault recognition, diagnostics, and exercise

TSPS No. 1 peripheral maintenance programs such as fault recovery, diagnostic, and exercises were also preserved in the TSPS No. 1B environment. These maintenance programs were emulated from the TSPS No. 1 environment, after necessary code and structural changes were made to provide interfacing compatibility with 3B20D hardware, firmware, and maintenance software capabilities. The function of fault recognition is to analyze failure conditions and quickly reconfigure the system to a working state with a minimum interruption to operational call processing. Testing in this area verified that faults were successfully detected and appropriate actions were taken.

Diagnostics are used in TSPS to test the various equipment units in response to fault-recognition requests or manual requests. These capabilities resolve hardware malfunctions by providing trouble-locating messages for use by maintenance craft to repair the faulty hardware. Testing in this area verified that the diagnostics properly detected the presence or absence of troubles and produced messages consistent with expected results without adversely affecting system operation.

Specific routine exercises are used to periodically examine the correct operation of the hardware and to detect various failures that may not show up in normal system operation. In general, exercises utilize the diagnostic capabilities described above to verify the condition of the equipment units on a periodic basis. Testing in this area demonstrated that the exercise routines operate in the expected manner.

#### 3.3.1 Test strategies

Sample physical fault insertion, power faulting, and data faulting, combined with manual actions initiated at the Local Maintenance Position, were the major means of verifying the proper operation of system maintenance functions. These techniques were used to develop specific tailored test strategies for each peripheral unit. Initially, tests were conducted in the absence of call load to verify the proper response. Subsequently, maintenance capabilities were tested in the presence of varying call loads to ensure proper interaction of call handling and maintenance functions.

Physical fault insertion was one of the primary methods of testing the operation of maintenance software and hardware. Physical fault

insertion verified proper cooperation of fault recognition, system reconfiguration, and diagnostics. It further provided a test of the Trouble Locating Manual (TLM) used by the craft to identify faulty circuit packs. It was impractical to do exhaustive fault insertion, such as that utilized in TLM generation. However, it was necessary to develop sample faults to verify that the maintenance software performed within the existing requirements.

Power faulting was the second major means of introducing hardware faults. Power faults were introduced selectively by blowing fuses or intentionally removing power to verify the ability of the system to detect the fault, take the appropriate equipment out of service, and produce the proper alarm and output messages to enable craft personnel to restore the system.

Data faulting was the third faulting technique used in testing maintenance actions. Data faulting is the application of erroneous data to the system. It measures the sensitivity of the system, the adequacy of defensive program checks, and the efficiency of data consistency audits. This type of faulting was useful, for example, in testing response to transmission irregularities between the base and remote subsystems, such as the Remote Trunk Arrangement and Position Subsystem No. 2.[11]

Various manual actions at the Local Maintenance Position were also used to verify the proper response of the system in areas such as reconfiguration, diagnostics, measurements, control and display capabilities, and exercises. These types of actions are usually taken by the craftspeople both on a routine basis and in emergency-action conditions.

#### *3.3.2 TSPS peripheral test sequence*

Communication between the SPC 1A and peripheral equipment is accomplished by the Central Pulse Distributor (CPD), Communications Bus Translator (CBT), Master Scanner (MS), Universal Trunk Scanner (UTSC), and Universal Trunk Signal Distributor (UTSD). Maintenance testing verified that the emulated maintenance programs detect malfunctions in the above units and take appropriate recovery and diagnostic actions.

The TSPS periphery consists of a number of functional entities or subsystems (see Table I) that are administered by duplicated controllers that interface with the SPC bus system. Extensive tests were performed to verify the operation and maintenance of these functional units under control of the TSPS No. 1B software.

In TSPS, trunks and service circuits provide the interface between the TSPS and external systems. For maintenance purposes, access to these circuits is required to evaluate performance and localize or

Table I—TSPS peripheral subsystems

| |
|---|
| Position and Trunk Link Network |
| Position Subsystem No. 1 |
| Peripheral Control Link |
| —Position Subsystem No. 2 |
| —Remote Trunk Arrangement |
| Station Signaling and Announcement Subsystem |
| Common Channel Interoffice Signaling Links |

isolate a trouble to a particular faulty unit. Comprehensive tests were required to ensure proper operation of these trunk and service circuit routines. Table II lists the TSPS trunks and service circuits for which maintenance routines were tested.

### 3.4 TSPS external interface tests

A number of systems external to TSPS provide operator administrative functions, or interact with TSPS for services and maintenance. In general, these are self-contained systems, but rely on communications with TSPS over data links and/or channels, thus requiring coordinated testing between the two systems. The communications generally rely on specific protocols or message types. Testing of external interfaces verified that the necessary protocols, message handling, and maintenance states were operational in the TSPS No. 1B environment. Some of the external interfaces include:

(*i*) External administrative centers
(*ii*) The Hotel Billing Information System (HOBIS)[8]
(*iii*) The Service Evaluation System (SES)[9]
(*iv*) The Billing Validation Application (BVA)[1]
(*v*) The Switching Control Center System (SCCS).[10]

The interfaces in items (*i*) through (*iv*) above required only verification that TSPS No. 1B had not introduced operational or maintenance problems. The SCCS interface [item (*v*) above] required substantive testing because of new interface hardware and software (see Ref. 12). This interface was functionally tested between the system laboratories of the SCCS and TSPS development organizations. Field-site testing verified these functional capabilities and stressed load-related features that could not be tested earlier.

### 3.5 3B20D/DMERT system testing

3B20D/DMERT testing evaluated 3B20D/DMERT hardware and software in the application environment. The 3B20D/DMERT tests concentrated on interfaces, resource utilization, and system response characteristics in the TSPS application environment. As such, overload response, resource audits, and software fault tolerance were extensively tested. The combined craft/machine interface and

Table II—Tests of TSPS trunk, service circuit, and maintenance circuits

| |
|---|
| Service Circuits |
| —Dial-Pulse receivers |
| —Multifrequency receivers |
| —Multifrequency outpulsers |
| —High-Impedance, Multifrequency receivers |
| —Coin Control and Ringback circuits |
| —Reorder Tone and Announcement circuits |
| —Audible Tone trunks |
| —Coin Detection and Announcement circuits—Type 1 |
| —Coin Detection and Announcement circuits—Type 2 |
| —Dual Tone Multifrequency Detection and Announcement circuits |
| —Multifrequency Announcement and Detection circuits |
| Trunk Test Panel |
| AMA maintenance |
| Recorded Announcement equipment |
| Tone and Interrupter circuit |
| Time of Day circuit |

DMERT administrative functions (field utilities and field update, for example) were also extensively exercised.

#### *3.5.1 Fault recovery, initialization, diagnostics, and overload*

The TSPS No. 1B maintenance strategy is based on the 3B20D/DMERT maintenance design augmented by additional capabilities unique to the TSPS application. These additions are mainly in areas where TSPS software directly interacts with the DMERT software, or when specific hardware interfaces are required for the TSPS application.

Specific fault-recovery strategies were developed by TSPS for the PSI, the interface between the 3B20D and the TSPS periphery.[12] In addition, the TSPS Application Integrity Monitor (AIM) was developed to directly interact with the DMERT system integrity monitor (SIM) to ensure the system integrity of TSPS No. 1B. DMERT system initialization and overload features were augmented by TSPS to establish an overall system initialization and overload strategy for the TSPS No. 1B.[13] These areas were tested extensively to determine the proper system response.

The 3B20D and PSI diagnostics, Trouble-Locating Procedures (TLP), and Routine Exerciser (REX) were tested by sample faulting techniques. Manual requests for removal/restoral were also used for verification of capabilities.

The TSPS No. 1 maintenance craft interface was replaced in TSPS No. 1B with a combined 3B20D Processor and TSPS maintenance craft interface. Testing in this area concentrated on combined craft response, software and hardware alarm interfaces, and an assessment of process priorities to provide sufficient terminal response time under load.

### 3.5.2 Administrative functions

Testing in this area concentrated in the areas of 3B20D recent change/verify (for equipment and system configuration database changes), system update (for installation of a "point" issue of a current generic or installation of a subsequent generic), and field update (for installation of one or more broadcast warning messages).

## 3.6 System evaluation runs

### 3.6.1 Purpose and definition

One goal of system testing was to monitor a set of quantified performance criteria to measure overall system quality and to ensure that system design goals were met. Of particular importance for TSPS No. 1B was the evaluation of system behavior at full-load call handling (maximum capacity). This was particularly important since a substantial number of TSPS No. 1 to TSPS No. 1B live in-service retrofits were scheduled for 1982 for offices at or approaching real-time overload of the SPC 1A. Since the Fresno site had a large amount of peripheral equipment, sufficient microprocessor-controlled call simulators and operator simulators were installed on a temporary basis to permit stressing the new TSPS No. 1B above the capacity design objective. This additional performance margin simulated variation among sites with respect to call mix and peakedness in the busy hour(s).

In the first quarter of 1981, a series of weekly System Evaluation Runs (SERs) were begun at both Fresno and San Antonio to measure system performance in an environment approximating post-cutover conditions. Typically, these SERs were executed on weekends and were 24 to 48 hours long. During these SERs the following conditions were met:

(*i*) Operating telephone company maintenance craft were assigned to monitor and maintain the office, performing all TSPS peripheral maintenance and trunk cutover testing simulating an in-service office.

(*ii*) Bell Laboratories and Western Electric site staff had instructed the maintenance craft in the use of normal 3B20D Processor and SPC 1B peripheral maintenance procedures. When craftspeople were troubleshooting equipment faults or potential design problems, normal field repair tools and techniques were used.

(*iii*) A simulated call load was applied to the system. At Fresno, the first quarter 1981 SER was conducted at a load equivalent to 25 percent of the system call capacity. This was gradually increased so that by mid-1981, SERs were conducted above the call-capacity design objective of the TSPS No. 1B.

### 3.6.2 Measurements and objectives

A team of designers, testers, and system analysts defined a set of eighteen measurements that were tracked as part of the SER each

Table III—Fresno Service Acceptance Test Performance* (9/27/81–10/5/81)

| | System Performance | First-Site Turnover Objective |
|---|---|---|
| Call Load (Percent of SPC 1A) | 175% | 160% |
| Mishandled Calls | 0.04% | <0.05% |
| Plug-in Replacements | 0 | <1/10 days |
| System Initializations | 0 | 0.02/wk |

* Includes normal maintenance and PT + T trunk testing.

week to summarize system stability and maintainability. For each measurement, a turnover objective was specified. If several results deviated widely from the objective, a high priority was placed on resolution of these problems. When the measurements in those areas approached the turnover objective, resources were redirected to improving other areas of system performance. Statistics were kept on the frequency and cause of initializations, interrupts, audits, and overloads. The duplex performance of the processor and its disk subsystem was tracked and the degree of automatic fault recovery and identification was evaluated. Hardware failure rates were closely monitored and compared with reliability models. All initial service objectives were achieved prior to cutover. Table III shows Fresno Service acceptance test performance versus key first-site turnover objectives.

## IV. SUMMARY

The strength of the integration and system test plan for the successful field introduction of the TSPS No. 1B involved the use of complementary techniques to set objectives and monitor progress. The comprehensive functional and regression testing, close tracking of correction and modification requests, and prompt integration and delivery of a diverse set of software changes were essential to the steady progress summarized by the periodically executed System Evaluation Runs.

An immediate benefit of this approach was the customer satisfaction expressed by Pacific Telephone and Southwestern Bell Telephone at turnover, cutover, and subsequent operation of their new TSPS No. 1B systems. Furthermore, the implementation of the comprehensive test program described herein is responsible for the excellent performance of the 37 offices now in service (35 of which were live in-service retrofits in high-traffic sites). Current Bell Laboratories and Western Electric efforts involve the continued coordination and support of the large number of TSPS No. 1B live in-service retrofits planned for the next few years.

## V. ACKNOWLEDGMENTS

The authors would like to acknowledge the dedicated efforts and innovative technical contributions of their staff and the test coordination provided by J. C. Dalby and his staff. Special recognition is also given to the 3B20D Development team, particularly J. H. Miller and her staff, for their valuable participation in the TSPS No. 1B testing program.

## REFERENCES

1. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
2. E. M. Prell, V. L. Ransom, and R. E. Staehler, "The Changing Role of the Operator," Int. Switching Symp., Paris, France, May 1979.
3. Special issue entitled "3B20 Processor & DMERT Operating System," B.S.T.J., *62,* No. 1, Part 2 (January 1983).
4. J. P. Delatore, D. Van Haften, and L. A. Weber, "System Verification and Evaluation Procedures," B.S.T.J., *58,* No. 6, Part 1 (July–August 1979), pp. 1335–46.
5. T. G. Hack, T. Huang, and L. C. Stecher, "Traffic Service Position System No. 1B: Software Development System," B.S.T.J., this issue.
6. W. F. Klinksiek and H. L. Mitchell, "3B20D Processor and DMERT Operating System: System Integration and Test," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 399–410.
7. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 191–206.
8. S. Michael and J. Vizcarrondo, "HOBIS: New Designs on Hotel Billing," Bell Lab. Rec. (January 1980), pp. 11–18.
9. T. R. Lehnert, "A Better Way to Measure the Quality of Telephone Service," Bell Lab. Rec., *59,* No. 6 (July–August 1981), pp. 186–9.
10. J. J. Bodnar, J. R. Daino, and K. A. Vandermeulen, "Traffic Service Position System No. 1B: Switching Control Center System Interface," B.S.T.J., this issue.
11. S. M. Bauman, R. S. DiPietro, and R. J. Jaeger, Jr., "Remote Trunk Arrangement: Overall Description and Operational Characteristics," B.S.T.J., *58,* No. 6, Part 1 (July–August 1979), pp. 1109–18.
12. G. T. Clark, H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
13. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Retrofitting the Processor

By J. C. DALBY, JR., D. VAN HAFTEN, and L. A. WEBER

(Manuscript received June 30, 1982)

*At the end of 1981, over 150 Traffic Service Position System No. 1 (TSPS No. 1) offices were in service, equipped with Stored Program Control No. 1A (SPC 1A) processors. Some of these sites had reached the system capacity with respect to real time or memory. The new SPC 1B, which contains a 3B20 Duplex (3B20D) Processor and a Peripheral System Interface (PSI), provides the TSPS No. 1B with additional processor capabilities for additional capacity and future features. This article discusses the techniques used for achieving a smooth retrofit from the TSPS No. 1 to TSPS No. 1B with virtually no interruption of call processing. Special procedures and tools were developed to introduce the SPC 1B onto existing buses and to verify the interfaces with existing peripherals by means of a cycle-stealing mechanism, while the SPC 1A continues to handle call processing. These procedures were used successfully at the first such retrofit in Redwood City, California, on March 13, 1982. During 1982, 34 additional sites will be retrofitted by Western Electric to accomplish the initial phase of the planned retrofits to TSPS No. 1B.*

## I. INTRODUCTION

Over 150 Traffic Service Position System No. 1 (TSPS No. 1) offices[1] are in service in the United States. These systems give fast, efficient toll operator services to over 95 percent of the Bell System main stations. The TSPS No. 1 consists of a Stored Program Control No. 1A (SPC 1A) and numerous peripheral units. The SPC 1A has performed well since its initial introduction into service in Morristown, New Jersey, in 1969.[2]

The development of the TSPS No. 1B introduced a modern 3B20

Duplex (3B20D) Processor with its peripherals and a Peripheral System Interface (PSI) into TSPS.[3] The 3B20D Processor and the PSI together are called the SPC 1B.[4] The PSI connects the 3B20D to the existing TSPS peripherals in such a manner that the interface to the SPC 1B is the same as the interface to the SPC 1A.

Since many TSPS No. 1 sites are approaching real time and memory exhaust and the new processor capabilities are necessary for new features, a procedure has been developed to allow in-service TSPS sites to retrofit to the SPC 1B. This procedure requires advanced planning for floor space, connectorization of existing buses, and some hardware modifications. Special procedures and tools are used to ensure that the SPC 1B can properly interface with the existing periphery before the new processor is in control of call processing. After cutover and following sufficient soak time on the SPC 1B during which assurance tests are run, the SPC 1A can be removed from the office.

Unlike commercial processor upgrades, where the system is taken off-line, the replacement of the SPC 1A with the SPC 1B must be performed with virtually no interruption of call processing. For this reason, special procedures and tools were developed to prepare the site and check the new equipment and interfaces while the existing system continues to handle telephone traffic. These procedures had to be thorough and be able to completely verify the hardware configuration, since upon cutover the new system must be able to handle a large volume of traffic. Furthermore, since the existing peripherals are maintained, and the new processor does not control the peripherals until cutover, the final cutover procedures must be straightforward and fast. During the retrofit, the operator traffic is interrupted for only approximately the length of a system initialization on the new system.

This article discusses the techniques used to achieve a smooth retrofit from the TSPS No. 1 to TSPS No. 1B. These procedures were successfully used at the first such retrofit in Redwood City, California, on March 13, 1982. During 1982 34 additional sites will be retrofitted by Western Electric to accomplish the initial phase of the planned retrofits to TSPS No. 1B.

## II. SITE PREPARATION

The first step in retrofitting an in-service site is to make the site compatible with the SPC 1B. Site preparation can be planned by the operating telephone company for any time prior to the actual retrofit. The site preparation, both bus modifications and auxiliary unit relocation, is performed according to Western Electric procedures. These procedures do not require special-purpose, processor-replacement software in TSPS No. 1.

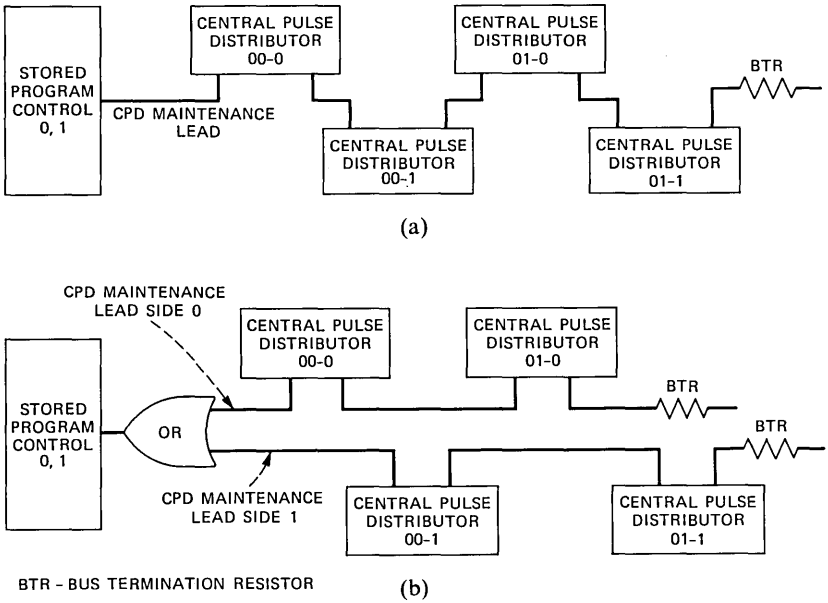



Fig. 1—(a) Simplex configuration for CPD maintenance lead. (b) Duplicated CPD maintenance lead.

### *2.1 Modifications of peripheral buses*

The TSPS No. 1 peripheral buses are modified to be compatible with the SPC 1B and to simplify introducing the SPC 1B. During this bus modification, call processing continues uninterrupted.

The Central Pulse Distributor (CPD) maintenance leads must be split to allow each of the 3B20D/PSI complexes access to the entire set of leads. At present the CPDs are daisy-chained along these complex CPD maintenance leads (see Fig. 1a*). These leads are duplicated by separating them and then feeding the EXCLUSIVE-OR of the two sets back to the processor (see Fig. 1b).

A set of miscellaneous leads are bundled into miscellaneous buses and are rerouted to the Communication Bus Translator (CBT). Each miscellaneous bus has a network reset lead, an auxiliary reset lead, a maintenance scanner lead, and three clock leads. The rerouting enables the clock leads to be disconnected from the SPC 1A when it is removed from the office.

Most TSPS No. 1 peripheral buses must be connectorized to allow insertion of the SPC 1B and the removal of the SPC 1A (see Fig. 2).

---

* Acronyms and abbreviations used in the figures and text of this paper are defined at the back of this Journal.

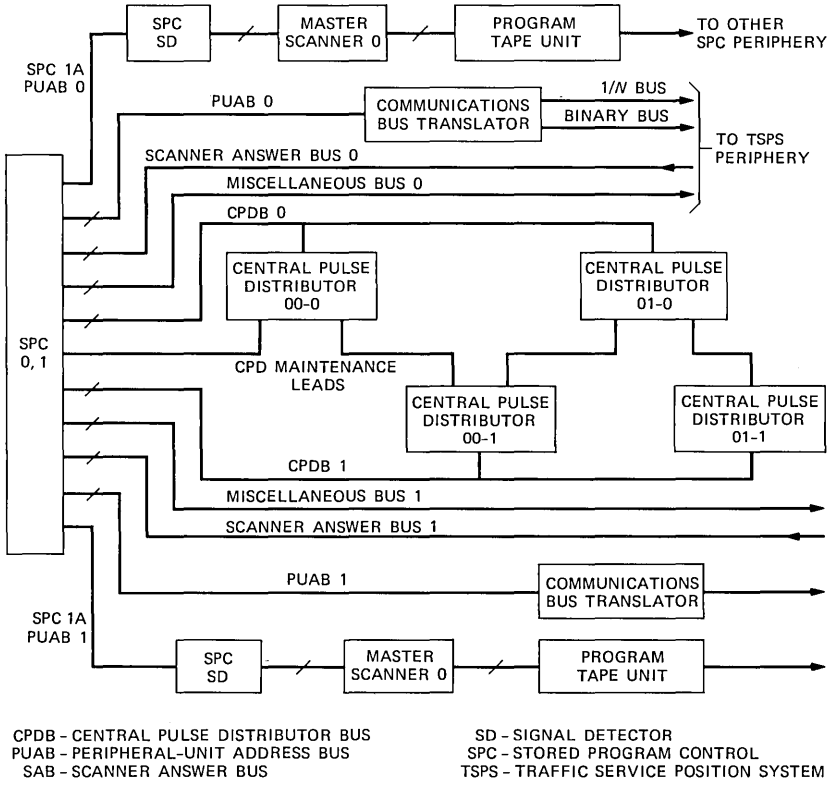



Fig. 2—TSPS No. 1 peripheral buses.

This is the most time-consuming activity of the processor replacement activities since 346 pairs of wires are involved. Connectorization is transparent to the SPC 1A operation. It requires a transition panel unit to be installed on site. The transition panel provides bus continuity for each pair of connectors that result. The panel is normally mounted on top of a cable rack near the location of the PSI frames or in an empty frame adjacent to the PSIs. The connectorization procedure is first run on one bus and then repeated for the mate bus.

In addition to the buses for the 3B20D Processor being connectorized, the SPC 1A Peripheral-Unit Address Bus (PUAB) and the one-out-of-$N$ (1/$N$) bus must be connectorized to accommodate the relocation of Master Scanner 0 (MS0).

### *2.2 Relocation of auxiliary units*

The SPC 1A Signal Distributor (SD), the Program Tape Unit (PTU), and MS0 reside on the SPC 1A PUAB. This bus is not utilized by the SPC 1B because the SD and PTU are replaced by 3B20D units.

However, MS0 must remain since it monitors TSPS units. This is resolved by moving the appearance of MS0 from the SPC 1A PUAB to the $1/N$ bus and by indicating the move in office data.

## III. RETROFIT PROCEDURES

After the site-preparation activities are completed, the actual retrofit procedures can begin. Two special software packages for the SPC 1A and the SPC 1B, plus special-purpose circuitry, are used to assist the retrofit.

### *3.1 Loading the SPC 1A retrofit programs*

Special retrofit software is loaded into the SPC 1A. It provides the capabilities for the initialization of TSPS periphery for 3B20D access tests, the bimodal feature of Station Signaling and Announcement Subsystem (SSAS) (see Section 3.2), and the ability to time-share buses with the 3B20D. The SPC 1A retrofit load overlays recent change programs and can be backed out whenever retrofit testing is at one of many safe stop points. Thus, the operating telephone company can have access to recent change capabilities when processor replacement programs are not in use.

### *3.2 Modification of bus access to SSAS announcement stores*

Processor replacement also requires the modification of the bus access to the SSAS announcement stores.[5] The Announcement Stores (AST) reside on the SPC 1A store bus (see Fig. 3). Since the store bus is not retained for the SPC 1B, the only access to the ASTs for loading announcements and for maintenance is through the SSAS controller after the SPC 1B retrofit. However, during the retrofit interval, the store bus access to the ASTs is left intact but not used. This allows emergency loading of ASTs if mate failures occur. The modification of bus access to the SSAS announcement stores is accomplished one SSAS side at a time. Therefore, for a short period, one SSAS side's AST is accessed via the SPC 1A store bus while the other SSAS side's AST is accessed through the SSAS controller. To handle this unique access situation, special-purpose bimodal software included in the SPC 1A retrofit program must remain loaded. This software ensures SSAS integrity during the bus modifications. As part of the modification the firmware in the SSAS controller is upgraded. This new firmware is designed to interface with both the SPC 1A and SPC 1B.

