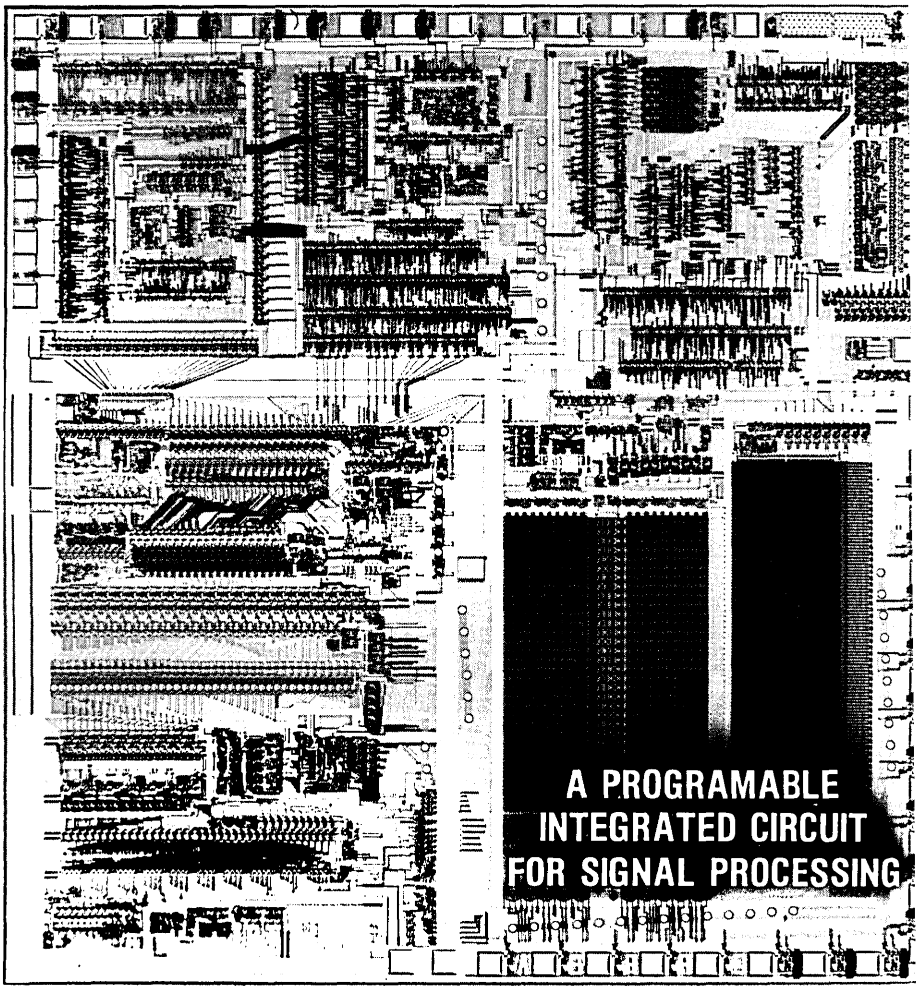
THE BELL SYSTEM TECHNICAL JOURNAL

SEPTEMBER 1981
VOL. 60, NO. 7, PART 2



THE BELL SYSTEM TECHNICAL JOURNAL

THE BELL SYSTEM TECHNICAL JOURNAL is published monthly, except for the May–June and July–August combined issues, by the American Telephone and Telegraph Company, C. L. Brown, Chairman and Chief Executive Officer; W. M. Ellinghaus, President; V. A. Dwyer, Vice President and Treasurer; F. A. Hutson, Jr., Secretary. Editorial inquiries should be addressed to the Editor, The Bell System Technical Journal, Bell Laboratories, Room WB 1L-331, Crawfords Corner Road, Holmdel, N.J. 07733. Checks for subscriptions should be made payable to The Bell System Technical Journal and should be addressed to Bell Laboratories, Circulation Group, Whippany Road, Whippany, N.J. 07981. Subscriptions $20.00 per year; single copies $2.00 each. Foreign postage $1.00 per year; 15 cents per copy. Printed in U.S.A. Second-class postage paid at New Providence, New Jersey 07974 and additional mailing offices.

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 WB 1L-331, Crawfords Corner Road, Holmdel, N.J. 07733. 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.

THE BELL SYSTEM TECHNICAL JOURNAL

DEVOTED TO THE SCIENTIFIC AND ENGINEERING
ASPECTS OF ELECTRICAL COMMUNICATION

Volume 60 | September 1981 | Number 7, Part 2

DIGITAL SIGNAL PROCESSOR

R. C. Chapman, *Guest Editor*

THE BELL SYSTEM TECHNICAL JOURNAL

DEVOTED TO THE SCIENTIFIC AND ENGINEERING ASPECTS OF ELECTRICAL COMMUNICATION

Volume 60 September 1981 Number 7, Part 2

Digital Signal Processor:

Overview: The Device, Support Facilities, and Applications

By J. R. BODDIE

(Manuscript received February 6, 1981)

This paper introduces the DSP, a new integrated circuit for digital signal processing. We describe the capabilities of the device and the tools available for operating it. Potential applications are also discussed. The paper is an overview of those that follow in this issue of the Bell System Technical Journal.

I. INTRODUCTION

The digital signal processor (DSP) is a new integrated circuit designed by Bell Laboratories and made by Western Electric Company. The device is one of the most complex high-performance circuits developed to date and will have a variety of telecommunications applications. This paper summarizes the capabilities of the DSP, describes user development tools, and lists potential applications.

The Bell System is rapidly applying digital technology to transmission, switching, and station equipment. When signals are encoded digitally, they are easily manipulated by computers and other systems that incorporate advances in very-large-scale integrated circuit technology (VLSI). The VLSI advantages include small size, high reliability, low cost, and low-power consumption. As this trend continues, it is possible to perform previous functions, as well as new ones not possible

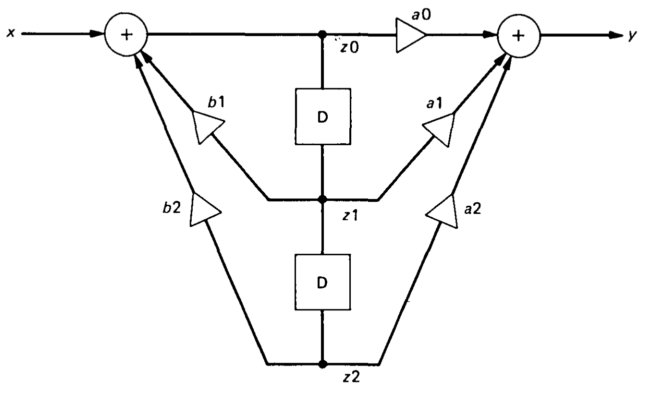


Fig. 1—Second-order filter section.

before, with digital techniques that were formerly performed with analog circuits.

Signal processing is the generation, filtering, detection, and modulation of signals. Most algorithms for signal processing repeatedly use multiplications and additions. A simple example is the second-order section used for filtering. (Refer to Ref. 1 for a more thorough introduction to digital filtering.) Figure 1 is a common schematic representation of the algorithm. The blocks represent delays or storage operations, the triangles are multiplications and the circles are additions. The $a0$, $a1$, $a2$, $b1$, and $b2$ in the structure are coefficients that determine the characteristics of the filter. The input, x, is a sequence of numbers representing a continuous waveform. Typically, a new input value is available every 125 μs. The output, y, is another sequence of numbers that must be computed using the algorithm at the same rate. The value, $z0$, is an intermediate result, and $z1$ and $z2$ are delayed values of $z0$. In order to achieve the real-time processing performance required by this example, five multiplications, four additions, and two data movements must be done in 125 μs.

The DSP was designed especially for this type of digital signal processing function. Customized by a program in an on-chip, read-only memory (ROM), the device can do over a million high-precision arithmetic computations per second. The key to the performance of the DSP is a parallel, pipelined architecture which provides maximum throughput by keeping all sections of the processor efficiently busy at all times.[2] The simplified block diagram of the DSP in Fig. 2 shows the organization of the processor as three independently controllable elements: a data arithmetic unit (AU) with multiplier, accumulator, and rounder; an address arithmetic unit (AAU) for controlling memory access; and an I/O unit to provide a serial data interface. A control unit (CU) synchronizes those elements and provides instruction decoding. Temporary results are stored in a read/write data memory (RAM). For program development and device testing, external memory can be used to replace the on-chip ROM. The DSP can do all the operations required to implement the second-order section of the example in 4 μs, that is, it can do 31 second-order sections in 125 μs. This speed is sufficient to implement a complete receiver for *TOUCH-TONE*® telephone service[3] or a low-speed modem using a single DSP. The DSP functions in a stand-alone fashion in many applications, but it can easily be interfaced to microprocessors or additional DSPs to achieve a greater degree of signal processing capability.

II. DEVELOPMENT OF THE DSP

The DSP is realized in N-channel MOS technology, using depletion loads. Packaged in a 40-pin DIP, it requires only a single 5-volt supply and runs at a 5-MHz rate. The circuit consists of approximately 45,000 transistors within a 68.5-mm^2 area.

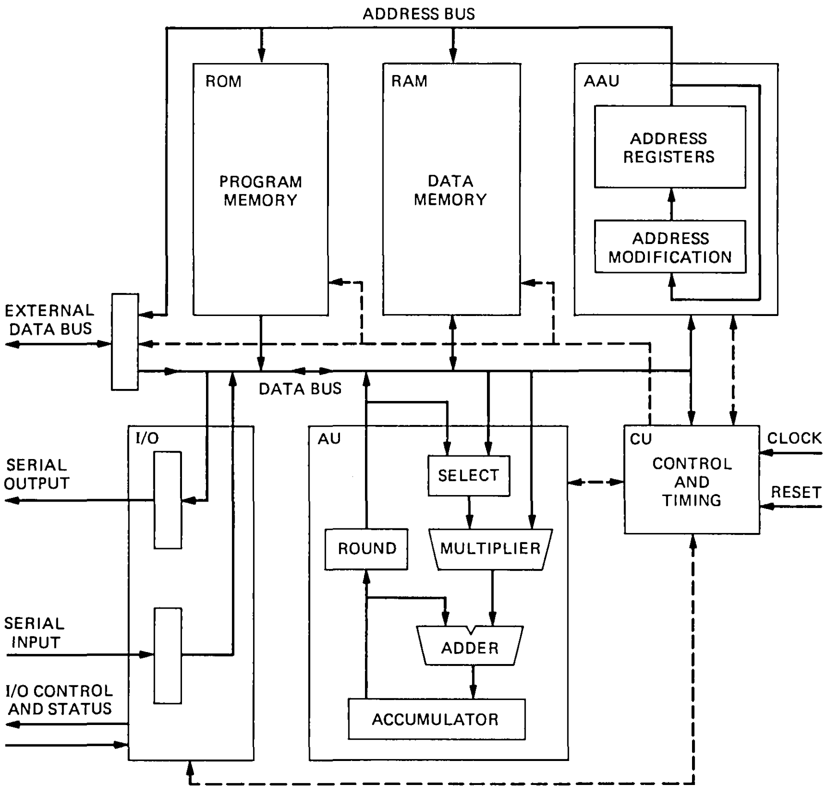


Fig. 2—Digital signal processor block diagram.

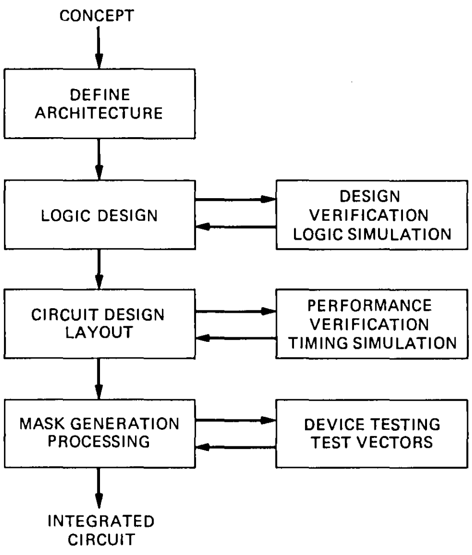

CONCEPT
DEFINE ARCHITECTURE
LOGIC DESIGN
DESIGN VERIFICATION LOGIC SIMULATION
CIRCUIT DESIGN LAYOUT
PERFORMANCE VERIFICATION TIMING SIMULATION
MASK GENERATION PROCESSING
DEVICE TESTING TEST VECTORS
INTEGRATED CIRCUIT

Fig. 3—Device development steps.

This level of integration and the performance requirements presented new challenges for circuit designers in the implementation and testing of the device. Figure 3 shows the major design steps from concept to final product. In the first step, the signal processing requirements of several benchmark applications were combined with the knowledge of what could be done within the limits of the technology. The result was a definition of the architecture, instruction set, and performance specifications.

The logic design work produced a gate-level description of the processor. A logic-level simulator program was used to verify the design. A TTL prototype of the device was also constructed which could emulate the DSP running at full speed. This proved useful for additional design verification and for the development of early DSP applications.

The circuit design and layout implemented the logic design with transistors. A custom layout style was used for the data AU and memories in which each device was created and connected to optimize circuit density, speed, and power. The circuit design and layout of the I/O, AAU, and CU was done using a technique of interconnecting standard predefined logic cells. Custom cells were defined where high-speed paths were required. A computer-aided design system was used to automatically place and connect the cells according to the logic description. This technique greatly reduced the design time of the project. The performance of the design was verified by a circuit simulator program. Computer aids were also used to check for viola-

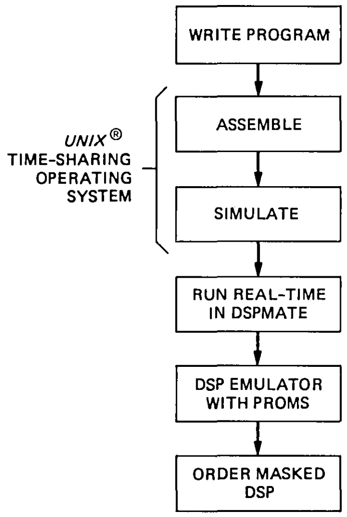


Fig. 4—Digital signal processor application development procedure.

tions of physical layout design rules and to determine the size and type of transistors and parasitic capacitances. In addition, computer plots of the circuit layout were visually inspected for design rule, functional and interconnect errors. Refer to Ref. 4 for details of this design.

Finally, masks were made and devices were fabricated, packaged, and tested. A sequence of inputs designed to test all DSP functions was used to test the devices, as well as to locate as many faults in the chip as possible.[5]

III. SUPPORT TOOLS FOR THE DSP

To facilitate the design of systems using the DSP, a comprehensive set of hardware and software design aids were developed. These tools can be illustrated with a typical application development process.

A DSP system development begins (as shown in Fig. 4) with the writing of a DSP program. The program is entered into a computer by a *UNIX** time-sharing operating system text editor. Digital signal processor programs are written in a unique assembly language which uses standard mathematical notation instead of a more conventional mnemonic format. This greatly improves the readability of programs which are usually arithmetic-intensive because of the nature of signal processing algorithms. An assembler program handles the peculiarities of this pipelined processor in translating DSP programs into a machine-executable code.[6]

* Registered trademark of Bell Laboratories.

The second step in program development is the simulated execution of the program by software simulation of the DSP.[7] The simulator allows the monitoring of operations in the DSP and the analysis of results at the programmer's computer terminal.

When the user is satisfied with the simulation results, the program can then be run on a DSP in real time. A hardware development system called DSPMATE (Fig. 5) does this while retaining most of the debugging aids of the simulator. The DSPMATE is a microprocessor-based system which can load DSP programs from the computer that hosts the development software. A numeric editor in the system can be used to enter or modify programs and data. The editor can also display a plot of the data on the user terminal. During the execution of a DSP program, I/O events and program flow can be captured and displayed. The DSPMATE can be interfaced to the user's system through a cable and 40-pin plug which is compatible with the DSP. Thus, the operation of the DSP can be monitored in the user's particular hardware environment. Auxiliary circuit boards for the DSPMATE provide analog-to-digital and digital-to-analog conversion, ROM programming, and multiple DSP emulation capabilities.

As program modifications become infrequent, the DSPMATE can be replaced in the user's prototype with a DSP EMULATOR (Fig. 6). This development tool is a small printed wiring board with a DSP and ultraviolet erasable, programmable ROMs for storing the user program.

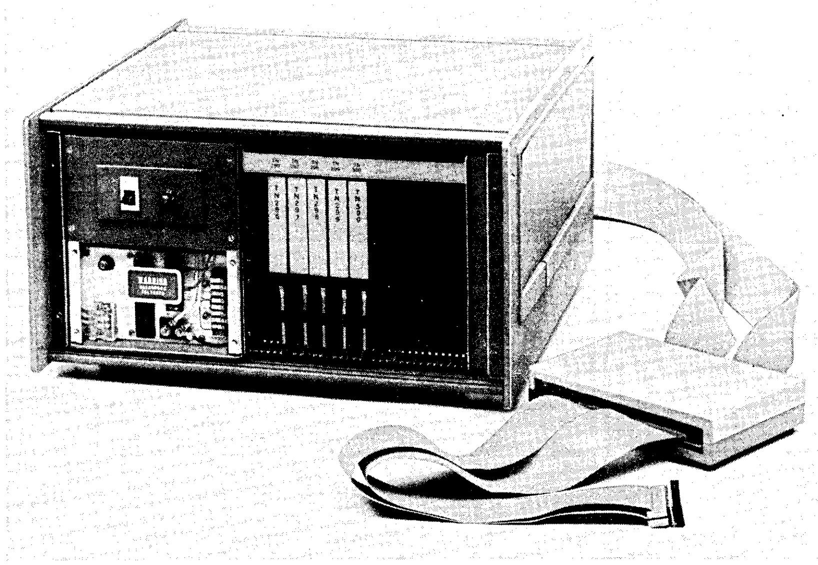


Fig. 5—Hardware development system (DSPMATE).

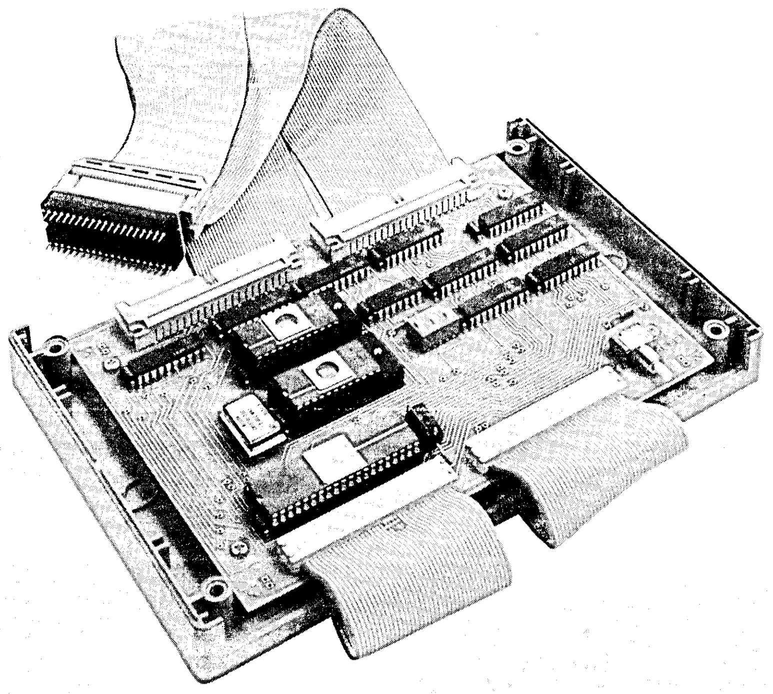


Fig. 6—Digital signal processor EMULATOR prototyping board.

A set of connectors allows the user to monitor the DSP with other test equipment.

Finally, the system developer can request fabrication of a DSP with an on-chip ROM coded with the debugged program.

Extensive documentation on the device, as well as on these tools, and a "hotline" for providing designers with quick answers to their questions, complete the support package.

IV. APPLICATIONS FOR THE DSP

A wide variety of applications exist for the DSP. Table I shows how the device fits into the spectrum of signal processing applications for telecommunications and the number (or fraction) of DSPs required for a particular application.

In this issue, we describe the following applications that represent a small fraction for which the DSP is being considered:

Table I—Application complexity of DSP

Application	Complexity (No. of DSPs)
Second-order section	0.03
ADPCM* coder	0.25
Dual-tone receiver	0.90
Modem, 1200-baud	1.50
Transmultiplexer	6.00

* Adaptive differential pulse-code modulation.

Transmission system measurements

To make measurements on transmission systems, it is useful to be able to generate complex signals comprised of sinusoidal components of arbitrary frequency, phase, and amplitude.[8] At the other end of the transmission system, loss and noise can be measured by using a high-precision filtering and averaging technique.[9]

Signaling receivers

The detection of tones for signaling systems is an example of applications that must be implemented digitally, because the overall system operates by signals that have been encoded into a digital format.[3,10]

Line treatment

An application which demonstrates the advantages of the high-precision arithmetic capability of the DSP is the treatment of special voice frequency metallic circuits.[11] Here, the DSP provides variable line equalization, gain, and balance functions.

Speech coding

Low bit-rate speech encoding has been made more viable with the DSP. Not only are such systems more economical, but now numerous techniques can be quickly developed and evaluated.[12,13,14]

Speech synthesis

Man-machine speech communications has long been a subject of intense research activity—usually requiring large general-purpose computers in a nonreal-time mode. The DSP makes real-time, high-quality speech synthesis a practical possibility.[15]

V. CONCLUSION

The development of the DSP is a significant accomplishment in many respects. Architecturally, it is a new type of parallel, pipelined processor which achieves a high degree of throughput for the particular job for which it was designed and, yet, it has the flexibility to do the nonsignal processing tasks when necessary. The circuit is one of the most complex and high-performance devices designed for manufacture. New tools were designed to support the processor and the development of signal processing systems. Many applications that were made with analog elements are now economically feasible through digital techniques, and new algorithms can be explored.

In the future, new DSP architectures will be developed to take advantage of advances in VLSI technology. Thereby, system designers will have the best signal processing components available for applying forward-looking solutions to needs of the Bell System.

VI. ACKNOWLEDGMENT

Many talented people contributed to the development and support of this complex device; however, D. C. Stanzione receives special recognition for his innovation, organization, and direction of the DSP project.

REFERENCES

1. E. J. Angelo, "Digital Signal Processor: A Tutorial Introduction to Digital Filtering," B.S.T.J., this issue.
2. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
3. J. R. Boddie, N. Sachs, and J. Tow, "Digital Signal Processor: Receiver for *TOUCH-TONE*® Service," B.S.T.J., this issue.
4. F. E. Barber et al., "Digital Signal Processor: An Overview of the Silicon Very-Large-Scale-Integration-Implementation," B.S.T.J., this issue.
5. I. I. Eldumiati and R. N. Gadenz, "Digital Signal Processor: Logic and Fault Simulations," B.S.T.J., this issue.
6. C. L. Semmelman, "Digital Signal Processor: Design of an Assembler," B.S.T.J., this issue.
7. J. Aagesen, "Digital Signal Processor: Software Simulator, B.S.T.J., this issue.
8. D. L. Favin, "Digital Signal Processor: Tone Generation," B.S.T.J., this issue.
9. S. P. Cordray, D. L. Favin, and D. P. Yorkgitis, "Digital Signal Processor: Power Measurements," B.S.T.J., this issue.
10. R. N. Gadenz, "Digital Signal Processor: Tone Detection for CCITT No. 5 Transceiver," B.S.T.J., this issue.
11. A. C. Bolling and R. L. Farah, "Digital Signal Processor: Voice-Frequency Transmission for Special-Service Telephone Circuits," B.S.T.J., this issue.
12. D. A. Berkley et al., "Digital Signal Processor: Adaptive Differential Pulse-Code-Modulation Coding," B.S.T.J., this issue.
13. R. E. Crochiere, "Digital Signal Processor: Subband Coding," B.S.T.J., this issue.
14. D. A. Berkley, N. S. Jayant, and C. A. McGonegal, "Digital Signal Processor: Private Communications," B.S.T.J., this issue.
15. M. R. Buric, J. Kohut, and J. P. Olive, "Digital Signal Processor: Speech Synthesis," B.S.T.J., this issue.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

An Overview of the Silicon Very-Large-Scale-Integration Implementation

By F. E. BARBER, T. J. BARTOLI, R. L. FREYMAN, J. A. GRANT, J. KANE and R. N. KERSHAW

(Manuscript received July 14, 1980)

A programmable digital signal processor integrated circuit has been designed as a general-purpose building block for a variety of telecommunication applications. The device, known as digital signal processor, is a single-chip integrated circuit fabricated in depletion-load NMOS *technology and packaged in a 40-pin* DIP. *This paper describes the silicon very-large-scale-integration (*VLSI*) implementation of the digital signal processor with emphasis on the circuit design phase. The specific areas discussed are choice of fabrication technology, layout styles, circuit design procedures, and circuit considerations.*

I. INTRODUCTION

A single-chip integrated circuit has been developed as a stand-alone part for digital signal processing. This device known as a digital signal processor (DSP) functions as a special-purpose microcomputer whose instruction set, arithmetic functions, and addressing capability are optimized for real-time signal processing. The primary sections of the DSP are a read only memory (ROM), a random access memory (RAM), an address arithmetic unit (AAU), an arithmetic unit (AU), a system controller, and appropriate input/output (I/O) circuitry.