After the SSAS bus access modification is complete, the system is ready to have the new SPC 1B introduced on the buses. Figure 4 shows the site configuration.

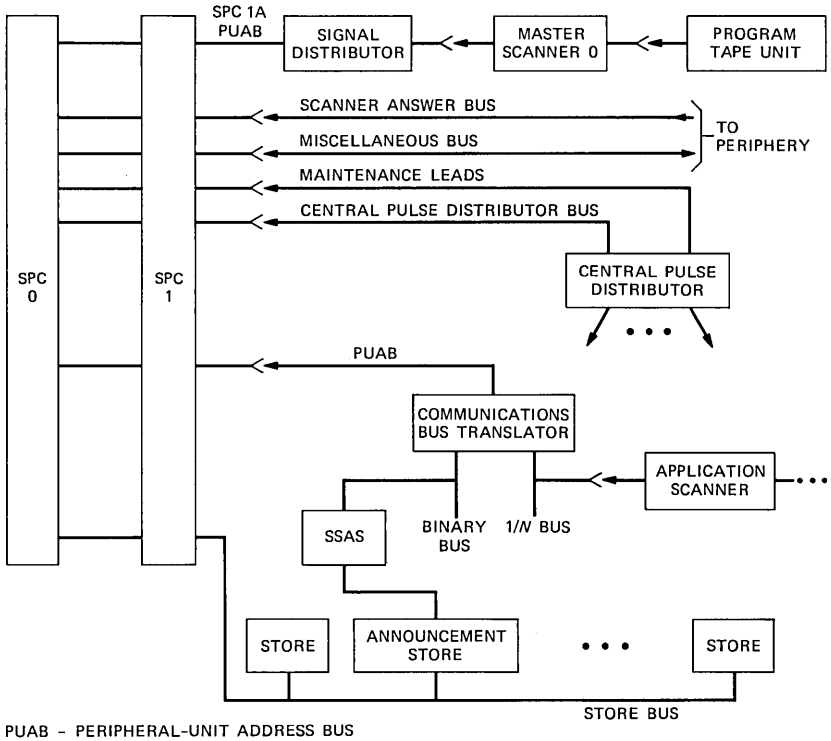



Fig. 3—Connectorized TSPS buses.

### *3.3 SPC 1B installation and testing*

In parallel with or subsequent to site preparation, the SPC 1B is installed. First, the 3B20D Processor is brought up in a stand-alone configuration by Western Electric according to standard procedures using the Duplex Multi-Environment Real Time (DMERT) operating system[6] and 3B20D diagnostics. Next, the PSIs are installed and connected to both halves of the 3B20D. Standard installation procedures are used to check out the connection. These procedures use the PSI diagnostic phases that test all internal PSI circuitry up to the bus drivers.

### *3.4 Insertion of the SPC 1B*

The SPC 1B is now ready to be inserted onto the TSPS peripheral buses. This involves removing the bus connectors on top of the transition panel and then inserting them into the inductor/transformer area of the PSI frame. As a result, the PSI not only has access to the TSPS peripheral buses, but also gives bus continuity when idled or

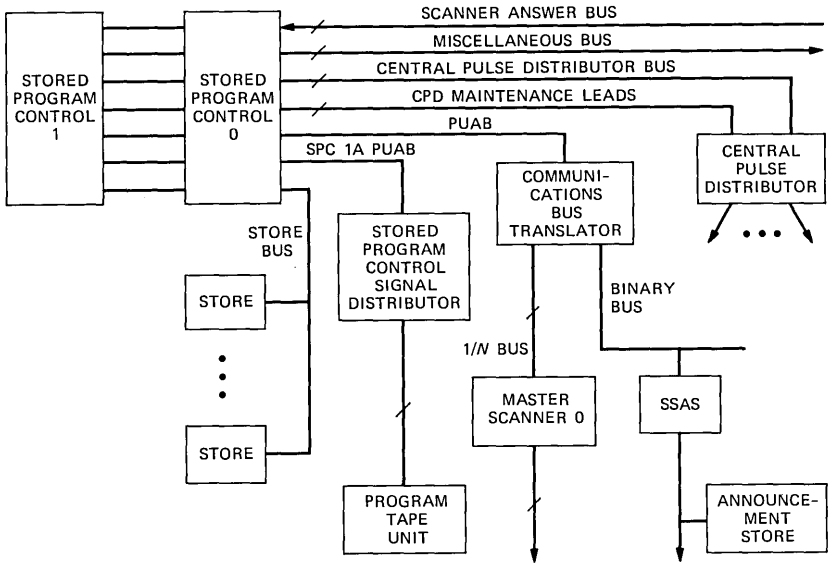



Fig. 4—Site preparation.

powered down. The retrofit software ensures that the SPC 1B complex never accesses the buses at the same time that SPC 1A does. The SPC 1A and SPC 1B operate in a time-sharing mode.

The insertion procedure connects one bus at a time utilizing existing TSPS diagnostics to verify correct movement. During the insertion the PSI access to the miscellaneous bus (see Section 2.1) is disabled to ensure that there is only one processor sending out signals over the clock and reset leads at any one time. Not until cutover does the SPC 1B have control of these leads.

### *3.5 SPC 1A/3B20D interface hardware and software*

Some additional special retrofit hardware and software is required to provide synchronization between the SPC 1A and the 3B20D for bus time-sharing. Every 25 ms the SPC 1A allows the 3B20D complete access to the TSPS peripheral buses for a period of 2 ms. It does this by sending the 3B20D a "start" (interrupt) signal. During the 2-ms interval, the SPC 1A keeps itself cycling. At the end of 2 ms, the SPC 1A sends the 3B20D a "stop" signal by disabling the PSI and then regains control of the buses. The SPC 1A sends out the "start" and "stop" signals via the unused CPD Execute leads in the office (see Fig. 5).

The SPC 1A retrofit software controls the bus time-sharing through

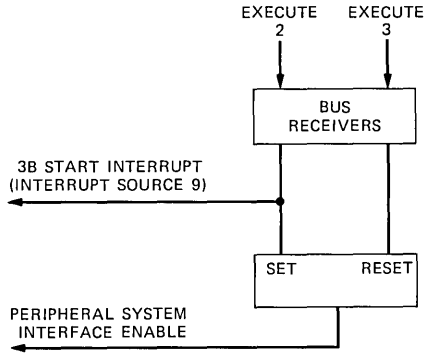



Fig. 5—Start and stop signals from SPC 1A.

the use of a base-level teletypewriter (TTY) message-handler program and a J-level program with a 25-ms entry rate. For a particular retrofit test, the SPC 1A software checks for the proper system configuration. If the system is not configured correctly, the retrofit test request is rejected; otherwise, the SPC 1A starts sending the proper start and stop pulses to the SPC 1B. Every 25 ms the SPC 1A sends an interrupt to the 3B20D ("go-ahead" signal) and, at the same time, an enable to the appropriate PSI frame. After the SPC 1A has cycled for 2 ms, it sends a disable to the PSI frame; the SPC 1A now has access to the peripheral buses for approximately 23 ms, and the cycle then repeats itself. Hence, the 3B20D must run its retrofit tests in 2-ms intervals so that it does not exceed its allotted time on the buses.

If for any reason the SPC 1A encounters a hardware interrupt during retrofit testing, it automatically stops sending interrupts to the 3B20D and initializes the SPC 1A retrofit software before exiting. The system must be restored to normal and the problem cleared before retrofit testing can continue.

The SPC 1B retrofit software has three major capabilities: an input message handler, an output message handler, and the retrofit test and control software. The input message handler, which is a user process, interprets the input message typed on the maintenance terminal. It then sends a message to the retrofit test and control software at the kernel level, which executes the requested test. Only one retrofit test can be run at a time. Most tests run in their entirety during the 2-ms interval. The emulated diagnostics, however, must run a section (<2 ms) at a time until completion. The retrofit control software ensures that this takes place. Finally, the output message handler (a user process) is used to print out any raw data words generated by the emulated diagnostics. This printing occurs on the maintenance terminal after the kernel process sends a message to the output message handler.

### *3.6 Testing of the 3B20Ds access to TSPS periphery*

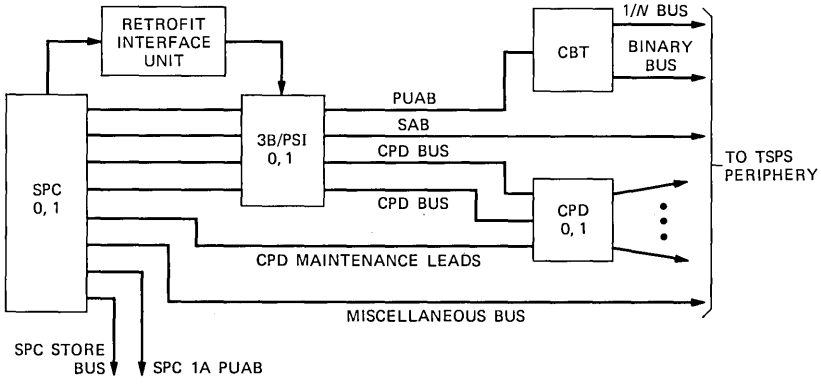
A special set of retrofit tests and procedures have been developed to verify that the processor has been replaced correctly. These procedures and tests verify the ability of the SPC 1B to drive the TSPS peripheral buses, and check that correct connections are made between the SPC 1B and TSPS peripherals. The system configuration during this testing is shown in Fig. 6. The retrofit tests include:

(*i*) SPC 1A/3B20D Interface Test—The interface test verifies numerous capabilities needed for subsequent tests. The SPC 1A must be sending start and stop signals to the SPC 1B complex, the time-share cable containing the CPD execute leads must be wired properly, and the hardware interface must be properly converting the signals from the SPC into a 2-ms enable pulse for the PSI.

(*ii*) CPD Scope Test—The CPD scope test verifies the proper bus connection and the system's ability to drive each lead. It involves having the SPC 1B send specific patterns of pulses equal to the lead number over the CPD leads and verifying them.

(*iii*) Communications Bus Translator (CBT) Scope Test—The CBT scope test verifies the proper connectorization and driving of the CBT leads. It involves having the SPC 1B send specific patterns of pulses equal to the lead number over the PUAB to the CBT and verifying them.

(*iv*) Central Pulse Distributor (CPD) Diagnostic—The CPD diagnostic is divided into two sections. One section runs with the CPDs inhibited. This section uses diagnostic circuitry in the CPDs to verify CPD bus input and appropriate responses. The other section runs with




CBT - COMMUNICATIONS BUS TRANSLATOR
CPD - CENTRAL PULSE DISTRIBUTOR
PSI - PERIPHERAL SYSTEM INTERFACE
PUAB - PERIPHERAL-UNIT ADDRESS BUS
SAB - SCANNER ANSWER BUS
SPC - STORED PROGRAM CONTROL

Fig. 6—Retrofit testing configuration.

the CPDs uninhibited. It checks the CPD operation by sending actual orders. This test marks the first time during retrofit testing that an actual CPD enable is sent over the buses. If either section fails, it prints out raw data words identical to the printout of the CPD diagnostic with the TSPS No. 1 generic.

(*v*) Station Signaling and Announcement Subsystem (SSAS) Loop-Around Test—The SSAS loop-around test consists of selected phases of the SSAS diagnostic. The Station Signaling and Announcement Subsystem (SSAS) receives a predetermined pattern via the CBT and returns the pattern on the Scan Answer Bus (SAB). The SPC 1B then checks to see if the loop-around was successful. This test verifies the integrity of CBT and SAB, and the integrity of all CPD and CBT bus routes to SSAS.

(*vi*) Miscellaneous Tests—The miscellaneous tests are required to check the remaining peripheral bus leads. These tests are run in two modes: one with the SPC 1A sending out pulses over these leads, the other with the 3B20D sending pulses. The SPC 1A-driven test checks the correct connections for the miscellaneous leads to the SPC 1B complex, while the 3B20D-driven test verifies its ability to send out proper pulses over these leads.

### *3.7 Mate SPC 1B complex*

Retrofit testing must be successful on both halves of the SPC 1B's (side 0 and side 1) before cutover. Therefore, all of the retrofit procedures that are run from one side must be executed from the mate side. This requires installing the SPC 1A/3B20D hardware interface on the mate SPC 1B complex. Once the hardware interface has been installed, retrofit testing can be performed from the mate SPC 1B.

## IV. OFFICE DATA TRANSFER AND RETROFIT

After the ability of the SPC 1B to interface with the buses is verified, the office data from the SPC 1A must be transferred to the SPC 1B. An on-line method has been developed to do this. Special wiring is installed between the SPC 1B and the teletypewriter buffer of the SPC 1A. This wiring allows the 3B20D to be viewed as a teletypewriter from the SPC 1A. After this connection is made, the 3B20D sends memory-read commands to the SPC 1A. The results of the reads are checked for correct parity and are reread when errors are found. The rereading ensures correctness. These reads continue until the entire office data spectrum is transferred.

After the office data have been transferred to the 3B20D, a special office data retrofit program is run. This program performs the required changes in the data structures and contents to make the data compat-

ible with TSPS No. 1B. It does not introduce new office-dependent data or modify any assignments specified in office-dependent data.

## V. CUTOVER PROCEDURES

Once all retrofit tests are successfully completed and the office data is transferred, the retrofit hardware interface is removed and the office data are merged with the official TSPS No. 1B generic software. To begin the actual cutover, the two miscellaneous bus connectors, which include the system clock leads, are connected to the inactive SPC 1B. The clock in the SPC 1A is stopped and the SPC 1B is initialized. Call processing is interrupted only from the time the SPC 1A clock is stopped until the resulting SPC 1B initialization is complete, usually less than 2 minutes. At this point, the 3B20D processors should be handling the TSPS call load. If any difficulties are encountered at this time, the SPC 1A configuration can still be brought up by powering down the PSI frames, restarting the SPC 1A clocks, and initializing the SPC 1A.

After cutover, a series of system diagnostics and verification tests are performed. In addition, a recommended soak interval of a few days is allowed to ensure system operation. Once the operating telephone company has determined that the TSPS No. 1B is performing satisfactorily, unneeded equipment can be removed by Western Electric. This removal requires no additional software or hardware capabilities.

## VI. SUMMARY

The first live retrofit was performed on March 13, 1982, in Redwood City, California. No major difficulties were encountered in this retrofit. During the remainder of 1982, 34 additional retrofits occurred.

## VII. ACKNOWLEDGMENTS

A number of people made contributions to the development and design of these retrofit procedures. Significant contributions were made by Carl Amodio, Ken Handy, Frank Maesky, and D. A. Wood of Western Electric and by Mark Koehler, Kevin Kulhanek, Ron Michelsen, J. D. Peterson, Dennis Shank, R. A. Tengelsen, Neil Walgenbach, Stan Windes, and several others of Bell Laboratories.

## REFERENCES

1. R. J. Jaeger, Jr., and A. E. Joel, Jr., "TSPS No. 1: System Organization and Objectives," B.S.T.J., *49,* No. 3 (December 1970), pp. 2417–43.
2. G. R. Durney, H. W. Kettler, E. M. Prell, G. Riddell, and W. B. Rohn, "TSPS No. 1: Stored Program Control No. 1A," B.S.T.J., *49,* No. 3 (December 1970), pp. 2445–508.
3. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.

4. G. T. Clark, H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
5. G. T. Clark, K. Streisand, and D. H. Larson, "TSPS No. 1: Station Signaling and Announcement Subsystem: Hardware for Automated Coin Toll Service," B.S.T.J., *58,* No. 6 (July-August 1979), pp. 1225–49.
6. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Capacity and Reliability Evaluation

By B. A. CRANE and D. S. SUK

(Manuscript received June 30, 1982)

*This paper describes the prediction and evaluation of the call-processing capacity and reliability resulting from use of the 3B20 Duplex (3B20D) Processor in the Traffic Service Position System No. 1B (TSPS No. 1B). The call-processing capacity was predicted using a processor real-time model whose parameter values were determined by laboratory and test-site measurements. The system reliability was predicted using Markov modeling techniques. Performing an evaluation during TSPS No. 1B development provided a means for monitoring progress toward meeting the capacity and reliability objectives.*

## I. INTRODUCTION

One of the development objectives of the Traffic Service Position System No. 1B (TSPS No. 1B) was to improve the call-handling capacity of the TSPS No. 1 by replacing the Stored Program Control No. 1A (SPC 1A) with the Stored Program Control No. 1B (SPC 1B).[1] The SPC 1B consists of the 3B20 Duplex (3B20D) Processor together with the Peripheral System Interface (PSI) unit, which adapts the 3B20D to existing TSPS peripherals. The 3B20D is microprogrammed to execute the SPC 1A instructions, thus allowing TSPS call-processing software developed for the SPC 1A to be ported to the SPC 1B with minimal changes. This emulated TSPS software executes as a kernel process under the DMERT operating system. References 2 and 3 contain further details on the SPC 1B architecture.

### *1.1 Call-processing capacity analysis*

The increased speed of the SPC 1B in executing the emulated SPC 1A instructions provides the increase in call-processing capacity. The

initial objective for capacity increase established for TSPS No. 1B was that the SPC 1B call-processing capacity should be at least 160 percent that of the SPC 1A.

Early in TSPS No. 1B development, a capacity prediction and evaluation plan was established for monitoring the progress in meeting the capacity improvement objective. This plan involved formulating a mathematical model of the SPC 1B real-time usage, where the parameters of this model represent the various call-processing and overhead activities performed by the processor. Laboratory and test-site measurements of these parameters during development provided, through use of the real-time model, estimates of the call-processing capacity. In this way, any problem areas having an adverse effect on call-processing capacity could be identified as candidates for improvement during continued development. This same real-time model, at the completion of development, has been incorporated into the TSPSCAP program[4] used by the operating telephone companies to determine the call-processing capacity of specific TSPS No. 1B sites. The formulation of this real-time model, the laboratory and test-site measurement techniques, and the resulting capacity performance data are described in subsequent sections of this paper.

### *1.2 System reliability analysis*

An important step in the development of highly reliable switching systems is the prediction of their reliability. To provide uninterrupted service, TSPS No. 1B has the same reliability objectives as other Bell System electronic switching systems (ESSs), namely: an average downtime of less than 3.0 minutes per year.[1] In TSPS No. 1B, most of the TSPS peripherals are retained and their maintenance strategy remains virtually unchanged from the TSPS No. 1. Thus, the reliability objectives of the TSPS peripherals will not change in TSPS No. 1B from 1.0 minute per year average downtime and, consequently, the SPC 1B reliability must achieve the objective of less than 2.0 minutes per year average downtime.

To predict the reliability of SPC 1B hardware, continuous-time, finite-state Markov models were used. The Markov model approach for the reliability calculation of repairable systems is described in Ref. 5. Throughout the development period of TSPS No. 1B, the reliability model was updated to accurately reflect architectural modifications or design changes in the subsystems. The reliability estimates of various configurations were compared to monitor the system reliability, to identify limiting subsystems, and to determine if modifications would improve the overall reliability. This will be described in subsequent sections of this paper.

## II. CAPACITY EVALUATION

### 2.1 Approach taken

The approach taken to model the real-time usage of the SPC 1B was to modify the real-time model of the SPC 1A.[6] Emulation of the SPC 1A code by the SPC 1B made this approach possible. The modified model contains parameters that represent the speedup in SPC 1B instruction execution relative to the SPC 1A and, also, the effects of the DMERT operating system.

These modifications were characterized by making measurements of real-time usage at various call loads ranging from idle to over 160 percent of the SPC 1A capacity. Call loads were applied through use of electronic, programmable call generators attached to TSPS trunks. The response of TSPS operators to these calls was simulated by other electronic, programmable units. The measurements of real-time usage were made by non-interfering monitoring equipment, which sampled and recorded the system execution state every 10 microseconds. Other measurements consisted of various TSPS traffic counts periodically printed out on the standard output devices.

### 2.2 TSPS No. 1 capacity analysis

Because the SPC 1A real-time model forms the basis for the SPC 1B real-time model, it is briefly described here. References 6, 7, and 8 should be consulted for greater detail.

#### 2.2.1 SPC 1A software architecture

During normal operation, most of the real-time usage of the SPC 1A occurs at two priority levels, called J-level and base level. J-level has the higher priority of the two, and is entered every 5 ms through a hardware interrupt to perform necessary input/output operations involved in communicating with the TSPS peripherals. Although a higher-priority H-level is also involved in these operations, H-level and J-level will hereafter be jointly referred to as J-level except where distinction is necessary. In the SPC 1A, base-level work has the lowest system priority and is performed whenever there are no higher-priority interrupts. Most of the call-processing work is performed in base level.

Each base-level program is assigned to one of five classes of work: A, B, C, D, or E. Each class is periodically visited by a control program to determine whether there is any work to do and to perform the work if present. The control program endlessly repeats the following fixed visitation sequence:

$$\cdots \text{ABACABADABACABABACABADABACABAE} \cdots .$$

We can see that from one class-E visitation to the next, termed an E-E cycle, the five classes are visited according to the ratio

$$A:B:C:D:E = 15:8:4:2:1.$$

Base-level programs are assigned to these classes in accordance with the acceptable delays in their execution; class A contains those programs requiring fastest response.

The time duration of an E-E cycle increases with the call load because, as the call load increases, there is more work to be done during each class visitation. However, a fixed amount of base-level work must be performed no matter what the call load is (e.g., determining if there is any work to do) and this work is referred to as the E-E cycle overhead.

#### 2.2.2 SPC 1A real-time model

The real-time model developed for the SPC 1A consists of the equation

$$900 = t_N N + T_{CR} + t_E E. \quad (1)$$

This equation expresses how a quarter-hour (900 seconds) of processor real time is shared by three different kinds of processor work: trunk-seizure work, represented by $t_N N$; constant-rate work, represented by $T_{CR}$; and E-E cycle overhead work, represented by $t_E E$. Each of these three terms is expressed in seconds per quarter-hour.

A TSPS call begins as a seizure (request for service) of a special TSPS trunk from a local office to a toll office. Most trunk seizures result in completed TSPS calls, but a few become uncompleted attempts because of customer abandonments, busy circuits, etc. Although these uncompleted attempts do not require as much processor real time as completed calls, they must be included as part of the processor real-time load. In the real-time equation, $N$ represents the number of trunk seizures per quarter-hour; and $t_N$ represents the average amount of processor real time (in seconds) required per trunk seizure. The value of $t_N$ depends on the mix of various types of completed TSPS calls and uncompleted attempts. About two-thirds of $t_N$ occurs in TSPS base level, and the other third in J-level. The TSPS call-processing capacity is expressed in terms of trunk seizures per quarter hour.

Constant-rate work is the processor work that is performed at fixed time intervals and is independent of trunk-seizure rate. For example, one type of TSPS trunk is scanned every 100 ms to determine whether a trunk seizure has occurred. The value of $T_{CR}$, in seconds per quarter hour, depends on the number of TSPS peripherals in use, and most of this time is spent in J-level.

The E-E cycle overhead work uses all processor real time not used by trunk seizures or constant-rate work. $E$ represents the number of E-E cycles that are executed per quarter hour, and $t_E$ represents the

average processor real time (in seconds) spent per E-E cycle in doing E-E cycle overhead work, which is independent of trunk-seizure load. By definition, all of this time occurs in base level.

Equation (1) is linear in terms of $E$ and $N$. Figure 1 plots $E$ as a function of $N$ for a typical SPC 1A site. Such a plot is referred to as a load line, which describes how the E-E cycle rate, $E$, varies with respect to the trunk-seizure rate, $N$. The slope of this load line is $-t_N/t_E$ and the intercept, corresponding to an idle system (i.e., when $N = 0$), is $(900 - T_{CR})/t_E$.

Also shown in Fig. 1 is a value of $E$, called $E_{MIN}$, which is the lowest E-E rate that can be sustained while still providing adequate system response. At rates below $E_{MIN}$ the visitation rate to the previously described base-level classes of work becomes too low and delays in serving requests become too long to meet service criteria. The trunk-seizure rate corresponding to $E_{MIN}$ is defined as the quarter-hour trunk-seizure capacity, $N_{CAP}$.