The ROM is organized as 1024 words by 16-bits per-word memory for storing the system programs and fixed data. The RAM is organized as 128 words by 20 bits per word and is used for storage of variable data and temporary results. The AAU generates the addresses for the RAM

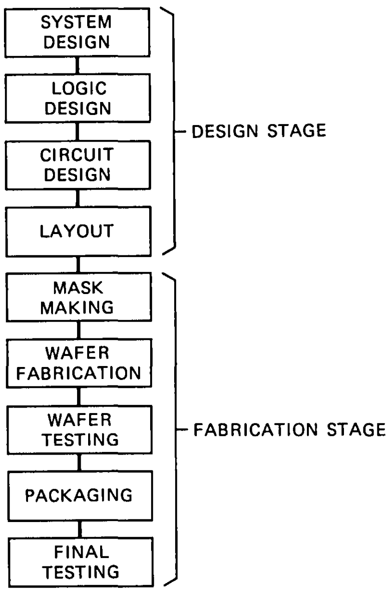


Fig. 1—Integrated circuit implementation process.

and ROM, as well as the addresses for an external ROM. The AU performs the necessary arithmetic operations for digital signal processing, e.g., 16- by 20-bit multiplication and 40-bit accumulation. The I/O unit will accept and generate a serial bit stream of either μ-255 law or linear PCM signal samples. The control unit decodes instructions and provides overall system coordination.

Figure 1 shows the process of implementing the DSP as an integrated circuit. This process is divided into a design stage and a fabrication stage. The design stage consists of four design areas: system, logic, circuit, and layout. In the case of the DSP, the system design is the process of defining the system architecture and instruction set.[1] The logic design assembles logic functions which meet the system's requirements. The circuit design implements the logic design with electrical devices so that the system's requirements are achieved. The fabrication of the integrated circuit starts with a circuit layout, which is the end result of the circuit design. The circuit layout is used to produce masks which are used in wafer fabrication to define the electrical devices. The completed wafers are tested to determine good devices that are then packaged. The completed integrated circuits are tested to verify that the system requirements are met.

This paper describes the circuit design phase of the integrated circuit implementation of the DSP.

II. TECHNOLOGY

The first consideration in circuit design is the choice of an integrated circuit technology. The choice of technology will impact the following areas: the system performance in terms of speed and power; the cost of the device which is related to silicon area or circuit density; and the type of logic functions which can be implemented by the specific technology.

The N-channel MOSFET (metal-oxide semiconductor, field effect transistor) or NMOS technology can easily implement the following logic functions: INVERT, NAND, NOR, XOR, XNOR, Flip-Flops, Shift Registers; and complex logic OR-AND-INVERT, AND-OR-INVERT. The MOS technology, by implementing these different forms of logic, can provide high functional density as compared to a technology which is restricted to fewer forms of logic gates.

The majority of DSP circuitry is a synchronous system wherein data are transferred between registers at a 5-MHz rate. The critical path is 10 logic gates deep; therefore, the technology must be capable of gate delays of 20 ns or less (worst case) at a power level compatible with the DSP level of integration.

The depletion-load NMOS technology was selected for implementing the DSP. This is an existing process presently used for manufacturing static memory devices. The existence of a manufacturable process has the advantage of decreasing the design time and decreasing the risk of developing a device. This technology provides the performance, speed and power, necessary for the DSP requirements, as well as allowing a high-density layout for implementing the 7300 TTL-equivalent gates of logic, 2.5 K bits of RAM and 16 K bits of ROM.

III. CIRCUIT LAYOUT

The final result of circuit design is a layout data base used to make masks. These masks are used in wafer fabrication to produce an integrated circuit. There are different layout approaches or styles to create and connect devices. These different styles optimize either circuit density or design time and time to produce final integrated circuits. The first style is known as custom layout, where each device is created and connected with the minimum restrictions. The custom layout style optimizes circuit density which results in the fastest speed and lowest power. The second style is known as polycell layout. This layout style uses established cells at a logic function level which have been designed, laid out, and electrically verified. A computer-aided design (CAD) system known as LTX[2] can automatically place and connect these polycell functions according to a logic description known as LSL.[3] The polycell layout style produces the shortest time interval from the start of the circuit design stage to completion of an integrated circuit meeting the system requirements.

The DSP memories, RAM and ROM, constitute a large number of transistors in a regular pattern. For this reason, it is desirable to use a custom layout style for memories to achieve the highest possible transistor density and consume the least amount of silicon area.

The AU performs a 16- by 20-bit multiplication and a 40-bit accumulation, along with other functions. These functions require logic which is performed on many bits in parallel, resulting in bit-oriented logic. These portions have the functional logic replicated up to 40 times depending upon word length required. Exemplary of the logic performed in the AU is the 20-bit add function that must be completed within one clock cycle. This path and other similar paths are up to 10 logic gates deep, resulting in the requirement that worst-case, critical-path gate delays must be less than 20 ns to fit within the 200-ns cycle. Consideration of the speed requirements in the AU and the regular characteristics of much of the logic, lead to the choice of a custom layout style in this section to optimize circuit density, speed, and power.

The IOAC section, which consists of the I/O, AAU, and control sections, has both random and bit-oriented logic. Custom layout was selected for the bit-oriented logic to obtain the higher circuit density. The IOAC random logic is 5 gates deep and results in the requirement that a worst-case gate delay must be less than 40 ns. Polycell layout could meet this speed requirement and was chosen for the random logic sections to decrease the design time of the project. Polycells for the depletion-load NMOS technology did not exist when the circuit design stage of the DSP was started; therefore, polycells were first defined and created before the IOAC random logic sections could be started.

Both custom and polycell layout styles were used in the DSP layout. Custom layout was used in the memory and bit-organized sections to optimize circuit density, speed, and power. Polycell layout was used in the random logic section to minimize design time. The resulting layouts have shown that the custom logic section achieved about twice the functional density with a factor of 3 improvement in the speed achieved for a given power level.

IV. CIRCUIT DESIGN PROCEDURES

Knowing the type of logic functions that could be implemented with the NMOS technology, the logic designer completed the logic description for the DSP. The circuit designer, having determined the layout style, had to convert the logic gates to NMOS transistors. Figure 2 shows this circuit design procedure. The proposed circuit is analyzed using a

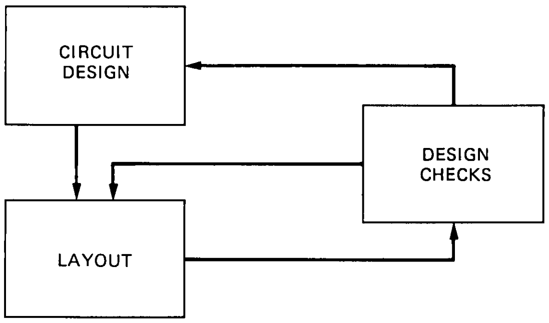


Fig. 2—Circuit design procedure.

circuit simulator such as SPICE.[4] After a successful simulation, the circuit is converted to mask levels on an in-house minicomputer graphics system. Physical layout design rules for lines and spaces are used to create geometries necessary for creating a mask for wafer fabrication. There are two computer aids for verifying the layout. The first aid checks for violation of the physical layout design rules. The second aid determines the size and type of transistors and the parasitic capacitance which is an important parameter for circuit performance. This circuit characterization represents a more accurate circuit description than was initially simulated before layout, and these results are used in the circuit simulator to determine final circuit performance. In addition, computer plots of the circuit layout are visually checked for design rule layout errors, functional errors, and interconnect errors.

V. CIRCUIT CONSIDERATION

The single most important factor which made the circuit design of the DSP practical is that the DSP is a synchronous system, wherein data flows between registers in a specified time interval. This allows for circuit simulations of relatively small sections since the exact timing is well known.

Signals, whether they are instructions or data, are passed between the major sections (AU, RAM, ROM, AAU, I/O, CONTROL) via a 20-bit data bus. The 5-MHz system clock has two phases known as a master and slave phase of 100 ns each. The data bus is charged during the slave phase and is discharged during the master phase by the sending port, if the data bit is true.

Synchronization between the various sections is achieved by distributing a master clock and a synchronization signal and locally regenerating the required clocks. De-skewing networks were included in each section to achieve precise clock synchronization even in the presence of master clock delays due to resistance-capacitance loading delay effects.

Power and ground bus distribution is important to maintain adequate noise margin. Each major section of the DSP has its own power and ground to minimize the accumulation of voltage drops and noise interactions due to transients. The size of these lines was made large enough to keep worst-case power and ground drops below 100 mv.

The input and output buffers were located at the edge of the chip and separate power and ground lines were also provided for these circuits. Both the peripheral placement of I/O buffers and provision for their separate power and ground bussing minimize externally induced noise effects on internal logic.

Internal testing probe pads were distributed along the data bus, the address bus, and selected control and timing signal paths so that the AU, RAM, ROM, and IOAC could be tested independent of each other for diagnostic testing and logic verifications. These pads did not increase

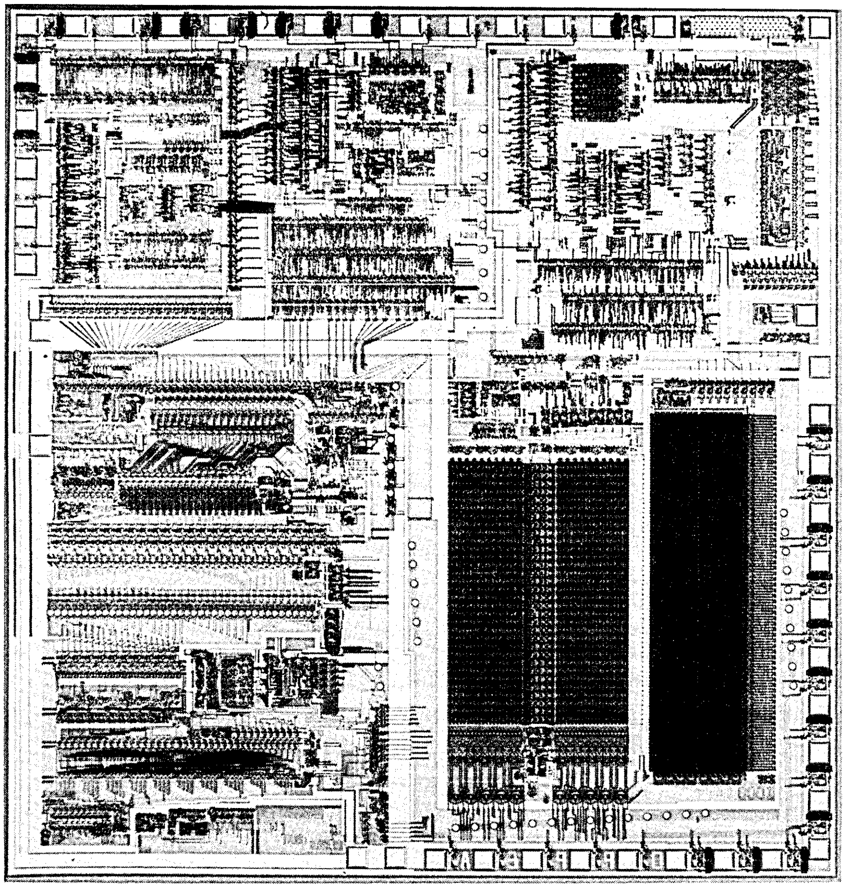


Fig. 3—Digital signal processor integrated circuit.

the final chip size and were invaluable in analyzing initial device testing results.

VI. CONCLUSIONS

The VLSI silicon implementation of the DSP has been described. The DSP integrated circuit meets the system's requirements—function, speed, and power. Figure 3 shows the chip, which contains 7300 TTL-equivalent logic gates, 16 K bits of ROM, and 2.5 K bits of RAM. The implementation requires 45,000 transistors and occupies an area of 62 mm^2 (8.1 by 8.6 mm).

VII. ACKNOWLEDGMENTS

We sincerely thank the many people at Bell Laboratories who contributed to the development of the DSP. In particular, we are very grateful to Bailey R. Jones for his efforts in wafer fabrication of the DSP.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX—A System for the Directed Automatic Design of LSI Circuits," Proc. 13th Design Automation Conference, San Francisco, California, June 28–30, 1976 pp. 399–407.
3. H. Y. Chang, G. W. Smith, Jr., and P. B. Walford, "Lamp: System Description." B.S.T.J. *53,* No. 8 (October 1974) pp. 1431–49.
4. L. W. Nagel and D. O. Pederson, "Simulation Program with Integrated Circuit Emphasis." Proc. Sixteenth Midwest Symposium on Circuit Theory, Waterloo, Canada, *1* (April 12–13, 1973), pp. VI.1.1.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Architecture and Performance

By J. R. BODDIE, G. T. DARYANANI, I. I. ELDUMIATI, R. N. GADENZ, J. S. THOMPSON, and S. M. WALTERS

(Manuscript received July 14, 1981)

This paper describes the DSP, a recently developed integrated circuit implementing a programmable digital signal processor. The single-chip device is fabricated in depletion-load NMOS and is packaged in a 40-pin DIP. It has the speed, precision, and flexibility for a variety of telecommunication applications. The processor can decode an instruction, fetch data, perform a 16- by 20-bit multiplication and a full 36-bit product accumulation in one machine cycle of 800 ns. This permits the realization of signal processing functions of such applications as dual-tone multifrequency receivers or low-speed data modems with a single device. The arithmetic precision of the processor is also sufficient for many voice signal applications.

I. INTRODUCTION

Digital signal processing has become more and more important in telecommunications. As new products and services are offered, the amount of required signal processing continues to increase. In addition, signals are becoming digital, especially in applications where the superior stability and accuracy of these signals is either necessary or more attractive. Digital signal processing is also prompted by the introduction of digital switching offices and digital transmission systems. It is made possible by the continuous, rapid growth of the silicon LSI and VLSI capabilities. The latter have made it inexpensive to build complex processors—so inexpensive that it is cost-effective even to use A/D conversion and digital signal processing in some analog systems. We indeed visualize the extension of the digital network all the way to the subscriber phone!

In this paper, we describe a single-chip, digital signal processor recently developed for Bell System use. The device, known as Digital Signal Processor (DSP) is a general-purpose building block which can be programmed to perform a variety of digital signal processing functions. Examples of these are filtering, equalization, modulation, tone detection, speech coding, and Fast Fourier Transform. The DSP is fabricated in depletion-load NMOS and packaged in a 40-pin DIP. It is customized to perform specific signal processing functions by means of an on-chip read only memory (ROM) containing the program and fixed data. The device also contains a random access memory (RAM) for writing and reading variable data, a Control Unit, an Arithmetic Unit (AU), an Address Arithmetic Unit (AAU), and appropriate Input/Output (I/O) circuitry. The DSP functions in a stand-alone manner, requiring only an external 5-MHz resonator or clock, or it may be directly interfaced with other processors to achieve a greater degree of signal processing capability.

The DSP programmability makes the device useful for a variety of telecommunication applications, and results in a shorter and less expensive system development cycle. Key elements in digital signal processing are adequate numerical precision and high-computation rates. The DSP offers both. Its 16- by 20-bit multiplier and 40-bit adder, running at 1.25 million operations per second, are unparalleled in other LSI processors.

The general DSP architecture is described in Section II. Section III centers on the DSP programming and includes a brief description of the instruction set. An example of a simple program is also given to illustrate the style of the input language. In Section IV, the DSP I/O interface is described. Finally, an overview of the DSP performance in typical filtering applications is given in Section V.

II. ARCHITECTURE

This section presents a description of the DSP architecture. As shown in Fig. 1, the principal features are as follows:

(*i*) a 1024-word by 16-bit ROM for instructions and fixed data;

(*ii*) a 128-word by 20-bit RAM for variable data;

(*iii*) an AAU which generates addresses for the ROM and RAM memories and performs post modification arithmetic on these addresses;

(*iv*) an AU which accepts a 16-bit and a 20-bit operand to form a 36-bit product, accumulates the product with a 40-bit accumulator, and rounds the accumulator to a 20-bit word (with overflow protection) for storage or output;

(*v*) an I/O section which serially receives and transmits either μ-255 law or linear PCM signal samples; and

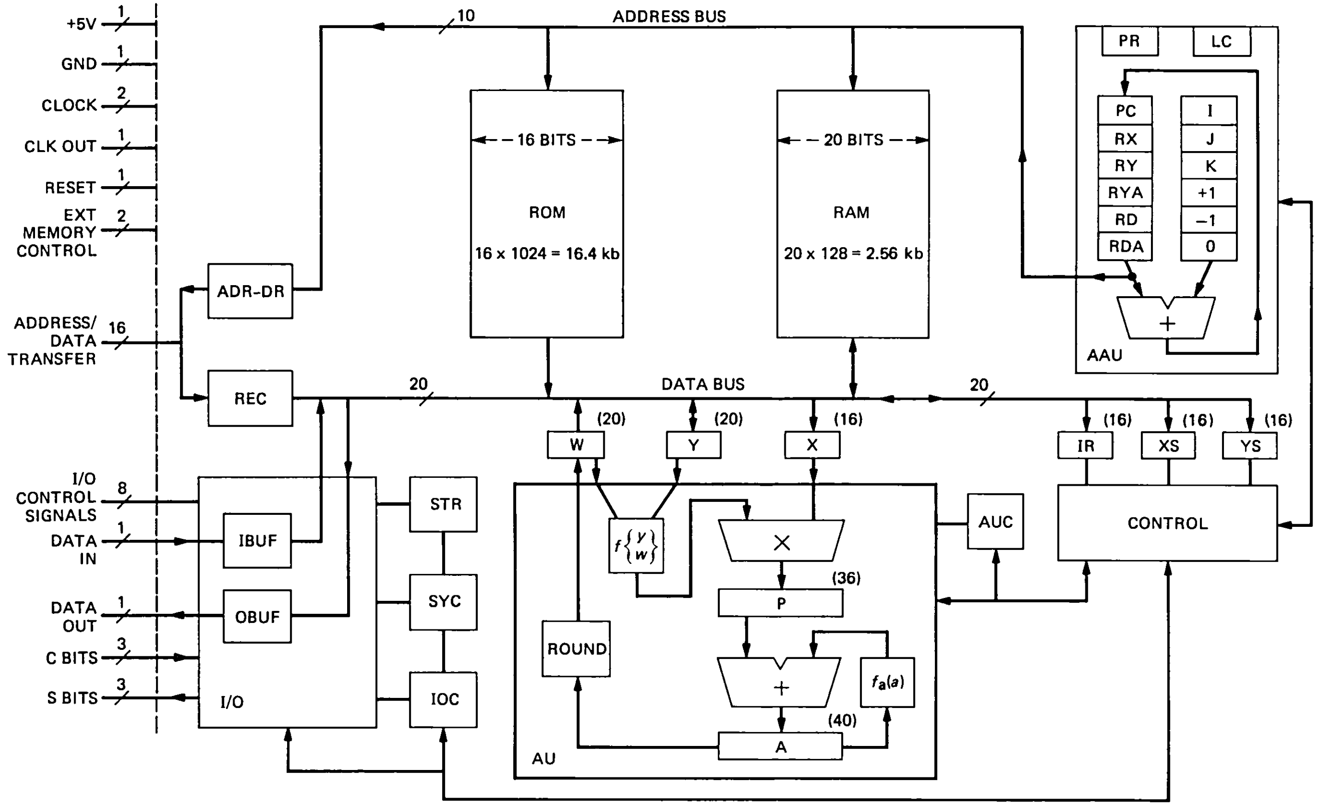


Fig. 1—Digital signal processor architecture.

(*vi*) a Control Unit for instruction decoding and overall system coordination.

The DSP is also able to access a 1024-word by 16-bit external ROM, with no reduction in processing speed. This feature is especially convenient during program development and testing. It is also useful for small volume applications in which the expense of programming the on-chip ROM is not justifiable.

The analysis of many digital signal processing algorithms reveals that they basically perform multiplications and additions. Therefore, the AU was designed to implement these operations efficiently. In its simplest form, the expression evaluated by the AU is

$$x \cdot y + a \rightarrow a$$

where,

x is the 16-bit coefficient in register X, and

y is the 20-bit data word in register Y.

Again, the word lengths for x and y were determined by examining the requirements of a variety of telecommunication applications. A good compromise was established between the hardware required to implement a given precision and the need for a general part, like the DSP, to cover most applications.

The 36-bit product, p, is summed with the 36 least-significant bits of the contents, a, of the 40-bit accumulator, A, and the result is written into A. When the value in A is needed outside the AU (e.g., to write to memory), the contents of A are truncated or rounded, overflow corrected (if necessary), then stored in the 20-bit AU output register, W. The contents of W can then be transferred to other parts of the DSP via the 20-bit data bus, or can be used as data for another arithmetic operation.

The AU is pipelined in three stages: (i) the formation of the product $x.y$, (ii) the addition of the product to a, and (iii) the transfer of a to W. Thus, while this transfer is in progress for any one expression, the addition of the product in P to the contents of A for the next expression is also being performed, and the formation of the product $x.y$ of the following expression is taking place. This pipelined structure keeps all parts of the AU busy at all times and allows the processor to maintain a high throughput.

The full capability of the AU is described by the more general operation

$$x \cdot f\begin{Bmatrix} y \\ w \end{Bmatrix} + f_a(a) \rightarrow a[\rightarrow w],$$

where

x = 16-bit coefficient which is read into register X from the 16 most-significant bits of the 20-bit data bus. This coefficient is normally fetched from ROM, but could also be fetched from RAM or the input buffer.

y = 20-bit data word which is fetched from RAM or the input buffer, and is read into register Y from the data bus. The contents of Y can also be written to the data bus.

a = contents of the 40-bit accumulator A. The four extra bits are provided for overflow protection.

w = rounded or truncated 20-bit AU output word which is stored in register W. The contents of W can be written to the data bus for storage in RAM or for output through the output buffer, or can be used instead of y in another arithmetic operation. The least-significant bit of w corresponds to bit 14 of a. This selection is consistent with the assumption that y and w are integers and that x is usually restricted by $-2 \leq x < 2$. However, other choices are possible by shifting a before reading it.

f = linear or nonlinear function of either y or w, such as the actual value, the absolute value, or the sign function (signum) of one of these variables.

f_a = arithmetic function of a (e.g., scaling of a by 2 or 8) or a logical function of a and p (p AND a).

The 16-bit processor instructions are stored in the ROM. When coefficients are fixed, they will also reside in ROM. Data for the algorithm, whether it comes from the input or is generated by the algorithm, may be stored in the 20-bit-wide RAM. In some applications (e.g., adaptive filters as required in echo cancelers) the coefficients are variable and are stored in the 16 most-significant bits of RAM locations.

Addresses for memory references are generated in the AAU. Four memory addresses, required to access the instruction, the coefficient, and the data (both read and write), are multiplexed onto the 10-bit address bus in each processor cycle, and the corresponding information is multiplexed onto the 20-bit data bus. The program in ROM is accessed by the address stored in register PC, the program counter. Fixed coefficients in ROM can also be addressed by PC. Alternatively, coefficients can also be addressed by the auxiliary register RX, which can point to either ROM or RAM. Data, which is read from RAM, can be addressed by RY or by an auxiliary register RYA, while data can be written to RAM by using addresses in RD or RDA. The primary use of the auxiliary registers RYA and RDA is to allow manipulation of temporary results in a separate section of data memory.

The AAU also provides a selection of possible increments for post-modification of these addresses. Under the direction of a given instruction, the contents of the address registers are applied to the address bus and then incremented in the AAU adder before being restored to the register, ready for subsequent use. The program and coefficients can be structured in ROM so that the contents of PC need only be incremented by +1. The contents of other address registers can be incremented by the amounts 0, +1, or −1, or by the contents of the 8-bit registers I, J, or K, as specified by the instruction. The program return register (PR) shown in Fig. 1, is used to provide a single level subroutine capability. The LC register is a 6-bit loop counter used to provide looping within an algorithm. All the registers mentioned above can be set to arbitrary values. This can be done unconditionally or subject to a particular condition being met.

Instructions from ROM are latched into the instruction register IR and subsequently decoded in the Control Unit. In some auxiliary instructions, e.g., a register set from ROM or a register load from RAM, a 16-bit argument follows the instruction; this argument goes to register XS or YS, respectively. The decoded signals are transferred from the Control Unit to the AU, the AAU, the I/O, and to the various registers, as needed. Arithmetic control information that is relatively invariant within an algorithm (e.g., the type of rounding arithmetic used in moving data from A to W, or the built-in scale factor used in some multiplier operations) is stored in the AUC register. The IOC register stores a similar type of information for the I/O (e.g., the I/O rate, or the size and format of the input and output data words).

The data interaction between the DSP and the outside world is carried out through the I/O structure. Inputs and outputs are processed through the buffer registers IBUF and OBUF, respectively. The I/O interface accommodates a serial transfer of 8-, 16-, or 20-bit words under the control of either the DSP or a variety of external devices (e.g., codecs, microprocessors). Different I/O rates and formats are available to the programmer to facilitate this interfacing. Additional details will be given in Section IV.