### 2.2.3 TSPSCAP program

TSPSCAP is an interactive, time-shared program used by the operating companies to determine the trunk-seizure capacity of specific TSPS sites. The user inputs the call mix and hardware configuration of a site, and TSPSCAP calculates the values of $t_N$, $T_{CR}$, and $E_{MIN}$ corresponding to these input values for use with the above real-time equation. TSPSCAP then outputs the value of $N_{CAP}$ for that site, together with auxiliary information such as an equation for the $E$

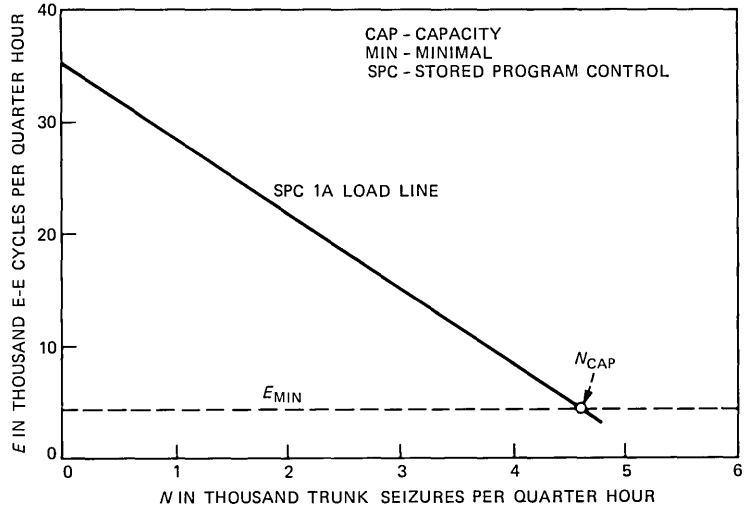



Fig. 1—Typical SPC 1A load line.

versus $N$ load line. This information is used by the operating telephone companies in growth planning to determine how close a TSPS No. 1 site is to its capacity limit.

### 2.3 TSPS No. 1B capacity analysis

As we mentioned earlier, emulation of the TSPS No. 1 software allows construction of the SPC 1B real-time model by modifying the SPC 1A real-time model. The modifications represent the speedup in instruction execution and the effects of the DMERT operating system. To understand these modifications the reader should know how the emulated code executes in the SPC 1B environment. This is briefly described below; Ref. 3 should be consulted for a more complete description.

#### 2.3.1 SPC 1B software architecture

**2.3.1.1 System execution levels.** The DMERT operating system has sixteen execution levels (ELs), numbered 0 through 15, that determine the relative priorities for process execution; EL 15 has the highest priority. Kernel processes can use ELs 15 through 2, and supervisor/user processes are restricted to ELs 1 and 0. The emulated TSPS call-processing software executes as a kernel process.

Table I shows the ELs for those processes that influence the SPC 1B real-time usage. H-level and high-priority J-level of the emulated TSPS process execute at EL 12, and low-priority J-level executes at EL 11. Base level executes at EL 5. The DMERT timer, at EL 15, provides a timing function for other processes by notifying a requesting process after a specified time period has elapsed. Processes involved with I/O, file management, and memory management execute at ELs 10, 7, and 2, respectively. The scheduler at EL 2 schedules the supervisor/user processes at ELs 1 and 0. Diagnostics for the 3B20D and PSI execute at EL 0, whereas diagnostics for the TSPS peripherals remain as part of class-E work in emulated TSPS base level at EL 5. The new TSPS craft interface software, which uses DMERT facilities to provide maintenance input-output message capability and system

Table I—DMERT execution levels

| Execution Level | Process |
|---|---|
| 15 | Timer |
| 12, 11 | TSPS J-Level |
| 10 | Disk Driver |
| 7 | File Manager |
| 5 | TSPS Base-Level |
| 2 | Memory Manager, Scheduler |
| 1, 0 | SPC 1B Diagnostics, TSPS Craft Interface, Other Supervisor/User Processes |

status display, also executes at EL 0. Software for the other TSPS output messages (e.g., for periodic traffic counts) remains as part of the emulated J-level and base-level code.

A 3B20/DMERT timer hardware interrupt occurs every 10 ms to service any timing requests. As with the SPC 1A, a J-level interrupt occurs every 5 ms. These two interrupts are synchronized such that the J-level interrupts lead the timer interrupts by 1 ms. As with the SPC 1A, the emulated base level executes whenever nothing at a higher EL is executing, with the exception that base level periodically relinquishes control (goes to sleep) so as to allow processes at lower ELs to execute.

Just before going to sleep, base level requests that the DMERT timer wake it after a specified period has elapsed. The low-level processes at ELs 4 through 0 can then execute, subject to interrupts by processes at higher ELs. However, if all the low-level processes complete their work before the timer awakens base level, then base level is prematurely awakened by a software interrupt and the pending timer request is deactivated. Thus, any real time not needed by the low-level processes is given back to base level, which uses this real time to execute additional E-E cycles.

***2.3.1.2 Speedup factors.*** The increased speed of the SPC 1B causes a net speedup in the execution of the emulated TSPS process relative to the SPC 1A. Not all portions of the emulated code experience the same degree of speedup, however, because of dependence on dynamic instruction mix, cache hit ratio, and ATB hit ratio.

The dependence on dynamic instruction mix occurs because some SPC 1A instructions could be emulated more efficiently than others. Also, in the SPC 1B, the execution time of some emulated SPC 1A instructions depends on what instruction options (e.g., rotating and masking) are exercised, whereas no such dependency exists in the SPC 1A.

To reduce memory access time, the SPC 1B employs a cache memory to contain the most recently accessed words of main memory. The cache is searched prior to each memory access and, if the word is in the cache (i.e., a cache hit), less real time is used because main memory need not be accessed. The cache is shared in common by all processes in the system.

The SPC 1B also employs eight Address Translation Buffers (ATBs), which speed up the task of translating from virtual memory address to physical memory address. Each ATB is essentially a cache memory that contains the physical addresses of the most recently accessed pages of virtual memory assigned to that ATB (a page is a 512-word block of main memory). If the page address is not in the ATB (i.e., an ATB miss), extra time is used in translation, which can

increase an instruction's execution time. To reduce ATB misses, J-level is exclusively assigned to one ATB and base level is exclusively assigned to another.

Parameters called "speedup factors" have been introduced to characterize the increased speed of the SPC 1B in executing the emulated TSPS process. Because of the effects of cache and ATB hits, speedup factors apply to execution of portions of code rather than to individual instructions. Thus, the speedup factor for a given portion of emulated code depends on the mix of executed instructions, and on the cache and ATB hit ratios experienced by those instructions.

*2.3.1.3 DMERT operating system.* For TSPS No. 1B, some real-time requirements of the DMERT operating system are application independent and others are application dependent. The application-independent requirements are for those functions that are necessary for maintaining a stable system environment. For example, the real time allocated to diagnose the 3B20D Processor would fall into this category. The application-dependent requirements are for those TSPS functions that make use of DMERT-supplied facilities. Two examples are: the real time required by the new TSPS craft interface, and the real time required to interface DMERT to TSPS J-level.

Parameters have been introduced that represent the combined TSPS-independent and TSPS-dependent DMERT real-time requirements for TSPS No. 1B. One parameter represents the combined high-level requirements (at ELs 15 through 5), and a second represents the combined low-level requirements (at ELs 4 through 0). Other parameters represent the real time used in handling TSPS J-level interrupts and base-level sleep requests.

### 2.3.2 SPC 1B real-time model

The SPC 1B real-time model is formed by adding speedup and operating system parameters to eq. (1) so as to obtain the new equation:

$$900 = t'_{\mathrm{N}}N' + T'_{\mathrm{CR}} + t'_{\mathrm{E}}\,E' + T'_{\mathrm{H}}, \tag{2}$$

where

$N'$ = trunk-seizures per quarter hour serviced by the SPC 1B
$E'$ = E-E cycles per quarter hour executed by the SPC 1B
$T'_{\mathrm{H}}$ = seconds per quarter hour used by high-level processes (at ELs 15 through 5) associated with TSPS-independent and TSPS-dependent DMERT work

and where $t'_{\mathrm{N}}$, $T'_{\mathrm{CR}}$, and $t'_{\mathrm{E}}$ are as defined in the following paragraphs.

The value of $t'_{\mathrm{N}}$, the average processor seconds per trunk seizure, is defined as

$$t'_N = \frac{t_{NB}}{K_{NB}} + \frac{t_{NJ}}{K_{NJ}}, \tag{3}$$

where

$t_{NB}$ = average processor seconds used per trunk seizure by the SPC 1A in base level

$K_{NB}$ = SPC 1B speedup factor for $t_{NB}$

$t_{NJ}$ = average processor seconds used per trunk seizure by the SPC 1A in J-level

$K_{NJ}$ = SPC 1B speedup factor for $t_{NJ}$.

Separate base-level and J-level speedup factors are defined because base-level and J-level each has its own dynamic instruction mix and, also, its own ATB and associated ATB hit ratio.

The value of $T'_{CR}$, the processor seconds per quarter hour of constant-rate work, is defined as

$$T'_{CR} = \frac{T_{CRB}}{K_{CRB}} + \frac{T_{CRJ}}{K_{CRJ}} + 18 \times 10^4 \, t_{DJ}, \tag{4}$$

where

$T_{CRB}$ = processor seconds per quarter hour used in constant-rate work by the SPC 1A in base level

$K_{CRB}$ = SPC 1B speedup factor for $T_{CRB}$

$T_{CRJ}$ = processor seconds per quarter hour used in constant-rate work by the SPC 1A in J-level

$K_{CRJ}$ = SPC 1B speedup factor for $T_{CRJ}$

$t_{DJ}$ = processor seconds used by the SPC 1B in handling each J-level interrupt.

Separate base-level and J-level speedup factors are defined for the same reasons stated above.

The value of $t'_E$, the processor seconds per E-E cycle to perform E-E cycle overhead work in base level, is defined as

$$t'_E = \frac{t_E}{K_E} + b(t_{DB} + \bar{s}A), \tag{5}$$

where

$t_E$ = processor seconds used per E-E cycle by the SPC 1A in performing E-E cycle overhead work in base level

$K_E$ = SPC 1B speedup factor for $t_E$

$b$ = number of base-level sleep periods executed per E-E cycle by the SPC 1B

$t_{DB}$ = processor seconds used by the SPC 1B in handling each base-level sleep-period request

$\bar{s}$ = average duration (in seconds) of each base-level sleep period

$A$ = average fraction of each base-level sleep period that is available to low-level processes (at ELs 4 through 0).

In this formulation it can be seen that the real time used by the low-level processes is treated as part of the SPC 1B E-E cycle overhead. The value of $A$ decreases as high-level interrupts increase and, therefore, $A$ decreases as call load increases.

The values of $b$ and $\bar{s}$ must satisfy the constraint

$$b\bar{s}AE^* = T'_L, \tag{6}$$

where

$T'_L$ = seconds per quarter hour to be allocated to low-level processes (at ELs 4 through 0) associated with TSPS-independent and TSPS-dependent DMERT work

$E^*$ = lowest SPC 1B E-E cycle rate at which $T'_L$ is to be allocated by base-level sleep periods.

At E-E cycle rates less than $E^*$, insufficient base-level sleep periods will occur to satisfy eq. (6). At E-E cycle rates higher than $E^*$, more than $T'_L$ can be used by low-level work if necessary.

The value of $b$ is a software parameter, and the value of $\bar{s}$ is determined by the value of $s$, which is another software parameter. When base level goes to sleep, it requests that it be awakened after $s$ milliseconds have elapsed. Because this request can be made at any time relative to the 10-ms DMERT timer interrupt, $\bar{s}$ is around 5 ms longer than $s$.

Equations (2) through (6) constitute the SPC 1B real-time model.

#### 2.3.3 Determination of real-time model parameters

The newly introduced SPC 1B real-time parameters have been characterized through measurements made in the TSPS system laboratories and at the test site in Fresno, California, prior to cutover. The basic measurement technique involved measuring the percentage of processor real time used at each execution level under a number of different loads applied to the system. Other auxiliary measurements were also made.

***2.3.3.1 Real-time measurement techniques.*** Processor real-time usage at the sixteen execution levels was measured through use of *Dynaprobe** monitoring equipment manufactured by the NCR COMTEN Corporation. The *Dynaprobe*, through means of high-impedance probes attached to the SPC 1B backplane, was used to sample the execution-level bits of the Program Status Word (PSW) every 10 microseconds to determine the relative frequencies of execution-level

* Registered trademark of NCR COMTEN Corporation.

occupancies. Other signals were also sampled at the same time to: (*i*) distinguish between emulated and native-mode code; (*ii*) count the number of times that each emulated SPC 1A instruction was executed during the measurement period; and (*iii*) measure the hit ratios experienced by the cache and by the base-level and J-level ATBs. The raw counts for all these data were written onto magnetic tape for subsequent off-line analysis.

During parameter measurement, simulated calls were generated by means of MICLOB (Microprocessor Controlled Load Box) units attached to the TSPS trunks. The response of TSPS operators, for those simulated calls that required operator assistance, was simulated by MOPS (Microprocessor Operator Position Simulator) units. The MICLOB and MOPS units are described in Ref. 9. Complete parameter characterization required taking measurements under various system conditions. Call loads were varied from zero to the maximum applicable simulated load. Different degrees of low-level activity were obtained by running processor and memory diagnostics and by causing different rates of output messages to be generated by the craft interface.

***2.3.3.2 Measurement of speedup factors.*** Values for each of the defined speedup factors were calculated from measurements taken at the Fresno test site. Figure 2 shows the calculated values for each of the speedup factors plotted with respect to $E'$, the SPC 1B quarter-hour E-E cycle rate. The value of $E'$ is inversely related to call load; $E' = 58{,}000$ corresponds to an idle system and $E' = 10{,}000$ corresponds to the maximum applied call load.

Figure 2 shows that the two J-level speedup factors, $K_{NJ}$ and $K_{CRJ}$,

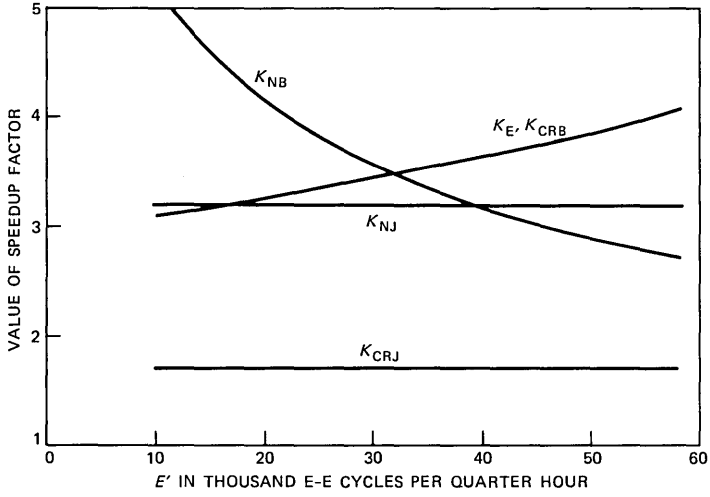



Fig. 2—Speedup factors.

are constant with respect to $E'$. The value of $K_{CRJ}$ is less than $K_{NJ}$ because J-level constant-rate work makes heavier use of the instructions that have relatively low-emulation efficiencies. Although the cache and ATB hit ratios associated with both of these speedup factors were observed to slightly increase with decreasing $E'$ (increasing call load), the effect of these changes was compensated for by a slight change in the dynamic instruction mix for $K_{NJ}$ and, for $K_{CRJ}$, a higher percentage of conditional transfers taken.

For base level, the three speedup factors, $K_E$, $K_{CRB}$, and $K_{NB}$ are seen to change with $E'$. The value of $K_E$, the speedup factor for E-E cycle overhead work, decreases with decreasing $E'$ (increasing call load) because of a marked decrease in the base-level cache and ATB hit ratios as $E'$ decreases. At zero call load, a relatively small portion of emulated code (the E-E cycle overhead work) is executed for a relatively high percentage of the time, causing the cache and ATB hit ratios to be at their highest values. The value of $K_{CRB}$, the speedup factor for base-level constant-rate work, was not measured directly but is set equal to $K_E$ because this type of work is quite similar to E-E cycle overhead work and because only a small percentage of real time (less than 2 percent) is involved.

The value of $K_{NB}$, the speedup factor for base-level trunk-seizure work, is seen to increase with decreasing $E'$ (increasing call load) even though the cache and ATB hit ratios are decreasing. This increase is caused by a decrease in the number of base-level instructions (excluding constant-rate and E-E overhead instructions) executed per trunk seizure as $E'$ decreases. Figure 3 shows this effect. Measured values of $I_{NB}$, the number of base-level instructions executed per trunk seizure, are plotted versus $E'$. The dependence of $I_{NB}$ is seen to be approximately linear with respect to $E'$ over a wide range of values.

Investigation has indicated that this effect is at least partly caused by queueing for busy facilities (e.g., digit receivers). During each E-E cycle, if a queue exists, an attempt is made to remove all entries from the queue. Those entries that cannot be removed remain for the next E-E cycle, thereby causing extra instructions to be executed. As the call load increases, the probability of queue formation also increases. The E-E cycle rate decreases, however, thereby producing a net decrease in the number of base-level instructions executed per trunk seizure. This effect also occurs with the SPC 1A, but to a lesser degree because, as will be seen, the E-E cycle rate of the SPC 1A is lower than that of the SPC 1B when both are operating at the same trunk-seizure rate.

Curves were fitted to the calculated values of the speedup factors shown in Fig. 2 to obtain expressions for the parameters used in the SPC 1B real-time model. These expressions are:

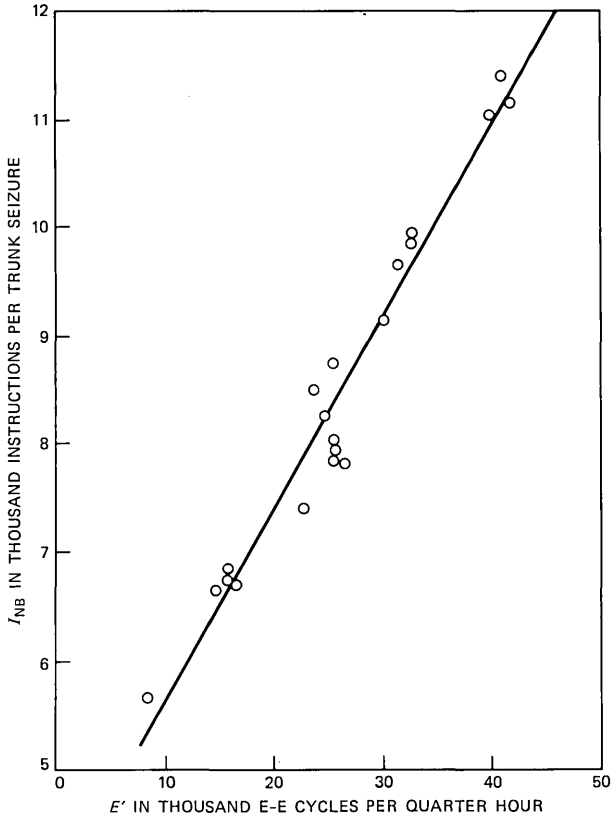



Fig. 3—Base-level instructions for each trunk seizure versus $E'$.

$$K_{CRJ} = 1.70;\ K_{NJ} = 3.20$$

$$K_{CRB} = K_E = \frac{8.2762}{(2.7929 - 13.104 \times 10^{-6}E')}$$

$$K_{NB} = \frac{20.698}{(1 + 45.884 \times 10^{-6}E')(2.8649 - 14.179 \times 10^{-6}E')},$$

where, as previously defined, $E'$ is the quarter-hour E-E cycle rate.

***2.3.3.3 Measurement of DMERT real-time requirements for TSPS No. 1B.*** Parameters representing the real-time requirements of DMERT for TSPS No. 1B combine both TSPS-independent and TSPS-dependent work. TSPS-independent DMERT work includes DMERT functional work (e.g., audits, timer, etc.) and maintenance work associated with the SPC 1B (e.g., 3B20D diagnostics). TSPS-dependent DMERT work includes the TSPS craft interface work and work associated with handling the TSPS J-level interrupts and base-level sleep-period requests. These parameters were characterized by

*Dynaprobe* measurements of the real time used at each execution level and under various system operating conditions.

The TSPS craft interface real time is primarily used in producing output messages for maintenance purposes, and is a function of message rate, message length, and the number of output devices in use. The message rate is, in turn, a function of call load. Characterization involved measuring the real-time cost on a per-character basis and analyzing output messages generated by SPC 1A sites to determine representative message rates and lengths.

Measurements of $T'_H$, the parameter combining TSPS-independent and TSPS-dependent real-time requirements for high-level DMERT work, yielded

$$T'_H = 45.2 + 10^{-3} N' \text{ s/QH},$$

where $N'$ is the TSPS No. 1B quarter-hour trunk-seizure rate. Measurements of $T'_L$, the parameter combining TSPS-independent and TSPS-dependent real-time requirements for low-level DMERT work, yielded

$$T'_L = 101.0 + 7.5 \times 10^{-3} N' \text{ s/QH}.$$

The value of $T'_L$ is the amount of real time that should be allocated to achieve satisfactory execution of low-level activities under worst-case conditions (e.g., high maintenance activity during call overload). Under normal conditions, the actual value of $T'_L$ is considerably less than this allocated value so that more real time is available to call processing.

To satisfy eq. (6), the values chosen for $b$, the number of base-level sleep periods per E-E cycle, and $s$, the requested duration of each sleep period, are

$$b = 5; s = 10 \text{ ms}.$$

*Dynaprobe* measurements also yielded

$$t_{DJ} = 84 \text{ microseconds}$$

for each J-level interrupt [see eq. (4)] and

$$t_{DB} = 1.5 \text{ ms}$$

for each base-level sleep request [see eq. (5)].

#### 2.3.4 Model evaluation

Figure 4 shows measured and predicted values of the quarter-hour E-E cycle rate, $E'$, plotted versus the quarter-hour trunk-seizure rate, $N'$, for the Fresno TSPS site. The SPC 1B real-time model was used to predict three different $E$ versus $N$ load lines, each corresponding to a different low-level activity rate. The separate load lines occur be-

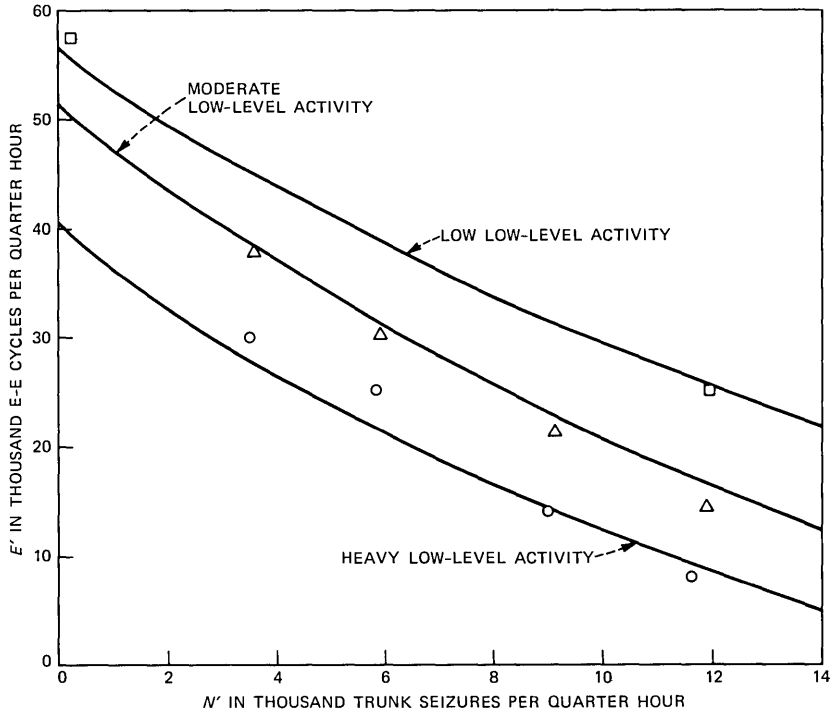



Fig. 4—SPC 1B load lines.

cause, as previously described, any real time not used by low-level processes is given back to base level, which uses this real time to execute additional E-E cycles.

The upper SPC 1B load line shows $E$ versus $N$ behavior when the low-level activity rate is low. Quarter-hour measurements were taken under these conditions at zero and the maximum applied load, and good agreement is seen between measured and predicted values. The middle SPC 1B load line corresponds to moderate low-level activity, and the lower SPC 1B load line corresponds to the condition when the low-level activity is heavy. Again, agreement between measured and predicted values is quite good.