The setting (under program control) of register SYC allows the user to suspend the DSP operation until a condition specified by one of the fields of this register is met. This can be used to synchronize the processor program with the data sample rate. The available conditions are input buffer full, output buffer empty, or the status of one of the two dedicated logical inputs C0 and C1. A control input, $\overline{\text{CST}}$, can be used to latch internally the values of C0 and C1. Similarly, the setting of register STR allows the user to output directly one or two logical signals (S0 and S1) and/or a synchronization pulse (STB).

The serial I/O and its control require another ten pins; they are described in Section IV. Sixteen of the 40 pins (DBS0-DBS15) are dedicated to the external data bus, which is used to access external ROM in place of the internal, mask-programmable ROM. The remaining eight pins are used as follows:

(i) two for the +5 V power supply (VCC) and ground (VSS);

(ii) three for the crystal connection (XTAL), the clock input (CLKIN), and the clock output (CLKOUT);

(iii) one for resetting the DSP to a starting point ($\overline{\text{REST}}$); and

(iv) two for external memory control ($\overline{\text{EXM}}$ and $\overline{\text{EXE}}$).

The external ROM is accessed by setting $\overline{\text{EXM}}$ low; $\overline{\text{EXE}}$ combined with CLKOUT allows the generation of signals needed to latch the address coming out of the DSP through the DBS pins, latch the data fetched from the external ROM, and enable these data onto the DBS pins.

III. PROGRAMMING

The DSP has two types of instructions: normal and auxiliary. The normal instructions control processor computations in the AU to evaluate the general expression given in the last section. The three AU operations of product formation, accumulation, and transfer to the AU output register W (if required), are fully completed in one cycle of the processor. The operations are performed in parallel, each one corresponding to the partial evaluation of a different expression.

For a normal instruction, a typical symbolic assembler input line consists of up to four expressions indicating

(i) the source and destination of the data to be transferred out of the AU, with the destination address register increment,

(ii) the control of the AU output register contents,

(iii) the function to be performed by the accumulator, and

(iv) the product to be formed by the multiplier, including a specification of the operand address registers and increments.

When true program constants are used for product operands they may be indicated directly in the expression rather than indirectly through an address register.

At the machine level, the 16-bit instruction has fields that control the above-mentioned functions, including the information needed to read the coefficient and data required in a later AU operation, and to write the result of a previous AU operation. Constants to be loaded into the X register are also 16 bits wide and are stored in ROM following the corresponding instruction.

Auxiliary instructions are used to control noncomputational aspects of the DSP, such as initialization of address registers and conditional inhibition of certain processor functions. They can also specify an additional set of computational operations for the AU, such as com-

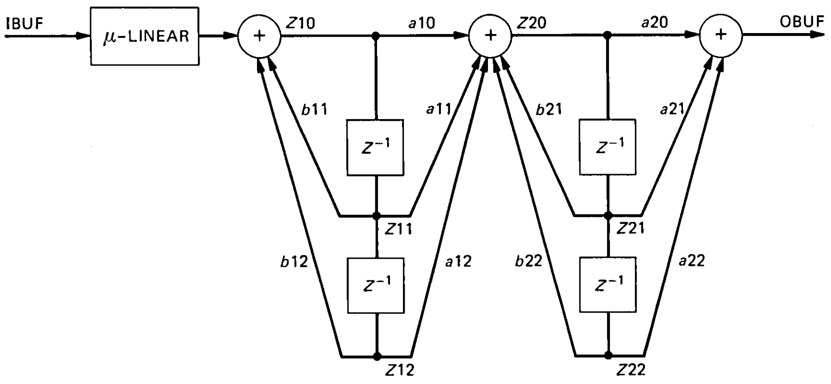


Fig. 2—Fourth-order recursive filter.

pressed/linear conversions or large shifts of the accumulator contents, which do not require the general argument flexibility available in normal computations. The assembler input for these instructions indicates in a simple format the special functions that they specify, as can be seen, e.g., in the register set instruction of the example below.

At the machine level, a 16-bit auxiliary instruction is always followed by a 16-bit argument which is interpreted either as an extension of the instruction itself or as data associated with the instruction. Both normal and auxiliary instructions have common fields that allow writing of previous results or fetching of information required for later operations.

Many features of the DSP are illustrated in a simple example of a fourth-order recursive filter shown schematically in Fig. 2 and in the assembler input code below. The filter has a μ-law input from the input buffer, two five-multiply second-order modules, and a linear output to the output buffer. The program begins at line 1 with a series of auxiliary instructions for initializing the DSP. The first seven instructions are unconditional register set operations. The constants IOC and AUC, to be written into the corresponding registers, reflect the desired options for I/O and AU operations. The increment registers I, J, and K are set to 1, −1, and −3, respectively. Registers RY and RD are set to 0, the address of the first RAM location. The constant SYC, to be written into the respective register, reflects the desired condition for suspending the DSP operation.

Assembler input code for fourth-order recursive filter

```
1:                      ioc = IOC;
2:                      auc = AUC;
3:                      i = 1;
4:                      j = −1;
```

```
 5:                                k = -3;
 6:                                ry = 0;
 7:                                rd = 0;
 8: loop:                          syc = SYC;
 9:                                a = p + a     p = mtl1 (ibufy);
10:                      w = a     a = p         p = mtl2 ();
11:   obuf = w                     a = p + a     p = b12**ry++i;
12:                                a = p + a     p = b11**ry++j;
13:                                a = p + a     p = a12**ry++i;
14:   *rd++i = y         w = a     a = p         p = a11**ry++i;
15:   *rd++i = w                   a = p + a     p = a10*w;
16:                                a = p + a     p = b22**ry++i;
17:                                a = p + a     p = b21**ry++j;
18:                                a = p + a     p = a22**ry++i;
19:                                pc = & loop;
20:   *rd++i = y         w = a     a = p         p = a21**ry++k;
21:   *rd++k = w                   a = p + a     p = a20*w;
```

The instruction in line 8 is the first instruction in the main operating loop of the program. (This loop processes each sample through the filter.) Its function is to suspend the processor until the selected external synchronizing event occurs. This is the method used in this example for synchronization with an external sample rate clock. Lines 9 and 10 are auxiliary instructions which perform the μ-law to linear conversion. This conversion is done on data which was fetched from IBUF. The accumulation, transfer to W, and write to OBUF in lines 9, 10, and 11 refer to the operations that were begun at the end of the loop. The practice of meshing the tail of the loop with its head is essential for writing low overhead code for this pipelined machine.

The RAM memory organization for this program is shown below:

Location	**Variable**
0	$Z12$
1	$Z11$
2	$Z22$
3	$Z21$

where the Zs are the state variables shown in Fig. 2. The values in registers I and J are used to modify the addresses in registers RY and RD so that these variable locations may be referenced. The K register resets these addresses after they are used for the last time in the loop with no additional overhead. The filter coefficients ($b12$, $b11$, $\cdots$, $a21$, $a20$) are stored in-line with the code.

The instruction that sets the PC for the end-of-loop branch is at line 19 instead of at the actual end of the loop. This is because of the pipelined architecture. When the machine is executing the branch instruction at line 19, it is already decoding the instruction at line 20 and is fetching the instruction at line 21. Therefore, the next instruction to be fetched will be at line 8.

In this example, there are only two DSP cycles of overhead in the loop (the setting of SYC and PC). The total loop has 14 cycles and could accommodate a sample rate of up to 89 kHz.

IV. INPUT/OUTPUT INTERFACE

The DSP architecture is designed to facilitate system interface with a minimum number of external components, if any. The I/O transfer of information is performed serially. The DSP I/O structure provides serial-to-parallel conversion of input data, and parallel-to-serial conversion of output data. Input and output operations are carried out in independent sections, thus, permitting them to be asynchronous with respect to each other, as well as with respect to the program execution. Several signals control the I/O transmissions.

Five DSP pins are dedicated to the input serial transfer and its control, and five pins are dedicated to the output. The beginning of a serial transfer is indicated by a synchronization signal present at the ISY (input synchronization) pin for an input, or at the OSY (output synchronization) pin for an output. Input data bits are received at pin DI and advanced into the IBUF register of the DSP by the clock signal present at pin ICK (input clock). Output data bits are available at pin DO and are shifted out ofthe OBUF register of the DSP by the clock signal present at pin OCK (output clock). The two enable lines $\overline{\text{CTR}}$ (not clear to read) and $\overline{\text{CTS}}$ (not clear to send) can be used to activate the input and output sections, respectively. A high level on one of these pins causes the DSP to be inactive on all the pins associated with that particular section, and tristates the corresponding off-chip drivers. This allows several DSPs to be switched on and off a single external I/O bus. The flags IBF (input buffer full) and OBE (output buffer empty) indicate the status of the respective buffers. These flags can be used to control external hardware and synchronize data transfers between the DSP and its peripherals. They can also be internally tested by certain DSP instructions.

The DSP I/O unit is programmable via the 10-bit IOC register. This register configures the input and output sections of the DSP to either generate or accept the clock and synchronization signals. If a section of the DSP generates these signals, it is said to be in the ACTIVE mode; otherwise, it is in the PASSIVE mode. The IOC also controls the length of the serial data transfer to be either 8, 16, or 20 bits. In addition, the IOC controls the I/O clock rate for active mode. The rate can be either ½, ¼, 1/32, or 1/64 of the DSP input clock. Finally, for both input and

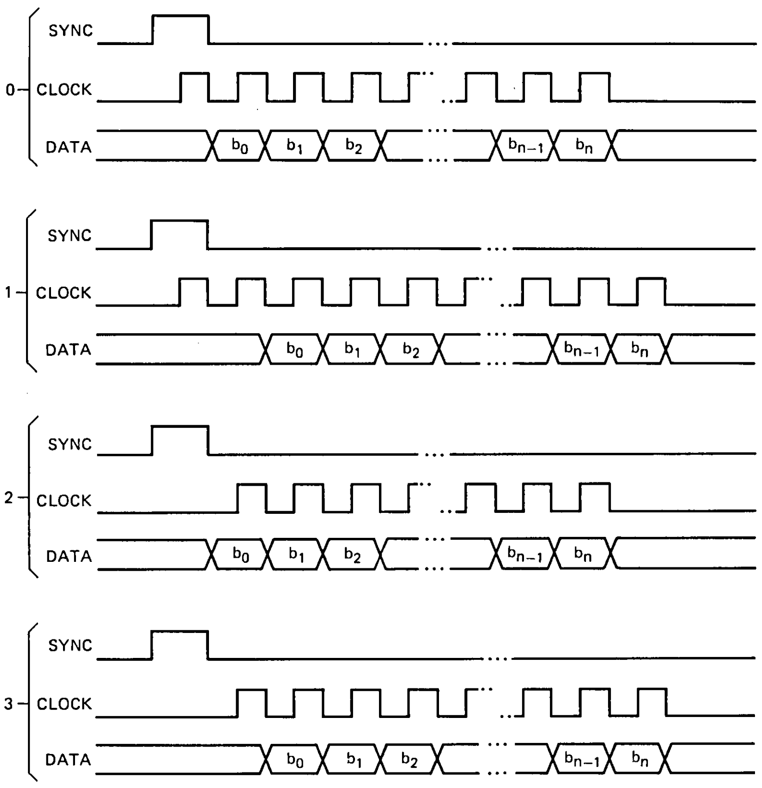


Fig. 3—Input/output active formats.

output, the IOC determines the timing relationship between the synchronization signal, the clock signal, and the first bit of data to be transferred. This is done by selecting one of four formats. Figures 3 and 4 display the various DSP formats for the active and passive modes, respectively. These formats allow the DSP to be readily interfaced to a variety of circuits and systems. In the ACTIVE mode, the DSP generates a burst of clock pulses whose number is a function of the selected format and the length of the serial transfer. In the PASSIVE mode, the synchronization and clock signals are supplied by an external source. The DSP accepts a continuous I/O clock. It should also be emphasized that the input and output sections are independently programmed except for the I/O transfer rate in the ACTIVE mode.

V. PERFORMANCE

The amount of signal processing that can be performed by the DSP depends on the cycle time t_c, which is the time for basic machine operations, such as a multiply, or the setting of a register. Specifically,

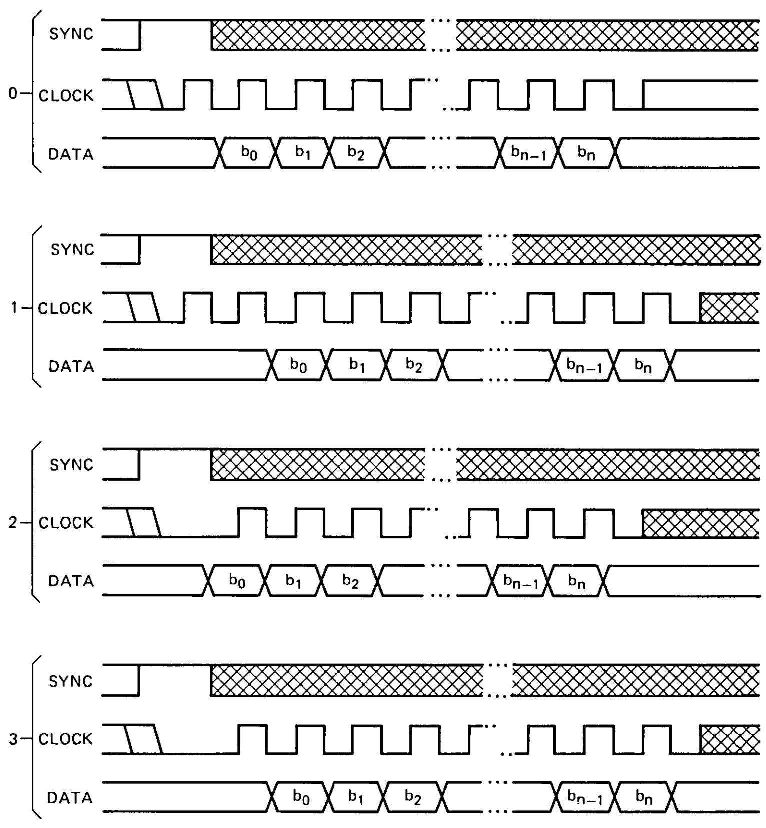


Fig. 4—Input/output passive formats.

if the sampling frequency is f_s (Hz), and the cycle time is t_c (s), the number of machine cycles available (AC) per sample period is

$$\text{AC} = \frac{1/f_s}{t_c}.$$

A basic machine operation requires four processor states. For a processor state period of 200 ns (5-MHz clock), the cycle time t_c is 800 ns; for an 8-kHz sampling rate, this provides 156 cycles per sample period.

The capacity for basic signal processing algorithm segments is now simply determined. For example, a five coefficient second-order recursive filter section (Fig. 2) requires five machine cycles and, thus, the DSP, running at 5 MHz, can execute 31 sections at the 8-kHz sampling rate. In practice, of course, one cannot implement as many sections since there will be a variety of other tasks, such as initialization and I/O, which will somewhat reduce the amount of filtering in an actual application.

Next, consider the size of the ROM. In most applications, each machine cycle requires one word of ROM for the instruction and one word of ROM for the coefficient. At the 8-kHz sampling rate, there are 156 available cycles; if all of these cycles are used in the algorithm, the ROM would need 156 + 156 = 312 words. However, in some applications, more than one coefficient can be associated with an instruction. An example is when a section of the algorithm is looped over more than once—each encounter of an instruction in the loop can be associated with a different coefficient. A study of such applications, as well as applications where two or more alternative programs must be resident in the processor at once, led to the 1024-word size for the DSP ROM.

The RAM size depends on the number of data words that need to be stored. For the recursive structures shown in Fig. 2, two items of data must be stored for each second-order section (SOS). Thus, if an 8-kHz sample rate is assumed and the maximum number of 5-multiply SOSS were programmed, then $31 \times 2 = 62$ words would be used. With 4-multiply SOSS, 78 words would be required. The RAM size for DSP is 128 words which is quite sufficient for the recursive filter applications. In the case of nonrecursive FIR filters, where one needs one storage location for each multiplication, this RAM size will allow a 128-tap filter. These results are summarized in Table I.

Table I—Performance features of DSP

16 by 20	Multiply-add (16-bit coefficient 20-bit data)
800 ns	Cycle time
16,384-bit	ROM (1024 words by 16 bits/word)
2,560-bit	RAM (128 words by 20 bits/word)
31	5-multiply SOSS @ 8-kHz sample rate
39	4-multiply SOSS @ 8-kHz sample rate
128	FIR taps @ 8-kHz sample rate

The level of performance is such that the signal processing required for a dual-tone multifrequency receiver, or a low-speed FSK modem can be implemented in a single DSP device.

Some further specifications of the DSP are given in Table II.

The circuit consists of approximately 45,000 transistors (with program ROM) within a 68.5-mm^2 chip area.

VI. CONCLUSION

The integrated circuit DSP described in this paper was designed not only to serve as a programmable processing element for general-

Table II—Specifications for DSP

Power supply	+5 V
Clock frequency	5 MHz
Max. I/O serial rate	4.5 Mb/s
Max. power dissipation	1.25 W
Package	40-pin DIP

purpose use in telecommunications applications, but to further enhance its use in these applications by reducing development effort. The DSP, therefore, has not only the processing capacity and precision for a number of common, small applications for both voice and data processing, but also may be easily interfaced to system data streams and to other processors to realize complex algorithms.

The DSP is the first element of a family of devices that is being developed at Bell Laboratories for digital signal processing in telecommunication applications.

VII. ACKNOWLEDGMENTS

The authors wish to acknowledge the efforts of F. E. Barber, T. J. Bartoli, D. B. Cuttriss, R. L. Freyman, J. A. Grant, B. R. Jones, J. Kane, R. A. Kershaw, C. R. Miller, H. E. Nigh, N. Sachs, W. A. Stocker, E. F. Schweitzer, and W. Witscher, Jr. in the implementation of this circuit and the guidance of H. C. Kirsh, R. A. Pedersen, and D. C. Stanzione throughout the project.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Logic and Fault Simulations

By I. I. ELDUMIATI and R. N. GADENZ

(Manuscript received December 30, 1980)

This paper illustrates a methodology for the design verification and testing of the Bell System digital signal processor. It is shown that a behavioral approach, as opposed to a structural approach, is advantageous for the generation of a first set of test vectors, since this set (i) exercises all the functions, as they are specified by the instruction set, and (ii) uncovers the bulk of the faults. The set can then be improved using the structural approach. The participation of the device designers in this process is essential. The relation between fault coverage and yield is also discussed. Theoretical relations are given which show how important it is to have a high-fault coverage (say, >95 percent) for VLSI chips.

I. INTRODUCTION

In this paper, we describe the logic simulation and fault analysis of a programmable VLSI digital signal processor (DSP) developed by Bell Laboratories.[1] The design of such a complex integrated circuit requires an extensive effort in the areas of design verification, testability, and fault coverage. Such an effort has a considerable impact upon the design cycle, yield, and reliability of the device.

In the following section we discuss design verification, which was done in software through computer simulations. Section III presents testing and the associated problem of generating test vectors. Computer simulations of the faulted circuit allow us to determine the fault coverage obtainable with a set of input vectors. The relation between fault coverage and true yield is discussed in Section IV.

II. DESIGN VERIFICATION

The design verification of a VLSI logic circuit could be done either in software via a computer model or in hardware by building a breadboard, or in both. The software approach is easier to set up and more flexible to use and modify. On the other hand, once built, a breadboard can be used not only for design verification but also for real-time testing and the development of support hardware. Early users can also benefit from it for their initial system design. However, a hardware model is usually built with SSI and MSI components and requires, therefore, an adaptation of the original circuit. The breadboard could be constructed so that it reflects the state of the circuit on a clock cycle basis, but it is very difficult to emulate dynamic structures and bus precharge circuits. As a result, design problems resulting from the use of such configurations may be masked in a breadboard.

Computer models for VLSI design verification may be a functional description in a high-level language, such as ADLIB,[2] a gate level description as in LAMP,[3] or a transistor level representation as in MOTIS[4] and SPICE.[5] Functional analysis provides a coarse simulation and its use is limited to the initial stages of the device conception. On the other hand, a transistor level description is quite complex and costly. It is most useful for the analysis of critical timing paths. A gate-level description can be utilized both for design verification and fault analysis. A further advantage is that it can also be used directly for automatic routing, as in LTX,[6] during chip layout. Computer aided automatic routing was used for the layout of several DSP sections that have relaxed performance requirements.

In the design verification of the DSP, a functional description language was used as a preliminary check for some particularly complex sections. LAMP was used throughout the design phases of the device, first to verify the logic design of the individual sections and then to simulate the complete device. In its final form, the LAMP computer model uses a gate level description for the random logic section, which consists of approximately 14,000 transistors, and a functional description for the memories. MOTIS and SPICE were also extensively used to analyze the behavior of the time-critical portions of the device.

Figure 1 illustrates the LAMP structure. The source file for the computer model is written in a language known as LSL-LOCAL (a combination of Logic Simulation Language and Logic Circuit Analyzer Language). The same description can be used for MOTIS and LTX. The use of a common source language for the logic and timing simulators, as well as the automatic router, has an obvious advantage toward generating an error free layout. The LSL-LOCAL provides a description of the various circuit components and their interconnections, using standard logic gates and, whenever possible, a library of NMOS subnet-

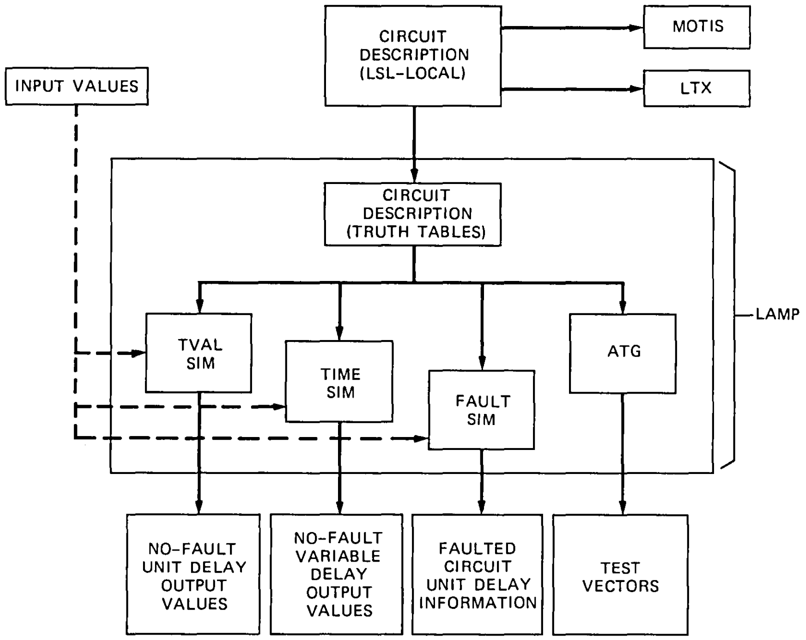

INPUT VALUES
CIRCUIT DESCRIPTION (LSL-LOCAL)
MOTIS
LTX
CIRCUIT DESCRIPTION (TRUTH TABLES)
LAMP
TVAL SIM
TIME SIM
FAULT SIM
ATG
NO-FAULT UNIT DELAY OUTPUT VALUES
NO-FAULT VARIABLE DELAY OUTPUT VALUES
FAULTED CIRCUIT UNIT DELAY INFORMATION
TEST VECTORS

Fig. 1—Lamp structure.

works or polycells. LAMP transforms this description into an object file which is a set of truth tables.

The LAMP true-value simulator uses the truth tables combined with a set of input vectors to check the behavior of the circuit under normal or unfaulted conditions. Each test vector specifies a set of values (1 or 0) at the circuit inputs for each clock phase. Gate delays are uniform (unitary) throughout the circuit. A zero gate delay can also be specified to better simulate the structural behavior of complex cells. For each vector, LAMP simulates the propagation of signals from inputs to outputs taking time steps equivalent to the unitary gate delay. By examining the values of the output signals, which are either 0, 1, or 3 ("don't know"), it is possible to verify the gate level performance of the circuit, as well as identify long circuit paths, races, and oscillations. An oscillation is declared if an output does not settle within a prudent number of time steps specified by the user.

The diagram in Fig. 2 outlines the steps followed in the design verification procedure for the DSP. Once the results of the true value simulation were satisfactory, timing simulations were carried out both on MOTIS and SPICE. MOTIS was used to check the overall timing performance of the random logic portion of the DSP (approximately 14,000 transistors). SPICE simulation was extensively used in areas where the device performance had critical timing requirements. These include the processor clocking system, the bus interface and precharge circuitry, critical paths with long delays or excessive loading, and places where races may occur. During the initial stages of layout, the timing simulations utilized estimated values of the parasitics; actual values were substituted at a later stage when needed.