### *2.3.5 SPC 1B capacity increase*

Figure 5 shows two $E$ versus $N$ load lines that indicate the increase in call-processing capacity provided by the SPC 1B. The upper load line depicts the $E$ versus $N$ behavior for the Fresno TSPS site as predicted by use of the SPC 1B real-time model for a typical low-level activity rate. The lower load line shows the $E$ versus $N$ behavior of the

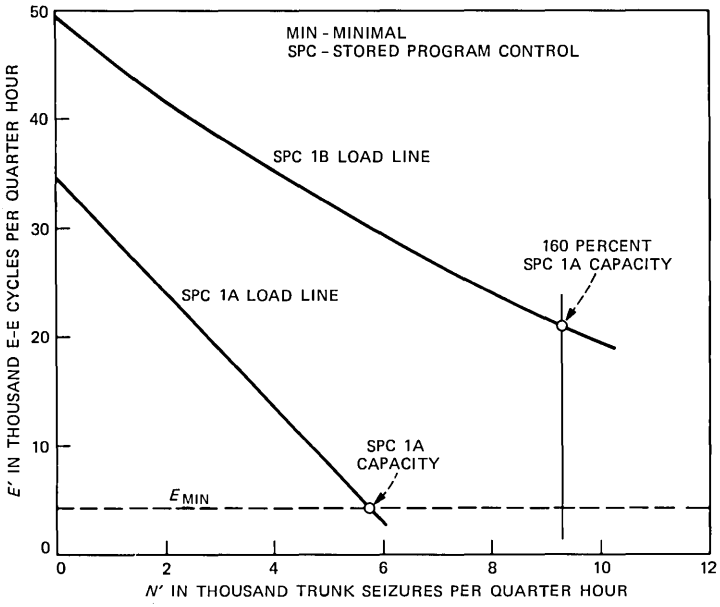



Fig. 5—Comparison between SPC 1A and SPC 1B load lines.

Fresno TSPS site as predicted using the SPC 1A real-time model, indicating how the site would perform if it were to use the SPC 1A. Both load lines assume the same call mix.

The SPC 1A load line shows that the SPC 1A would reach its capacity at about 5800 trunk seizures per quarter hour, since it is at that trunk-seizure rate that the E-E cycle rate equals 4180 E-Es per quarter hour, the SPC 1A value of $E_{MIN}$. Analysis and experiments conducted at Fresno indicate that $E'_{MIN}$ for the SPC 1B should be less than $E_{MIN}$. Therefore, since SPC 1B measurements were conducted at Fresno at around 9300 trunk seizures per quarter hour with good system performance (see Fig. 4), it can be concluded that the capacity of the SPC 1B is at least 160 percent of the SPC 1A capacity. Furthermore, because the SPC 1B E-E cycle rate at 9300 trunk seizures per quarter hour is high with respect to the indicated value of $E_{MIN}$, it appears that the SPC 1B capacity is comfortably greater than 160 percent of the SPC 1A. This additional capacity serves as a margin to accommodate variation among sites with respect to call mix and peakedness in busy-hour load.

### *2.3.6 TSPSCAP program for TSPS No. 1B*

A TSPSCAP program was developed for the TSPS No. 1B incorporating the SPC 1B real-time model. As has been seen, a load-line

equation for the SPC 1B is considerably more complex than for the SPC 1A, and depends to a large extent on the amount of SPC 1B diagnostic and craft interface activity. Therefore, instead of providing a load-line equation, the TSPSCAP program provides calculated values of $E'$ and $N'$ which can be used to plot two load lines for the site in question. These two load lines, similar to the upper and lower load lines shown in Fig. 4, define what can be termed as a load-zone of normal system behavior. That is, the quarter-hour E-E cycle and trunk-seizure measurements for a site that is experiencing normal operation should fall within this load zone.

## III. RELIABILITY EVALUATION

### *3.1 Reliability requirements*

The SPC 1B reliability requirements are similar to those of a traditional ESS-type processor, having four fault categories: hardware faults, recovery deficiencies, procedural errors, and software deficiencies.[10]

Hardware failures are allocated 0.4 minute of downtime per year. The SPC 1B is divided into three subsystems, each of which is duplicated to achieve high reliability. Thus, one failure in one side of a subsystem will not cause a system outage. Hardware faults can cause a system outage only when both sides of a subsystem are experiencing failures (i.e., before the first failure is repaired, another failure occurs on the other side of the subsystem). When this occurs, the system is unable to establish a working configuration until one side of the failed subsystem is repaired and system integrity is reestablished. The hardware reliability is a function of the failure rates of the subsystems, the system architecture, and the repair rates of the subsystems.

Recovery deficiencies are allocated 0.7 minute of downtime per year. When a hardware failure condition is detected, an automatic fault-recovery action occurs to establish a working configuration. Unsuccessful recovery actions are classified as recovery deficiencies. These are due to either design errors or limitations in fault-recovery programs.

Procedural errors are allocated 0.6 minute of downtime per year. An improper maintenance procedure can cause a system outage. Providing easy-to-follow documentation and reducing the number of manual steps help to minimize procedural errors.

Errors in operational programs and data are allocated 0.3 minute of downtime per year. The amount of bootstrap time required to recover the system from software deficiencies is considered to be a part of system downtime under this category. To minimize this source of downtime, overall software execution is monitored continually, data

integrity is checked using extensive auditing procedures, and thorough system integration tests are performed after program changes are introduced.

All four potential causes of system outage are closely interrelated. For example, improper procedures combined with certain hardware faults may prevent system recovery. In this paper, only SPC 1B outages induced by hardware faults are considered.

### *3.2 Reliability estimates*

#### *3.2.1 Reliability model*

To provide a basis for the relationship between the reliability model and the system architecture, a brief review of the SPC 1B architecture is presented. A complete description of the SPC 1B architecture can be found in Ref. 11, and a more detailed description of 3B20D architecture can be obtained from Ref. 12.

As shown in Fig. 6, the SPC 1B consists of three subsystems or communities: a duplicated 3B20D Control Unit and PSI (CU/PSI), a duplicated Input/Output Processor (IOP), and a duplicated Disk File Controller with Movable Head Disk (DFC/MHD). Either half of the duplicated CU/PSI community can access either side of the duplicated TSPS peripheral bus system. The IOP community has duplicated

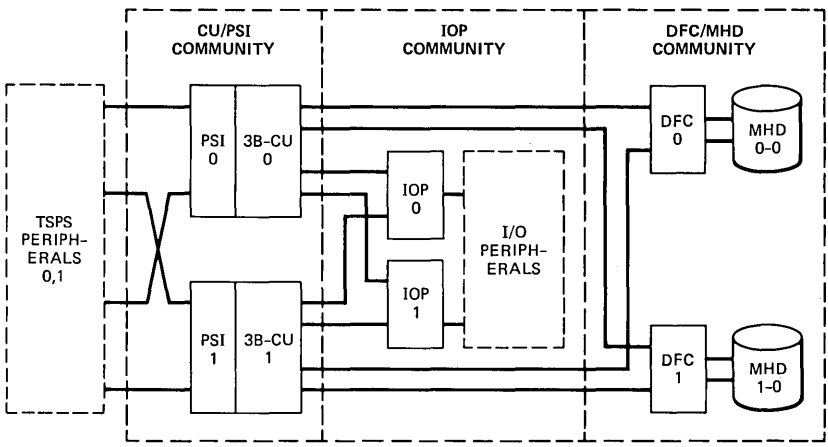



Fig. 6—SPC 1B architecture.

IOPs, each controlling a number of Peripheral Controllers (PCs). Each type of PC is designed to control a specific peripheral device, such as tape drives, teletypewriters, etc. The DFC/MHD community consists of a pair of DFCs, each capable of controlling one or more associated MHDs. One MHD on each half of the DFC/MHD community contains software required to bootstrap the SPC 1B.

Certain reliability measures are required to predict the probability of successful operation of the system. In expressing reliability of a switching system, "availability" is a more widely used term. Availability is defined as the fraction of time, on the average, that a system is expected to be in an operating condition. The availability of a switching system is a function of the system architecture and of subsystem failure rates and repair rates. Estimating availability of a system requires a mathematical model that can reflect the system architecture appropriately. The continuous-time, finite-state Markov model was used for the SPC 1B availability calculation, where an exponential probability distribution was assumed for the failure rates and repair rates. Detailed descriptions of how to use the Markov model to calculate system availability can be found in Refs. 5 and 13.

The reliability model for each of the three communities contains three states: a "duplex state," a "simplex state," and a "down state." The duplex state is a state where both halves of the community are fault-free and operational. Upon detecting a fault in either half of the community, a transition to the simplex state occurs. The rate of the transition is determined by the failure rate of the community. While a community is in the simplex state, one of two transitions is possible. A transition to the duplex state could occur if the faulty half of the community is repaired before a failure occurs in the other half of the community. On the other hand, a transition to the down state may occur if a new fault is detected in the remaining half before the initial fault is successfully repaired. A transition from the down state to the simplex state occurs when one of the faulty halves is repaired and put back to service. The rates of transitions, from the down state to the simplex state and from the simplex state to the duplex state, are determined by the repair rates, which are the reciprocals of corresponding mean time to repairs (MTTRs) for the community.

The probability that a community is in the down state is defined as the unavailability of the community. The SPC 1B is considered out of service when any of the CU/PSI, IOP, or DFC/MHD subsystems of the SPC 1B are in a down state. Hence, the unavailability of the system can be obtained by calculating the sum of unavailabilities of these three communities. The expected downtime per year for the SPC 1B can be estimated directly from the unavailability of the system.

### 3.2.2 Availability estimates and modifications

To evaluate the unavailability of each community, the reliability model was converted to a set of simultaneous equations where the unknowns are the probabilities of the states. Programs were written to solve the sets of equations corresponding to various architectural configurations. When these programs are used, sensitivity of the system downtime to the architectural variations as well as to the parametric values such as repair rates and failure rates of each community could be investigated. Coefficients of the equations were determined by the failure rates and repair rates of each community. Failure rates of the three communities were estimated from their component failure rates.

The repair rate of each community is estimated from the MTTR of mechanical failures, the MTTR of electrical failures, and a craft dispatch time. The MTTR of mechanical failures is considered separately from the MTTR of electrical failures because, for an MHD, the MTTR of mechanical failures is an order of magnitude longer than that of electrical failures. To minimize the MTTR, extensive diagnostic programs are included in the TSPS No. 1B, which can locate a fault within the resolution of three circuit packs. Detailed descriptions of diagnostic programs can be found in Refs. 11 and 14. A craft dispatch time is added to the MTTR when determining the repair rates of each community because the SPC 1B can be maintained by craft personnel located at a remote site. The dispatch time depends on the average travel time from the remote site and on the ratio between the average staffed hours and unstaffed hours per day of the TSPS No. 1B office.

The failure rates of the CU/PSI, DFC, and IOP are principally due to electrical failures. On the other hand, the failure rate of the MHD is due to roughly half mechanical and half electrical failures. Consequently, the MTTR of an MHD is much longer than the MTTRs of the other units. For the TSPS No. 1B application, the MTTR of an MHD has been improved through use of a spare MHD for each system.

Evaluation of the reliability model using current parameters shows that the reliability objectives for the TSPS No. 1B have been met.

## IV. CONCLUSION

This paper has described the prediction and evaluation of the call-processing capacity and system reliability of the SPC 1B. The call-processing capacity has been estimated through means of a processor real-time model whose parameter values have been determined by laboratory and test-site measurements. The system reliability has been predicted through use of Markov modeling techniques. Performing this evaluation during TSPS No. 1B development to monitor progress was instrumental in meeting the capacity and reliability objectives.

## V. ACKNOWLEDGMENTS

G. J. Kujawinski and the authors worked together as a team in evaluating the SPC 1B capacity; his contributions were essential to the success of the effort.

## REFERENCES

1. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
2. N. X. DeLessio and N. A. Martellotto, "Traffic Service Position System No. 1B: System Description," B.S.T.J., this issue.
3. R. J. Gill, G. J. Kujawinski, and E. H. Stredde, "Traffic Service Position System No. 1B: Real-Time Architecture Utilizing the DMERT Operating System," B.S.T.J., this issue.
4. L. B. Brisson and R. L. Potter, "A Time-Shared Program for Predicting TSPS No. 1 Capacity," Proc. Int. Conf. Commun., Seattle, 1973.
5. J. A. Buzacott, "Markov Approach to Finding Failure Times of Repairable Systems," IEEE Trans. Reliability, *R-19* (November 1970), pp. 128–34.
6. R. J. Jaeger, Jr. and R. L. Potter, "Analysis of Processor Usage in Stored Program Controlled Telephone Switching Systems," Proc. Int. Conf. Commun., Philadelphia, PA, 1972.
7. Special issue entitled "Traffic Service Position System No. 1," B.S.T.J., *49*, No. 10 (December 1970).
8. N. Farber, "A Model for Estimating the Real-Time Capacity of Certain Classes of Central Processors," Paper 426, Proc. Sixth Int. Teletraffic Congress.
9. J. J. Stanaway, Jr., J. J. Victor, and R. J. Welsch, "Software Development Tools," B.S.T.J., *58*, No. 6, Part 1 (July 1979), pp. 1307–34.
10. P. W. Bowman, M. R. Dubman, F. M. Goetz, R. F. Kranzmann, E. H. Stredde, and R. J. Watters, "Maintenance Software," B.S.T.J., *56*, No. 2 (February 1977), pp. 255–87.
11. G. T. Clark; H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
12. M. W. Rolund, J. T. Beckett, and D. A. Harms, "3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., *62*, No. 1, Part 2 (January 1983), pp. 191–206.
13. M. D. Soneriu and D. S. Suk, "Markov Model for Estimating the Reliability of Duplicated and Repairable Computing Systems," Nineteenth Annual Tech. Symp., Pathways to System Integrity, Washington, D. C. (June 1980), pp. 87–95.
14. J. L. Quinn, F. M. Goetz, and R. L. Engram, "3B20D Processor & DMERT Operating System: Diagnostic Tests and Control Software," B.S.T.J., *62*, No. 1, Part 2 (January 1983), pp. 367–81.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Switching Control Center System Interface

By J. J. BODNAR, J. R. DAINO and K. A. VANDERMEULEN

(Manuscript received July 30, 1982)

*Over the years, the centralization of the maintenance and operation of Stored Program Control Systems (SPCS) has proved to be an economically attractive and effective methodology. At the core of the centralized maintenance plan is the Switching Control Center (SCC), which has responsibility for the surveillance and control of a number of SPCS. This center is supported by the Switching Control Center System (SCCS), which is a minicomputer-based system that provides automation of, or mechanized support for, the functions of the SCC. Since SCCS support for the Traffic Service Position System No. 1 (TSPS No. 1) was already available, these capabilities needed to be carried forward to support the operation and maintenance of TSPS No. 1B. In addition, the use of the new 3B20D Processor with a new craftperson interface allowed for a number of improvements and extensions. The SCCS interface to TSPS No. 1B has made use of new technology and techniques by incorporating microprocessors, video terminal interfaces, and BX.25 protocol in the design. The result is a flexible interface with software-driven craft displays.*

## I. CENTRALIZED MAINTENANCE—OVERVIEW

Since the early 1970's the Bell Operating Companies (BOCs) have been taking increasing advantage of the concept of centralized maintenance of switching systems. To economically operate, administer, and maintain electronic switching systems, operations centers were formed where technical experts equipped with computer-based support systems can apply their skills to many switching systems. In the case of the Traffic Service Position System (TSPS), the daily maintenance and operations functions may be provided by the Switching Control Center (SCC).

If a TSPS is maintained and operated from an SCC, the SCC manager has overall responsibility for the quality of service provided by the TSPS and the cost of maintaining the system. To provide high-quality service at a reasonable cost, a number of functions must be performed efficiently at the SCC. Since in this environment the TSPS is unattended, except when field personnel are assigned to specific tasks in the office, the SCC personnel monitor the real-time status of the TSPS on a continuous basis. In the event of failure of equipment in the TSPS complex, the SCC exercises appropriate controls to ensure service protection. The cause of the failure is then analyzed and isolated. Finally, when the trouble is identified, field personnel are dispatched to perform any necessary on-site repair.

To accomplish these functions, a number of work positions have been established in the SCC and individual craft have been assigned particular responsibilities.

The Office Controller is responsible for real-time surveillance of several switching systems and as such has the most frequent interface to the TSPS. (Most BOCs will maintain several types of electronic switching systems from each SCC.) The controller responds to system alarms and takes necessary service-protection action. The Analyzer is responsible for those more complex problems that require sectionalization and trouble identification. The Analyzer interacts with the TSPS to collect additional data and uses previously reported data to isolate system troubles. The craftperson at the trunk-maintenance position is responsible for identifying faulty trunks and directing their repair. In addition, dispatch and administration functions exist in the SCC to identify work for field forces, facilitate proper information flow, and keep records.

## II. TRADITIONAL INTERFACE

The No. 2 Switching Control Center System (No. 2 SCCS) began deployment in the mid-1970's as a computer-based support system to aid the SCC personnel in performing their jobs. It has evolved over time by providing new features and interfacing to Stored Program Controlled Systems (SPCS) such as TSPS No. 1.[1,2] With the development of TSPS No. 1B, a parallel SCCS development was done to integrate the new system into the existing operations environment while taking advantage of opportunities afforded by the new 3B20D Processor.

### *2.1 The TSPS No. 1—SCCS interface*

The traditional interface provided by No. 2 SCCS consists of a dedicated channel from a TSPS No. 1 that parallels the Maintenance

Teletypewriter (MTTY), plus a dedicated channel that terminates on an E2A telemetry unit at the TSPS No. 1 Master Control Center (MCC). At the SCC, the TTY channel is connected to a minicomputer system, known as the Computer Subsystem (CSS) and the telemetry channel terminates on a telemetry unit that collects data from many switching offices. This basic architecture is shown in Fig. 1. The TTY channel is a 110-baud asynchronous serial communication line. The data output by the TSPS No. 1 on this channel reports detailed information in line-oriented output messages suitable for hard-copy teleprinters. In addition, line-oriented input commands may be sent to the TSPS No. 1 on this channel. The TTY channel data are collected and stored in the CSS. The data are available for real-time alerting and long-term analysis.

The telemetry system serves the purpose of reporting real-time equipment status and allows controls to be activated remotely. At the TSPS No. 1 MCC an E2A remote unit is provided to collect detailed

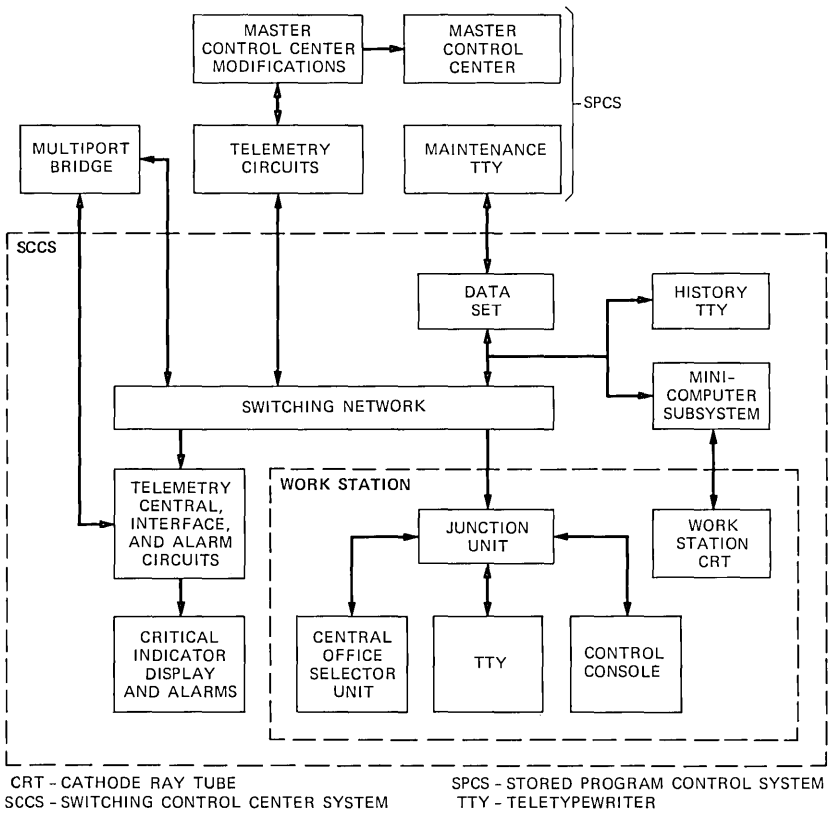



Fig. 1—TSPS No. 1-SCCS interface.

status of TSPS No. 1 hardware states. This is accomplished by connecting discrete scan points in the E2A telemetry unit to individual lamp displays on the MCC. Control capability is provided by individually connecting relay contact closures in the E2A telemetry unit in parallel with the pushbutton keys on the MCC. The interface is thus hardwired in place.

An E2A central unit in the SCCS polls up to 16 remote E2A units on a continuous basis. The polling process takes place over a data network that has a multiport bridge at its hub. The central unit transmits a request for data along with a unique address of one of the remote units. The remote units on the network respond only when their address is contained in the polling request. The message format used consists of start and stop bits, 16 data bits, and a 7-bit cyclic redundancy check sum. This data word contains a summary of the status of critical hardware units in each office. The data are displayed in real-time on a wall-mounted status panel called the Critical Indicator Panel (CIP) in the SCC.

To gain access to more detailed status-and-control capability, a control console at the SCC can be connected to the E2A remote unit at the TSPS No. 1. This is accomplished by the use of switching equipment in the SCCS to disconnect the E2A from the multiport network and connect it on a point-to-point basis with a control console at a work station in the SCC. Two types of consoles exist for interconnection with a TSPS No. 1. The original type of console provided was a wired logic console with a fixed display and keyboard. The display consisted of status lamps with labels that corresponded to labels printed on the MCC. Pushbutton keys on a keyboard were labeled with the same designations as used on the keys at the MCC.

Housed inside the console was an E2A central unit which was configured to collect all the data available from the remote unit. Its memory was hardwired to the lamps on the display panel. Such a console was then limited to functioning only with a TSPS No. 1. Any changes or additions to the MCC required corresponding changes to the consoles at the SCC. A second type of console is now provided, which consists of a CRT and microprocessor. In this type, the microprocessor communicates with the E2A remote unit to collect status information. The interface to the craft at the SCC consists of a CRT display. On this display, the lamps and keys on the MCC are represented by their labels, as shown in Fig. 2. Active status or controls are shown by displaying the label in reverse video (black characters on a white field). Keys are distinguished by preceding the label with a plus sign. Keys are activated by cursor positioning. This software-driven display allowed for the use of the same physical equipment with other types of switching systems.

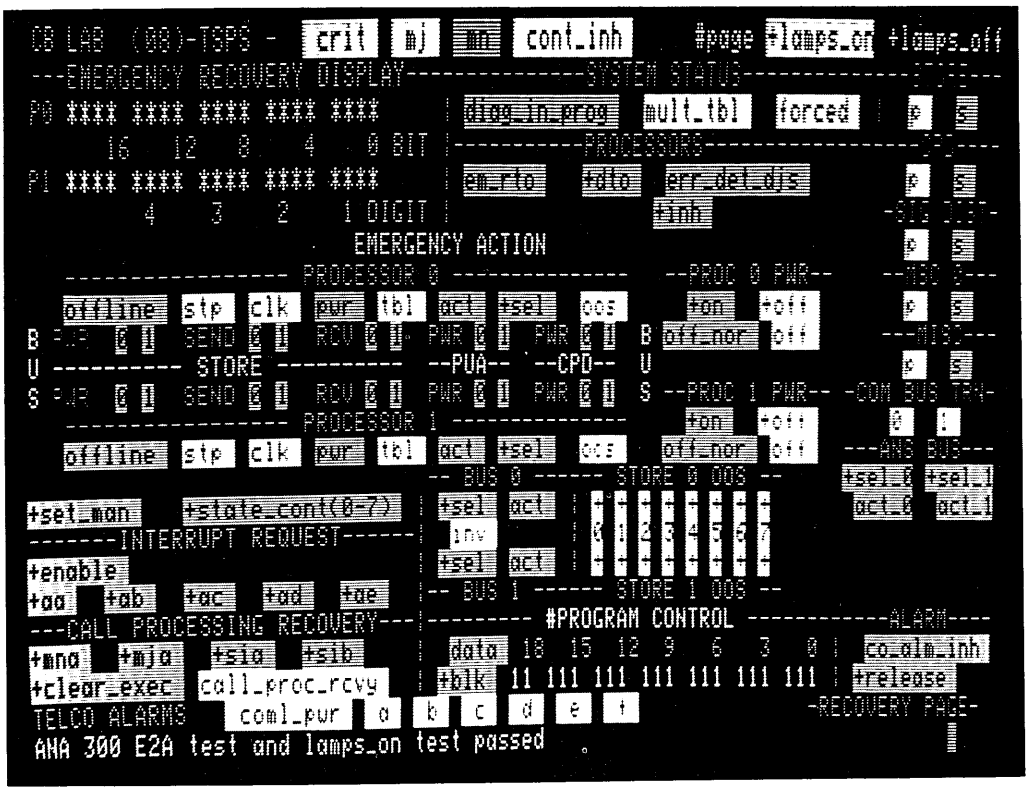



Fig. 2—TSPS No. 1 MCC display page.

### 2.2 Human interface

A number of human interfaces are provided to alert SCC craft to trouble conditions and to allow for further data collection, analysis, and control. As data are collected via the telemetry network in real time, alarms and status of critical equipment units are displayed on a highly visible CIP. This provides alarm visibility and long-term equipment status information to the SCC managers and craft. TTY messages are examined in real time as they are received and stored by the CSS. Messages reporting alarmed events are used to trigger a video display on an alarm monitor, which is a large CRT device. The identity of the switching system reporting the trouble and the alarm level are displayed along with a one-line summary of the condition being reported.