III. TESTING AND FAULT ANALYSIS

The DSP architecture facilitates testability and program development. The DSP is customized to perform signal processing functions by means of an on-chip ROM which stores both program and fixed data. However, the ability to access an external ROM is also provided. This external memory interface feature allows emulating the DSP program and provides a means for device testing. Address information and data are multiplexed on the external bus pins, with hand shaking signals indicating the presence of address or need for data. These signals can also be used by an automatic tester to either force an input vector or compare output data. In addition, to help in the debugging process, the chip layout was partitioned and internal pads were provided, thus

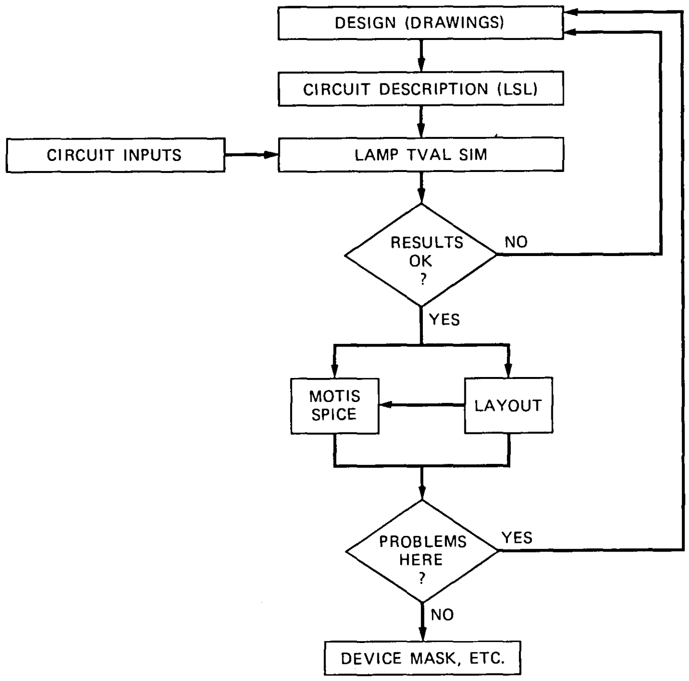


Fig. 2—Steps in the design verification.

allowing the possibility of independently exercising and testing each of the DSP sections.

The input vectors needed for design verification were selected so as to exercise all DSP functions, which are specified by the instruction set. These functions were combined with data streams of either alternating zeros and ones and their complements, or specific data patterns for functions that exhibit a known pattern sensitivity. The task of generating the vectors was further simplified by the use of the DSP Assembler, which translates a functional input into machine code and deals with some specific architectural features of the DSP, such as pipelining and skewing of certain instruction fields. In addition, using this behavioral or functional approach, the expected outputs were easily predicted. In summary, this approach to generate the vectors required for design verification proved adequate.

The same set of vectors served as an initial input to LAMP for the fault simulations, and was able to detect the bulk of the faults. As a result, it became the main portion of the test vectors subsequently used for testing the DSP devices. Recently, Szygenda suggested that it seems reasonable to expect that this procedure will be successful.[7] Our experience confirms that this is the case. Thus, we believe that the behavioral approach to test vector generation is to be preferred to the structural approach, at least as a first step. The latter approach aims at sensitizing each node of the circuit and propagating the effects to the outputs, using a set of vectors which may not represent necessarily meaningful device functions. The process is both lengthy and costly. Currently available programs for automatic test generation (such as the ATG feature of LAMP or Teradyne's P400) are also based on the structural approach; as a result, they are limited in capability and expensive to use, especially for devices with such complexity as the DSP.

The faults exercised in LAMP are gate inputs and outputs stuck at either zero or one, with the excitation and observation points being at the pins. For each input vector, LAMP considers one fault at a time and compares the outputs of the faulted and unfaulted circuits. A fault is detected if a change is observed at the output pins. Faults that have equivalent effects on the output are collapsed in order to reduce computational cost. Also, once a fault has been detected, it is possible to remove it from the list of faults, so that the following vectors will not have to consider it. The LAMP simulation provides a list of all-test-passed (ATP) faults, as well as information on possible races and oscillations caused by the faults. From these data, the fault coverage and subsequent steps to improve it can be determined.

Figure 3 displays the fault coverage given by LAMP versus the number of test vectors used to verify the DSP random logic. The fault

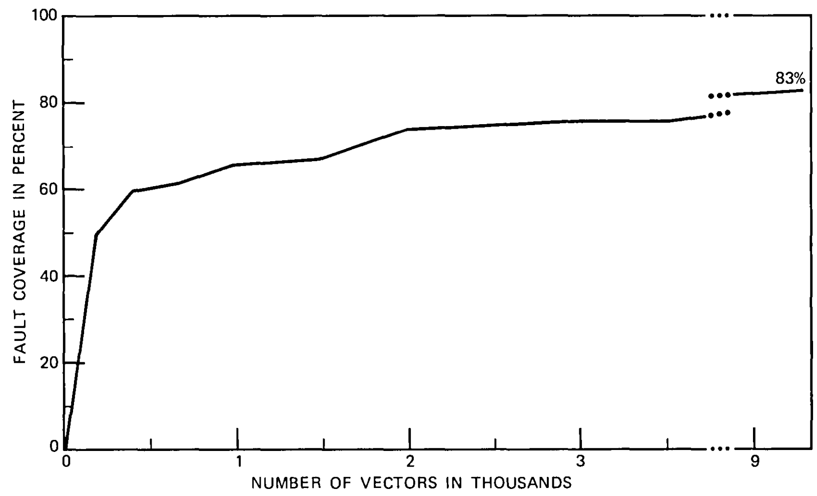


Fig. 3—Fault coverage for DSP random logic.

coverage achieved with ~9000 vectors generated via the behavioral approach was 83 percent. This is an excellent starting point in the quest for a high fault coverage. Our goal was to obtain a fault coverage in excess of 95 percent. Analysis of the undetected faults and the structures used in the circuit implementation revealed that the actual fault coverage is significantly higher than the value given by LAMP. This is because faults not observable due to built-in circuit redundancies can be disregarded along with the faults which are not detected in the simulator but will be detected in the actual circuit. The DSP timing utilizes a four-phase clock with non-overlapping master and slave pulses in each phase. These pulses are used to achieve signal transfers between registers. The master and slave pulses are generated locally in each of the DSP sections, and are kept synchronous through a universal synchronizing signal. This clocking scheme is tolerant to certain classes of faults such as stuck slaves if the data is synchronous. When reclocking is done at the boundaries of the DSP sections, some faults may be disregarded if SPICE simulations indicate proper timing at those boundaries. Other examples of faults that can be neglected are the ones resulting in switched depletion loads being always on, if the device meets the maximum power dissipation requirement, and the undetected faults that are due to unassigned instruction fields.

To achieve high reliability, the test vectors should guarantee an extensive fault coverage which should cover not only the device functions, but also the structures used for the circuit implementation. This is especially important because the DSP is programmable and the test vectors are designed to be independent of the program in the on-chip ROM to avoid costly test program development. A more in-depth look at the circuit was required to detect at least part of the remaining faults and further improve the fault coverage beyond 95 percent. Additional specific sequences of test vectors had to be generated by means of a structural approach. These sequences were applied to exercise the faults not previously detected and to propagate their effects to the output pins.

The vectors used to test the DSP random logic exercise only a limited number of RAM locations. Therefore, the RAM is further tested by writing into it and reading from it standard checkerboard patterns. When reading, the contents of the different memory locations are accumulated into a checksum which is sent to the output and compared with the expected value. The contents of the on-chip ROM is also verified via a checksum test. The DSP external memory interface makes it possible to treat the contents of the internal ROM as data which is fetched sequentially to compute the checksum. The result is sent to the output where it is compared with the precalculated value, determined from the user's program. This value is the only difference in the complete testing patterns of different DSPs. The additional vectors required to test the RAM (~3600) and the ROM (~4300) bring the total number of test vectors used for the DSP to slightly above 20 K.

IV. FAULT COVERAGE AND YIELD

The fault coverage provides a measure of the fraction of the faults detected by a given set of test vectors. A fault coverage less than 100 percent implies that some devices which passed the test may fail to execute the user's program. This could be the result of using certain program sequences or data patterns that exercise faulted nodes not covered by the test vectors. The presence of such devices affects the reliability and the eventual cost of the host system. Identifying faulty devices during incoming inspection, if any, or during system subassembly has some impact on the cost. Failure in the field results not only in a reduced system reliability, but also in a higher replacement cost and the possibility of loss of service. Therefore, it is very important to identify faulty devices as much as possible during device testing. In this section, the relationships among yield, fault coverage, and chip area are discussed.

The reduction in a wafer yield y can be attributed to two sources. The first one is the existence of area defects, i.e., defects that cause whole portions of a wafer to provide no good devices. This area defect condition is represented by the parameter y_0 in the equation below. The second source is the existence of fatal point defects which are randomly distributed over the wafer area where good chips can be found. These assumptions result in the following expression for the yield y of a wafer

$$y = y_0 e^{-DA}, \tag{1}$$

where

y_0 = the area defect yield factor,

D = the point defect density, and

A = the chip active area.

In order to account for the spread of the random defect density D among wafers, Murphy[8] suggested that the defect density is distributed according to a probability density function. Assuming a gamma distribution for the defect density D, the average yield Y, for a very large number of wafers, is given by[9-11]

$$Y = \frac{Y_0}{(1 + \lambda D_0 A)^{1/\lambda}}, \tag{2}$$

where

Y_0 = the average area defect yield factor,

D_0 = the average value of the defect density, and

λ = the variance of the defect density.

It should be noted that the gamma distribution provides the best fit to experimental yield data.[9] In addition, depending on the value of λ, it encompasses several distributions which were proposed earlier. (See Refs. 8 and 11 to 13.) Therefore, eq. 2 will be used to study the relationship between fault coverage and yield.

A fault coverage less than 100 percent indicates a lack of observable exercise for some of the logic gates making up the circuit. Assuming a uniform distribution of logic gates over the chip active area, the effective chip area A_p being probed can be expressed as a function of the fault coverage as follows,

$$A_p = FA, \tag{3}$$

where F is the fractional fault coverage for a given set of test vectors. The ratio between the true yield Y_t, obtainable with a 100 percent fault coverage, and the yield at probe Y_p is then given by

$$\frac{Y_t}{Y_p} = (1 + \lambda F D_0 A)^{1/\lambda} \cdot (1 + \lambda D_0 A)^{-1/\lambda}. \tag{4}$$

This expression can be used to determine the fault coverage required to achieve a desired value for Y_t/Y_p:

$$F = \frac{(Y_t/Y_p)^\lambda\,(1 + \lambda D_0 A) - 1}{\lambda D_0 A}. \tag{5}$$

The dependence of the yield on the fault coverage for several values of λ and a $D_0 \cdot A$ of 3, is displayed in Table I.

The parameter λ is a function of the fabrication facility and could be determined from the yield data. For simplicity, assume that in the limit λ approaches zero. Then eq. 2 reduces to

$$Y = Y_0 e^{-D_0 A}, \tag{6}$$

and eq. 5 becomes

$$F = 1 + \frac{1}{D_0 A} \ln \frac{Y_t}{Y_p}. \tag{7}$$

This expression is plotted in Fig. 4 for various values of $D_0 \cdot A$. The figure emphasizes the need for extensive fault coverage as the value of $D_0 \cdot A$ is increased. For example, in order to achieve a value of 0.9 for Y_t/Y_p and assuming a value of 3.0 for $D_0 \cdot A$, the fault coverage should be 96.5 percent.

V. CONCLUSIONS

A methodology for the design verification, fault analysis and testing of a programmable VLSI device was presented. Logic design verification was performed at the gate level through LAMP true-value simulations. Sections of the device having critical timing requirements were verified via MOTIS and SPICE.

A behavioral approach to test vector generation, in which all the device functions were exercised with appropriate data, proved adequate for the design verification. Through fault analysis, it was found that these vectors also uncovered the bulk of the faults and, therefore, could be used for testing the device. The structural approach, which is both lengthy and costly, was used only to generate additional vectors in order to further improve the fault coverage.

The need for an extensive fault coverage, and its impact on the device cost and reliability, was emphasized. A relationship between

Table I—Y_t/Y_p vs. F for $D_0 \cdot A = 3$

	Y_t/Y_p		
F[%]	$\lambda = 0$	$\lambda = 1/3$	$\lambda = 1$
80	0.549	0.729	0.850
85	0.638	0.791	0.887
90	0.741	0.857	0.925
95	0.861	0.927	0.963
98	0.942	0.970	0.985

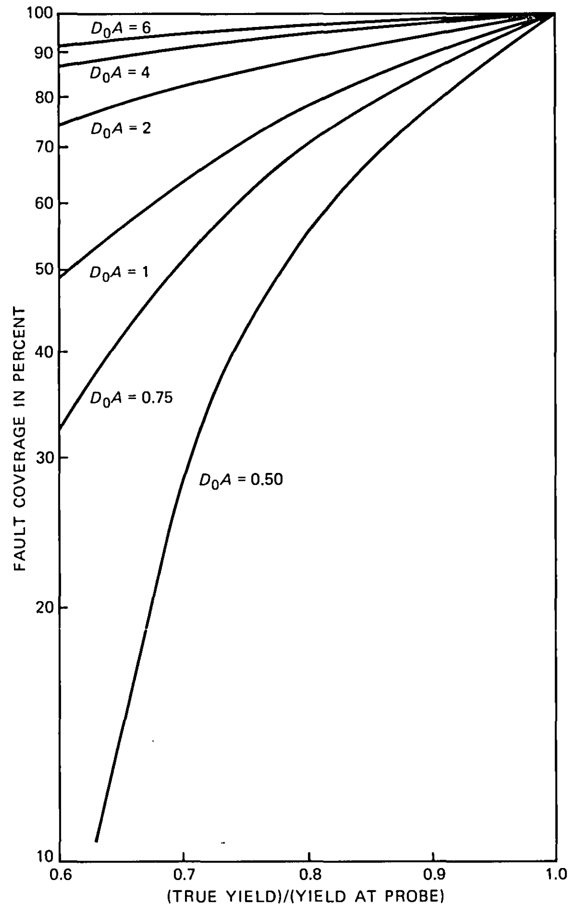


Fig. 4—Fault coverage as a function of Y_t/Y_p and D_0A.

fault coverage, yield and chip area was established. The analysis shows that it is important to have a fault coverage in excess of 95 percent for chips with large areas.

VI. ACKNOWLEDGMENTS

The authors wish to thank J. R. Boddie, N. J. Elias, H. Shichman, D. C. Stanzione, and R. L. Wadsack for useful discussions and comments.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.

2. D. D. Hill, "ADLIB: A Modular, Strongly-Typed Computer Design Language," Proc. Fourth Int. Symp. Computer Hardware Description Languages, Palo Alto, California, October 1979, pp. 75-81.
3. "LAMP: Logic Analyzer for Maintenance Planning," several papers in B.S.T.J., *53,* No. 8 (October 1974), pp. 1431-555.
4. B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS: An MOS Timing Simulator," Trans. on Circuits and Systems, *CAS-22,* No. 12 (December 1975) pp. 901-10.
5. L. W. Nagel and D. O. Pederson, "SPICE—Simulation Program with Integrated Circuit Emphasis," Memorandum No. ERL-M382, Electronics Research Laboratory, University of California, Berkeley, April 12, 1973.
6. G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX—A System for the Directed Automatic Design of LSI Circuits," Proc. 13th Design Automation Conference, San Francisco, California, June 28-30, 1976, pp. 399-407.
7. S. A. Szygenda, "Recent Results on Simulation and Testing for Large Scale Networks," Workshop on Large Scale Networks and Systems, IEEE 1980 Symp. on Circuits and Systems, Houston, Texas, April 28-30, 1980, pp. 22-5.
8. B. T. Murphy, "Cost-Size Optima of Monolithic Integrated Circuits," Proc. IEEE, *52* (December 1964), pp. 1537-45.
9. C. H. Stapper, "Defect Density Distribution for LSI Yield Calculations," IEEE Trans. on Electron Devices, *ED-20,* No. 7 (July 1973), pp. 655-7.
10. C. H. Stapper, "On a Composite Model to the IC Yield Problem," IEEE J. of Solid State Circuits, *SC-10,* No. 6 (December 1975), pp. 537-9.
11. J. Sredni, "Use of Power Transformations to Model the Yield of ICs as a Function of Active Circuit Area," Proc. Int. Electron Device Meeting, Washington, D.C., December 1975, pp. 123-5.
12. A. G. F. Dingwall, "High Yield Processed Bipolar LSI Array," Int. Electron Devices Meeting, Washington, D.C., October 1968.
13. J. E. Price, "A New Look at Yield of Integrated Circuits," Proc. IEEE, *58* (August 1970), pp. 1290-1.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Software Simulator

By J. AAGESEN

(Manuscript received June 23, 1980)

*One of the development aids for the digital signal processor (*DSP*) is a software simulator,* dspsim, *which runs interactively under the UNIX* operating system. It is a program debugging tool which can be used without access to the* DSP *hardware environment. It allows the user to monitor run-time characteristics of* DSP *programs which cannot be observed using the device itself. It is very flexible in providing capabilities for single or multiple program stepping, setting and modifying conditional breakpoints, examining register contents and generating data plots on the terminal.*

I. INTRODUCTION

A number of development tools have been designed for the single-chip digital signal processor (DSP).[1] This article describes a software simulator for the DSP, dspsim, which runs under the *UNIX* operating system. The simulator provides an interactive program development and debugging facility which operates exclusively in the *UNIX* environment with no need for the DSP and associated hardware. It includes general input/output handling and offers great flexibility in its ability to access registers, set breakpoints, and take specified action when prescribed conditions are met. Also, it has the capability of printing x-y plots on the terminal. Execution can be interrupted at any time for observation of register contents, change in breakpoint conditions, etc., after which execution can be resumed without loss of continuity. Creation of programs is facilitated by the DSP assembler[2] which generates a file that the simulator can load directly into its program

* Registered trademark of Bell Laboratories.

memory. Diagnostic messages are printed in response to erroneous operations and special DSP conditions.

This paper covers the architecture of the simulator, the handling of DSP conditional auxiliary instructions, and a discussion of the simulator commands. It concludes with a brief terminal session illustrating the operation of the simulator.

II. ARCHITECTURE

2.1 Overview

A block diagram of dspsim is shown in Fig. 1. The DSP box represents the simulation of the basic DSP architecture as described in Ref. 1, excluding the RAM and the ROM. The operation of the simulator is controlled by the simulator executive system which interprets commands and invokes required utility routines. A number of files are associated with the simulator. The RAM file corresponds to the random-access memory of the DSP. The program file (PGM) provides the read-only memory function. The input stack (IS), performing the function of a signal source, contains data that are to be read into the input

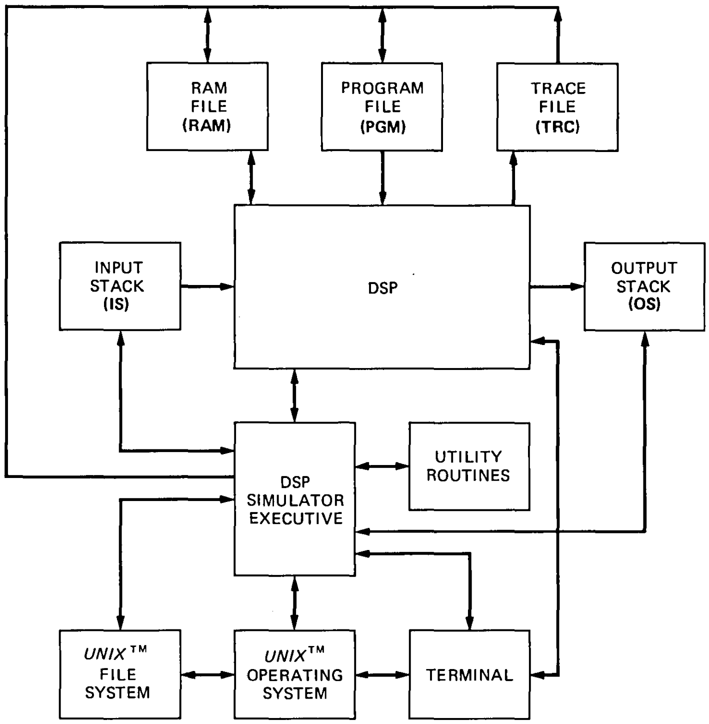


Fig. 1—Block diagram of DSP simulator.

buffer, IBUF, of the DSP. The output stack (OS) collects output data from the DSP output buffer, OBUF. The trace file (TRC) keeps a record of program branches. The simulator can access files in the *UNIX* file system. This provides for off-line storage of DSP files so that DSP files can be loaded from and written to *UNIX* files.

2.2 Data formats

Data to be entered into registers directly from the terminal or from *UNIX* files may be hexadecimal, octal, binary, decimal integer, or decimal fixed-point numbers. Also, data can be entered in μ-255 companding format (chord and mantissa) and as linear data with a special prefix indicating conversion to μ-255 upon loading. The latter is convenient when a linear input data file exists and the μ-255 processing performance of the DSP program is to be evaluated.

2.3 File formats

Files for the DSP are arrays in memory. They are classified into file types in accordance with the word length of the data they accommodate. The file types, characterized by their data structure and simulator application, are as follows:

- 10-bit address data (TRC file)
- 16-bit data (PGM file)
- 20-bit data (RAM file)
- mixed data word length (input and output stacks)

The DSP chip transfers data from and to the outside world via serial channels. The I/O control register determines the number of bits to be transferred in a particular operation. Data words are stored in the most significant bits of the 20-bit IBUF. When a 16-bit word, for example, is transmitted to IBUF, an inherent scaling by the factor 16 takes place. Since the simulator cannot tell from a data word, per se, what its intended bit-length is, files of mixed data lengths have a length identifier associated with each word, specifying 8, 16, or 20 bits. When data are read from IS into IBUF, the simulator first checks for agreement between the word length identifier and the input number field of the I/O control register; if no discrepancy is detected, the data transfer takes place with the proper bit alignment, otherwise an error message will be given. The data words in the DSP OBUF are right-adjusted so no bit-shifting is required on transfer to OS. The word length information from the output number field of the I/O control register is, however, carried over to the OS. The identical formats of IS and OS allow output data to be used as input in a subsequent run of a DSP program.

The *UNIX* files are in ASCII format. They contain a FILETYPE declaration which must match the file type of the DSP file into which it is loaded. Appropriate word length symbols are appended to data in *UNIX* files containing mixed data word lengths.

A data line (where data is linear) may have the format

data[* scale factor][+ offset].

This can be used in editing an existing file into a new one with scaled and offset data values. The appropriate arithmetic is performed when the file is loaded.

UNIX files may contain more input or output data than the corresponding simulator IS or OS can accommodate. The input file will be automatically loaded into the IS when the stack is exhausted; this will continue until all *UNIX* file data have been used. Repeated "writes" of the OS to a *UNIX* file, within the execution of a DSP program, will append data to that file until execution of the particular DSP program is terminated.

III. CONDITIONAL OPERATIONS

The DSP has four control/status lines which correspond to the following four bits of the synchronization control register:

IBF	Input Buffer Full
OBE	Output Buffer Empty
c0	External Control Signal
c1	External Control Signal

These control lines are hardware driven and, therefore, have no predictable logical state during the execution of a DSP program. If an auxiliary instruction is conditional, control bits must be available at the time the instruction is executed. The simulator handles the control bit setting through its communications links with the external operating environment, namely the terminal or the *UNIX* file system, in the following ways:

(*i*) Default. A request is printed on the terminal for the value of the control bit IBF, OBE, c0, or c1. Execution resumes when the control bit value is entered.

(*ii*) Optional. The control bits can be read from a *UNIX* file specified as an argument to the GO command. This is used when the execution of the DSP program requires input of numerous control bits.

IV. COMMANDS

The command repertoire includes *UNIX* type commands for file handling and editing. It also includes commands for re-initialization of the simulator, setting and reading DSP registers, and transfer of files between the simulator and *UNIX* environments.

The WHEN operation is used to perform checks on breakpoint variables during execution of the DSP program and invoke simulator commands when breakpoint conditions are met. A breakpoint can be set on any DSP register value, on accumulator overflow, and on the number of DSP cycles executed; it can also be set to occur after a specified number of input or output operations have taken place and can be implemented for the *n*th time the program counter, or input or output data, match their corresponding breakpoint parameters. This permits the execution of complex test scenarios.

There are three simulator commands associated with the WHEN operation. SC sets the breakpoint parameters and DC lists the table of current parameter values. The WHEN command itself sets the test conditions and simulator commands to be carried out during execution of the program. It has the format:

```
WHEN[(expression){commands}]
```

The expression has the structure

```
cond op cond op cond ···
```

where cond is any test variable name. This implies that the variable in the logical expression becomes "true" when the breakpoint variable matches the check value. The logical operator NOT, OR, or AND is designated by op. As an example, the following simulator command lines

```
sc pc = 10
sc a = 1234.5
when (pc|a){dmp pc; dmp y}
```

will result in the printing of the DSP program counter value and Y-register, Y, when the program counter, PC, equals 10 or the accumulator, A, equals 1234.5.

The ED command invokes the *UNIX* text editor which operates on ASCII files. Thus, the *UNIX* files can be edited directly, whereas the simulator and DSP files are translated into ASCII files during the editing process and back into numerical format on completion of the editing. The translations are done automatically and are not visible to the user. The editor is useful, for example, in creating or altering input data files or filter coefficient files for the simulator's RAM.

The DMP command is used to print the contents of all or, through appended arguments, selected DSP registers. The plot command, PLT, produces x-y plots of data files. It facilitates automatic or specified scaling and shifting of data origin. It can be used in comparing segments of input data with the corresponding processed data.

The GO command initiates execution and, through a number of arguments, controls various I/O and diagnostics options. While the GO command provides for continuous execution of a DSP program, the STEP command executes the number of DSP cycles specified in its associated argument.

V. TERMINAL SESSION

The usage of the simulator is illustrated by an application of the DSP as a tone generator. The terminal session is recorded in Fig. 2. The simulator is invoked from the *UNIX* shell level by the DSPSIM command. The simulator command level is indicated by a ":" prompt character. First, the simulator's program memory is loaded, using the LD command, with the hexadecimal object file tone440, which was generated previously by the DSP assembler from a source program. Next, a breakpoint is set on an accumulated number of outputs equal to 70. The WHEN command is used to specify the actions to be taken when the breakpoint is reached. The actions are:

1. Write the OS into the *UNIX* file tone440.out.
2. Plot the data in tone440.out on the terminal.
3. Stop execution.

Finally, the execution is initialized with the GO command (the −m flag suppresses certain diagnostic messages). Although this terminal

```
$ dspsim
VERSION 2.7 (Mar 1, 1980)
: ld pgm tone440
: sc nout=70
: when (nout){wr os tone440.out;plt tone440.out;stop}
: go -m

*10**3
 140+
 130+
 120+
 110+
 100+
  90+
  80+
  70+
  60+   **                **                ***               ***
  50+  *  *              *  *              *   *             *   *
  40+ *    *            *    *
  30+                          *           *     *           *     *
  20+*      *          *
  10+                           *         *       *         *       *
   0+--------*--------*----------------------------------------------------
 -10+                           *       *         *       *         *
 -20+         *      *
 -30+                            *     *           *     *           *
 -40+          *    *                 *                               *
 -50+           *  *              *                 *   *                 *
 -60+            **                ***               ***               ***
    -+---------+---------+---------+---------+---------+---------+---------+
     0        10        20        30        40        50        60        70
```

Fig. 2—Terminal session on DSP simulator.

session is not an exhaustive demonstration of the simulator features, it should give a general flavor of the simulator operation.

VI. ACKNOWLEDGMENT

Grateful thanks go to Stephen M. Walters, who did some preliminary work in translating the DSP functions into simulator software; both he and James R. Boddie offered helpful suggestions on the operation of the simulator. Also, special thanks go to Robert L. Farah who, through diligent use of the simulator, discovered a number of abnormalities which were subsequently diagnosed and corrected.

REFERENCES

1. J. R.. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. C. L. Semmelman, "Digital Signal Processor: Design of the Assembler," B.S.T.J., this issue.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Design of the Assembler

By C. L. SEMMELMAN

(Manuscript received June 13, 1980)

In addition to the features normally provided by assemblers, the digital signal processor assembler handles the multistatement-per-instruction format required by a pipelined machine and provides several other capabilities required by the digital signal processor architecture. In describing the manner in which this was accomplished, more attention is devoted to matters of interest to the user of the assembler and less to its internal construction. The use of "lex" to write the parser subroutine is described, and possible future enhancements are discussed. An example illustrates the digital signal processor assembler's input language and its outputs.

I. INTRODUCTION

The assembler for the digital signal processor (DSP), as for any processor, converts programs written in a symbolic language into the corresponding machine language, and provides various convenience features for use by the DSP programmer.

The architecture of the DSP is designed for a maximum speed of operation in applications which differ markedly from those of ordinary computers. As a result, the DSP assembler contains features which are unique to this DSP application. At the assembly language level, this results in a complex programming language, which the programmer must understand thoroughly in order to produce correct and efficient programs. The assembler, of course, must accept every legal instruction in this input language and produce machine language corresponding to every operation in the DSP repertoire. This language differs from standard assembly languages in several major respects, as described below.

II. ARCHITECTURAL FEATURES OF THE DSP AND THEIR EFFECT ON THE ASSEMBLER DESIGN

The DSP has a number of unusual architectural features which affect the design of the assembler. Following an overview, these features are described from a user's point of view and their effects on the assembler's design and operation are discussed. Boddie et al.[1] give a more complete description of the machine architecture.

2.1 Overview of DSP architecture

The DSP contains 1024 words of 16-bit ROM memory for program storage and 128 words of 20-bit RAM memory for data. The processor contains a 16- by 20-bit integer multiplier, whose output is fed into a 40-bit integer accumulator. From there, data may be sent to a 20-bit W register, before being sent to storage or back to the multiplier for further calculation. Input and output are handled through 8-bit buffers, with automatic serial-parallel conversion and external synchronization. The DSP has four kinds of special purpose registers, including five registers for indirect addressing of data (direct addressing is not allowed), four registers for increments and counting, two for setting the DSP ground rules, and others for instruction counting, return address storage, synchronization, and status output. A separate adder is used to increment the addresses stored in the indirect addressing registers.

The multiplier, accumulator, W register, and data storage functions are separately programmed and controlled by different fields in the machine language word. There are two different classes of instructions: arithmetic and auxiliary, and they make different uses of some of the bits of the machine language word. The DSP uses the value of one of the instruction fields to distinguish between the two classes. Arithmetical instructions occupy only one machine word and may specify that the next machine word contains a numerical value for immediate input to the multiplier. Auxiliary instructions occupy two machine words, and bits in the second word further distinguish between arithmetical-auxiliary and non-arithmetical-auxiliary subclasses.

2.2 Pipeline architecture

The term "pipeline" refers to the fact that the processor has several hardware components which perform different operations simultaneously and pass data from one component to the next as through a pipe. In the DSP, these components are a multiplier, an accumulator, the W register and a memory. Data flow from the multiplier to the accumulator, then to the W register and to storage. Each of the hardware components is under the command of a specific group of bits in the instruction, which selects the exact operation to be performed out of the group available to the particular component.

Figure 1 shows how the pipeline functions. The columns correspond to the hardware components, and the rows to instructions which are executed sequentially in time. Data move diagonally. Each component can accept and process data in each instruction cycle, if commanded to do so. Although it takes four cycles for data to progress through all four components, the DSP accepts a new input argument and produces a new output each instruction cycle. This increases the data processing speed appreciably.

For each instruction, the assembler must accept up to four statements, one for each hardware component, and combine the corresponding bit patterns to form the complete instruction. The statements are usually written in the left-to-right sequence shown in Fig. 1, as this encourages the following interpretation:

(*i*) Store the present contents of the W (or Y) register.

(*ii*) Reload the W register from the accumulator.

(*iii*) Reload the accumulator using the present contents of the product register.

(*iv*) Calculate a new product from the x and y arguments specified.

An instruction containing four such statements might be

```
*rda = w   w = a   a = p + a   p = *rx++j * ibufy.
```

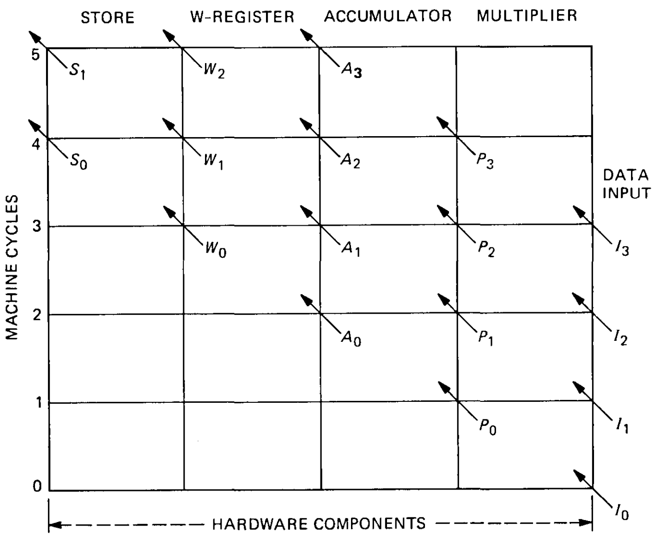


Fig. 1—Pipelining of data.

2.3 *Advanced fetches of x and y fields*

The x and y fields in a DSP instruction specify the two input arguments to the multiplier. The sources for the x and y data must be specified in the instruction that is fetched two cycles before the instruction that operates on the data. The assembler accepts the x and y source specifications on the same line as the operation (as shown in the previous example) and then "skews" them to the x and y fields of the instruction two instructions previous. The four-statement instruction shown above would appear in machine language as in Fig. 2.

2.4 *Instruction fetched two cycles before execution*

Each instruction is fetched from read only memory (ROM) two cycles before it is executed. This allows time to decode the remaining instruction fields before execution begins. If this only resulted in a two-cycle time delay between fetching an instruction and executing it, programmers would be able to ignore the delay completely. Unfortunately, however, this delay in execution also applies to jump instructions. The two instructions that follow a jump are already in the operating hardware when the jump takes effect and their x and y fields will affect instructions that follow the jump. They may differ from the x and y fields that would be fetched if the jump destination were reached by normal program counter incrementing. Fields $x1$ and $y1$ and fields $x3$ and $y3$ in Fig. 3 both refer to the same operation instruction located at the destination.

The current version of the assembler cannot determine if these two sets of x and y fields should be alike or if they may be different. The programmer must answer this question. The assembler tests and reports a difference as a warning message.

Both the advanced fetching of the x and y fields described above and this instruction fetching in advance of execution are forms of pipelining, and they create problems quite unlike those encountered in the writing of standard assemblers.

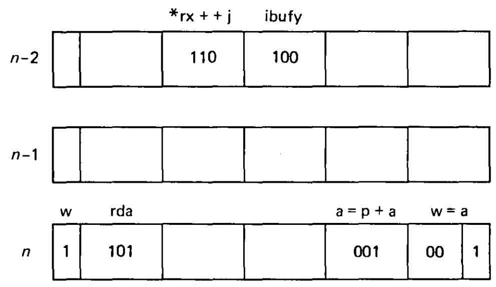


Fig. 2—Fields x and y advanced by two instructions.

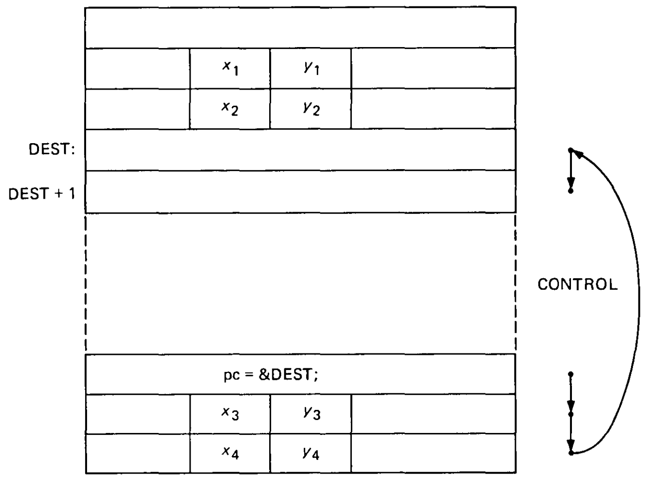


Fig. 3—Multiple x and y fields apply to same instructions.

2.5 Other hardware features

The other hardware features mentioned above cause few problems for the assembler and even eliminate some of the usual assembler functions. Input and output are treated as if they were two storage locations. The special purpose registers appear in the source program only to be loaded with a constant. When they are used for other purposes a key word, which also specifies the application, conceals their identifiers. Even statements specifying arithmetical operations do not have to be analyzed as the assembler recognizes the entire statement and translates it into a single bit pattern. For example, the entire statement

$$a = p + a$$

is identified as one key "word" and translated into the binary number 001. A multiplication statement, such as

p = *rx++j * ibufy,

contains two key words, *rx++j and ibufy, which become 110 and 100. Multiplication is implied and there is no place for the product to go except into the product register, so it need not be specified.

2.6 Summary

The unusual architecture of the DSP produces some strange results both in the assembler operation and in the appearance of source programs. The frequent use of key words and the prohibition of direct addressing combine to make variable names almost disappear from the source program. Their only use is to prime registers for indirect addressing. Pipelining also helps to conceal the programmer's intent. As a result, source programs are more difficult to read and good comments are more important than in most assembly language programs.

III. EXTERNAL FEATURES OF THE DSP ASSEMBLER

In the development of the assembler, a number of policy questions were considered and settled before program writing was started. These decisions are discussed below.

3.1 Environment for development and customer use

Because of the widespread use of the *UNIX** time-sharing operating system[2] at Bell Laboratories, no alternative was given serious consideration. This choice makes it easy for programmers to prepare their DSP source language programs using the *UNIX* text editor, store them as files, and have the assembler pick them up for processing. Assembler output files can be listed and retained for testing with the DSP simulator,[3] and can be converted to PROM or ROM mask formats.

3.2 Special DSP hardware features

The programmer is required to place the individual statements which make up an instruction, so that the pipeline operations will be performed on the correct data. The programmer must also place jump statements properly, as the assembler does not advance them by two instructions. This is considered proper as the programmer must understand the effects of the advanced x and y field fetches.

The assembler does advance the x and y fields for the programmer. This results in a more readable source program and eliminates one source of possible error in the assembler input. The assembler also deduces, from the key words found, which of the various instruction classes and sub-classes the programmer is using.

3.3 Input language characteristics

The input language syntax resembles that of the language C. This choice was made because many DSP programmers were familiar with that language.

To facilitate programming for people who do not need the full mnemonic content of the C-like constructs, a set of three-character alternates is available. Thus, a programmer has the choice of entering either

```
*rx++i        or        rxi
```

* Registered trademark of Bell Laboratories.

to mean "the quantity pointed to by the contents of the RX register, which is post-incremented by the contents of register I."

The assembler also provides a limited macro facility. It permits a single instruction macro to be defined and called within the source language input file. Argument substitution is not implemented. For more elaborate facilities, the programmer may use the C-language preprocessor, which allows multi-instruction capability, nesting of macros, and argument substitution.

3.4 Handling of errors

The assembler reports each error in a message which appears on the user's terminal. Each error is classified as "fatal" or "for information," and, where possible, each message contains the number of the line in the input file where the error was detected. Any fatal error will prevent the writing of the PROM programming file, and no error or combination of errors is capable of stopping the assembler before it reaches the end of the source file.

IV. IMPLEMENTATION POLICIES AND DETAILS

The programming methods described here were selected because they were convenient in developing the assembler and led to good quality code.

4.1 One-pass assembler

Because so few variable names are expected in the DSP programs, it seemed reasonable to assemble as much of the program as possible on the first pass over the source file. This decision forces the programmer to define macros before calling them and to assign variables to RAM storage before referring to them, which appear to be reasonable restrictions.

The assembler handles labels in the following manner. When a label definition is encountered, the assembler puts the memory address and the current ROM address in a label definition table. When a label reference is encountered, the memory address and the ROM location of the reference are added to a table of label references. At the end of the source program file, the assembler moves the label definition ROM locations into the corresponding label reference locations. Any undefined references cause fatal errors.

4.2 Organization of the DSP assembler

A very brief description of the assembler is as follows: the main program calls the parser, which is a subroutine written by "lex."[4,5] It returns a value identifying the token found. Control passes through two levels of "switch" statements to a block of code where the bits corresponding to the token are moved into the machine word, and some flags are set or some table entries made. These actions are repeated until the end of file is encountered. Then label references are resolved, output files written, and messages written for the user.

The following paragraphs elaborate on this brief description, but still give only an overview of the methods used. Tables I through VIII in the Appendix show the statements which may appear in each class of instruction and the tokens which are permitted in each statement. Reference to these tables may be helpful in reading the following paragraphs.

4.2.1 Token identifiers

The numerical values used to identify the tokens are octal numbers whose "hundreds" digit identifies a family of tokens and whose "tens" and "units" digits indicate the member of that family. In addition to the families of tokens shown in Appendix A, there is a utility family, whose members are semicolons, label definitions, macro starts, ends and calls, numerical values, comments, RAM assignments, dimensions, and subscripts. Some of these items are described more fully in the next section.

4.2.2 Utility functions

Several members of the utility family are described below. Label definitions have already been discussed, and comments, dimensions, and subscripts do not need extensive coverage.

4.2.2.1 Semicolon. The semicolon is used to mark the end of each DSP instruction. Its appearance initiates the clean-up after one instruction and the priming required for the next.

4.2.2.2 Macros: start, end, and call. A macro is defined as in the following example:

{Macname rdx = y   a = p   p = axi * ryk},

where the braces signal the start and end of the definition. At the left brace, the parser is called to read the macro's name, which is saved in a table. The statements in the macro are assembled in a macro table location, rather than in their final ROM position. The right brace causes the assembly location to revert to the normal ROM position.

Mentioning the name of the macro causes the saved bits to be placed in the proper position. The DSP programmer can add additional statements to the macro before the semicolon completes action on that instruction.

4.2.2.3 RAM variables. A RAM assignment statement appears as follows:

ram Z, ABC, TABLE[10];

The key word "RAM" causes control to go to a block of code where each token is read. The assembler adds each variable name, its dimension which defaults to 1, and the RAM location assigned to it, to a table of RAM variables. At the semicolon, control returns to the normal loop.

4.2.2.4 Numbers. Numbers may be used for several different purposes: as an immediate value to act as the x input to a multiplication, to build a table in ROM memory, to load a register, as a dimension in a RAM variable definition or as a subscript in a RAM variable reference. Thus, the processing of a number depends on the context of its use.

4.2.3 Flags and error detection methods

Fifteen different flags are used in the assembler. Among their uses are recording the presence of a member from each family of tokens, the arithmetical or auxiliary class of an instruction, and whether the instruction occupies two ROM words or one. The flags are used in tests for correctness of the source code and in steering the assembler.

The tests for errors are quite thorough. They may include some tests for errors which can never occur, as it was easier to include the test than to prove the impossibility of the error. The parser rules include one which detects any character which is not a letter, a number or one of the specified group of punctuation symbols. When such a character appears, a message to the user is sent and a fatal error recorded.

4.2.4 The file 'dsp.listing'

This file is the output medium by which the assembler communicates its results to the DSP programmer. A sample of the output is shown in Fig. 6. The tables referring to macros and labels are prepared initially as separate disk files and later concatenated with the file containing the remainder. This file is prepared by rewinding and rereading the input source program, matching each line to the assembled code.

V. RESULTS

An assembler for DSP programs has been written and functions as described in the preceding paragraphs. The assembler contains about

```
# include "dspbq.h"
"Two Biquad Sections in Cascade"
"Dial Tone Rejection Filter"
ioc = 0502;        /* 8 bits in, 16 out */
auc = 067;
i = 1;
j = -1;
k = -3;
loop: syc = 1;
bqic
bquad (1.,-1.957,1., -1.112,.544)
bquad (1.,-1.891,1.,-1.7328,.94297)
bqnoc (loop)
```

Fig. 4—Input file for biquadratic filter using macro cells and C-preprocessor.

```
Completed scan of source file. 0 fatal errors.
Files b.out and d.out were written to disk
dsp.listing was written to disk.
To save files, mv them to other names.
```

Fig. 5—Digital signal processor assembler messages to user in a successful assembly.

3500 lines of code and comments and has been in use for over a year in a wide variety of applications. Its output has been used as input to the simulator program,[5] thus, assuring compatibility.

VI. FUTURE ENHANCEMENTS

Enhancements to the DSP assembler fall into near-future and more remote-time categories.

```
D.S.P. ASSEMBLER, [data of version]
    Two Biquad Sections in Cascade
    [data and time of assembly]

    MACRO DEFINITIONS
    MACRO NAME   ADVCMD   CMD    DATA   W/A   SOURCE
                  octal   octal  octal

    LABELS
    ROM LOC.   LABEL     REF. AT LOC.
      dec.                dec.
       10      loop        44

    RAM VARIABLES
    RAM LOC.   DIMENSION    NAME
      dec.       dec.

LOC.  COMMAND     DATA      X    Y   LINE                    SOURCE
dec.   octal      octal              dec.

                                          "Two Biquad Sections in Cascade"
                                          "Dial Tone Rejection Filter"
  0 : 00 00 10 : 15 0502 (aux w   )   3 AN  ioc = 0502;
  2 : 00 00 10 : 14 0067 (aux w   )   4 AN  auc = 067;
  4 : 00 00 10 : 10 0001 (aux w   )   5 AN  i = 1;
  6 : 00 00 10 : 11 7777 (aux w   )   6 AN  j = -1;
  8 : 00 00 10 : 12 7775 (aux w   )   7 AN  k = -3;
 10 : 00 00 10 : 16 0001 (aux w   )   8 AN  loop:   syc = 1;
 12 : 00 04 10 : 02 0000 (aux iny )   9 AN            ry = 0;
 14 : 00 00 10 : 04 0000 (aux w   )  10 AN            rd = 0;
 16 : 00 21 01 :  000020 (imm ryi)   11 AA                                     p = mtl1(iny);
 18 : 00 22 01 :  000031 (imm ryj)   12 AA                          a = p      p = mtl2( );
                                          obuf = w                  a = p + a
 20 : 14 21 10 :  156457 (imm ryi)   14 N              p = -.544* ryi;
 22 : 00 21 10 :  043453 (imm ryi)   15 N                           a = p + a  p = --1.112*ryj;
 24 : 00 20 10 :  040000 (imm w  )   16 N                           a = p + a  p = 1.*ryi;
 26 : 01 21 01 :  101301 (imm ryi)   17 N   rdi = y    w = a        a = p      p = -1.957*ryi;
 28 : 11 22 10 :  040000 (imm ryj)   18 N   rdi = w                 a = p + a  p = 1.*w;
                                                                    a = p + a
 30 : 00 21 10 :  141646 (imm ryi)   20 N              p = -.94297* ryi;
 32 : 00 21 10 :  067346 (imm ryi)   21 N                           a = p + a  p = --1.7328*ryj;
 34 : 00 20 10 :  040000 (imm w  )   22 N                           a = p + a  p = 1.*ryi;
 36 : 01 00 01 :  103372 (aux w   )  23 N   rdi = y    w = a        a = p      p = -1.891*ryi;
 38 : 11 00 10 :  040000 (aux w   ) ·24 N   rdi = w                 a = p + a  p = 1.*w;
                                                                    a = p + a
 40 : 00 00 11 :  000001 (aux w   )  26 AA             ;
 42 : 00 00 01 :  000100 (aux w   )  27 AA             w = a;
 44 : 00 00 10 : 00 0012 (aux w   )  28 AN             pc = &loop;
 46 : 00 00 00 :  000000 (aux w   )  29 AN  ; ;
 48 : 00 00 00 :  000000 (aux w   )  29 AN
 50 : 00 00 00 :  000000 (aux w   )  30 AN  ; ;
 52 : 00 00 00 :  000000 (aux w   )  30 AN
```

Fig. 6—Digital signal processor assembler "dsp.listing" file.

```
base = 16
size = 8
*Two Biquad Sections in Cascade
0000: 00 08 d1 42
0002: 00 08 c0 37
0004: 00 08 80 01
0006: 00 08 9f ff
0008: 00 08 af fd
000a: 00 08 e0 01
000c: 01 08 20 00
000e: 00 08 40 00
0010: 04 41 00 10
0012: 04 81 00 19
0014: c4 48 dd 2f
0016: 04 48 47 2b
0018: 04 08 40 00
001a: 14 41 82 c1
001c: 94 88 40 00
001e: 04 48 c3 a6
0020: 04 48 6e e6
0022: 04 08 40 00
0024: 10 01 86 fa
0026: 90 08 40 00
0028: 00 09 00 01
002a: 00 01 00 40
002c: 00 08 00 0a
002e: 00 00 00 00
0030: 00 00 00 00
0032: 00 00 00 00
0034: 00 00 00 00
```

Fig. 7—File "b.out" written by DSP assembler.

Among the early improvements are additional macro-libraries and better syntax checking. The philosophy now in use is that of checking for specific errors. Because programmers are so ingenious at devising novel mistakes, it appears that the strategy should be reversed. It would be better to accept only code which conforms exactly to established forms, rejecting everything else.

More difficult, and correspondingly more valuable, are features that would simplify preparation of DSP programs for the user. This immediately suggests a compiler. However, the pipeline features of the DSP hardware will require the solution of design problems more complex than those for a standard compiler.

Register arithmetic is another area in which assistance to programmers would be valuable. Much of the speed advantage of the DSP comes from the planned use of automatically incremented registers for indirect addressing. Unplanned or random addressing of memory would forfeit this advantage. A compiler, then, should optimize the register use and incrementing. It might also have to change the locations in which the data are stored.

VII. EXAMPLES OF DSP ASSEMBLER INPUT AND OUTPUT

The following example was taken from a biquadratic filter program and shows the use of the macro library and preprocessor. The operations are primarily numerical calculations.

Figure 4 shows the input file required to program a two section filter.

```
Two Biquad Sections in Cascade
filetype i
0x0008
0xd142
0x0008
0xc037
0x0008
0x8001
0x0008
0x9fff
0x0008
0xaffd
0x0008
0xe001
0x0108
0x2000
0x0008
0x4000
0x0441
0x0010
0x0481
0x0019
0xc448
0xdd2f
0x0448
0x472b
0x0408
0x4000
0x1441
0x82c1
0x9488
0x4000
0x0448
0xc3a6
0x0448
0x6ee6
0x0408
0x4000
0x1001
0x86fa
0x9008
0x4000
0x0009
0x0001
0x0001
0x0040
0x0008
0x000a
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
```

Fig. 8—File "d.out" written by DSP assembler.