The office controller is responsible for responding to the real-time events reported on the CIP and the alarm monitor. The controller's work station is equipped with two CRTs that serve as additional interfaces to the system. One CRT, known as the CSS work station, is connected to the CSS. Via this device the controller can examine previously reported data that have been stored by the CSS. In addition, the CSS work station can be placed in the monitor mode where it is connected to the TSPS No. 1 via the CSS and becomes the functional equivalent of the maintenance TTY at the TSPS No. 1. That is, in this mode the work station receives each character output by the TSPS No. 1 as it reaches the CSS and any characters typed by the user are sent directly to the TSPS No. 1.

The second CRT is the microprocessor-driven console described above, which interfaces to the telemetry network. This equipment constitutes the control console and is the functional equivalent of the local MCC lamp and key panel. Thus the office controller has the same capabilities at the SCC work station that a craft at the TSPS No. 1 has via the maintenance TTY and MCC panel. The control console and CSS work station functions are provided on separate terminals and via separate processors and links in order to enhance the availability of the interface in the face of a simplex outage.

The analyzer work station is also equipped with these devices. The analyzer, however, makes more use of SCCS programs in the CSS which analyze large volumes of data and reduce them to summarized outputs and reports. For example, filtering and patterning capabilities allow for the identification of repetitive events and aid in the trouble-sectionalization process. While the office controller uses the control console for service-protection purposes, the analyzer uses it for more detailed data collection and observation of system performance under specific conditions.

It is this operations environment into which the new TSPS No. 1B

was integrated. The new interface serves as the foundation for remote maintenance of TSPS No. 1B as well as other 3B20D Processor applications.

## III. THE TSPS NO. 1B—SCCS INTERFACE

### *3.1 Elements of the new local maintenance position*

With the use of the 3B20D Processor and the DMERT operating system came the opportunity for new flexibility at the local maintenance position.[3] The MCC and the MTTY of the TSPS No. 1 have been replaced by the Maintenance CRT (MCRT) and a Receive-Only Printer (ROP).[4] The ROP provides a paper history function by recording each maintenance event or alarm condition in the form of line-oriented output messages.

The MCRT has several modes of operation. The first, the Emergency Action Interface (EAI), can be thought of as the replacement of the emergency recovery functions of the MCC. Actually, the EAI provides a hardware control function that is independent of the system software. It provides recovery capabilities for the TSPS No. 1B even in the absence of software sanity. The MCRT also has a test input/output mode. This allows the user to input line-oriented commands and to receive responses to these commands as well as spontaneously generated output. This is equivalent to the old MTTY input and output mode. In addition, the MCRT has a Control and Display mode (C&D), which provides the user with page-oriented displays of system status and with a menu of commands used to affect the status or configuration of the system. The C&D mode provides a very effective human interface for the daily functions of trouble isolation and correction. Finally, the MCRT provides continuous status information on the health of the system via a critical indicator display.

These different modes of operation are provided by dividing the screen into several sections and providing keys for toggling between modes (see Figs. 3 and 4). The top of the CRT is a header line. The second and third lines of the MCRT contain critical indicator information. The fourth line is reserved for command input. Lines five through twenty-two are used alternatively to display menus and pages in the C&D mode or to display the EAI page. Finally, any lines that are unused by the page that is currently being displayed are used for text input and output. At least two lines are always available for text input and output. However, if the currently displayed page does not use the entire page region, the remaining area may also be used for output.

The MCRT and the ROP interface to the 3B20D Processor through a firmware controlled unit called the Maintenance Teletype Controller (MTTYC). This unit provides the EAI mode and interfaces between

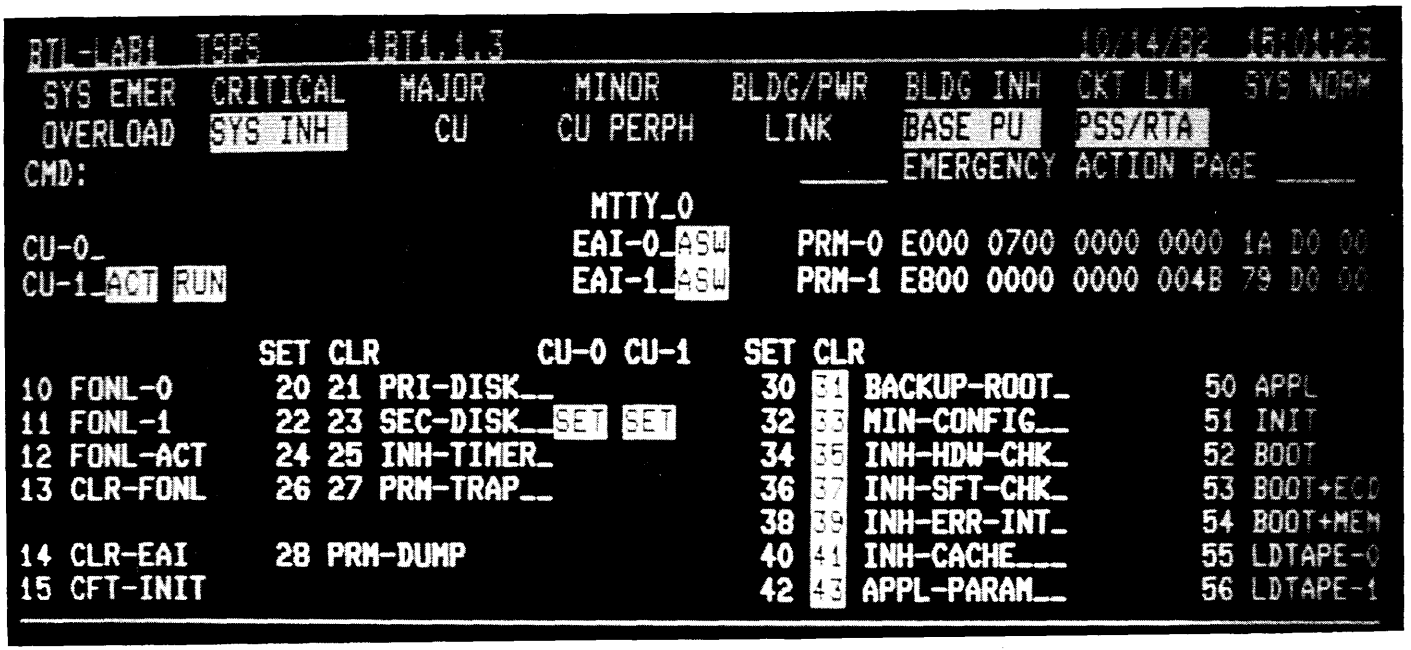



Fig. 3—TSPS No. 1B EAI page.

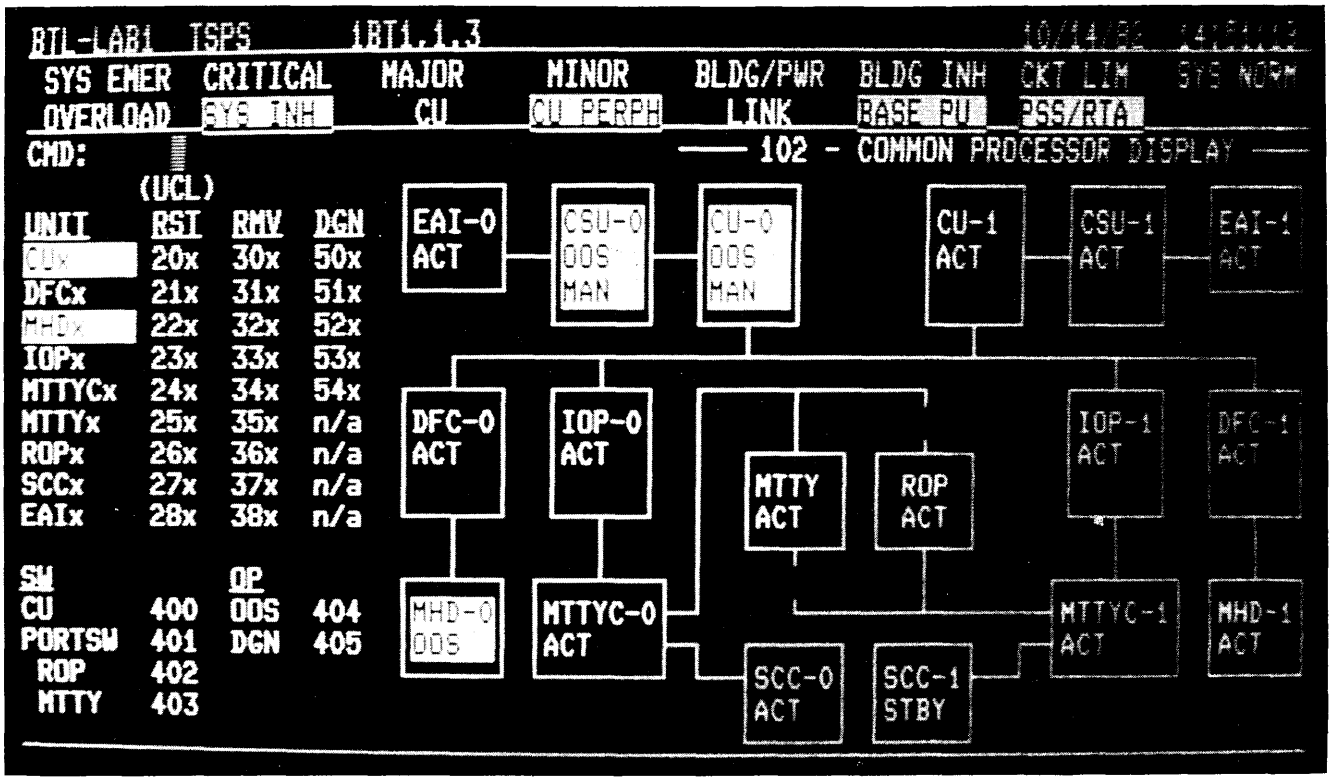



Fig. 4—TSPS No. 1B control and display page.

the terminals and the 3B20D Processor in other modes. The firmware in the MTTYC is responsible for controlling the MCRT in the EAI mode to allow access to the system in the absence of 3B20D Processor sanity. The MTTYC also controls the interface to the SCCS. All other modes of operation and information on the MCRT are controlled by software in the 3B20D Processor.

### *3.2 Elements of the remote maintenance interface*

The allocation of information and modes of operation at the SCC is consistent with both the local interface at the TSPS No. 1B and the existing SCC functions. The critical indicator information is displayed on the CIP as was done for the TSPS No. 1. In addition, the text input/output and the data on the ROP are logged in the CSS. This provides the work station with the Monitor mode and allows the CSS to drive the alarm monitor. This will also allow the existing work station features for the controller and the analyzer to operate as before. Also, the EAI interface is available to the Control Console (CC) just as the MCC functions of the TSPS No. 1 were provided there. This allows the controller to recover the TSPS No. 1B when necessary. Providing this function on the CC gives added assurance that it will be available in the event that the CSS is unavailable.

Furthermore, to provide consistent operation with the local interface, a CRT display mode essentially identical to that at the MCRT is provided on both the CC and the CSS work station. The only exception is that the CC is not provided with the text input/output functions at the bottom of the screen. This means that both the CSS work station and the CC have access to the EAI mode and the C&D mode, and the CSS work station has full TTY input/output capabilities. Also, all the mechanized features provided by the CSS to support the controller, analyzer, and trunk-maintenance positions remain available.

To communicate between the TSPS No. 1B and SCCS in the C&D mode, a language was defined with which the TSPS No. 1B tells the SCCS what should be displayed on the screen. The SCCS then sends each user command to the TSPS No. 1B, which processes the command and sends back a description of the display, complete with all the up-to-date status information. The SCCS translates that data stream into one that is understood by the SCC terminal. The language that is used to communicate this information is known as Virtual Terminal Protocol (VTP). By this method, the SCCS is not required to have complete up-to-date information describing each of the many C&D pages provided by TSPS No. 1B. This helps keep the system relatively independent with respect to changes and enhancements.

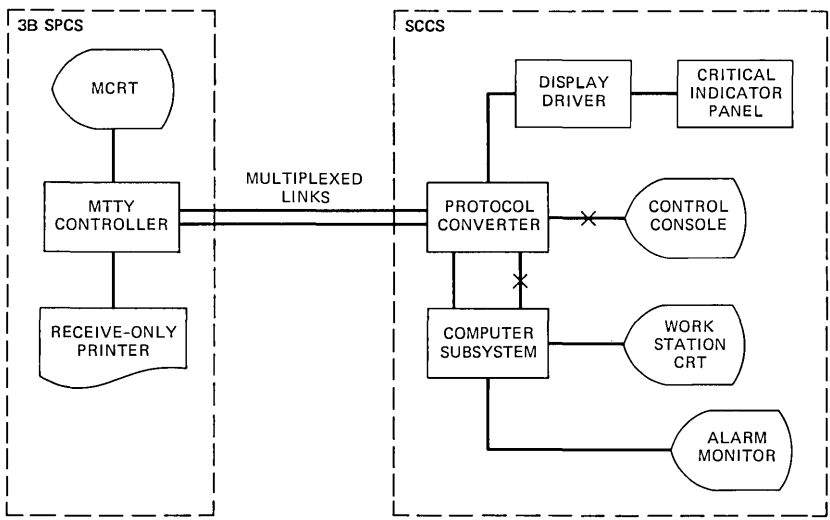
With respect to the EAI mode and the critical indicators, however, much less information is involved and it is less subject to change. The

EAI consists of a single page, which is controlled by the firmware in the MTTYC. There are up to twelve critical indicators, the definition of which is very stable for each SPCS. Therefore, the SCCS is capable of generating a critical indicator display as well as an EAI display, and the TSPS No. 1B simply sends the data required to describe the status of each of the indicators or display components.

### *3.3 Basic architecture of the interface*

Since the existing TTY and E2A interfaces do not have the capacity to support the amount and form of information that must be exchanged between the TSPS No. 1B and SCCS, a new interface was designed. In particular, the information necessary to drive the EAI and C&D modes requires highly reliable transmission of a significant amount of binary information. The new interface that was designed for communication between the TSPS No. 1B and the SCC, Fig. 5, consists of a pair of 2400-baud synchronous links between the MTTYC and a new device at the SCCS called the Protocol Converter (PC). One link is normally active while the other is used as a backup and is therefore normally in the standby state. The link carries seven logical streams of data under the BX.25 protocol using the virtual channel assignment capabilities of that protocol. The virtual channels (VC) are assigned as follows:

VC1—Emergency action interface
VC2—Critical indicator information




CRT - CATHODE RAY TUBE
MCRT - MAINTENANCE CRT
MTTY - MAINTENANCE TELETYPEWRITER

SCCS - SWITCHING CONTROL CENTER SYSTEM
SPCS - STORED PROGRAM CONTROL SYSTEM

Fig. 5—TSPS No. 1B—SCCS interface.

VC3—C&D interface
VC4—TTY input and acknowledgment
VC5—TTY output
VC6—Spare
VC7—Spare

The two links provide simple backup for each other. If either end detects a link outage at level 2 of the protocol, it will switch to the other link and establish communication on that link. The function of the PC is to handle the BX.25 protocol and to divide the virtual channels into separate physical connections bound for the different subsystems of the SCCS.

The PC is a firmware-controlled microprocessor system. Each microprocessor can support up to three 3B20D-based systems. The PC is housed in a newly designed cabinet, known as the T-cabinet, which can hold up to three PCs. In addition, this cabinet contains one spare PC, which can be manually activated to replace any one of the three active PCs, should they fail.

The PC sends the critical indicator information contained on VC2 to another firmware-controlled microprocessor called the display driver, which interfaces to the CIP. The display driver is also housed in the T-cabinet and can support up to 18 systems.

The PC sends and receives the TTY information contained on VC4 and VC5 to and from the CSS. The interface is a 1200-baud asynchronous serial communication line similar to the MTTY interface in the previous architecture. Since this is a character-oriented interface and the BX.25 link to the TSPS No. 1B is a message-oriented interface, the PC provides for the translation between the two. The fact that the TSPS No. 1B emulates the TSPS No. 1 has allowed the software in the CSS—used to assist the SCC in the maintenance and operation of TSPS—to be carried forward to support TSPS No. 1B with only minor modification. That is, most information received from a TSPS No. 1B, except that which deals directly with the processor itself, is the same in form and content as in TSPS No. 1. Thus, all the mechanized tools provided by the SCCS are still applicable with very little modification. This fact significantly enhances the ease with which the TSPS No. 1B can be integrated into the SCC environment as it gradually replaces the existing TSPS No. 1's.

To provide the MCRT function, the PC multiplexes the data contained on VC1, VC2, and VC3 onto a single physical link which can be connected to either the control console or the CSS, depending on whether the control console or CSS work station is currently accessing the office. This interface is also a 1200-baud asynchronous communication line, and a simplified version of levels 2 and 3 of the BX.25 protocol has been designed for this link. This allows for the three logical data streams to coexist on this link and also allows for some

error checking to increase the reliability of the interface. In addition, this link allows the user to input C&D commands or EAI commands into the TSPS No. 1B through the PC.

In summary then, the interface described in this section permits the distribution of functions among the subsystems of the SCCS, which is consistent with existing operations and allows the SCC user to access the TSPS No. 1B in the same manner as the user at the local maintenance position. The control console has access to the EAI, critical indicator, and C&D information when it is attached to the office. In addition, the CSS work station can "display" the office information and receive the same view as the MCRT at the TSPS No. 1B, including the text input/output functions at the bottom of the CRT. Finally, all the status and alarm information output at the ROP and all the input/output activity on the TTY channel—both from the SCC and the local position—are logged in the history files in the CSS. This information then becomes the basis for the alarms displayed on the alarm monitor and for the sophisticated tools provided by the SCCS to support the controller, analyzer, and trunk-maintenance positions at the SCCS.

## IV. OPERATIONAL IMPACT

The TSPS No. 1B-SCCS interface introduces several new features to the SCC craft. The use of software and microprocessor technology at the SCC and the TSPS No. 1B have enabled various design objectives to be met. The available technology allows the use of the BX.25 protocol providing a multiplexing arrangement on the TSPS No. 1B-SCCS link. Intelligent control equipment administers the BX.25 protocol that provides error detection and correction on the link. Software-controlled video displays replace hardwired control panels provided on the TSPS No. 1.

The 3B20D Processor will be used in other Western Electric switching systems, which will permit the development of a common SCC interface for all such systems, These, and other new features and capabilities described below, take advantage of the available new technology. This section discusses how the features and capabilities of the architecture described in Section III affect the SCC operations.

### *4.1 Consistent operation with local TSPS No. 1B maintenance terminal*

The CSS work station operation with the TSPS No. 1B is essentially identical with the on-site TSPS No. 1B maintenance terminal. In addition to all capabilities provided at the on-site TSPS No. 1B terminal, the CSS work station has access to the features provided by the SCCS computer subsystem. In the EAI and C&D modes, the control console also has identical operation as the on-site TSPS No. 1B maintenance terminal. Both the SCC work station and the control

console have full access to the EAI page to monitor and initiate TSPS No. 1B status and recovery functions. By reducing the differences between the local and remote interfaces, training requirements are reduced and craft efficiency is enhanced.

### *4.2 Consistent operation with existing SCC*

The design provides the TSPS No. 1B with support which is operationally similar to other SPCSs interfacing to the SCCS.

The TSPS No. 1B may be monitored and controlled from the CSS work station with the CSS providing the support functions of logging, sorting, and browsing. All the tools provided by the CSS for the support of TSPS No. 1 are extended to TSPS No. 1B. This means that operational procedures that are in place with respect to TSPS No. 1 are still applicable. The alarm monitor display of alarms is also fully functional.

In the event of a CSS failure, the control console provides craft access to the TSPS No. 1B as it does in other SPCS-SCCS interfaces. Alarms are displayed on the critical indicator panel in a standard line-up arrangement as with other systems. All CSS features operate in the same manner as with previous systems.

While enhancements have been made to the human interface, the basic SCC functions (described in Section II) have been maintained with the TSPS No. 1B.

### *4.3 Improvements to the human interface*

Since the TSPS No. 1B introduces many new features and capabilities in the maintenance terminal, these features are also available at the CSS work station and control console as a result of the new TSPS No. 1B-SCCS interface.

For example, in addition to the text input/output message capability, used in the TSPS No. 1, the TSPS No. 1B-SCCS interface includes the C&D mode. The C&D mode provides a series of displays that give current status of various sections of the system. These displays may be requested by the craftperson using a menu selection process. The menu lists several possible display pages with a three-digit-selection menu number. In addition, the user can initiate many of the typical maintenance and recovery actions required to operate a TSPS No. 1B using menu numbers provided on these displays. This provides a significant improvement in speed and accuracy for frequently used functions over the text input and output capabilities of the TSPS No. 1 and SCCS interface.

The EAI page, Fig. 3, follows the same format as other display pages. From this page the craftperson is able to initiate various recovery functions using a two-digit menu selection. Also available on this page

is progressive system status during recovery sequences. This is a significant improvement over the previous MCC interface of the TSPS No. 1 in several ways. In particular, the EAI provides more information—because of the video display—and more consistency between the emergency interface and other kinds of access to the system. Consistency between the local interface and the remote interface at the SCC is also enhanced. In the TSPS No. 1 the local interface was a hardware panel of lamps and keys and the remote interface used a video display and keyboard to emulate the local interface. In the new design the local and remote interface are identical. The system also provides four special function keys at the video terminal. These keys are designed to allow the craft fast and convenient use of the capabilities of the human interface. The keys provide one-button operation to perform the following functions:

(*i*) Request the EAI page (EA DISP)

(*ii*) Exit the EAI page (NORM DISP)

(*iii*) Change between the text input and menu input mode (CMD/MSG)

(*iv*) Retire alarms (ALM RLS).

To illustrate the advantages of the control and display operation available with the interface between the TSPS No. 1B and the SCCS, consider the example of the alerting and analysis of a major alarm associated with a moving head disk in the TSPS No. 1B. As with other systems, a TSPS No. 1B major alarm results in a visual indication at the critical indicator panel, a one-line alarm summary message is displayed at the alarm monitor, and, if activated, an audible alarm sounds. If the controller then connects the CSS work station to the office in the display mode, the display will indicate the alarm with the "Major" indication and flashing "CU PERPH" (Control Unit Peripheral problem) in reverse video in the critical indicators section of the display (second line on the screen).

The controller can retire the alarms by depressing the alarm-release special function key. This will return the major indication to normal on the screen and result in "CU PERPH" in solid reverse video. The controller can then select the CU PERPH page from the page index by entering the three-digit menu number associated with that page. The CU PERPH page will display the status of the 3B20D Processor and peripherals. On the display, the moving head disk will be indicated as out-of-service. Once called in by the controller, the analyzer may then request diagnostics be run on the disk using either a menu-selection number (from the menu of commands displayed on the left part of this display page) or, by using the special function key to change to the text input mode and typing a text diagnostic message. The result of this diagnostic should lead the analyzer to further action, (e.g. dispatch to repair or replace the faulty hardware).

Video displays and menu selection provide an effective human interface, which results in a low probability for human errors occurring and a high level of job satisfaction.

### *4.4 High reliability*

To meet the high reliability required of the TSPS No. 1B-SCCS interface for unattended operation of TSPS, the interface is designed with a highly redundant architecture. The 2400-baud data link between the two systems is fully duplicated and the BX.25 protocol provides error detection and correction. The TSPS No. 1B components that control and administer the SCCS interface (e.g., the MTTYC) are fully duplicated and provide full access both to the local and remote users in the event of single-unit failures.

The SCCS components are also backed up. In particular, the T-cabinet provides for a sparing arrangement in the event of a PC failure, and the duplication of capabilities at both the control console and CSS work station allows these terminals to back each other up in the event of failures in either subsystem. Finally, the CIP provides alarm and status information even in the event of a CSS failure that disables the alarm monitor.

In addition, this design provides for some improvement in the capabilities provided to the SCC when single-link failures occur. In previous SPCS-SCCS interfaces, the text message data and the E2A telemetry data containing critical indicator information and emergency action information were transmitted over separate facilities. Therefore, if one facility failed, the SCC lost partial capability. The TSPS No. 1B-SCCS interfaces, using two fully duplicated links, provide the ability to transmit all the necessary data with no loss of capabilities at the SCC in the event of a single transmission-channel failure.

## V. CONCLUSION

The SCCS interface to TSPS No. 1B has made use of new technology and techniques by incorporating microprocessors, video terminal interfaces, and BX.25 protocol in the design. The result is a flexible interface with software-driven craft displays. At the SCC, the craft interface devices are the same physical devices used to interface with older technology SPCSs, yet they provide new video displays and menu selection inputs. Thus, the new technology of TSPS No. 1B can be maintained from the SCC as it is introduced into the network and can also be used to enhance SCCS craft interfaces while remaining consistent with overall SCC operations.