The use of four macro calls reduces the amount of typing required of the programmer, and the probable number of errors. Figure 5 shows the messages the programmer receives on the terminal at the conclusion of a successful assembly. Figure 6 is the file "dsp.listing."

The files "b.out" and "d.out" are shown in Figs. 7 and 8, respectively. The file "d.out" is the input file to the simulator, DSPMATE, and the ROM programming utilities. The file "b.out" is a more readable output file, giving both ROM locations and machine language in hexadecimal machine language.

VIII. CONCLUSIONS

An assembler for the DSP differs from conventional assemblers in many interesting respects. Many of the standard principles of assembler design either do not apply or do not provide benefit. On the other hand, many novel problems arose, for which standard techniques were of little assistance.

The current version of the assembler is considered to be a useful, reliable tool for programmers to use today. There are several areas in which greater assistance for DSP programmers can be provided and improvements in those areas are anticipated.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. T. H. Crowley, "UNIX Time-Sharing System: Preface," B.S.T.J., *57,* No. 6, Part 2 (July-August 1978), pp. 1897-2304.
3. J. Aagesen, "Digital Signal Processor: Software Simulator," B.S.T.J., this issue.
4. M. E. Lesk, "Lex—A Lexical Analyzer Generator," Comp. Sci. Tech. Rep. No. 39, Bell Laboratories (October 1975).
5. S. C. Johnson and M. E. Lesk, "UNIX Time-Sharing System: Language Development Tools," B.S.T.J., *57,* No. 6 (July-August 1978), pp. 2155-75.

Appendix

Table I—Normal instructions

NOTHING	NOTHING	a = p	p = XSRC*YSRC
DEST = y	w = a	a = p + a	p = XSRC*w
DEST = YSRC		a = p − a	p = XSRC*c
DEST = w		a = p + 2*a	p = XSRC*abs (YSRC)
		a = p + 8*a	p = XSRC*abs (w)
		a = p + a/2	p = XSRC*c*sgn (YSRC)
		a = p + a/8	p = XSRC*c*sgn (w)
		a = p & a	

Notes:
(1) If YSRC occurs in column 4, DEST = YSRC may not be used in column 1. Instead, use DEST = y.
(2) If w is used in column 4, DEST = YSRC may not be used in column 1.
(3) If the second instruction following this one is a normal instruction in which XSRC refers to RAM, NOTHING must be selected for column 1.

Table II—Auxiliary arithmetic instructions

NOTHING	NOTHING	NOTHING	NOTHING
DEST = y	w = a	a = p	p = YSRC
DEST = YSRC	w = ltm1(w)	a = p + a	p = w
DEST = w	w = ltm2(w)	a = p − a	p = mtl1 (YSRC)
		a = p + 2*a	p = mtl2 ( )
		a = p + 8*a	
		a = p + a/2	
		a = p + a/8	
		a = p & a	
		a = a ≪ 14	
		a = a ≪ 18	

Note:
See Notes in Table I.

Table III—Nonarithmetic auxiliary instructions

NOTHING	NOTHING
DEST = y	REG = VALUE
DEST = YSRC	REG = &LABEL [*N*]
DEST = w	REG = &RAMVAR [*N*]
	REG = YSRC
	if (CONDITION) doset ( )
	if (CONDITION) doau ( )
	if (CONDITION) dowt ( )
	if (lc−−! = 0) doset ( )
	return

Notes:
(1) See Note 1 in Table I.
(2) An instruction containing only a semicolon is a no op.
(3) VALUE represents a number −− integer, real, octal or hex.
(4) &LABEL [*N*] represents the *N*th word in ROM memory after the address of the label. If $N = 0$, [*N*] may be omitted.
(5) &RAMVAR [*N*] represents the address of the $(N + 1)$th location in an array called RAMVAR, stored in RAM memory. If $N = 0$, [*N*] may be omitted.

Table IV—Conditions

ibf	IBUF full
obe	OBUF empty
c0	C0 = 1
c1	C1 = 1
a==0	a equal to zero
a < 0	a less than zero
a > 0	a greater than zero
1c! = 0	lc not equal to zero

Table V—Destinations (DEST)

Form 1	Form 2
obuf	out
*rda	rdz
*rda++	rdp
*rda−−	rdm
*rd++i	rdi
*rd++j	rdj
*rd++k	rdk

Table VI—Y sources (YSRC)

Form 1	Form 2
ibufy	iny
*rya	ryz
*rya++	ryp
*rya−−	rym
*ry++i	ryi
*ry++j	ryj
*ry++k	ryk

Table VII—X sources (XSRC)

Form 1	Form 2	
x	olx	previous value of x
VALUE	VALUE	immediate data
*rx++i	axi	RAM address
*rx++j	axj	RAM address
*rx++k	axk	RAM address
*rx	axz	RAM address
*rx++	axp	RAM address
*rx−−	axm	RAM address
ibufx	inx	
*(rom+rx++i)	rxi	ROM address
*(rom+rx++j)	rxj	ROM address
*(rom+rx++k)	rxk	ROM address
&LABEL [*N*]	&LABEL [*N*]	
&RAMVAR [*N*]	&RAMVAR [*N*]	

Note:
See Notes 3, 4, and 5 of Table III.

Table VIII—Registers (REG)

pc	program counter
rx	pointer for x data
ry	pointer for y data
rya	alternate pointer for y data
rd	pointer for write destination
rda	alternate pointer for write destination
i	auto-increment for memory pointer
j	auto-increment for memory pointer
k	auto-increment for memory pointer
lc	loop counter
auc	AU control
ioc	I/O control
syc	synchronization
str	status output

Note:
auc, ioc, syc, and str cannot be set by y sources.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

A Tutorial Introduction to Digital Filtering

By E. J. ANGELO, JR.

(Manuscript received July 11, 1980)

Very-large-scale integration (VLSI) of digital electronic circuits has changed the hardware aspects of digital filters in a major way so that the use of such filters as components in commercial systems has become both economically feasible and technically desirable. Thus, large numbers of system engineers and circuit designers are now finding a need to learn about the properties of such filters, how they are used, and how they are designed. This paper is a first step toward meeting that need.

I. INTRODUCTION

The possibility of doing filtering and other signal-processing operations by numerical means instead of by traditional analog means has been known and studied for 20 years or longer. However, until recently the hardware for the physical realization of digital filters has been bulky, power-hungry, and expensive, and for this reason the digital filter has not been suitable for use as a component in commercial systems. Thus, interest in digital filters has been limited to a relatively small number of specialists doing research in this and related fields, where size, power consumption, and cost are not primary considerations.

However, VLSI has changed this condition drastically. It has reduced the size, power consumption, and cost of digital filters to the point where their use as a system component is both economically feasible and technically desirable. As a result, large numbers of system engineers and circuit designers are finding a need to learn about digital filters. Therefore, there is a place for tutorial material addressed specifically to the needs of these people.

This paper is an attempt to meet this need and addresses system engineers and circuit designers having no previous experience with digital filters or sampled-data systems. However, they are assumed to have a good understanding of the Laplace transform and its use with signals, differential equations, and electric circuits. We hope to provide a good understanding of the fundamentals of digital filtering and a strong foundation for further study of the subject. To reach these objectives most effectively, an effort is made to avoid all unnecessary abstractions. Generality is sacrificed for the sake of simplicity.

II. ELEMENTS OF DIGITAL FILTERING

This section gives an introduction to digital filtering in terms of the elementary circuit shown in Fig. 1. This simple circuit can be used to illustrate how filtering is done in the digital domain, in contrast with the more usual case of filtering in the analog domain.

The circuit is described by the following single node equation for the single unknown voltage v_2:

$$\frac{v_2 - v_1}{R} + C\frac{dv_2}{dt} = 0. \tag{1}$$

Rearranging the terms in this equation yields

$$v_2 + RC\frac{dv_2}{dt} = v_1, \tag{2}$$

which is a first-order linear differential equation in the unknown voltage v_2.

In this example, v_1 and v_2 are understood to be information bearing signals. For example, v_1 may be the output voltage of a strain gauge, the output of an accelerometer, or the output of a telephone transmitter. The information—strain, acceleration, or speech—is represented by the amplitude of v_1. The voltage v_2 represents the information after it has been processed by the RC circuit (filter). The processed form of the information, v_2, may be more valuable than the original form, v_1, because, for example, high-frequency noise has been removed from the

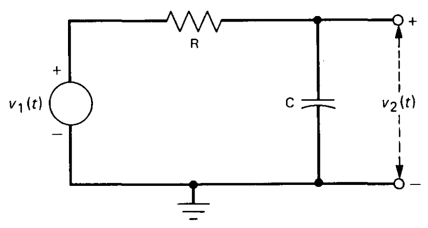


Fig. 1—An elementary filter.

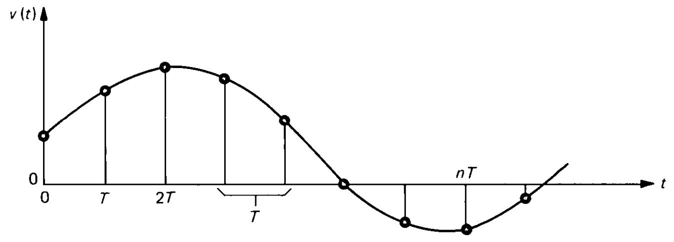


Fig. 2—Waveform of an analog signal and discrete samples of the signal.

signal. The amplitudes of the voltages v_1 and v_2 are the physical analogs of the original information—strain, acceleration, or speech—and the physical system represented by Fig. 1 is said to be an analog system.

Consider further the signal $v_2(t)$, for example. In the mathematical representation of eq. (2), voltage v_2 represents a continuum; that is, it can represent the values of any and all real numbers, and there are no real numbers that cannot be values of v_2. Thus, v_2 can change smoothly from any one value to another without any jumps or discontinuities. Similarly, time t in eq. (2) represents a continuum, and it can change smoothly from one value to another without any jumps or discontinuities. Moreover, when $v_2(t)$ represents a physical quantity, as it always does in the systems under study here, it is defined (has a numerical value) for every value of time t. These analog considerations are mentioned here because, in contrast, matters are quite different in digital filters and in digital systems.

Equation (2) can be solved easily by analytic means for the unknown voltage v_2 when the input voltage v_1 is a sinusoidal function of time, an exponential function of time, or a step function of time. It can also be solved analytically, but with more difficulty, when v_1 is a square wave and also, with still more difficulty, when v_1 is a more general periodic function of time.

When the input signal voltage in Fig. 1 is a more complicated function of time than in the few examples cited above, it is usually not practical, or even possible, to solve eq. (2) by analytic means. In such cases, however, it is possible to obtain an approximate solution by numerical methods. These numerical methods provide the basis for digital filtering. The numerical methods appropriate to this study are based on considering only discrete values of the signals, voltages v_1 and v_2 in Fig. 1, chosen at uniformly spaced instants of time. These discrete values of the signals are called samples of the signals. Figure 2 shows the waveform of a continuous analog signal voltage, and uniformly spaced samples of the signal are indicated on this diagram.

If waveforms for the analog signals v_1 and v_2 in eq. (2) exist, then sequences of samples similar to the one in Fig. 2 also exist for v_1 and v_2. Assuming the same sampling instants for the two signals, these sequences can be represented symbolically, starting at some instant designated $t = 0$, as

$$\begin{aligned} &v_1(nT): v_1(0), v_1(T), v_1(2T), \cdots, v_1(kT), \cdots, \\ &v_2(nT): v_2(0), v_2(T), v_2(2T), \cdots, v_2(kT), \cdots. \end{aligned} \tag{3}$$

Now, if the time interval between samples is sufficiently small, the derivative in eq. (2) can be approximated at time $t = nT$ by

$$\frac{dv_2}{dt} \approx \frac{v_2(nT) - v_2[(n-1)T]}{T}. \tag{4}$$

As T is made smaller, the approximation becomes better. Then, if T is made sufficiently small, eq. (2) can be approximated for one instant of time, $t = nT$, by substituting eq. (4) into eq. (2) to get

$$v_2(nT) + \frac{RC}{T} v_2(nT) - \frac{RC}{T} v_2[(n-1)T] = v_1(nT). \tag{5}$$

Equation (5) is a linear difference equation that approximates the linear differential equation given by eq. (2) above for one instant of time.

The difference eq. (5) offers a possibility that is not offered by the differential eq. (2); it can be solved explicitly for the response $v_2(nT)$ to obtain

$$v_2(nT) = \frac{1}{1 + RC/T} v_1(nT) + \frac{RC/T}{1 + RC/T} v_2[(n-1)T] \tag{6}$$

$$= av_1(nT) + bv_2[(n-1)T]. \tag{7}$$

This equation gives the present sample of the response voltage $v_2(nT)$, a single sample value, in terms of the present sample of the input voltage, $v_1(nT)$, and the immediately preceding sample of the response voltage, $v_2[(n-1)T]$. If $v_1(nT)$ is a known sample of the input, and if $v_2[(n-1)T]$ is known from a previous calculation of $v_2(nT)$, then the present value of $v_2(nT)$ can be calculated by simple arithmetic from eq. (7), assuming, of course, that the coefficients a and b are known. This calculation can be repeated for successive values of $v_1(nT)$ and $v_2[(n-1)T]$ to obtain the sequence of output samples for $v_2(nT)$.

In the calculation described above, the present response sample $v_2(nT)$ depends on the present input sample $v_1(nT)$ and on the immediately preceding response sample $v_2[(n-1)T]$. Thus, eq. (7) is a kind of recursion formula in which each calculation of $v_2(nT)$ provides the value of $v_2[(n-1)T]$ for the next calculation in the sequence. The start-up behavior for this system raises an additional question, but one of no serious consequence. If the start-up instant is designated $t = 0$, which for the discrete samples corresponds to $n = 0$, then $v_1(0)$ is assumed to be known, and in addition, the value of $v_2(-T)$ is needed for the last term in eq. (7) when $n = 0$. This last requirement is similar to the requirement that the initial conditions be known when a differential equation is to be solved. If the value of $v_2(-T)$ is known from physical considerations, there is no problem. If $v_2(-T)$ is not known, a reasonable value, such as zero, can be assumed, and the sequence of calculations can begin. In this case, the first few samples of $v_2(nT)$ that are calculated will depend on the assumed initial value of $v_2[(n-1)T]$. However, the effect of the assumed initial condition decays with time, and it soon disappears.

Example 1

The simple filter of Fig. 1 is represented approximately by the difference eq. (7),

$$v_2(nT) = av_1(nT) + bv_2[(n-1)T].$$

Suppose that the input to the filter is

$$\begin{aligned} v_1(nT) &= 1, \qquad n = 0, \\ &= 0, \qquad n \neq 0, \end{aligned}$$

and suppose that

$$v_2(-T) = 0,$$

then, the response $v_2(nT)$ can be calculated as follows:

$$\begin{aligned}
v_2(0) &= a + 0 = a, && \text{for } n = 0 \\
v_2(T) &= 0 + ba = ab, && \text{for } n = 1, \\
v_2(2T) &= 0 + b(ab) = ab^2, && \text{for } n = 2, \\
v_2(3T) &= 0 + b(ab^2) = ab^3, && \text{for } n = 3, \\
&\vdots \\
v_2(nT) &= 0 + bv_2[(n-1)T] = ab^n, && \text{for } n, \\
&\vdots
\end{aligned}$$

It follows from eqs. (6) and (7) that the coefficients a and b are nonnegative and less than unity. Thus, the response $v_2(nT)$ decays with time and tends to zero as n increases without limit. This problem is treated again in Section 4.2 from a different point of view.

In the approximate representation of the circuit in Fig. 1 by the difference eq. (7), the time continuum does not exist; it has been replaced by a sequence of discrete instants of time. The sequence of samples $v_2(nT)$, for example, is a discrete-time signal, in contrast to the corresponding $v_2(t)$, which is a continuous-time signal. The discrete-time signal is defined (that is, it has a specific numerical value) only at the discrete instants of time $t = nT$, whereas the continuous-time signal is defined for every instant of time. The samples of the discrete-time signal $v_2(nT)$, for example, have finite amplitudes and zero time duration.

As shown above, the value of the present response sample $v_2(nT)$ in the circuit shown in Fig. 1 can be calculated from eq. (7) by using simple arithmetic. Only multiplication and addition (of signed numbers) are required, and it is particularly useful in this study to think of this arithmetic as being performed on a pocket-sized electronic calculator. This is true because the calculator is a digital machine that has much in common with the digital filter. All of the terms on the right-hand side of eq. (7) are entered into the calculator in digital form (decimal digits) through the keyboard, and the result of the calculation, $v_2(nT)$, is presented in digital form (decimal digits again) on the output display of the calculator.

Any given electronic calculator has a fixed number of digits in its output display, and it follows from this fact that only a finite number of discrete values can be displayed on the output. Thus, if the calculator is used to evaluate $v_2(nT)$ from eq. (7), then v_2 can no longer represent a continuum of values; it can represent only a finite number of discrete values given by the digits displayed on the output of the calculator. The information in this case is not represented by the amplitude of a physical variable, but rather, it is represented by the digits displayed by the calculator. Therefore, by definition, the information is not in analog form, and because of its form, it is called digital information. The set of digits representing the information is called a digital signal. In the case of binary systems, the information is represented by the binary-digit (bit) patterns associated with binary numbers.

Since the digital signal produced by the calculator and by the digital filter can have only a finite number of "allowed" values, the signal is quantized. One result of quantization is the introduction of random errors called quantization noise. Another result is the existence of nonlinear feedback loops in many digital filters with the likelihood of self-sustaining oscillations called limit cycles. However, these are problems associated with the design and performance of the hardware used to realize digital filters, and it is not appropriate to discuss them here. Detailed treatments of these problems are given in Refs. 1 through 3. The remainder of this paper assumes that the number of digits available for representing signals is unlimited.

In effect, sampling the analog signal shown in Fig. 2 changes the continuous-time representation of the signal to a discrete-time representation. Converting the analog-sample amplitudes in Fig. 2 to equivalent digital values changes the continuous-amplitude representation to a discrete-amplitude representation.

To summarize developments up to this point, the circuit of Fig. 1 is chosen as a simple filter to be studied for the purpose of getting an introduction to the ideas of digital filtering. The circuit is analyzed by standard techniques to obtain an analog representation in terms of the differential equation (2). Then, we imagine that the signal voltages v_1 and v_2 are sampled to obtain the difference eq. (7) as an approximation to the differential equation. Next, we envision an electronic calculator for evaluating eq. (7) by numerical methods. Data is entered into the calculator in digital form, and the result of one calculation is the value of one sample of the filter response $v_2(nT)$ in digital form. This cycle of calculation is repeated successively with successive samples of the input voltage $v_1(nT)$ to obtain successive values of the output voltage $v_2(nT)$.

Consider the case in which the input to the filter in Fig. 1 is a voice signal. To represent this signal with good accuracy by a sequence of samples, the signal must be sampled about 10,000 times per second. (The sampling is examined in detail in Section 3.2.) This fact implies that for each one-second interval of speech, about 10,000 samples of the response $v_2(nT)$ must be calculated. Although the calculation of each output sample is simple enough, calculating 10,000 of them with a manually operated calculator takes quite a while.

The stage is now set for the introduction of the digital signal processor as a means for implementing digital filters. The digital signal processor is a digital device (binary digits) that has been especially designed to perform the arithmetic required in the repetitive evaluation of the difference equation described above. It does the arithmetic automatically at very high speed under program control. When programmed to solve eq. (7) it can accept a new input signal sample in digital form, calculate the corresponding response sample, and deliver the response to the output all in less than 5 μs. Thus, the signal processor can receive a new input sample and calculate the corresponding response in a tiny fraction of the interval between successive input samples, an interval of 100 μs at 10,000 samples per second. It follows from these facts that the processor, with its blazing speed, can operate in real time, solving eq. (7) almost instantly for the response to each input sample and then waiting for the next input sample to come along. Thus, by solving eq. (7) in real time, the processor produces in sampled digital form the same response to the input signal v_1 as the filter in Fig. 1, within the accuracy permitted by the approximations involved in deriving eq. (7) and in sampling v_1.

The filter shown in Fig. 1 is an analog computer that solves the differential eq. (2) in real time. Similarly, the digital signal processor is a digital computer that solves the difference eq. (7) in real time. To the extent that eq. (7) is a good approximation to eq. (2), the processor is a good approximation to the filter in Fig. 1.

The ideas developed above make the concepts of digital filtering and the use of the digital signal processor for its realization seem to be quite simple. While the basic ideas are indeed perfectly straightforward, the implementation of high-performance filters by these means in a realistic system environment presents a challenge.

First, the question of how well difference eq. (7) approximates differential eq. (2) has been raised above. Insofar as the filtering operation is concerned, this question requires a detailed answer and a more complete mathematical formulation of the problem than we have so far given. The remainder of the paper concerns this and related problems.

Second, the simplicity of the digital filter as presented above is genuine, but in a way it is deceptive. The starting point for the presentation above is chosen to bypass all of the challenging preliminary work that is needed to put the real engineering problem into a form that can be implemented by the digital signal processor.

The linear difference equation is the central element in the concept of the digital filter. In the example represented in Fig. 1, the difference equation is obtained by approximating the differential eq. (2), which, in turn, is obtained directly from the circuit assumed in Fig. 1. However, the design of real filters rarely has this kind of starting point. For reasons which stem partly from technological heritage and partly from mathematical tractability, filter design usually starts with a specification of the frequency characteristics that are desired of the filter: Pass-band characteristics and frequencies, stop-band characteristics and frequencies, delay characteristics, and the like. Then, in order to obtain a realization in digital form of a filter having these specified characteristics, it is necessary to determine, in some way, a linear difference equation describing a filter having the specified characteristics. The digital signal processor is then programmed to evaluate this difference equation by arithmetic operations.

The most usual way of designing a digital filter is to start with classical analog-filter theory. Given a realizable set of frequency characteristics, classical theory can be used to derive the analog transfer function for an analog filter having the specified characteristics. Corresponding to this transfer function there is always a differential equation such as the one in eq. (2), for example. In principle, it would be possible to proceed as in the example in Fig. 1 and use this differential equation to derive an approximately equivalent difference equation. However, an alternative procedure proves to be more fruitful.

Every differential equation relating an output signal and its derivatives to an input signal and its derivatives gives rise, through the Laplace transform, to an analog transfer function. Similarly, as is shown in Section IV, every difference equation relating the present and past values of an output signal to the present and past values of an input signal gives rise, through the z transform, to a digital transfer function. It is shown in the remainder of this paper that the digital transfer function is related to the digital filter and its frequency characteristics in much the same way as the analog transfer function is related to the analog filter. The z transform is a special form of the Laplace transform that is developed in some detail in Section 3.1. Furthermore, we show in Section VI that an analog transfer function can be transformed into a digital transfer function in such a way that the frequency characteristics of the two functions are related in a precisely known manner. Thus, in many cases classical techniques can be used to derive a prototype analog transfer function that can be transformed into a digital transfer function having the desired frequency characteristics. The difference equation corresponding to this digital transfer function can be derived easily, as shown in Section 4.4, and it can then be implemented with hardware to obtain a physical realization of the digital filter.

The method outlined above is the most common, but not the only, method used for designing digital filters. The remainder of this paper gives the details of the method. However, the treatment is necessarily introductory, and makes no attempt to provide any expertise in the field. (See Refs. 1 through 3 for a detailed treatment of the subject.)

III. THE z TRANSFORM FOR SAMPLED SIGNALS

Section II introduced digital filtering in terms of a simple example. The example reveals some of the approximations involved in digital filtering, and it points out the need for a more comprehensive mathematical formulation of the problem. The z transform is the mathematical tool that is extensively used for this purpose. The objective of this section is to present the z transform and to develop its properties to the extent required by this paper.

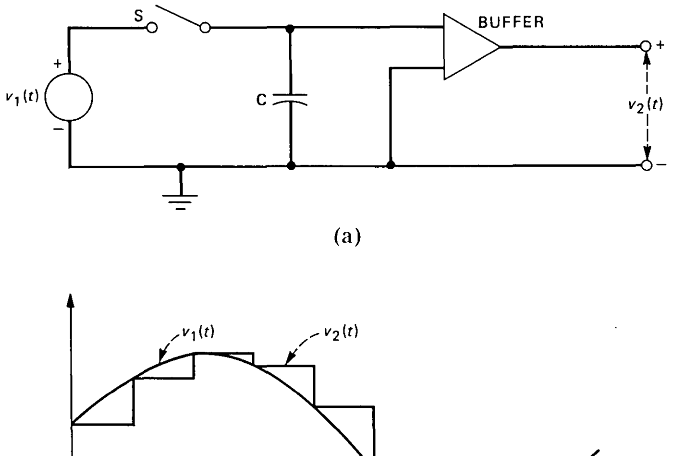
3.1 Definition and elementary properties of the z transform

In general, digital filtering and digital signal processing concern processing signals that are characterized by a sequence of values, normally instantaneous samples of a continuous-time signal, that are uniformly spaced in time. Such signals are discussed in Section II. For the purpose of this study, we consider only signals that are zero for time t less than some instant designated $t = 0$. It is also assumed that the signal to be sampled is continuous at every sampling instant, except possibly at $t = 0$, in which case the value of the signal at $t = 0+$ is taken by convention.

Figure 2 shows a continuous-time signal and a sequence of discrete samples of that signal. The first step in the method used here for developing the z transform is to note that signal samples of zero duration cannot exist in any physical system. Any sampling operation that is implemented in hardware is necessarily associated with a holding operation that produces signal samples of nonzero duration. The most widely used sample-and-hold circuit has the form shown in Fig. 3a, and when its input voltage is the continuous-time signal voltage shown in Fig. 2, the circuit produces a stair-step output-voltage waveform shown as v_2 in Fig. 3b. It is also important to note that digital filters are often followed by a digital-to-analog converter to provide an output signal in analog form. In many cases the converter produces a stair-step output waveform like the one shown in Fig. 3b. Thus, stair-step waveforms play a central role in the analysis of digital filters and of sampled-data systems in general.

A great deal of valuable information about the sampling process and the stair-step output waveform can be obtained from the Laplace transform of the stair-step signal. This transform is

$$V_2(s) = \int_0^\infty v_2(t)e^{-st}dt. \tag{8}$$



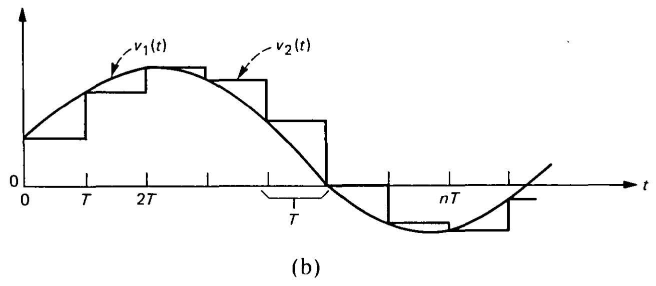


Fig. 3—Sample-and-hold circuit. (a) Circuit. (b) Waveforms.

(The convention followed in this paper is to use lower-case letters to represent instantaneous values of time varying quantities and to use capital letters for transforms and other quantities that are not functions of time.) At first glance, evaluating the integral in eq. (8) may seem to present a serious problem because of the stair-step form of $v_2(t)$. However, a little further thought shows how this problem can be reduced to the very simple problem of calculating the transform of a constant. It follows from the definition of the integral that the integral in eq. (8) can be expressed as the sum of infinitely many integrals, each spanning the time interval of one step in the waveform of $v_2(t)$. Thus, eq. (8) can be expressed as

$$V_2(s) = \int_0^T v_1(0)e^{-st}dt + \int_T^{2T} v_1(T)e^{-st}dt + \cdots + \int_{nT}^{(n+1)T} v_1(nT)e^{-st}dt + \cdots. \quad (9)$$

This equation uses the fact that during each step in the waveform, $v_2(t)$ is constant and equal to the value of $v_1(t)$ at the beginning of the step.

Equation (9) can be written more briefly as

$$V_2(s) = \sum_{n=0}^{\infty} \int_{nT}^{(n+1)T} v_1(nT)e^{-st}dt. \quad (10)$$

Evaluating this integral and inserting the limits yields

$$V_2(s) = \sum_{n=0}^{\infty} v_1(nT) \left(\frac{e^{-snT} - e^{-s(n+1)T}}{s}\right). \quad (11)$$

Factoring e^{-nsT} out of the parentheses in eq. (11) produces

$$V_2(s) = \sum_{n=0}^{\infty} v_1(nT)e^{-nsT} \left(\frac{1 - e^{-sT}}{s}\right). \quad (12)$$

Now the factor in the parentheses is independent of n; hence, it can be factored out of the summation to produce

$$V_2(s) = \frac{1 - e^{-sT}}{s} \sum_{n=0}^{\infty} v_1(nT)e^{-nsT}. \quad (13)$$

The summation in eq. (13) contains the values of all the samples of the input signal $v_1(t)$, and it also contains the instant of time $t = nT$ at which each sample occurs. Thus, it contains complete information about the sequence of samples $v_1(nT)$. It is, therefore, common practice to associate the summation with the process of sampling the input signal. The factor multiplying the summation depends on the fact that the output signal $v_2(t)$ is a stair-step wave, but it is totally independent of the input signal $v_1(t)$. This factor is the same in the Laplace transform of every stair-step wave, and it is commonly associated with the hold part of the sample-and-hold operation. Since this factor is the same for every stair-step wave, it is of minor importance in the design of digital filters, although its effect must always be accounted for at some point in the design.

As a result of the relations described above, it proves to be very useful to break eq. (13) into two parts,

$$\hat{V}_1(s) = \sum_{n=0}^{\infty} v_1(nT)e^{-nsT} \tag{14}$$

and

$$H_h(s) = \frac{1 - e^{-sT}}{s}, \tag{15}$$

so that (13) can be written as

$$V_2(s) = H_h(s)\hat{V}_1(s). \tag{16}$$

The circumflex ($\hat{}$) is used to distinguish the function in eq. (14) pertaining to the sample values of $v_1(t)$ from the Laplace transform $V_1(s)$ of the unsampled signal $v_1(t)$. The subscript h on the left side of eq. (15) signifies that H_h is associated with the hold part of the sample-and-hold operation.

The separation of eq. (13) into two parts given by eqs. (14) and (15) is a purely algebraic operation; it is not based on any consideration of any physical system. Therefore, readers should not feel frustrated if their attempts to ascribe a physical significance to these separate relations produce results that are less than completely satisfactory. However, when eqs. (14) and (15) are multiplied together, as in eq. (16), the result is always the Laplace transform of a stair-step wave in which the heights of the successive steps are equal to the values of the successive samples in eq. (14).

In eq. (14) the complex-frequency variable s appears only in the combination e^{-nsT}. Therefore, it proves to be quite convenient to define a new symbol,

$$z = e^{sT}, \tag{17}$$

so that eq. (14) can be written more simply as

$$V_1^*(z) = \sum_{n=0}^{\infty} v_1(nT)z^{-n} \tag{18}$$

$$= v_1(0) + v_1(T)z^{-1} + v_1(2T)z^{-2} + \cdots$$
$$+ v_1(nT)z^{-n} + \cdots. \tag{19}$$

This is the z transform of the sequence of sample values $v_1(nT)$. The transform, as given by eq. (18), is an infinite series in the variable z. Since eq. (18) is a power series, it can be shown by conventional methods that if the sequence $v_1(nT)$ is bounded, then the series converges for all values of z such that $|z| > 1$.

The relations developed above can be illustrated with the aid of Fig. 4. Figure 4a shows the beginning of a sequence of samples, $f(nT)$. The z transform of this sequence can be written by inspection with the aid of Fig. 4a; it is

$$F^*(z) = \sum_{n=0}^{\infty} f(nT)z^{-n}. \tag{20}$$

Furthermore, a stair-step wave, $f_a(t)$, can be constructed on the samples of Fig. 4a as shown in Fig. 4b. The height of each successive step is equal to the value of each successive sample in Fig. 4a. Now the Laplace transform of the stair-step wave can be written. Using eq. (17) to replace z in eq. (20) yields

$$\hat{F}(s) = \sum_{n=0}^{\infty} f(nT)e^{-nsT}. \tag{21}$$

Then, eqs. (15) and (16) lead to

$$\begin{aligned} F_a(s) &= H_h(s)\hat{F}(s) \\ &= \frac{1 - e^{-sT}}{s} \sum_{n=0}^{\infty} f(nT)e^{-nsT}. \end{aligned} \tag{22}$$

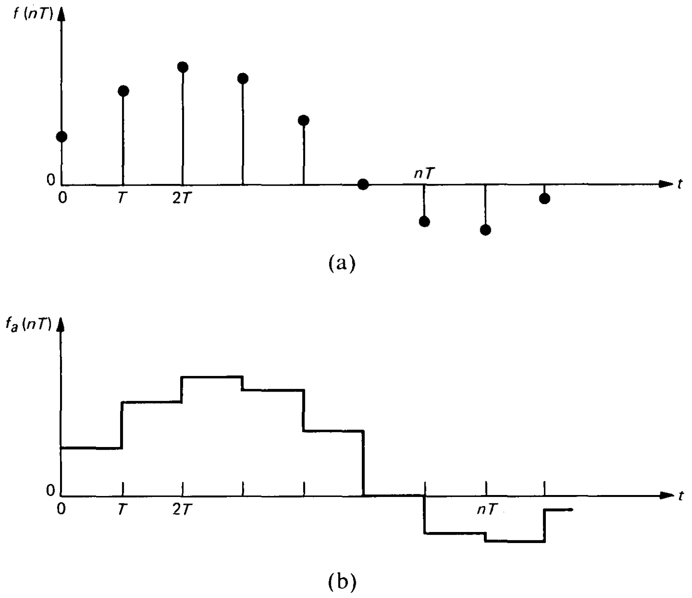


Fig. 4—Sampled data. (a) Sequence of samples. (b) Stair-step waveform.

This is the Laplace transform of the stair-step wave.

Example 2

Consider the short sequence of samples

$$f(nT): 2, 3, 1, 0, 0, 0, \cdots.$$

The z transform of the sequence is, by inspection

$$F^*(z) = \sum_{n=0}^{2} f(nT)z^{-n} = 2 + 3z^{-1} + z^{-2}. \tag{23}$$

Using eq. (17) to replace z in eq. (23) yields

$$\hat{F}(s) = 2 + 3e^{-sT} + e^{-2sT}. \tag{24}$$

The Laplace transform of the corresponding stair-step function is given by eqs. (15) and (16) as

$$\begin{aligned} F_a(s) = H_h(s)\hat{F}(s) &= \frac{1 - e^{-sT}}{s}(2 + 3e^{-sT} + e^{-2sT}) \\ &= \frac{1}{s}(2 + 3e^{-sT} + e^{-2sT} - 2e^{-sT} - 3e^{-2sT} - e^{-3sT}) \\ &= \frac{2}{s} + \frac{1}{s}e^{-sT} - \frac{2}{s}e^{-2sT} - \frac{1}{s}e^{-3sT}. \end{aligned} \tag{25}$$

If $u(t - nT)$ represents the unit step function delayed by nT units of time, then the inverse Laplace transform of eq. (25) can be written term-by-term as

$$f_a(t) = 2u(t) + u(t - T) - 2u(t - 2T) - u(t - 3T). \tag{26}$$

As a quick sketch of eq. (26) shows, it is the stair-step wave based on the given arbitrary sequence of samples.

Example 3

Suppose that the input voltage to the sample-and-hold circuit in Fig. 3a is a unit-step function so that samples $v_1(nT)$ are unity for all nonnegative n including $n = 0$. The z transform for this sequence is, from eq. (18):

$$V_1^*(z) = \sum_{n=0}^{\infty} z^{-n}. \tag{27}$$

This series can be summed to obtain

$$V_1^*(z) = \frac{1}{1 - z^{-1}}, \qquad |z| > 1. \tag{28}$$

The series in eq. (27) converges to this value for all values of z such that $|z| > 1$.

The important point in the above examples is that given an arbitrary sequence of uniformly spaced sample values, the z transform of the sequence can be written by inspection. Then, with no effort at all, the Laplace transform of the associated stair-step wave can be written, and from the Laplace transform the frequency spectrum of the wave can be calculated. The frequency spectrum is, of course, a central idea in the classical methods for the analysis and design of filters.

Similarly, given the z transform expressed as a power series in z^{-1}, the corresponding sequence of sample values can be written directly by inspection of the coefficients in the power series. In this way, the inverse of the z transform can be calculated. In the study that follows, there will be numerous occasions to write z transforms and their inverses by these simple procedures.

The z transform defined by eq. (18),

$$V_1^*(z) = \sum_{n=0}^{\infty} v_1(nT)z^{-n}, \tag{29}$$

is obtained from the Laplace transform in eq. (14),

$$\hat{V}_1(s) = \sum_{n=0}^{\infty} v_1(nT)e^{-nsT}, \tag{30}$$

by defining the symbol

$$z = e^{sT}. \tag{31}$$

The relations between z and s and between $V_1^*(z)$ and $\hat{V}_1(s)$ are important in the following study; therefore, we examine them here. For every point in the s plane, eq. (31) specifies just one point in the z plane. Conversely, however, every point in the z plane corresponds, through eq. (31), to infinitely many points in the s plane. This matter is examined in more detail later. The process by which a point in one plane is transferred to the other plane is called "mapping," and the law that governs the mapping process in the present case is eq. (31). At corresponding points in the two planes, eq. (31) is satisfied, and then eqs. (29) and (30) are equivalent with

$$V_1^*(z) = \hat{V}_1(s). \tag{32}$$

The important relationship between the z and s planes can be explored further by considering the special case in which the complex-frequency variable s takes on purely imaginary values, $s = j\omega$, and s is, thus, restricted to points in the s plane lying on the imaginary axis. The corresponding values of z are given by eq. (31) as

$$z = e^{sT} = e^{j\omega T}. \tag{33}$$

Under this condition,

$$|z| = |e^{j\omega T}| = 1, \tag{34}$$

and all values of z satisfying this relation correspond to points on the unit circle in the z plane, a circle centered at the origin and having unity radius. That is, every point on the $j\omega$ axis in the s plane corresponds to a point on the unit circle in the z plane, or, more simply, the $j\omega$ axis in the s plane maps onto the unit circle in the z plane.

It also follows from eq. (33) that

$$z = 1 \qquad \text{when} \quad \omega = 0. \tag{35}$$

Furthermore, as ω increases in the positive direction from zero, the angle of z, which is just ωT rad, increases positively, and as the point s moves up the $j\omega$ axis in the s plane, the point z moves continuously in a counterclockwise direction around the unit circle in the z plane. These relations are illustrated in Figs. 5a and b. (The three parts of Fig. 5 represent the same z plane; three parts are used to avoid putting too much information into one diagram.)

Then, from eq. (33), as ω increases,

$$z = -1 \qquad \text{when} \quad \omega T = \pi \text{ rad}. \tag{36}$$

If the sampling frequency is defined as

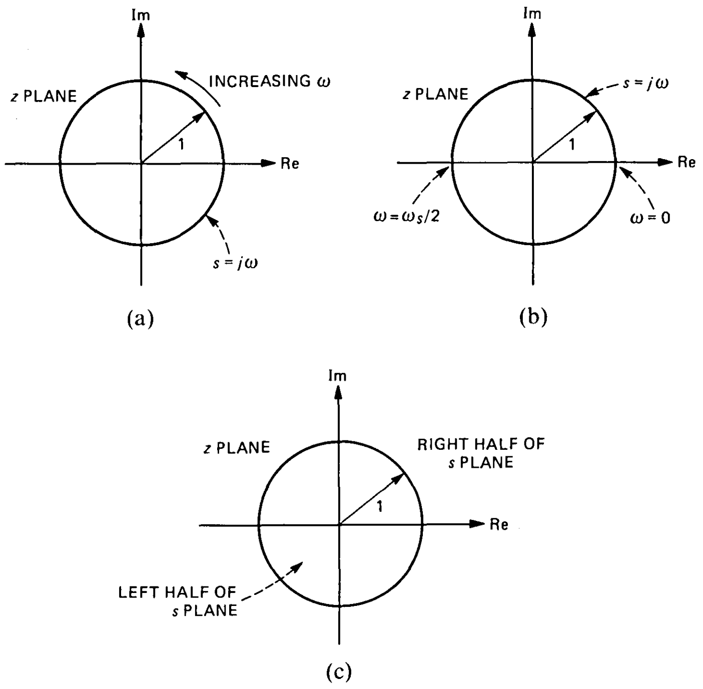


Fig. 5—Relations between the z plane and the s plane.

$$f_s = \frac{1}{T} = \frac{\omega_s}{2\pi}, \tag{37}$$

then eq. (36) can be written as

$$z = -1 \qquad \text{when} \quad \omega = \frac{\omega_s}{2} \tag{38}$$

or when

$$f = \frac{f_s}{2}. \tag{39}$$

This relation is shown in Fig. 5b. Thus, as ω increases positively from zero to $(\omega_s/2)$ rad/s, the point z moves counterclockwise around the upper semicircle in Fig. 5a.

In a similar way, as ω increases negatively from zero to $(-\omega_s/2)$ rad/s, the point z moves clockwise around the lower semicircle in Fig. 5a.

To pursue this matter further, let $s = j\omega$ move up the $j\omega$ axis without limit. It then follows from eq. (33) that the corresponding point in the z plane moves around the unit circle shown in Fig. 5 repeatedly, without limit. Thus, z passes through any given point on the unit circle an unlimited number of times as $j\omega$ increases without limit, and that single point on the unit circle in the z plane corresponds to infinitely many points uniformly spaced on the $j\omega$ axis in the s plane.

To explore this matter further, note that

$$\exp(j\omega T) = \exp[j(\omega T + 2\pi k)], \qquad k = 0, \pm 1, \pm 2, \cdots, \tag{40}$$

and using eq. (37),

$$\exp(j\omega T) = \exp[j(\omega + k\omega_s)T], \qquad k = 0, \pm 1, \pm 2, \cdots. \tag{41}$$

But this is the definition of a periodic function of ω with, in this case, a period equal to ω_s, the sampling frequency in radians per second. Moreover, with $s = j\omega$, eq. (30) becomes

$$\hat{V}_1(j\omega) = \sum_{n=0}^{\infty} v_1(nT)e^{-jn\omega T}, \tag{42}$$

and it follows from eq. (41) that $\hat{V}_1(j\omega)$ is also periodic with a period equal to ω_s. Thus, the frequency spectrum of the sampled wave is a periodic function of ω, a consequence of the sampling operation. We examine this further in the next section.

Returning to eq. (31) and substituting $s = \sigma + j\omega$ yields

$$z = e^{sT} = e^{\sigma T}e^{j\omega T}, \tag{43}$$

and

$$|z| = e^{\sigma T}. \tag{44}$$

Since T is always positive, it follows that

$$|z| < 1 \quad \text{when} \quad \sigma < 0. \tag{45}$$

Thus, the entire left half of the s plane is mapped by eq. (43) into the interior of the unit circle in the z plane as indicated in Fig. 5c. It is easy to show in a similar manner that the right half of the s plane is mapped into the exterior of the unit circle in the z plane. These relations are of basic importance when considering the stability (freedom from growing transients) of digital filters.

3.2 Frequency spectra of sampled signals

In Section 3.1 a continuous-time signal voltage $v_1(t)$, having a Laplace transform $V_1(s)$, is applied to the input of the sample-and-hold circuit shown in Fig. 3a. The output of this circuit is the stair-step signal voltage $v_2(t)$ shown in Fig. 3b. The Laplace transform of $v_2(t)$ is found in Section 3.1 to be

$$V_2(s) = H_h(s)\hat{V}_1(s), \tag{46}$$

where

$$H_h(s) = \frac{1 - e^{-sT}}{s}, \tag{47}$$

and

$$\hat{V}_1(s) = \sum_{n=0}^{\infty} v_1(nT)e^{-nsT}. \tag{48}$$

In Section 3.1, we show that with $s = j\omega$, $\hat{V}_1(j\omega)$ is a periodic function of ω with a period equal to ω_s, the sampling frequency in radians per second. However, the analysis given there provides no information about $\hat{V}_1(j\omega)$ beyond the fact that it is periodic. The objective here is to extend that analysis.

Since $\hat{V}_1(s)$ in eq. (48) depends on the sample values of $v_1(t)$, it seems reasonable that $\hat{V}_1(s)$ should be related in some way to $V_1(s)$, the Laplace transform of $V_1(t)$. This, in fact, is the case. It can be shown that

$$\hat{V}_1(s) = \frac{1}{T}\sum_{n=-\infty}^{\infty} V_1(s - jn\omega_s), \tag{49}$$

where, again, ω_s is the sampling frequency. The case of greatest interest is that in which $s = j\omega$, and

$$\hat{V}_1(j\omega) = \frac{1}{T}\sum_{n=-\infty}^{\infty} V_1[j(\omega - n\omega_s)]. \tag{50}$$

This result shows, again, that $\hat{V}_1(j\omega)$ is a periodic function of ω, and it shows further that the function consists of $V_1(j\omega)$, the transform of $v_1(t)$, repeated periodically along the $j\omega$ axis with a spacing equal to ω_s.

Equation (50) is a very important relation that has been proved in the literature many times by various methods. (See Refs. 1 through 4.) All of the methods have one thing in common—they are not simple. Furthermore, the proof does not contribute any useful engineering insights; all of the results of interest to system and circuit designers are contained in the end result, eq. (50), and hence, the proof is not included here.

Equation (48) corresponds to a z transform expressed in terms of the sample values of $v_1(t)$. This expression is in a form especially suitable for time-domain studies, such as those involving difference equations and their numerical solution by digital filters. Equation (50) corresponds to the same z transform, but it is expressed in terms of $V_1(j\omega)$, the transform of $v_1(t)$. This expression is in a form especially suitable for frequency-domain studies, such as those involving the frequency characteristics of sampled signals and digital filters.

With $s = j\omega$, eqs. (46), (47), and (50) can be used to express the transform of the stair-step output voltage delivered by the sample-and-hold circuit of Fig. 3a as

$$V_2(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega T} \sum_{n=-\infty}^{\infty} V_1[j(\omega - n\omega_s)]. \tag{51}$$

This equation gives the transform of the output voltage V_2 in terms of the transform of the input voltage V_1 and a weighting factor associated with the hold part of the sample-and-hold operation. Consider the summation on the right-hand side of eq. (51). Figure 6 illustrates the important relations expressed by this summation. Figure 6a shows a possible frequency spectrum $|V_1(j\omega)|$ for the signal at the input of the circuit in Fig. 3a. Figure 6b shows a possible spectrum of the summation in eq. (51), generated by sampling the input signal. As specified by eq. (51), this spectrum is exactly the spectrum of Fig. 6a repeated periodically on the ω axis with the period ω_s.

The first thing to be noticed about the spectrum shown in Fig. 6b is that, under the conditions pictured, this spectrum contains the undistorted spectrum of Fig. 6a, the spectrum of the original signal before sampling. Thus, it contains in undistorted form all of the information in the original signal. Clearly, this is true only if the maximum frequency ω_m in the original signal is less than $\omega_s/2$, half the sampling frequency. If ω_m exceeds this limit, the periodically repeating spectra will overlap, and the information contained in the original spectrum shown in Fig. 6a will be irreversibly distorted. Distortion arising from this source is called fold-over error or, more commonly, aliasing.

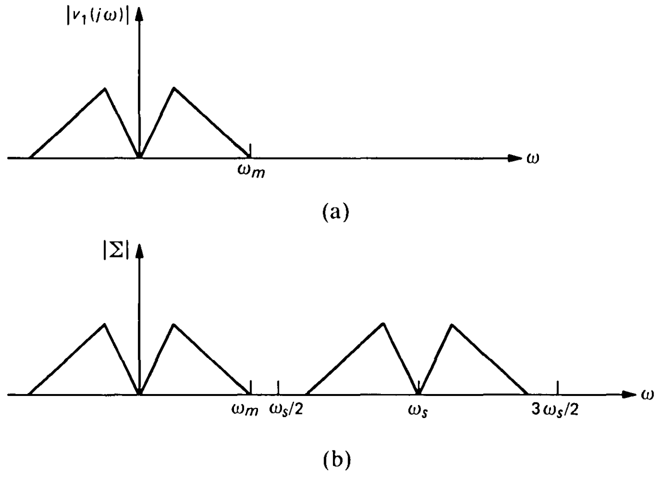


Fig. 6—Frequency spectra. (a) Spectrum of the signal before sampling. (b) Spectrum of the sum in eq. (51).

Armed with these results, it is appropriate to return now to eq. (51), the complete transform of the stair-step output voltage of the sample-and-hold circuit in Fig. 3. This equation is

$$V_2(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega T} \sum_{n=-\infty}^{\infty} V_1[j(\omega - n\omega_s)] \tag{52}$$

$$= \frac{1 - e^{-j\omega T}}{j\omega T} \{V_1(j\omega) + V_1[j(\omega \pm \omega_s)] + \cdots\}. \tag{53}$$

If the sample-and-hold circuit is designed so that there is no fold-over error, or aliasing, then it follows from Fig. 6 and eq. (53) that an ideal low-pass filter can be used to recover the spectrum of the input signal $V_1(j\omega)$ from the spectrum of the stair-step output signal $V_2(j\omega)$. The output from such a low-pass filter is, from eq. (53),

$$V_3(j\omega) = \frac{1 - e^{-j\omega T}}{j\omega T} V_1(j\omega). \tag{54}$$

Thus, $V_1(j\omega)$ is recovered from the output of the sample-and-hold circuit, but it is weighted by another function of ω representing the filtering action of the sample-and-hold circuit. The low-pass filter used to recover $V_1(j\omega)$ is called a reconstruction filter. It is understood that $V_1(j\omega)$ is zero at all frequencies outside the passband of the reconstruction filter.