## VI. ACKNOWLEDGMENTS

The authors would like to recognize the efforts of a large number of

people over a several-year period in the development and testing of the TSPS-SCC interface.

## REFERENCES


1. R. J. Jaeger, Jr. and A. E. Joel, Jr., "Traffic Service Position System No. 1," B.S.T.J., *49*, No. 10 (December 1970), pp. 2417–31.
2. R. E. Staehler and W. S. Hayward, Jr., "Traffic Service Position System No. 1, Recent Developments," B.S.T.J., *58*, No. 6, Part 1 (July–Aug. 1979), pp. 1109–1367.
3. G. T. Clark, H. A. Hilsinger, J. H. Tendick, and R. A. Weber, "Traffic Service Position System No. 1B: Hardware Configuration," B.S.T.J., this issue.
4. T. G. Hack, T. Huang, and L. C. Stecher, "Traffic Service Position System No. 1B: Software Development System," B.S.T.J., this issue.

THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 3, March 1983
*Printed in U.S.A.*

## *Traffic Service Position System No. 1B:*

# Long-Range Planning Tools

By P. L. BASTIEN and B. R. WYCHERLEY

(Manuscript received August 9, 1982)

*This article describes the issues involved in operator services planning and the structure of the computer tools developed to address these issues. The approach to long-range planning of operator services networks is very flexible, and the availability of the Traffic Service Position System No. 1B further enhances this flexibility. This planning effort is a complex process that has been automated by computer tools that are both accurate and user-friendly.*

## I. INTRODUCTION

The planning of Traffic Service Position System (TSPS) installations and growth is rather different from that usually encountered in engineering central office equipment.[1] One of the reasons for this is the great flexibility that the engineer has in selecting the placement of operator position subsystems and how traffic will be routed to the various TSPSs that serve a particular area. The availability of the TSPS No. 1B offers new choices: formerly, when a TSPS exhausted its call processing capacity the alternatives were to divert the traffic overload to another TSPS that had surplus capacity or to purchase a new TSPS and divert load to it. Under the new system a TSPS No. 1 can be retrofitted to a TSPS No. 1B with much greater call capacity.[2] The resulting efficiencies, including avoidance of trunk rearrangements, trunk splintering, and opening of new operator groups, provide significant economic benefits to the operating telephone company. The TSPS No. 1B also provides the engineer more opportunities to consolidate, that is, to retrofit a subset of a group of TSPS No. 1's to TSPS No. 1B's and retire some or all of the remaining TSPS No. 1's in the group.

The Operator Services Traffic Network Planning System (OSTNPS) is a tool designed to aid telephone companies in making long-range plans for TSPS and related network evolution in order to guide short-range planning activities.[3] OSTNPS has been available to telephone companies since November 1981. Using interactive programs and a *UNIX** operating system environment, OSTNPS lets the user make the critical decisions while taking over the detailed calculations that can often severely limit the number of alternatives that a planner considers. OSTNPS combines maximum user flexibility with fast turn-around: the user can generate and analyze an alternative satisfying all the necessary constraints in about two hours of terminal time (CPU time is negligible). Since a typical alternative may involve 45 nodes and an associated network configuration over a 20-year period, design considerations and human factors engineering were of the utmost importance in developing OSTNPS.

Many of the factors to be considered existed for the TSPS No. 1 before the development of the TSPS No. 1B; the general approach taken with the program design made addition of the new degree of freedom relatively simple. This article describes the problems involved in planning TSPSs, and the software programs developed to guide the user in this planning.

## II. PLANNING ISSUES

### *2.1 Long-range planning*

The purpose of long-range planning is to evaluate the results of the most likely trends in traffic, services, and technology in order to optimize the long-term results of short-range decisions. The usual long-range plan covers a period of 20 years. The procedure consists of generating a set of alternatives and comparing the economic worth of each. The alternatives usually include a continue-as-is base plan with which the other alternatives are compared.

Because 20-year forecasts involve many potential contingencies, the long-range planner must apply judgment and experience to select the most reasonable alternatives and to perform appropriate sensitivity analysis on these alternatives.

### *2.2 Operator services long-range planning issues*

The choices specific to operator services include the following.

#### *2.2.1 Purchase of new nodes*

Usually the purchase of a new TSPS, Remote Trunk Arrangement

---

* Trademark of Bell Laboratories.

(RTA), or Position Subsystem (PSS) is made to relieve the exhausts of existing nodes.[4] The planner must decide where to place the new node and where its traffic is to come from. For example, a new TSPS may be loaded in such a way as to delay its exhaust and that of neighboring systems for as long as possible, or it may be placed in a way such that its resultant traffic is routed over facilities that are as short as possible. Usually a new PSS is placed where there is a large operator labor market. The issues involved in an RTA purchase will be discussed in Section 2.2.4.

#### *2.2.2 Retrofit to TSPS No. 1B*

A TSPS No. 1B has greater real-time and memory capabilities than a TSPS No. 1. If the TSPS No. 1 exhausts real time or memory, then the savings in rearrangements and/or new node purchases must be weighed against the costs of retrofit and, in a growth environment, the eventual purchase of a new node.

#### *2.2.3 Load balancing*

Load balancing consists of moving trunks, and hence traffic, from one node to another in such a way that all exhausts are relieved. Load balancing eliminates the capital expenditure of a new operator node in the given year, but the purchase is only deferred and load balancing can be quite expensive in itself because of trunk rearrangements.

#### *2.2.4 RTA issues*

An RTA concentrates trunks and homes them in on a nearby TSPS via base-remote (BR) trunks.[4] The planning issues specific to RTAs are as follows.

- Direct versus RTA trunking. If a set of local offices is located a long distance (say 100 miles) from the TSPS on which the offices are to be homed, one alternative is to trunk them directly to the TSPS and another is to concentrate these trunks via an RTA. In the former case, trunk and facility costs dominate; in the latter, purchase of an RTA is necessary. In general, purchasing an RTA is desirable if many trunks are involved and the distance to the TSPS is large.
- Colocated RTA. If a TSPS reaches a condition of trunk exhaust, an RTA may be purchased and colocated with the TSPS: the RTA concentrates some of the TSPS incoming trunk calls and relieves the exhaust. Further, a colocated RTA-TSPS pair gives more flexibility in load balancing if the TSPS later becomes exhausted for any other reason: the RTA could then be rehomed on a nearby TSPS with spare capacity. An RTA rehome is defined as a move of its base-remote trunks; if the RTA were not there, a

move of about eight times as many TSPS incoming trunks would be necessary.

#### 2.2.5 PSS rehoming

TSPS operators are assigned to PSSs, consisting of groups of operator consoles and the associated intelligence.[4] A move involving TSPS/RTA traffic usually means that position requirements at that TSPS will change. If a sufficient amount of traffic is moved from one TSPS to another, one or more PSSs may be rehomed between these TSPSs. The planner generally has great flexibility in these rehomings except for mileage constraints.

#### 2.2.6 Impact on toll trunking

Sometimes a move of TSPS/RTA traffic impacts on toll trunking as well. Since TSPS and RTA incoming trunks are bridged between the local office and toll switch, a move of these trunks implies a corresponding connect/disconnect of trunks at the associated toll switches. Moreover, such a move will also cause a change in intertoll trunking. Every local area generates operator traffic that returns to itself via the toll switch. When TSPS/RTA incoming trunks are moved from one toll switch to another, the traffic must have a path from the second toll switch back to the first via Direct Distance Dialing (DDD) trunks. This path would not have been necessary had the move not taken place. On the other hand, an RTA rehome does not change toll trunking since only BR trunks are involved. Detailed examples of these issues will be considered in Section IV.

### 2.3 New considerations with TSPS No. 1B

A TSPS No. 1 that exhausts real time or memory can be relieved by retrofitting a 3B20D Processor in place of the present Stored Program Control No. 1A—a procedure that converts TSPS No. 1 to TSPS No. 1B.[2] The demand for the 3B20D Processor for TSPSs existed because real time or memory is the limiting factor on most TSPSs today. From a planning point of view, this method of relief is easiest because no traffic rearrangements are needed.

#### 2.3.1 Selective conversion to TSPS No. 1B

Because of capital constraints, a planner may convert some but not all exhausted TSPS No. 1's to TSPS No. 1B in a given year. This strategy introduces new effects on planning: a combination of conversion to TSPS No. 1B, TSPS/RTA purchases, and load balancing may take place. Consider the following example. Suppose that both TSPSs A and B (both TSPS No. 1's) exhaust real time in some study year and that a single conversion to TSPS No. 1B is to be made in that

year. The planner retrofits A, giving it extra real-time capacity. To relieve B, the planner takes the following steps: two RTAs are purchased, colocated with B, loaded with incoming trunks from B, and homed on A. B is thus relieved and survives real-time exhaust two more years, at which time B is retrofit and the RTAs are rehomed back to B.

#### *2.3.2 TSPS consolidation*

Planning issues are different in low-growth regions: the added real-time capacity of TSPS No. 1B increases the possibility of TSPS consolidation. First a consolidation candidate must be selected from a group of nearby TSPSs. It may be the oldest, most centrally located, or most lightly loaded TSPS of the group. The TSPS is retired by moving all of its traffic to the other TSPSs in the group. After consolidation, the remaining TSPSs may have to be converted to TSPS No. 1B earlier than if consolidation had not taken place, since they will exhaust sooner.

## III. USER-CONTROLLED PLANNING TOOL

### *3.1 Design considerations*

OSTNPS was developed so as to combine extreme flexibility with very fast turnaround. Extreme flexibility means that the user can generate any alternative desired—the programs check that no constraints are violated along the way. Very fast turnaround means that the input database and modeling must be both representative and simple, so that generation and analysis of an alternative can be done in an hour or two. A typical operator services alternative consists of 5 TSPSs, 8 RTAs, 20 PSSs, and a dozen toll switches all evolving as a function of time with consolidation, purchase of new nodes, or load balancing. The usual study period is 20 years so that, with a base year, there may be up to 21 toll-connect and intertoll sets of trunk requirements ("trunk fields") associated with the alternative. A typical number of such solutions is ten.

OSTNPS does not optimize but rather lets users generate their own alternatives. For a given study area, it is not known how close a given plan is to the optimum because the optimum is not known to begin with. However, analysis of all possible alternatives for a given simple study area suggests that the economic worth of a typical "intelligent" user plan will be within 5 percent of the optimal plan.

### *3.2 Modeling considerations*

Modeling considerations are as important as design considerations, because inputs must be at once representative and simple if the objective of fast turnaround is to be met.

### 3.2.1 Input database

For mechanized long-range planning, it is desirable to obtain reasonable results using data that are easy for users to collect. OSTNPS requires comparatively little user input data, and all data come from standard sources. Implied in the small size of the input data base are several assumptions—summarized below—used to make forecasts.

The input database contains only three types of call volumes: total trunk seizures on TSPS (whether or not an operator is accessed), trunk seizures that reach an operator for any purpose, and trunk seizures that reach an operator only for Operator Number Identification (ONI) of the calling party. These three volumes, for every TSPS and study year, are the primary inputs into all of the forecasting algorithms.

Some of the algorithms are linear regressions based on these volumes. One such regression forecasts required operator consoles (positions), the independent variable in this case being the product of the number of attempts reaching an operator and the average operator work time per call (both input database items). In reality, the relationship is not quite linear because large operator teams are more efficient than small operator teams. However, this lack of linearity occurs primarily in lightly loaded TSPSs; for moderately to heavily loaded TSPSs, a linear relation is sufficient.

Simplifying assumptions are also made in modeling the required TSPS trunks. First, minor trunk types are considered as a single category, a simplification that cuts down required input data considerably. Second, for each trunk type considered by the program, it is assumed that groups are of equal size, and that each group handles the same number of attempts with the same holding time per attempt. The average trunk group size is an input item, and the holding time is derived from input items. These approximations are quite suitable for long-range planning, and because of them, the input database is made far easier for the user to create.

### 3.2.2 Network

Even though OSTNPS is fundamentally an operator node planner, there must be a way to estimate the cost of trunk movements resulting from shifts in loads. Trunk movements occur in the toll-connect and intertoll fields. In the toll-connect field, it is adequate to assume that when a deload occurs, the appropriate number of trunks are disconnected from the deloaded operator node and the same number reconnected to the other (if the two operator nodes are colocated, the costs are different than if they are not, and this is properly taken into account). This information, together with percentages of 2-wire and 4-wire bridges, facilities, and average mileages between the local offices and toll switches before and after, is sufficient to get a good estimate

of the toll-connect movement costs. Intertoll movement also occurs after a deload since the load on all toll switches is affected, depending on community of interest. It would be unnecessarily complex to reengineer the toll network after each deload or purchase. Instead, the change in offered load between every pair of toll switches is estimated and converted to trunks. Because intertoll trunk groups are generally larger and thus more efficient than toll-connect trunk groups, it is assumed that an intertoll trunk carries twice the load as a toll-connect trunk.

## IV. STRUCTURE OF PROGRAMS

### *4.1 Introduction*

The job of generating and evaluating a long-range operator services alternative can be split into four separate tasks, each programmed as a separate OSTNPS module. Figure 1 shows a general flowchart of the process, and Fig. 2 illustrates the scope of each of these programs in the overall operator services network.

(*i*) Node exhaust. The Node Exhaust program calculates requirements for each TSPS/RTA node in each year, informs the user about possible exhausts, and lets the user relieve the exhausts by purchasing new nodes, moving traffic, or converting to TSPS No. 1B. This program does not involve itself with the network connecting the nodes—this task is performed later.

(*ii*) PSS. The PSS program, given overall position requirements from the Node Exhaust program, assigns specific PSSs for these requirements and builds the network joining those PSSs to the TSPSs.

(*iii*) Remainder of network. The Network program assigns specific trunk groups between local areas, TSPS/RTA's, and toll switches. As discussed in Section 4.4, the detailed network can be reconstructed given the global requirements output by the Node Exhaust program for each TSPS/RTA.

(*iv*) Translation for Economic Analysis. The Translator program combines the output of the preceding two programs and transforms it into a "network file" for the economic evaluator program TASP (Toll Alternatives Studies Program).[5] TASP, a very detailed economic evaluator based on Capital Utilization Criteria (CUCRIT), requires a user-supplied "cost file" as well as the above network file.

Following is a detailed description of each of these programs.

### *4.2 Node Exhaust program*

#### *4.2.1 Summary*

The Node Exhaust program is the first module of OSTNPS. Starting with a set of user-supplied input data describing the operator nodes

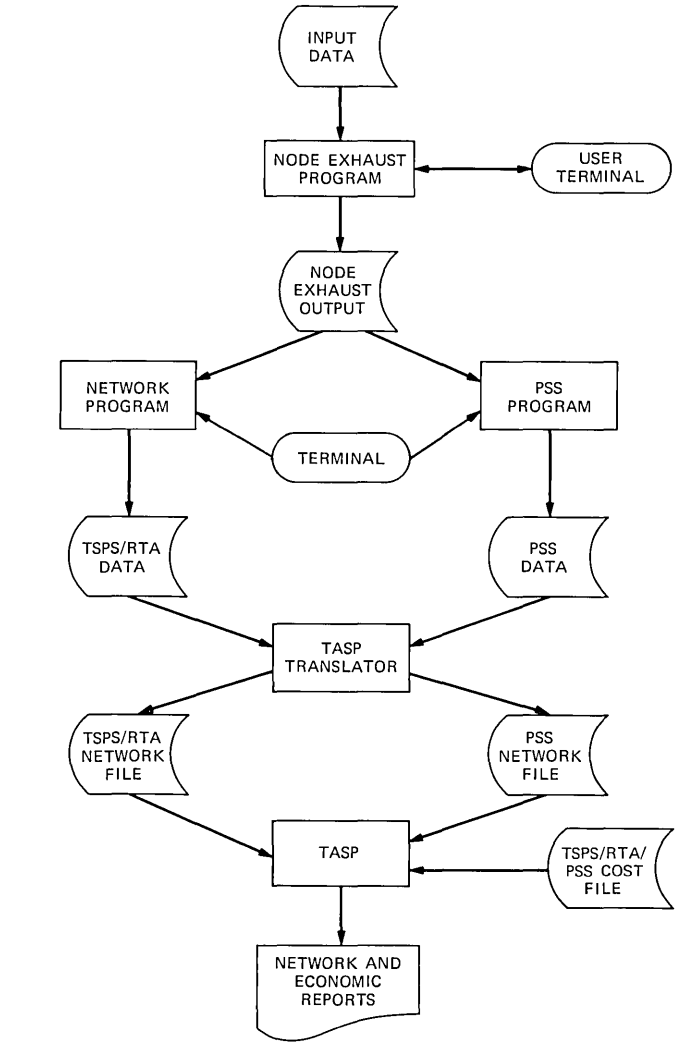
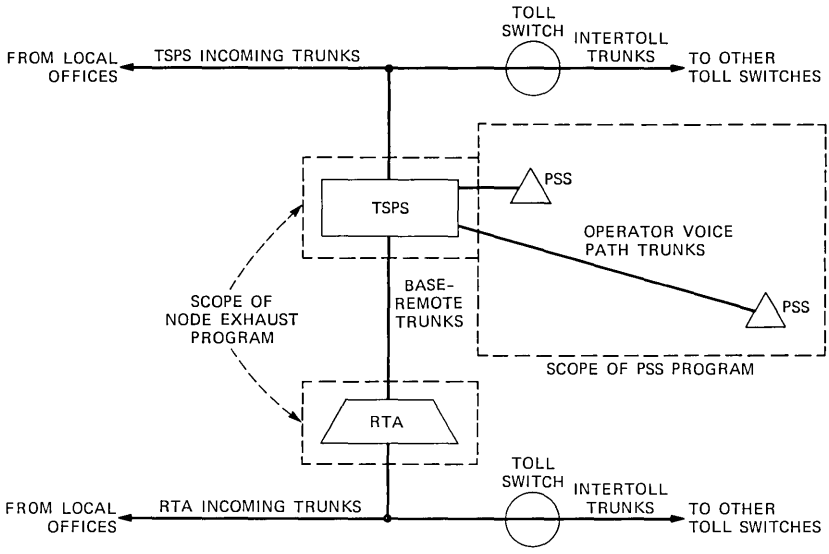



Fig. 1—Flowchart of OSTNPS.

(TSPSs and RTAs) in the study area, the program generates part of an alternative interactively. The alternative is refined further in succeeding OSTNPS modules. Figure 3 shows a general flowchart of the Node Exhaust program.

For each study year, the program forecasts requirements for relevant TSPS/RTA items and compares these numbers with their respective




PSS - POSITION SUBSYSTEM
RTA - REMOTE TRUNK ARRANGEMENT
TSPS - TRAFFIC SERVICE POSITION SYSTEM

Fig. 2—Scope of OSTNPS program.

capacities. If an exhaust occurs, the program prints out the operator node that is exhausted, along with the cause(s) of exhaust, and a complete status report on all other operator nodes. The user is given an opportunity to purchase new nodes, convert to TSPS No. 1B, or to move traffic from one node to another. The program will proceed to the next study year only if there are no unrelieved exhausts. After completion of the last study year, the alternative is stored in a data base to be used by subsequent modules of OSTNPS.

It should be emphasized that the user interacts with the program on a year-by-year basis while the program is running. It is not necessary for the user to know exactly what traffic moves and node purchases to make before the study begins.

#### *4.2.2 Description of exhaust causes*

The Node Exhaust program considers TSPS/RTA exhaust causes of several types. The most common type of exhaust for a TSPS No. 1 is real time; with the development of TSPS No. 1B and its associated increase in real-time capacity, a TSPS may exhaust owing to a number of other causes. Following is a description of other exhaust causes.

***4.2.2.1 TLN terminations and individual trunk types.*** Various types of

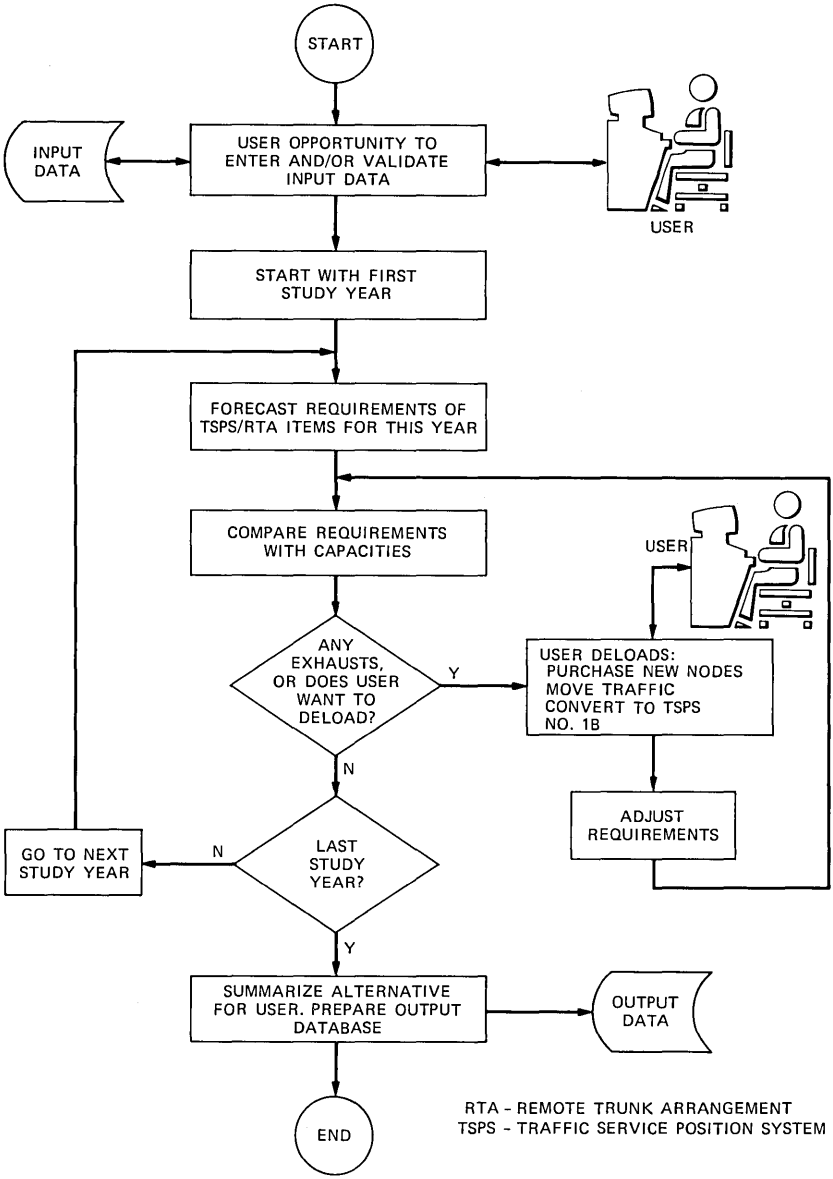



Fig. 3—Flowchart of Node Exhaust program.

TSPS trunks terminate on Universal Trunk Frames (UTFs) which in turn attach to the Trunk Link Networks (TLNs). These trunk types include: incoming ("universal") trunks, Transfer Centralized Automatic Message Accounting (XCAMA) trunks, Base-remote (BR) trunks, and all other TSPS trunks considered as a single category.[6]

A TSPS becomes TLN-exhausted if, after engineering each TLN trunk type and fitting these trunks on UTFs, the required number of UTFs (including an administrative margin) exceeds the hardware limitation. In addition, a TSPS can exhaust if one of the individual trunk types exceeds its software, design, or hardware limitation.

*4.2.2.2 PLN appearances and positions.* The Position Link Networks (PLNs) contain terminations by which operator positions and various service circuits (receivers, outpulsers, announcement circuits, etc.) connect to the TSPS. As in the TLN case, a TSPS becomes PLN-exhausted if, after engineering positions and service circuits, the total number of PLN appearances exceeds hardware limitations.

*4.2.2.3 TSPS network.* TSPS traffic travels between trunks requiring service and circuits/operators providing service via the TSPS network. A given trunk can connect to a given position or service circuit via one of eight paths composed of "A," "B," and "C" links. The TSPS network is limiting if its offered Erlang load is sufficiently great to result in a blocking probability greater than 0.001.

*4.2.2.4 TSPS office data memory.* TSPS memory is subdivided into two types: the generic program and office data. Office data memory contains information specific to the design and traffic characteristics of a given TSPS site. Conversion of a TSPS No. 1 to a TSPS No. 1B increases the total amount of addressable memory as well as real time.

### *4.2.3 Input data*

The input data to the Node Exhaust program include, for each TSPS/RTA in the study area, base year requirements for various trunk types as well as current and forecast call volumes of various types.

The following data are required:

- Call volumes: trunk seizures, position seizures, and XCAMA trunk seizures
- Trunks: base and RTA incoming trunks, XCAMA trunks, and all other trunks except BR
- Real-time capacity in trunk seizures
- Operator Average Work Time (AWT).

### *4.2.4 Forecast of TSPS/RTA items*

For a given study year, the Node Exhaust program uses these input data to forecast TSPS/RTA items that determine exhausts. The program uses simple formulas or linear regressions in place of detailed calculations.

Figure 4a shows a sample printout of these items from an actual study. At user request, this printout will appear in any study year while the Node Exhaust program is running.

| | TSP1 | TSP2 | TSP3 |
|---|---|---|---|
| BASE INC TSZ | 5226 | 11730 | 8241 |
| XCAMA TSZ | 800 | 1300 | 1200 |
| RTA RTA1 - TSZ | 2264 | 0 | 0 |
| RTA RTA2 - TSZ | 1144 | 0 | 0 |
| RTA RTA3 - TSZ | 1116 | 0 | 0 |
| RTA RTA4 - TSZ | 1400 | 0 | 0 |
| RTA RTA5 - TSZ | 0 | 0 | 984 |
| RTA RTA6 - TSZ | 0 | 0 | 1230 |
| RTA RTA7 - TSZ | 0 | 0 | 1038 |
| RTA RTA8 - TSZ | 0 | 0 | 1146 |
| TOTAL NON-XC TSZ | 11150 | 11730 | 12639 |
| TOTAL TSZ | 11950 | 13030 | 13839 |
| R-T CAPY (TSZ) | 16007 | 16587 | 17621 |
| REALTIME - % RTC | 74 | 78 | 78 |
| BASE INC TKS | 555 | 1786 | 959 |
| XCAMA TKS | 24 | 36 | 24 |
| RTA RTA1 - INC TKS | 283 | 0 | 0 |
| RTA RTA2 - INC TKS | 143 | 0 | 0 |
| RTA RTA3 - INC TKS | 186 | 0 | 0 |
| RTA RTA4 - INC TKS | 280 | 0 | 0 |
| RTA RTA5 - INC TKS | 0 | 0 | 246 |
| RTA RTA6 - INC TKS | 0 | 0 | 246 |
| RTA RTA7 - INC TKS | 0 | 0 | 173 |
| RTA RTA8 - INC TKS | 0 | 0 | 191 |
| OTH BASE TKS, EX BR | 73 | 213 | 68 |
| TLN - TOTAL UTF | 5 | 8 | 6 |
| POSITIONS (TRAFFIC) | 71 | 80 | 53 |
| PLN - TOTAL APPR | 276 | 300 | 251 |
| NETWORK (ERLANGS) | 75 | 92 | 105 |
| MEMORY (OFC NMCDS) | 4 | 5 | 5 |
| POSITION SEIZURES | 7205 | 8082 | 5552 |

Fig. 4(a)—Node Exhaust printout before user move—1984.

The following forecasts are made:

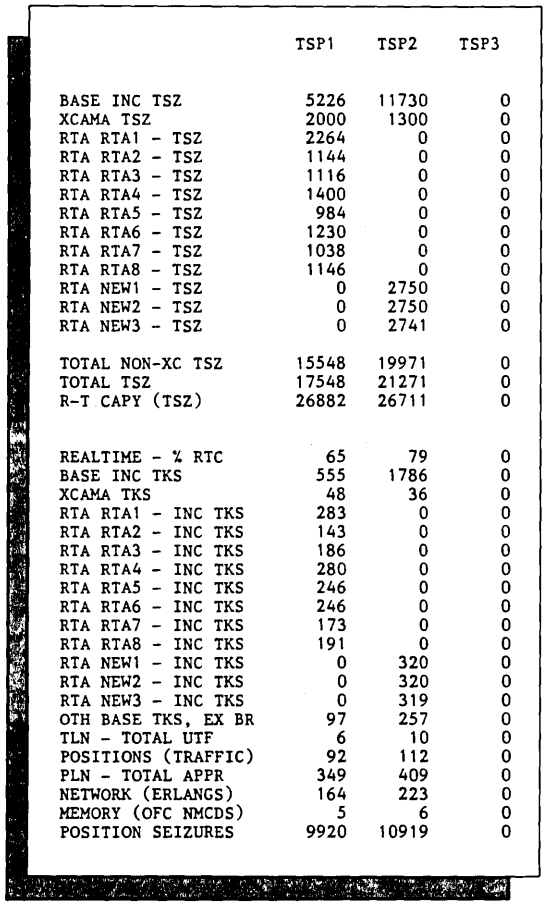
- Real-time usage
- Trunks of various types
- Memory
- Operator positions
- Service circuits
- TSPS network load.

#### 4.2.5 Exhaust prediction

Once the above items are forecast in a given study year for every TSPS/RTA in the study, the program compares these numbers with their respective upper limits. The program thus determines for each TSPS/RTA whether it is exhausted and for what reasons.

#### 4.2.6 User-specified changes to network

Input commands to the Node Exhaust program allow the user to purchase new operator nodes or move traffic from one node to another



| | TSP1 | TSP2 | TSP3 |
|---|---|---|---|
| BASE INC TSZ | 5226 | 11730 | 0 |
| XCAMA TSZ | 2000 | 1300 | 0 |
| RTA RTA1 - TSZ | 2264 | 0 | 0 |
| RTA RTA2 - TSZ | 1144 | 0 | 0 |
| RTA RTA3 - TSZ | 1116 | 0 | 0 |
| RTA RTA4 - TSZ | 1400 | 0 | 0 |
| RTA RTA5 - TSZ | 984 | 0 | 0 |
| RTA RTA6 - TSZ | 1230 | 0 | 0 |
| RTA RTA7 - TSZ | 1038 | 0 | 0 |
| RTA RTA8 - TSZ | 1146 | 0 | 0 |
| RTA NEW1 - TSZ | 0 | 2750 | 0 |
| RTA NEW2 - TSZ | 0 | 2750 | 0 |
| RTA NEW3 - TSZ | 0 | 2741 | 0 |
| TOTAL NON-XC TSZ | 15548 | 19971 | 0 |
| TOTAL TSZ | 17548 | 21271 | 0 |
| R-T CAPY (TSZ) | 26882 | 26711 | 0 |
| REALTIME - % RTC | 65 | 79 | 0 |
| BASE INC TKS | 555 | 1786 | 0 |
| XCAMA TKS | 48 | 36 | 0 |
| RTA RTA1 - INC TKS | 283 | 0 | 0 |
| RTA RTA2 - INC TKS | 143 | 0 | 0 |
| RTA RTA3 - INC TKS | 186 | 0 | 0 |
| RTA RTA4 - INC TKS | 280 | 0 | 0 |
| RTA RTA5 - INC TKS | 246 | 0 | 0 |
| RTA RTA6 - INC TKS | 246 | 0 | 0 |
| RTA RTA7 - INC TKS | 173 | 0 | 0 |
| RTA RTA8 - INC TKS | 191 | 0 | 0 |
| RTA NEW1 - INC TKS | 0 | 320 | 0 |
| RTA NEW2 - INC TKS | 0 | 320 | 0 |
| RTA NEW3 - INC TKS | 0 | 319 | 0 |
| OTH BASE TKS, EX BR | 97 | 257 | 0 |
| TLN - TOTAL UTF | 6 | 10 | 0 |
| POSITIONS (TRAFFIC) | 92 | 112 | 0 |
| PLN - TOTAL APPR | 349 | 409 | 0 |
| NETWORK (ERLANGS) | 164 | 223 | 0 |
| MEMORY (OFC NMCDS) | 5 | 6 | 0 |
| POSITION SEIZURES | 9920 | 10919 | 0 |

Fig. 4(b)—Node Exhaust printout after user move—1984.

at any time. These steps are mandatory if there is an unrelieved exhaust; otherwise they are optional (as in the case of TSPS consolidation). The possible options are as follows:

- The user may convert a TSPS No. 1 to TSPS No. 1B.
- The user may purchase a new TSPS or RTA.
- The user may move TSPS or RTA incoming trunk seizures from one node to another.
- The user may rehome an RTA from one TSPS to another.
- The user may move XCAMA trunk seizures from one TSPS to another.
- The user may retire a TSPS or RTA.

If exhausts are still not relieved after such a series of steps, the

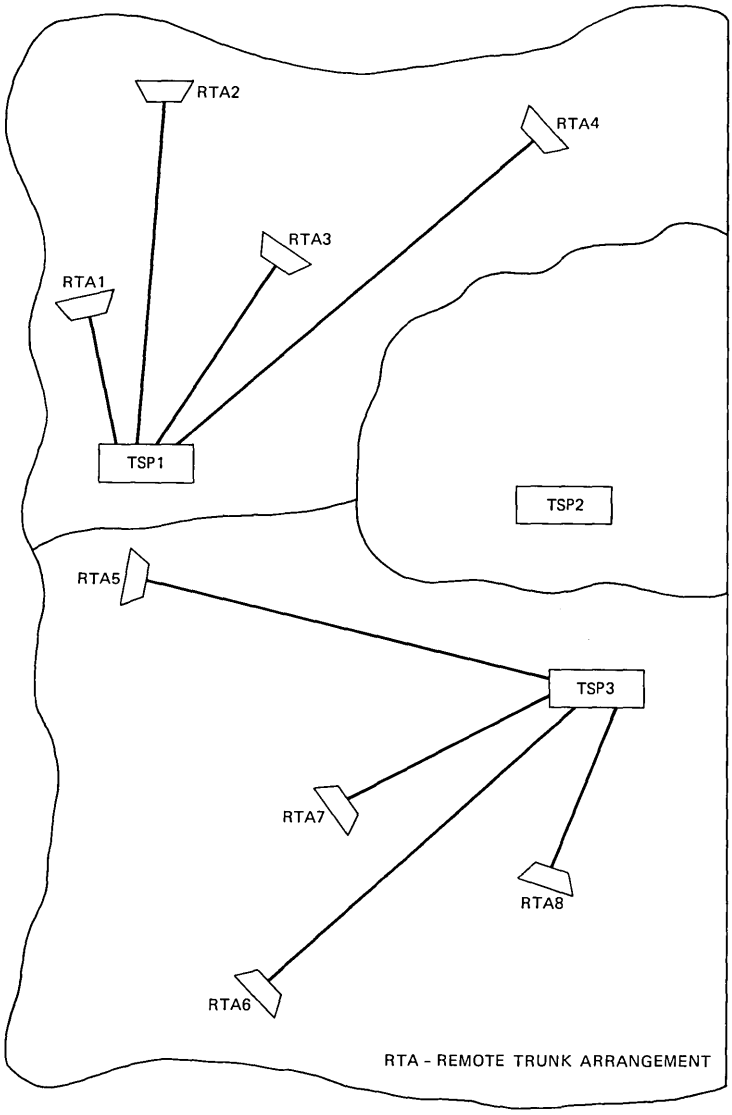



Fig. 5(a)—TSPS/RTA configuration before user move.

program will not proceed to the next study year. The user must start over with the given study year and try another strategy.

#### *4.2.7 Example*

Figures 4 and 5 illustrate the results of a set of moves made during a run of the Node Exhaust program. Figure 4 lists the relevant forecasts

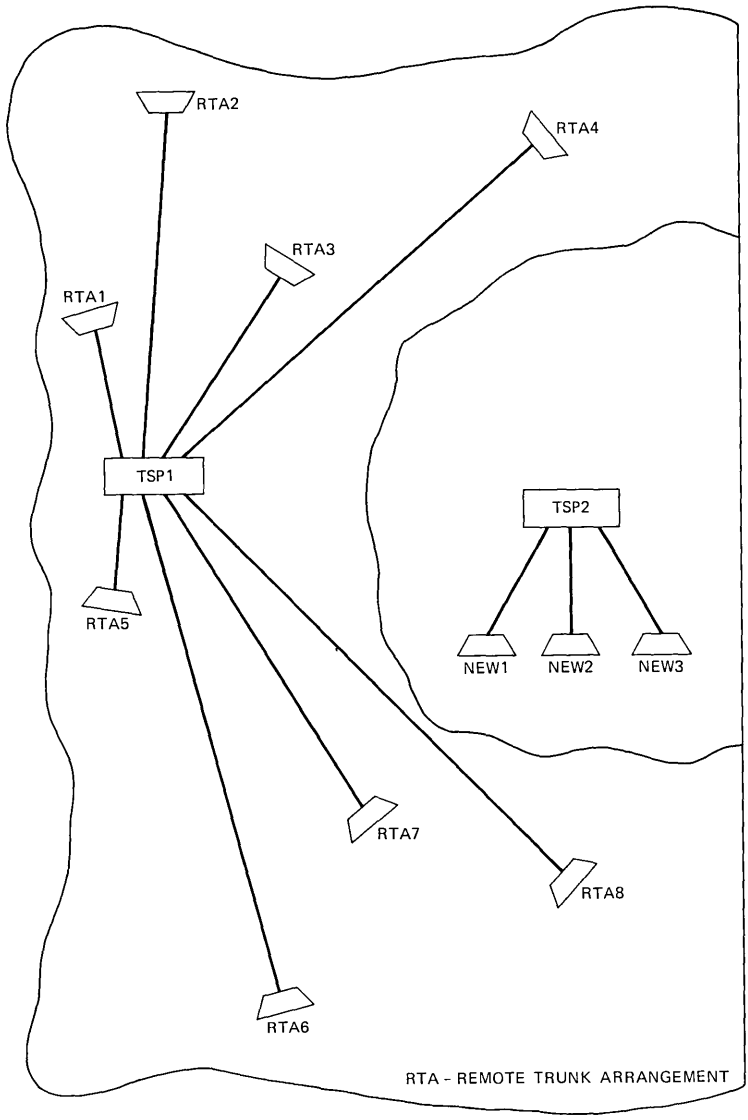



Fig. 5(b)—TSPS/RTA configuration after user move.

before and after the moves, and Fig. 5 shows a pictorial representation of this situation.

The study area consists of three TSPSs: TSP1, TSP2, and TSP3, all of which are TSPS No. 1's. These are the columns in Figures 4a and 4b. Each row represents a given forecast item for the study year, in this case, 1984. The tables are split into two sections: "User-Controlled

Items" (trunk seizures which the user may directly move from one operator node to another) and "Other Relevant Items" (exhaust-determining items which indirectly change as a result of such a move).

To evolve from Fig. 4a to 4b, the user retires TSP3 in the following way. First, RTAs RTA5, RTA6, RTA7, and RTA8, presently homed on TSP3, are rehomed to TSP1. Second, RTAs NEW1, NEW2, and NEW3 are purchased, homed on TSP2, and loaded with all of TSP3's base incoming traffic. Finally, all of TSP3's XCAMA traffic is moved to TSP1, leaving TSP3 with no traffic and hence retired. The remaining TSPSs TSP1 and TSP2 exhaust real time as a result of the move, and they are converted to TSPS No 1B's.

#### *4.2.8 Completion of run*

Once all years in the study period have been covered, the relevant information for the alternative is printed on the user's terminal. The information is also organized into an output database for use by the PSS and Network programs described below.

### *4.3 PSS program*

#### *4.3.1 Summary*

The main function of the PSS program is to establish a schedule for Position Subsystems (PSSs). This task can be separated from all others because its only input is the total required number of operator positions on each TSPS for each study year. Numbers of positions are established in the Node Exhaust program depending on incoming traffic to the TSPSs. Once the schedule is established, an additional task is to obtain facilities between each PSS and its home TSPS.

*4.3.1.1 PSS schedule.* The schedule is determined by the following considerations:

- Total number of positions required on each TSPS in each study year.
- Equipment actually in the field in the base year.
- The absolute and recommended maximum number of positions per PSS.
- The type of PSS (PSS1 or PSS2).*
- Minimization of the number of PSSs in the study, subject to administrative constraints and operator availability.

The solution consists in presenting to the user a schedule that is compatible with all the constraints, as in Fig. 6a, and letting the user modify it as desired by means of interactive commands, as in Fig. 6b.

Following is a description of Fig. 6a. First, equipment in the field

* PSS1 positions are no longer being manufactured, and PSS1s remain capped at their present position levels.

| | 1980 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TSPS1 | | | | | | | | | | | | | | | | | | | | | |
| PS11* | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 |
| PS12 | 30 | 18 | 23 | 27 | 33 | 34 | 36 | 38 | 40 | 42 | 45 | 48 | 50 | 50 | 28 | 30 | 33 | 35 | 38 | 40 | 43 |
| PS13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS2 | | | | | | | | | | | | | | | | | | | | | |
| PS21* | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 |
| PS22* | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 |
| PS23 | 30 | 0 | 0 | 0 | 14 | 14 | 17 | 20 | 23 | 25 | 28 | 32 | 35 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS3 | | | | | | | | | | | | | | | | | | | | | |
| PS31* | 55 | 55 | 55 | 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PS32* | 28 | 28 | 28 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PS33 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS4 | | | | | | | | | | | | | | | | | | | | | |
| PS41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| PS42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 27 | 29 | 32 | 34 | 37 | 40 | 43 |

Fig. 6(a)—A first-order PSS field.

| | 1980 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TSPS1 | | | | | | | | | | | | | | | | | | | | | |
| PS11* | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 |
| PS12 | 30 | 18 | 23 | 27 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 28 | 30 | 33 | 35 | 38 | 40 | 43 |
| PS13 | 0 | 0 | 0 | 0 | 3 | 4 | 6 | 8 | 10 | 12 | 15 | 18 | 20 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS2 | | | | | | | | | | | | | | | | | | | | | |
| PS21* | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 | 55 |
| PS22* | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 47 |
| PS23 | 30 | 0 | 0 | 0 | 14 | 14 | 17 | 20 | 23 | 25 | 28 | 32 | 35 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS3 | | | | | | | | | | | | | | | | | | | | | |
| PS31* | 55 | 55 | 55 | 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PS32* | 28 | 28 | 28 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PS33 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TSPS4 | | | | | | | | | | | | | | | | | | | | | |
| PS41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| PS42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 27 | 29 | 32 | 34 | 37 | 40 | 43 |

Fig. 6(b)—PSS field after the first move.

(base year) is entered by the user (1980 in this case). PSS1's, for which positions are capped, are marked with a "*". For the study years (1981–2000), the scheduler bases its calculations on position requirements and purchases new PSSs where needed.

Given this PSS schedule, the user can modify it with the following interactive commands:

- For a given TSPS, the user may redistribute the number of PSS2 positions.
- The user may rehome a PSS2 from one TSPS to another.

- The user may or may not release unneeded PSS2 positions for reuse.

Consider Fig. 6b. The first move consists in all excess positions beyond 30 in the base year being moved from PS12 to PS13 for all study years up to 1994. A possible second move consists of making PS33, PS13, and PS42 the same PSS by rehoming PS33 (originally homed on TSPS3) to TSPS1 in 1984, then to TSPS4 in 1994.

*4.3.1.2 PSS facilities.* Once the schedule is established, the program asks the user for facility data between TSPSs and the PSSs. This, together with the schedule, completely determines the input to the economic analysis program.

### *4.4 TSPS/RTA Network program*

Besides the network joining TSPSs to PSSs, there is the network joining local offices and toll switches to TSPS/RTA's. This network can be divided into three distinct portions:

- Toll connect trunks (local office to TSPS/RTA to toll switch).
- Base remote trunks (RTA to TSPS).
- Intertoll trunks.

Figure 7b shows an example of such a network.

The Node Exhaust program allocated total trunk seizures amongst the TSPS/RTA's without making any assumptions about the network. The separation of exhaust planning and network planning allows simplified program design, and allows the user to concentrate on each issue separately. The method used to model the network allows this separation to work; the network program guides the user through the steps of establishing the network corresponding to the alternative formulated by the Node-Exhaust program.

At the beginning of the study, there is by definition a one-to-one correspondence between "local areas" and TSPS/RTAs (Fig. 7a). User-specified traffic moves made during a node exhaust run may affect the network in the following ways (see Fig. 7b):

- A move of base-remote trunks, also called an RTA rehome (a) in Fig. 7b.
- A move of toll-connect trunks (b) in Fig. 7b.
- Changes in the intertoll network. A certain percentage of operator traffic leaving a local area must go back to the same local area via its toll switch. Corresponding to each toll-connect move (b) is an increase in required intertoll trunks (b'). No such intertoll adjustment is necessary for a base-remote move.

The Network program obtains the following information from the user:

- Toll switch information: names, types, homing TSPS/RTAs.
- Base-remote mileage and facility information.

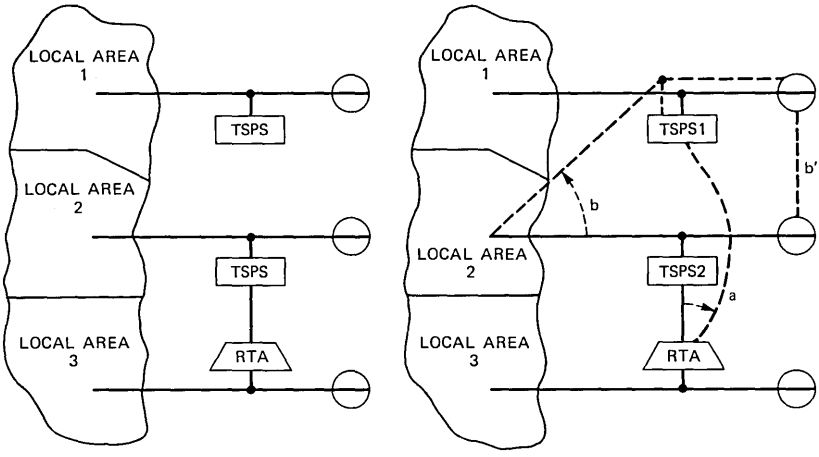



Fig. 7—(a) Relationship of local offices and toll switches to TSPS/RTA's—the correspondence is one to one. (b) Effect on network of user-specified traffic moves during a Node Exhaust run.

- Toll-connect mileage and facility information, including the percentage of 2-wire and 4-wire bridges.
- Community of interest and intertoll mileage/facility information.

### *4.5 Economic evaluation and translator programs*

Economic evaluation of the alternatives is performed by TASP, which has already been mentioned. Telephone company planners are very familiar with this program, which has been in use for years. TASP requires a considerable amount of input, even for studies involving only a few switches. A fourth program, the Translator program, transcribes the output of the three preceding programs into TASP format without user intervention. The PSS and TSPS/RTA Network programs write files in a compact language which is translated into TASP language by the Translator program.

## V. RUNNING THE PROGRAMS

The generation and economic evaluation of a complete alternative with OSTNPS involves the following steps to be taken by the user:

- Run the Node Exhaust program.
- For the above Node Exhaust program alternative, run one or a few appropriate PSS alternatives.
- Run each of these PSS alternative through TASP and choose the best.

- Run the Network program for the above Node Exhaust program alternative.
- Concatenate the best PSS alternative with the Network program alternative, and run this complete alternative through TASP.

This TASP output constitutes the economic evaluation of this complete alternative, to be compared with others.

## VI. CONCLUSION

OSTNPS is an operator services long-range planning tool. It allows the user to analyze in a short time different strategies of operator services network evolution. For high-growth areas, OSTNPS evaluates various types of load-balancing alternatives versus purchase of new equipment. For low-growth areas with underutilized nodes, OSTNPS helps the user determine whether or not to retire those nodes. For both of these cases, OSTNPS allows the user to easily incorporate the conversion to TSPS No. 1B into the planning process. OSTNPS is not an optimizer, but instead lets users generate and evaluate their own alternatives.

## VII. ACKNOWLEDGMENTS

The authors wish to thank R. G. Crafton of AT&T, who contributed in innumerable ways, and the operator services planning organizations of Mountain Bell Telephone and Bell of Indiana, who helped us during program development.

## REFERENCES

1. R. J. Jaeger, Jr. and A. E. Joel, Jr., "TSPS No. 1: System Organization and Objectives," B.S.T.J., *49,* No. 10 (December 1970), pp. 2417–43.
2. R. E. Staehler and J. I. Cochrane, "Traffic Service Position System No. 1B: Overview and Objectives," B.S.T.J., this issue.
3. R. G. Crafton, "Network Planning for Operator Services Systems," Int. Switching Symp., Montreal, 1981, Session 21, C Paper 3, pp. 1–5.
4. S. M. Bauman, R. S. DiPietro, and R. J. Jaeger, Jr., "TSPS No. 1: Remote Trunk Arrangement: Overall Description and Operational Characteristics," B.S.T.J., *58,* No. 6 (July–August 1979), pp. 1119–35.
5. B. H. Fetz and P. M. Moricz, "Tipping the Scales for No. 4 ESS," Bell Lab. Rec., *58,* No. 5 (May 1980), pp. 138–45.
6. W. K. Comella, C. M. Day, Jr., and J. A. Hackett, "TSPS No. 1: Peripheral Circuits," B.S.T.J., *49,* No. 10 (December 1970), pp. 2561–623.

## ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| 3B20D | 3B20 Duplex Processor |
| 3bas | 3BD assembler |
| 3bcc | 3BSGS C compiler |
| 3bld | 3B20D link editor |
| 3bldp | 3B20D process loader |
| 3BPEST | 3B20D inhibits |
| 3BSGS | 3B20D Software Generation System |
| ACHI | application channel interface |
| ACPDL | application control process data library |
| AIM | application integrity monitor |
| ALU | arithmetic/logic unit |
| AMA | automatic message accounting |
| APE | advanced processor editor |
| ARA | answer register A |
| ARB | answer register B |
| AST | announcement stores |
| ASW | all seems well |
| ATB | address translation buffer |
| AWT | average work time |
| BBR | buffer bus to register |
| BPI | bit per inch |
| BR | base remote |
| BTR | bus terminating resistor |
| BVA | billing validation application |
| CALE | call to emulation |
| CAR | channel address register |
| CBB | constant to buffer bus |
| CBT | communications bus translator |
| CC | central control |
| CC | control console |
| CCIO | central control input/output |
| CCIS | common channel interoffice signaling |
| CD | control and display |
| C&D | Control and Display mode |
| CDR | channel data register |
| CDT | control display and test |
| CHK/GEN | check/generator |
| CIP | Critical Indicator panel |
| CLRFLG | clear flag |
| CMON | craft monitor |
| CMS | Change Management System |
| CPDB | central pulse distributor bus |

| | |
|---|---|
| CPD | central pulse distributor |
| CRC | Change Review Committee |
| CR | correction report |
| CRT | cathode ray tube |
| CSOP | controller of output spooler process |
| CSS | computer subsystem |
| CU | control unit |
| CUCRIT | Capital Utilization Criteria |
| CU/PSI | control unit/peripheral system interface |
| CUPERPH | control unit peripheral problem |
| DAP | display administration process |
| DFC | disk file controller |
| DFC/MHD | disk file controller with removable head disk |
| DGDL | diagnostic data library |
| DIAMON | diagnostic monitor |
| DIST | distributes |
| DMA | direct memory access |
| DMERT | Duplex Multi-Environment Real-Time |
| DMU | data manipulation unit |
| DRMO | detect rightmost one |
| DSCH | dual serial channel |
| DUC | dual utility circuit |
| DZRMO | detect and zero rightmost one |
| EA | enable address |
| EAI | emergency action interface |
| ECD | equipment configuration data |
| ECD | Equipment Configuration Database |
| ECD/SG | Equipment Configuration Database/System Generation Database |
| ECIO | executive control for input/output |
| ECMP | executive control for the main program |
| EGBN | execute go back to normal |
| EIH | error-interrupt handler |
| EL | execution level |
| ERD | emergency recovery display |
| ESS | Electronic Switching System |
| EXC | execute |
| FLZ | find low zero |
| FSI | file system interface |
| FTS | field test set |
| GBNHJ | go back to normal H-level or J-level |
| GRASP | Generic Access Program |
| HOBIS | Hotel Billing Information System |
| HSR | hardware status register |

| | |
|---|---|
| I | inhibit interrupt |
| IB | instruction buffer |
| ILAF | interrupt-level activity flags |
| IM | interrupt mask |
| INT | interrupt |
| INTS | interrupt source register |
| I/O | input/output |
| IOP | input/output processor |
| IS | interrupt source |
| ISR | interrupt source register |
| JCL | job control language |
| LMP | local maintenance position |
| LOTS | local toll simulator |
| LSS | Laboratory Support System |
| MAIS | maintenance interrupt sources |
| MCC | maintenance control center |
| MCC | master control center |
| MCCD | maintenance control center data |
| MCCI | maintenance control center interrupts |
| MCH | maintenance channel |
| MCRT | maintenance CRT |
| MESS | microprocessor service evaluation system simulator |
| MHD | moving head disk |
| MICLOB | microprocessor-controlled load box |
| MIP | micro-level test set interface program |
| MIRA | maintenance input request administrator |
| MLTS | micro-level test set |
| MNA | minor audit initialization |
| MOPS | microprocessor operator position simulator |
| MR | maintenance register |
| MR | memory to register |
| MR | modification request |
| MSEC | millisecond clock state |
| MS | master scanner |
| MTTR | mean time to repairs |
| MTTYC | maintenance teletype controller |
| MTTY | maintenance teletypewriter |
| MVS | multiple virtual storages |
| OBB | ones to buffer bus |
| ONI | operator number identification |
| OSTNPS | Operator Services Traffic Network Planning System |
| OST | operating system trap |
| PA | present address |
| PBCP | peripheral bus control process |

| | |
|---|---|
| PC | peripheral control |
| PC | peripheral controller |
| PC | protocol converter |
| PCB | process control block |
| PCPEIH | processor control process error-interrupt handler |
| PCPMD | processor control process maintenance driver |
| PD | peripheral device |
| PD | power distribution |
| PEST | interrupt inhibits |
| PIR | program interrupt request |
| PLN | position link network |
| PPR | pulse point register |
| PROM | programmable read-only memory |
| PSI | peripheral system interface |
| PSIDGDR | peripheral system interface diagnostic driver |
| PSIDIAGC | peripheral system interface diagnostic control |
| PSIEIH | peripheral system interface error-interrupt handler |
| PSS | position subsystem |
| PSS | Program Support System |
| PSW | program status word |
| PTU | program tape unit |
| PUAB | peripheral-unit address bus |
| PUMS | peripheral-unit maintenance summary |
| RAM | random access memory |
| RBB | register to buffer bus |
| RC | reply check |
| RCVP | recent change/verify position |
| RETE | return to emulation |
| REX | routine exerciser |
| RJE | remote job entry |
| RM | register to memory |
| RMU | rotate mask unit |
| ROP | read-only printer |
| ROP | receive-only printer |
| RRM | registers to memory |
| RTA | remote trunk arrangement |
| SAB | scanner answer bus |
| SAR | store address register |
| SCAN | scans |
| SCCS | Source Code Control System |
| SCCS | Switching Control Center System |
| SCC | Switching Control Center |
| SCR | store control register |
| SC/SD | scanner/signal distributor |

| | |
|---|---|
| SDR | store data register |
| SD | signal distributor |
| SDS | Software Development System |
| SER | system evaluation run |
| SES | Service Evaluation System |
| SG | system generation |
| SIA | system initialization A |
| SIB | system initialization B |
| SIM | system integrity monitor |
| SIR | store instruction register |
| SLS | single-line simulator |
| SMT | switch mode and transfer |
| SPC | stored program control |
| SPCS | Stored Program Control System |
| SSAS | station signaling and announcement subsystem |
| STF | some tests failed |
| SWAP | Switching Assembly Program |
| TASP | Toll Alternatives Studies Program |
| TLM | Trouble Locating Manual |
| TLN | trunk line network |
| TLP | trouble-locating procedures |
| TOD | time of day |
| TPCP | TSPS display page control process |
| TR | trouble report |
| TSIP | TSPS spooler input process |
| TSPNF | TSPS non-frozen development |
| TSPS | Traffic Service Position System |
| TSPSCAP | Traffic Service Position System Real-Time Capacity Program |
| TU | tape unit |
| TUS | Test Utility System |
| UCB | unit control block |
| UCG | universal call generator |
| ULARP | user-level automatic restart process |
| UTF | universal trunk frame |
| UTSC | universal trunk scanner |
| UTSD | universal trunk signal distributor |
| VC | virtual channels |
| VTP | virtual terminal protocol |
| WRMI | we really mean it |
| XCAMA | transfer centralized automatic message accounting |

## CONTRIBUTORS TO THIS ISSUE

**R. Ahmari,** B.S.E.E., 1966, University of Tehran; M.S. (E.E.), 1969, and Ph.D. (E.E.), 1972, Illinois Institute of Technology; Assistant Professor, Manhattan College, New York, 1972–1973; Bell Laboratories, 1973—. Mr. Ahmari worked on system planning for private networks, design and development of fault-tolerant systems, and system testing and integration for TSPS. He is currently supervising a group engaged in software design, and development coordination of TSPS No. 1B Generic 1BT2. Mr. Ahmari is a Registered Professional Engineer in the State of Illinois. Member, IEEE.

**Pierre L. Bastien,** B. S. (Physics), University of Montreal; Ph.D. (Physics), University of California at Berkeley; Bell Laboratories, 1973—. Upon joining Bell Laboratories in 1973 Mr. Bastien first studied the effects of No. 4 ESS failure on the toll network and local offices. More recently he has worked on long-range planning tools: The Operator Services Traffic Network Planning System for operator services; and the Switching Studies Tool, for toll planning. Currently, Mr. Bastien is a Member of Technical Staff in the SPC Network Planning Methodology Group working on development of a planning tool for long-range toll switch studies and point-of-presence (interconnection) planning for post-divestiture.

**Joseph J. Bodnar,** B.S.E.E., 1969, Rutgers University, M.S.E.E., 1971, Polytechnic Institute of Brooklyn; M.S. (Advanced Management), 1980, Pace University; Bell Laboratories, 1969–1976; AT&T, 1976–1979; Bell Laboratories, 1979—. Mr. Bodnar worked on the development of the Switching Control Center System at Bell Laboratories. He then joined AT&T Network Operations where he was responsible for TSPS and ESS operations methods and technical support. He returned to Bell Laboratories in systems engineering and is currently Supervisor of the Switching Control Center Systems group. Member, Tau Beta Pi, Eta Kappa Nu.

**G. T. Clark,** B.S.M.E., 1952, Bradley University; Western Electric, 1956–1961; Bell Laboratories, 1961—. Mr. Clark was first engaged in the physical design of step-by-step common control equipment and later worked on the design of 758C PBX equipment. He has coordinated the physical design of TSPS No. 1 equipment, including detail design of network, position subsystem, TTY trunk and buffer, the station signaling and announcement subsystem, Peripheral System

Interface frame, and processor replacement for TSPS No. 1B. He was also engaged in the physical design of AIS equipment, including the file subsystem No. 2. He is currently working on physical design of new features for TSPS No. 1B.

**J. I. Cochrane,** B.S.E.E., 1962, M.S.E.E., 1966, Georgia Institute of Technology; Ph.D., E.E. (Communication Theory), N.Y.U.; Honeywell, Inc., 1963–1965; Bell Laboratories, 1966—. From 1966–1969, Mr. Cochrane worked on the design of prototype signal processors for ballistic missile defense radars. In 1969 he was appointed Supervisor of a group responsible for the system design of advanced missile defense radars. From 1972–1973 he supervised the group responsible for the Perimeter Acquisition Radar software performance requirements. From 1973 to 1976 he supervised the formulation of long-range plans for the Navy Telecommunications System. From 1976 to 1977 he supervised the determination of the requirements for new features and equipment to be used in the operations and administration of voice-grade switched networks for large corporate customers. In 1977 and 1978 he supervised a group responsible for formulating and analyzing concepts for new services for large corporate customers using emerging telecommunications technology. In 1978 he was appointed Head of the Network Operations Planning Department responsible for formulating long-range plans for computer-based support systems to support Bell System network operations. In 1981 Mr. Cochrane was appointed Director of the Network Switching and International Systems Engineering Center; this center is responsible for the Bell Laboratories systems engineering work on Operator Services systems and for much of the planning and systems engineering work in support of AT&T International. Licensed Professional Engineer (NJ); member, Tau Beta Pi, Eta Kappa Nu.

**Bently A. Crane,** B.S. (Physics), 1954, M.S. (Physics), 1955, University of Michigan; M.S.E.E., 1959, New York University; Bell Laboratories, 1957—. Mr. Crane was involved in exploratory development of advanced computers for ballistic missile defense systems until 1972. He then began Bell System work, which has included exploratory development of auxiliary processors and voice message systems. Mr. Crane is currently a Consultant in the Operator Services Project Planning Department, working on TSPS system performance evaluation. He recently received the Bell Laboratories Distinguished Technical Staff Award.

**John R. Daino,** B.S.E.E., 1967, Syracuse University; M.S.E.E.,

1969, Ohio State University; AT&T, 1977–1980; Bell Laboratories, 1967–1977, 1980—. After joining Bell Laboratories in 1967, Mr. Daino worked on switching systems development in No. 5 Crossbar and No. 3 ESS. In 1972, he transferred to systems engineering to work on billing systems for electronic and electromechanical switching systems. Mr. Daino transferred to AT&T Network Design in 1977 where he was responsible for ESS generic planning as well as individual switching projects. In 1980, he returned to Bell Laboratories in his present position as Supervisor of the Suburban and Local Systems Central Office Operations group.

**John C. Dalby,** Jr., B. S. (Applied Mathematics), 1968, M.S.E. (Computer, Information, and Control Engineering), 1969, University of Michigan; Masters of Philosophy (Computer Science), 1977, Rutgers University; Bell Laboratories, 1970—. Since joining Bell Laboratories Mr. Dalby has been involved in TSPS No. 1 development designing call-processing and maintenance software and writing system development requirements for new TSPS No. 1 features. Presently, he supervises a group doing No. 5 ESS Operator Services Position System Planning. Member, Tau Beta Pi.

**Noel X. DeLessio,** B.S.E.E., 1960, M.S.E.E., 1961, Ph.D. (E.E.), 1966, Polytechnic Institute of Brooklyn; Bell Laboratories, 1966—. Mr. DeLessio worked on Safeguard system design and supervised guidance design for the SPRINT missile system. Subsequently, he supervised the Exploratory Development Group of the Operator System Laboratory and is currently Head of the No. 5 ESS OSPS Development Department in the Operator Services and Digital Switching Applications Laboratory.

**Richard S. DiPietro,** B.S. (Engineering Science), 1970, Northwestern University; M.S.E.E., 1972, New York University; Bell Laboratories, 1970—. Mr. DiPietro worked on military system performance evaluation until 1974. He then worked on hardware design and system testing for the remote TSPS trunk arrangement. Since 1977, he has been Supervisor of TSPS field support, system test, and call-programming groups.

**R. J. Gill,** B.S., 1970 (Computer Science), Purdue University; M.S., 1971 (Applied Mathematics), University of Michigan; Bell Laboratories, 1970—. Mr. Gill began his work at Bell Laboratories on the Safeguard ballistic missile defense system in the area of software

architecture, design, and analysis. From 1974 through 1981 he worked on TSPS, designing call-processing software, doing exploratory development, and then supervising the Processor Application Group for TSPS No. 1B. In 1982 he became Supervisor of the OSPS Architecture Group for the No. 5 ESS Operator Services Position System.

**T. G. Hack,** B.S., 1966 (Mathematics), Xavier University; M.S., 1967, Ph.D, 1970 (Applied Mathematics), Purdue University; Bell Laboratories, 1966—. Mr. Hack initially worked on software generation tools for the No. 2 ESS and No. 3 ESS projects and then in the area of change management for C-language-based developments. In 1979, he became Supervisor of a group responsible for the software development environment for the TSPS project. He currently supervises a group responsible for the development of operational software for No. 5 ESS.

**Harry A. Hilsinger,** B.S.E.E., 1954, Newark College of Engineering; M.S.E.E., 1958, New York University; Bell Laboratories, 1954—. Mr. Hilsinger initially worked on airborne guidance systems for the Nike Zeus project. In 1961 he became Supervisor of a group doing physical design for the first electronic PBX system. He has been participating in switching system physical design since that time and is currently responsible for physical design associated with Network Operator Services. Member, Tau Beta Pi, Eta Kappa Nu.

**Teddy Huang,** B.S. (Electrical Engineering), Purdue University; M.S. (Electrical Engineering), MIT; Ph.D (Electrical Engineering), Purdue University; Bell Laboratories, 1972—. Since joining Bell Laboratories, Mr. Huang has been engaged in system simulation, maintenance software, and test utility systems development. He is currently a Supervisor of a group responsible for the overall integrity of Operator Services Position Systems.

**Gary J. Kujawinski,** B.S.E.E., 1975, M.S.E.E., 1977, University of Illinois; Bell Laboratories, 1977—. Mr. Kujawinski has worked on the TSPS No. 1B system in the software emulation, performance measurement, and field support areas. He currently supervises the TSPS Test Facility Group. Member, IEEE.

**N. A. Martellotto,** B.E.E, and B.S. (Applied Mathematics), 1957, Georgia Institute of Technology; M.E.E, 1959, New York University;

M.B.A., 1970, University of Chicago; Bell Laboratories, 1957—. Starting with the Bell System Data Processor project in 1957, where he did logic design and programming, Mr. Martellotto has been involved with computers and software development throughout his career at Bell Laboratories. He worked on EPBX and No. 1 ESS and holds a patent related to the basic notion of ESS generic programs. In 1966, he became Head of the Indian Hill Computation Center (IHCC). In 1976, he resumed design and development of ESS software development support programs and other related work. In late 1979, Mr. Martellotto became DMERT project manager and for the next two years was involved with all aspects of the project, from operating system development to field support. He is now Head of the Software Development Systems Department at Indian Hill. Member, IEEE, Tau Beta Pi, Eta Kappa Nu.

**Sherman C. Reed,** B.S.E.E., 1956, University of Okalahoma; M.S.E.E., 1958, Newark College of Engineering; Bell Laboratories, 1956—. At Bell Laboratories, Mr. Reed had many assignments in ballistic missile defense activities from 1956 to 1974. He began work on TSPS in 1974 and has been involved with call programming and operator actions software design, new feature planning development coordination and system testing. He is currently Supervisor of the TSPS No. 1B 1BT2 System Testing Group. Member, Tau Beta Pi, Eta Kappa Nu, Pi Mu Epsilon.

**Robert E. Staehler,** B.S.E.E., 1947, The College of the City of New York; M.S.E.E., 1948, Polytechnic Institute of Brooklyn; Bell Laboratories, 1948—. Mr. Staehler's early work was on No. 5 Crossbar, toll signaling systems, and trainers for guided missile systems. In 1953, he worked on the development of electronic switching systems, specifically, the processor memory for the experimental central office in Morris, Illinois, and the processor logic and call memory for No. 1 ESS. He was appointed Director of the Electronic Switching Projects Laboratory in 1964 with responsibility for special applications for No. 1 ESS to military and data networks, including No. 1 ESS AUTOVON. In 1968, he became Director of the Electronic Systems Design Laboratory with responsibility for development of the 1A Processor for No. 1 ESS and No. 4 ESS. In 1976, he became Director of the Network Operator Services and Digital Switching Applications Laboratory with responsibility for developing operator services for both domestic and international applications, along with the No. 5 ESS remote switching vehicles. Senior member, IEEE. Member, Eta Kappa Nu, Tau Beta Pi, Sigma Xi.

**L. C. Stecher,** B.S., 1967, Loyola University; M.S., 1968, Northwestern University; Ph.D., 1972 (Applied Mathematics/Computer Science), Northwestern University; Bell Laboratories, 1970—. Mr. Stecher was involved in the design and development of the No. 4 ESS maintenance and call-processing subsystems. In 1975, he became Supervisor of the No. 4 ESS trunk maintenance development effort and later became Supervisor of an exploratory effort to define new network services for the evolving Stored Program Controlled Network. In 1980, he became Head of a department responsible for development of software for the Traffic Services Position System. This software interacts with operators, customers, and the network to handle toll and assistance traffic. In addition, the department is responsible for the Software Development System and Programmer Support Systems used in the Operator Services and Digital Switching Applications Laboratory.

**E. H. Stredde,** B.S.E.E., 1966, M.S.E.E., 1967, University of Illinois; Bell Laboratories, 1967—. Mr. Stredde has developed maintenance and operational software for No. 1 ESS, the 1A Processor, No. 4 ESS, and TSPS. He is currently Supervisor of the Remote Switch Module Switch Maintenance Group. Member, Eta Kappa Nu, Tau Beta Pi.

**D. S. Suk,** B.S.E.E., 1969, Seoul National University; M.S. (E.E.), 1977, and Ph.D. (E.E.), 1978, University of Iowa; Bell Laboratories, 1978—. Mr. Suk has been involved in maintenance software design and the evaluation of TSPS No. 1B performance and reliability. He is currently working on project management and software development methodology. Member, IEEE, Eta Kappa Nu.

**James H. Tendick,** B.S.E.E., 1977, University of Illinois; M.S.E.E., 1978 Stanford University; Bell Laboratories, 1977—. Mr. Tendick was initially involved in the hardware design of the Peripheral Systems Interface for the TSPS No. 1B. He was also involved in system testing at the first live TSPS No. 1B site in Fresno, California. Other work activities included speech synthesis and recognition and field support. Presently, he is supervising a group responsible for system test and integration of the current TSPS Generic. Member, Tau Beta Pi, Phi Kappa Phi, Eta Kappa Nu, Sigma Xi.

**Kendra A. VanderMeulen,** B.S. (Mathematics), 1973, Marietta College; M.S. (Computer Science), 1977, Ohio State University; Bell Laboratories, 1973—. Ms. VanderMeulen has participated in the de-

sign, development, testing, and maintenance of minicomputer-based operations systems for service evaluation and SPCS maintenance. She is currently Head of the SCCS Applications Development Department responsible for the development of SCCS application software, development of a world-class switching operations system for the international market, and development of an operations maintenance system for minicomputer maintenance and operations centers.

**Daniel Van Haften,** B.S., M.S. (Mathematics), 1970, Michigan State University; Ph.D. (Electrical Engineering), 1977, Stevens Institute of Technology; Bell Laboratories, 1970—. Mr. Van Haften initially worked on the Safeguard project. In 1974 he joined the Network Operator Services Laboratory, where he was involved in TSPS system testing, field support, call-processing development, and processor replacement development. In 1981 he became involved in No. 5 ESS site testing in Seneca, Illinois, and is presently Supervisor of the Factory Test Software Design Group in the Local Digital Switching Software Laboratory. Member, Phi Beta Kappa, Phi Kappa Phi, Pi Mu Epsilon.

**Laurance A. Weber,** B.E.E., 1945, Cornell University; M.E.E., 1955, Polytechnic Institute of Brooklyn; Bell Laboratories, 1946—. Mr. Weber was initially involved in the design of signaling circuits. Later he participated in the design of circuits for crossbar tandem systems. Following this assignment, he was appointed Supervisor in charge of designing data sets for the mechanization of TWX service. He was appointed Head of the 101 ESS Design Department in 1960. He has had subsequent assignments in No. 2, No. 2B, and No. 3 ESS. He is presently Head of the Operator Services and Digital Switching Application Laboratory System Testing and Laboratory Administration Department. Member, IEEE, Tau Beta Pi, Sigma Xi, Eta Kappa Nu.

**R. A. Weber,** B.S.E.E., 1970, Iowa State University; M.S.E.E., 1971, Stanford University; Bell Laboratories, 1970—. Mr. Weber was initially a member of the Ocean Systems Laboratory, where he worked on underwater cable systems. In 1974 he transferred to the Operator Services and Digital Switching Applications Laboratory, where he has done circuit design and diagnostic programming on the Remote Trunk Arrangement and the SPC 1B Processor. He currently is the Supervisor of a group responsible for the TSPS System software, which interfaces with the DMERT operating system. Member, Tau Beta Pi.

**J. R. Williams,** B.S.E.E., 1960, Vanderbilt University; M.S.E.E., 1961, University of Illinois; U. S. Navy Submarine Service, 1961–1964;

Bell Laboratories, 1964—. Mr. Williams has had a variety of assignments in Electronic Switching System development. His early work included assignments in system design, test system development, and operational software design for No. 1 ESS ADF, a store and forward message switching system. In 1969, he became involved in maintenance planning and hardware design for No. 4 ESS. Later assignments included responsibilities in the area of No. 4 ESS maintenance and call-processing software development. In 1977, he became responsible for No. 5 ESS maintenance planning, and in 1979, joined the TSPS project with responsibility for operational software development. In 1980, he assumed responsibility for planning future operator system developments and for system testing new TSPS features. In 1982, Mr. Williams returned to No. 5 ESS development, with responsibility for maintenance software.

**Bruce R. Wycherley,** B.A., 1976, M.A., 1978 (Mathematics), University of Oklahoma; Bell Laboratories, 1978—. At Bell Laboratories, Mr. Wycherley has worked on the development of TSPS market models, examined TSPS limiting components, and developed operator services long-range planning tools for use by telephone companies. He is presently working in the Operator Services Planning group, investigating applications of automatic speech recognition and studying the impact of divestiture on TSPS. Member, Pi Mu Epsilon, Phi Beta Kappa.

**THE BELL SYSTEM TECHNICAL JOURNAL** is abstracted or indexed by *Abstract Journal in Earthquake Engineering, Applied Mechanics Review, Applied Science & Technology Index, Chemical Abstracts, Computer Abstracts, Current Contents/ Engineering, Technology & Applied Sciences, Current Index to Statistics, Current Papers in Electrical & Electronic Engineering, Current Papers on Computers & Control, Electronics & Communications Abstracts Journal, The Engineering Index, International Aerospace Abstracts, Journal of Current Laser Abstracts, Language and Language Behavior Abstracts, Mathematical Reviews, Science Abstracts* (*Series A, Physics Abstracts; Series B, Electrical and Electronic Abstracts; and Series C, Computer & Control Abstracts*), *Science Citation Index, Sociological Abstracts, Social Welfare, Social Planning and Social Development*, and *Solid State Abstracts Journal*. Reproductions of the Journal by years are available in microform from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.


Bell System