Equation (54) can be rearranged to give further insight in the following manner. Factoring $e^{-j\omega T/2}$ out of the numerator of the fraction yields

$$V_3(j\omega) = \frac{e^{j\omega T/2} - e^{-j\omega T/2}}{j2(\omega T/2)} e^{-j\omega T/2}\, V_1(j\omega)$$

$$= \frac{\sin(\omega T/2)}{\omega T/2} e^{-j\omega T/2}\, V_1(j\omega). \tag{55}$$

Then the magnitude of $V_3(j\omega)$ is

$$|V_3(j\omega)| = \left|\frac{\sin(\omega T/2)}{\omega T/2}\right| |V_1(j\omega)|. \tag{56}$$

With the sampling frequency $f_s = 1/T$ as before,

$$\frac{\omega T}{2} = \frac{\omega}{2f_s} = \frac{\pi\omega}{\omega_s}, \tag{57}$$

and

$$|V_3(j\omega)| = \left|\frac{\sin(\pi\omega/\omega_s)}{\pi\omega/\omega_s}\right| |V_1(j\omega)|. \tag{58}$$

The first factor on the right in eq. (58) is the magnitude of the filter function associated with the sample-and-hold circuit. It is the (sin x)/x function that occurs frequently in the study of signals and the response of linear systems to signals. A plot of this function is shown in Fig. 7. As eq. (58) shows, the spectrum of the signal at the input to the sample-and-hold circuit is multiplied by this weighting curve. Figures 6 and 7 should be compared in the light of this fact. Note that the (sin x)/x weighting function in this case has zeros at ω equal to $\pm\omega_s, \pm 2\omega_s, \cdots, \pm k\omega_s, \cdots$.

In the design of precision digital filters the effects of the (sin x)/x function in Fig. 7 must be accounted for at some point in the design. In some cases, the reconstruction filter, discussed in connection with

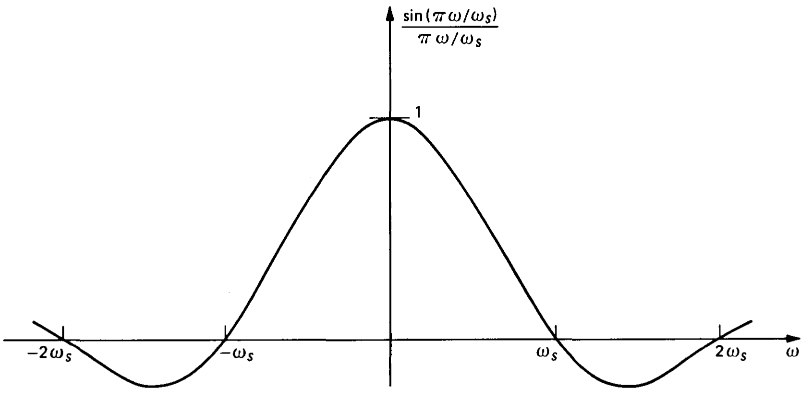


Fig. 7—Plot of the factor in eq. (58).

eq. (54), is designed to compensate for the (sin x)/x function over the band of frequencies occupied by the signal.

The developments above provide a quantitative answer to the important question of how frequently a signal must be sampled to ensure that the train of samples constitutes an accurate representation of the information contained in the signal. The results presented above show that the original information can be recovered with good accuracy if there is no fold-over error, and they also show that irremovable distortion is introduced when fold over occurs. Fold over and distortion are avoided if the maximum frequency f_m contained in the original signal is less than half the sampling frequency. Thus, for no distortion, it is required that

$$f_m < \frac{f_s}{2} = \frac{1}{2T}. \tag{59}$$

This result was first published by H. Nyquist, and the particular sampling frequency $f_s = 1/T = 2f_m$ is called the Nyquist frequency or the Nyquist rate.

From these results, it follows that the original signal must be band-limited so that a sampling frequency greater than $2f_m$ can be chosen. Furthermore, after a sampling frequency f_s has been chosen, it is still usually necessary to pass the input signal through a band-limiting filter to prevent high-frequency noise and spurious signals from being folded back on top of the baseband signal. Such band-limiting filters are usually called antialiasing filters.

It is shown above that when a signal is sampled, the information in the original signal can be recovered from the sampled signal with the aid of an ideal low-pass reconstruction filter, provided that the signal is sampled at a frequency at least twice the highest frequency contained in the original signal. In practice, however, ideal filters are not available, and practical filters require a band of frequencies in which to make the transition from the passband to the stopband. Thus, in order to avoid distortion of the information in practice, the signal must be sampled at a frequency somewhat greater than twice the maximum frequency in the original signal. Under this condition, the lobes of the periodic frequency spectrum shown in Fig. 6b are separated by an interval on the frequency axis. This interval is called a guard band, and it provides the band needed by the low-pass reconstruction filter to make the transition from the passband to the stopband.

Voice signals in telephony are usually band-limited so that the maximum frequency is in the range between 3 and 3.5 kHz, depending on the type of service involved. The sampling frequency used with such signals is usually 8 kHz, providing a guard band of from 1 to 2 kHz, depending on the maximum frequency in the signal.

IV. TRANSFER FUNCTIONS IN THE z DOMAIN

The z transform is developed in Section 3.1 as a special form of the Laplace transform with a new variable $z = e^{sT}$. The Laplace transform leads to the highly useful concept of the transfer function in the frequency domain for systems processing continuous-time signals. These transfer functions often take the desirable form of rational functions of the complex frequency s. In a similar way, the z transform leads to an equally useful transfer function in the z domain for systems processing discrete-time signals. These transfer functions often take the desirable form of rational functions of the variable $z = e^{sT}$. The objective of this section is to develop the z-domain transfer function and to examine its use in the design of digital filters.

4.1 The time-shift theorem

In the case of the Laplace transform, there are many theorems stating the mathematical properties of the transform, and similarly, there are many theorems stating basic properties of the z transform. However, among all of these z-transform theorems, there is only one that is important in this study of digital filters. This is the time-shift theorem, and it is important in deriving the z-domain transfer function from the linear difference equation (see Section II).

To develop this theorem, consider the function $f(t)$ such that

$$f(t) = 0 \qquad \text{for} \quad t < 0. \tag{60}$$

Consider also the same function delayed in time, $f(t - kT)$, where k is a positive integer and

$$f(t - kT) = 0 \qquad \text{for} \quad t - kT < 0. \tag{61}$$

The z transform of the sequence of samples representing $f(t)$ is obtained, using eq. (18), by changing t to nT and writing

$$F(z) = \sum_{n=0}^{\infty} f(nT)z^{-n} \tag{62}$$

$$= f(0) + f(T)z^{-1} + \cdots + f(nT)z^{-n} + \cdots . \tag{63}$$

Similarly, if the delayed signal is sampled at the same instants, the transform of the delayed signal is obtained by replacing t with nT and writing

$$F_d(z) = \sum_{n=0}^{\infty} f[(n-k)T]z^{-n} \tag{64}$$

$$= f(-kT) + f[(1-k)T]z^{-1} + \cdots + f(-T)z^{-(k-1)}$$
$$+ f(0)z^{-k} + f(T)z^{-(k+1)} + f(2T)z^{-(k+2)} + \cdots . \tag{65}$$

But according to eq. (61), all terms in eq. (65) for which $n - k$ is less than zero are themselves zero, and, thus, eq. (65) reduces to

$$\begin{aligned} F_d(z) &= f(0)z^{-k} + f(T)z^{-(k+1)} + f(2T)z^{-(k+2)} + \cdots \\ &= z^{-k}[f(0) + f(T)z^{-1} + f(2T)z^{-2} + \cdots]. \end{aligned} \tag{66}$$

Comparing eqs. (66) and (63) yields

$$F_d(z) = z^{-k}F(z). \tag{67}$$

Thus, if the z transform of the sequence $f(nT)$ is $F(z)$, then for any positive integer k, the z transform of the sequence, $f[(n - k)T]$, is $z^{-k}F(z)$.

4.2 The transfer function

The s-domain transfer function evolves from the differential equation and the Laplace transform. In a similar manner, the z-domain transfer function evolves from the difference equation and the z transform. A simple example serves as a satisfactory introduction to this topic, and such an example is provided by the circuit of Fig. 1, the corresponding difference eq. (7), and the related discussion in Section II.

The difference eq. (7) is

$$v_2(nT) = av_1(nT) + bv_2[(n - 1)T]. \tag{68}$$

This expression represents one sample value of the response $v_2(nT)$. This present value of v_2 depends on the present value of the input signal $v_1(nT)$ and also on the previous sample value of v_2. The totality of sample values of the input v_1 and the response v_2 for nonnegative values of n is given in Section II as

$$\begin{aligned} &v_1(nT)\text{: } v_1(0), v_1(T), v_1(2T), \cdots, v_1(kT), \cdots, \\ &v_2(nT)\text{: } v_2(0), v_2(T), v_2(2T), \cdots, v_2(kT), \cdots. \end{aligned} \tag{69}$$

The z transform of $v_1(nT)$ can be written, from eq. (18), as

$$V_1(z) = \sum_{n=0}^{\infty} v_1(nT)z^{-n}, \tag{70}$$

and the z transform of $v_2(nT)$ is

$$V_2(z) = \sum_{n=0}^{\infty} v_2(nT)z^{-n}. \tag{71}$$

Similarly, eq. (68) can be transformed by multiplying by z^{-n} and summing over all nonnegative values of n; the result is

$$\sum_{n=0}^{\infty} v_2(nT)z^{-n} = a \sum_{n=0}^{\infty} v_1(nT)z^{-n} + b \sum_{n=0}^{\infty} v_2[(n - 1)T]z^{-n}. \tag{72}$$

Now, using eqs. (70) and (71) together with the time-shift theorem (eqs. (64) and (67)), eq. (72) can be written simply as

$$V_2(z) = aV_1(z) + bz^{-1}V_2(z). \tag{73}$$

Equation (73) can now be solved to obtain the response $V_2(z)$ as an explicit function of the input $V_1(z)$, and the result is

$$V_2(z) = \frac{a}{1 - bz^{-1}} V_1(z). \tag{74}$$

The z-domain transfer function is then defined as

$$H(z) = \frac{V_2(z)}{V_1(z)} = \frac{a}{1 - bz^{-1}}. \tag{75}$$

In this way, given a linear difference equation, the corresponding z-domain transfer function can be determined. The result is a rational function of the variable z, and the coefficients in the transfer function are the coefficients in the difference equation. For digital filters the coefficients are normally real numbers. Of much greater importance to the design of digital filters, however, is the fact that this process works equally well in the opposite direction. Given a z-domain transfer function that is a rational function of z with real coefficients, a corresponding linear difference equation can be determined. This reverse operation is of basic importance because the digital filter performs the filtering function by the numerical evaluation of the difference equation, as discussed in some detail in Section II. The required difference equation is almost always obtained from a z-domain transfer function.

Of further importance in this connection is the fact that there are many ways in which s-domain transfer functions for continuous-time signals can be transformed into approximately equivalent z-domain transfer functions for discrete-time signals. Thus, frequency-domain specifications for a filter can be used with classical procedures to obtain an s-domain transfer function, the s-domain transfer function can then be transformed into a z-domain transfer function, and finally, the difference equation required by the digital filter can be determined from the z-domain transfer function. These matters are discussed in more detail in Section VI.

The term *z-domain transfer function* is often replaced by the term *digital transfer function* to emphasize the fact that $H(z)$ is associated with a digital filter. Similarly, to preserve the distinctions, the s-domain transfer function is often called an *analog transfer function.* These terms are used in the remainder of this paper.

Returning to eq. (74) and using the definition in eq. (75), the transform of the response can be written as

$$V_2(z) = \frac{a}{1 - bz^{-1}} V_1(z) = H(z)V_1(z). \tag{76}$$

Some further insights can be gained by considering the special case in which the input signal has a z transform $V_1(z) = 1$. This transform corresponds to a sample sequence $v_1(nT)$ that has a value of unity for $n = 0$ and a value of zero for all other values of n. Such a signal is sometimes called a unit-sample signal. With this input, eq. (76) becomes

$$V_2(z) = H(z) = \frac{a}{1 - bz^{-1}}, \tag{77}$$

and it represents the unit-sample response of the difference equation (68). (For future use, note the similarity between the unit-sample response in this case and the unit-impulse response in the case of the analog transfer function.) The inverse transform of eq. (77) gives the time-domain response $v_2(nT)$ to the unit-sample input. The inversion can be performed by expanding the right-hand side of eq. (77) in a power series in z^{-1} and then reading off the coefficients of successive powers of z^{-1}. One way to get such a power series in this case is to perform algebraically the division indicated by eq. (77); when this is done, the result is

$$V_2(z) = H(z) = a + abz^{-1} + ab^2z^{-2} + \cdots . \tag{78}$$

The coefficients in the successive terms of this series are the uniformly spaced samples of the unit-sample response $v_2(nT)$; they are

$$v_2(nT)\text{: } a, ab, ab^2, \cdots , ab^n, \cdots . \tag{79}$$

Note that exactly this result is obtained in Example 1 of Section II by direct evaluation of difference eq. (7).

The sample values given in eq. (79) lie on a decaying exponential curve having the equation

$$v_2(t) = ab^{t/T}. \tag{80}$$

This assertion can be verified by substituting $t = nT$ and comparing the result with the general term in the sequence given by expression (79). It follows from eqs. (6) and (7) that b is nonnegative and less than unity. Thus, $v_2(t)$ in eq. (80) decreases by the factor b (less than unity) in every interval of duration T, which is in agreement with eq. (79).

The sequence of sample values given in expression (79) constitutes the inverse transform of the digital transfer function for the filter in Fig. 1, and it is also the response of the filter to a unit-sample input. In the study of the Laplace transform, the inverse transform of the analog transfer function is identified with the response of the filter to a unit-impulse input. The terminology developed for the analog transfer function is carried over to the digital transfer function, and the inverse of the digital transfer function is also called the impulse response of the filter, although in fact it is the unit-sample response. Thus expression (79) is called the impulse response of the digital transfer function in eq. (77). Note that the inverse transform given by expression (79) consists of a sequence of infinitely many samples. Many digital filters have impulse responses of this form, and as a class they are called infinite-impulse-response (IIR) filters.

The digital transfer functions that are useful in the design of digital filters are usually rational functions of z that can be written in the form

$$H(z) = \frac{N(z)}{D(z)}, \qquad (81)$$

where N and D are polynomials in z. The impulse response (unit-sample response) associated with these functions can be determined by writing N and D in ascending powers of z^{-1} and performing the indicated division, as was done to obtain eq. (78). In general, the impulse response is a sequence of infinitely many samples as in eq. (78). However, there is a subset having the form

$$H(z) = N(z) = a_0 + a_1 z^{-1} + \cdots + a_N z^{-N}, \qquad (82)$$

and members of this subset are sometimes used as the basis for digital filters. In this case, the impulse response has a finite number of terms, $a_0, a_1, \cdots a_N$, given by eq. (82), and no division is needed to determine them. Filters of this class are called finite-impulse-response (FIR) filters. These filters are the digital counterparts of the classical analog transversal filters realized with the aid of electrical delay lines. See Refs. 1 through 3 for further information on FIR filters.

4.3 Frequency characteristics of the digital transfer function

Section 4.2 shows how the digital transfer function can be developed from a linear difference equation. The main result is given by eqs. (74) and (75), and it has the form

$$V_b(z) = H(z) V_a(z), \qquad (83)$$

where $H(z)$ is the digital transfer function and V_a and V_b are transforms of the input and output signals, respectively. At this point, it is possible to examine to some extent how the transfer function performs as a filter and to gain some understanding of its frequency characteristics and how they affect the transmission of signals.

This study is based on the results developed in Section III; therefore, it is helpful to revert to the symbolism used in eqs. (14) through (19). Thus, eq. (83) is rewritten as

$$V_b^*(z) = H(z) V_a^*(z). \tag{84}$$

The transform $V_b^*(z)$ represents a sequence of discrete-time sample values $v_b(nT)$. However, filter characteristics, as understood by engineers, have meaning only in terms of continuous-time analog input and output signals. Therefore, it is assumed here that the digital transfer function (filter) is followed by a digital-to-analog converter that generates a stair-step wave based on the sequence of sample values $v_b(nT)$, as illustrated in Fig. 4, to produce an output in analog form.

The Laplace transform of this stair-step wave is given by eq. (16) as

$$V_c(s) = H_h(s) \hat{V}_b(s). \tag{85}$$

The next step concerns certain definitions that are made in Section 3.1. The identities below are not functional relations—they are pure definitions. The quantity $V_b^*(z)$ in eq. (84) is defined by eqs. (14), (17), and (18) to be

$$\sum_{n=0}^{\infty} v_b(nT) e^{-nsT} \equiv \hat{V}_b(s) \equiv V_b^*(e^{sT}) \equiv V_b^*(z), \tag{86}$$

and similarly, by definition,

$$\sum_{n=0}^{\infty} v_a(nT) e^{-nsT} \equiv \hat{V}_a(s) \equiv V_a^*(e^{sT}) \equiv V_a^*(z). \tag{87}$$

Now, substituting $z = e^{sT}$ into eq. (84) and using eq. (86) leads to

$$V_b^*(e^{sT}) = H(e^{sT}) V_a^*(e^{sT}) = \hat{V}_b(s). \tag{88}$$

Rearranging this equation and using (87) produces

$$\hat{V}_b(s) = H(e^{sT}) V_a^*(e^{sT}) = H(e^{sT}) \hat{V}_a(s). \tag{89}$$

This result can now be substituted into eq. (85) to obtain

$$V_c(s) = H_h(s) H(e^{sT}) \hat{V}_a(s), \tag{90}$$

for the transform of the stair-step output.

For $s = j\omega$, eq. (90) becomes

$$V_c(j\omega) = H_h(j\omega) H(e^{j\omega T}) \hat{V}_a(j\omega), \tag{91}$$

and substituting eq. (50) for $\hat{V}_a(j\omega)$ yields

$$V_c(j\omega) = \frac{1}{T} H_h(j\omega) H(e^{j\omega T}) \sum_{n=-\infty}^{\infty} V_a[j(\omega - n\omega_s)]. \tag{92}$$

This is the transform of the stair-step output of the digital filter followed by a digital-to-analog converter.

To interpret eq. (92), it is best to start at the far right and work back to the left. The quantity $V_a(j\omega)$ is the transform of the analog signal at the input to the filter before it is sampled. The offset $n\omega_s$ and the summation represent the periodic frequency spectrum generated by sampling the input signal; an example of such a spectrum is shown in Fig. 6. The factor $H(e^{j\omega T})$ multiplying the summation in eq. (92) is the digital transfer function expressed as a function of $s = j\omega$. The magnitude and phase characteristics of this factor modify the frequency spectrum of the sampled input signal in the usual way, and by this process the filter performs its function. The design of digital filters is concerned mainly with this factor. The factor $H_h(j\omega)$ is given by eq. (15) with $s = j\omega$. This factor, together with the multiplier $1/T$, contributes the $(\sin x)/x$ weighting function shown in eq. (58) and Fig. 7.

In summary, the frequency characteristics of the digital filter followed by a digital-to-analog converter are obtained [apart from the $(\sin x)/x$ factor] from the digital transfer function $H(z)$ with $z = e^{j\omega T}$. This transfer function is, of course, closely related to the linear difference equation from which it is derived and which is implemented by the digital filter.

4.4 Difference equations from the transfer function

Section 4.2 shows how the digital transfer function can be determined from a given difference equation. This section is concerned with the inverse operation, determining the difference equation from a given digital transfer function. The importance of this operation lies in the fact that the digital filter operates, as described in Section II, by calculating values given by a difference equation. However, filter specifications are usually given in terms of frequency characteristics, and such specifications lead to transfer functions. Thus, the required difference equation is usually obtained from a digital transfer function.

The transfer functions that are appropriate for this study are rational functions of z with real coefficients, expressed as

$$H(z) = \frac{N(z)}{D(z)},$$

where N and D are polynomials in z. In the study of these functions, we note that if the polynomials N and/or D are of degree greater than two, and especially if the roots of N or D are located close together in the z plane, then practical realization of the corresponding digital filter often proves to be unsatisfactory. The performance of the filter is so sensitive to the values of the coefficients in N and D that high-quality performance cannot be obtained. Therefore, special procedures must be followed to make possible the realization of precision digital filters of high complexity. One special procedure that can be followed is to decompose the complex filter into a cascade of noninteracting biquadratic (second-order) sections, plus possibly a single first-order section. This procedure may not be the best way to solve the problem, but it is simple, it always works, and, therefore, it is widely used. (It is noted in passing that exactly the same problem arises in the design of active RC analog filters, and it is often solved in the same way.) Thus, in the remainder of this paper, the most complex transfer function to be considered, a basic building block, is the biquadratic function of the form

$$H(z) = \frac{a_0 + a_1z^{-1} + a_2z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}}. \tag{93}$$

If the response of the sampled-signal system corresponding to eq. (93) is designated $r(nT)$ with the z transform $R(z)$, and if the stimulus (input) is designated $s(nT)$ with the transform $S(z)$, then eq. (93) can be used to write

$$R(z) = \frac{a_0 + a_1z^{-1} + a_2z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}} S(z). \tag{94}$$

At this point, it is appropriate to note for future use that eq. (94) can be written in the alternative form

$$R(z) = a_0 \frac{1 + a_1'z^{-1} + a_2'z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}} S(z). \tag{95}$$

When several such sections are connected in cascade, it may be possible to lump some or all of the scale factors a_0 into a single scale factor for the entire filter. The result is a reduction in the number of multiplications that the hardware must perform, a fact that is discussed further in Section V. However, a potential problem exists in this connection, because the signal level throughout the filter must be kept in a suitable range. If the signal level is too large, some register in the filter will overflow and cause distortion. If the signal level is too small, the signal-to-noise ratio will suffer. Thus, it may be necessary to distribute the scale factors throughout the filter.

Equation (94) can be rearranged algebraically to obtain

$$R(z) = a_0S(z) + a_1z^{-1}S(z) + a_2z^{-2}S(z) - b_1z^{-1}R(z) - b_2z^{-2}R(z). \tag{96}$$

From the definition of the z transform in eq. (18), it follows that

$$S(z) = \sum_n s(nT)z^{-n}, \tag{97}$$

where it is understood that the summation is over all nonnegative values of n. Similarly, the time-shift theorem, eqs. (64) and (67), yields

$$z^{-1}S(z) = \sum_n s[(n-1)T]z^{-n}, \tag{98}$$

and

$$z^{-2}S(z) = \sum_n s[(n-2)T]z^{-n}. \tag{99}$$

Using equivalences of this kind throughout eq. (96) yields

$$\begin{aligned}\sum_n r(nT)z^{-n} = \sum_n a_0 s(nT)z^{-n} &+ \sum_n a_1 s[(n-1)T]z^{-n} \\ &+ \sum_n a_2 s[(n-2)T]z^{-n} - \sum_n b_1 r[(n-1)T]z^{-n} \\ &- \sum_n b_2 r[(n-2)T]z^{-n}.\end{aligned} \tag{100}$$

The desired difference equation is the inverse of this transform equation. The inversion is performed by removing the summations and cancelling the common factor z^{-n}; the result is

$$\begin{aligned}r(nT) = a_0 s(nT) &+ a_1 s[(n-1)T] + a_2 s[(n-2)T] \\ &- b_1 r[(n-1)T] - b_2 r[(n-2)T].\end{aligned} \tag{101}$$

The digital filter can be programmed to perform, in real time, the multiplications and additions of signal samples and coefficients required to evaluate the right-hand side of eq. (101) for each successive sample of the response $r(nT)$. A simple example of this operation is presented in Section II. According to eq. (92) and the related discussion, the frequency characteristics of the resulting digital filter are given by $H(e^{j\omega T})$, the transfer function of eq. (93) with $z = e^{j\omega T}$.

Each specific biquadratic filter section is characterized completely for the filter hardware by the coefficients a_j and b_k of the difference eq. (101). (The quantities r and s are time varying data values.) These coefficients are typically part of the program for the filter, and they are stored with the program in the read-only memory (ROM) of the hardware. It is also significant to note that the coefficients in eq. (101) are identical with the coefficients in the digital transfer function of eq. (93). Thus, when the digital transfer function for the biquadratic section has been determined, the design is complete, and the programming of the signal processor can begin.

The present value of the response $r(nT)$ in eq. (101) depends on both the present and past values of the stimulus and also on past values of the response. Or, stated another way, the response $r(nT)$ calculated at the present will be used in calculating future values of the response. Thus, eq. (101) is a kind of recursion formula for calculating successive values of the response. For that reason, filters of this class are called recursive filters. Note that the transfer function for the FIR filter, given by eq. (82), has all its b coefficients equal to zero. Thus, the response of the FIR filter does not depend on past values of the response, and hence, there is no recursion. Filters of this class are called nonrecursive filters.

V. NUMERICAL SOLUTION OF DIFFERENCE EQUATIONS

Any biquadratic digital filter is described by a linear difference equation of the form given by eq. (101). The a_j and b_k on the right-hand side of this equation are coefficients of the difference equation (filter), and they are usually stored in the ROM of the signal processor. The quantities r and s are either past data values stored in the random-access memory (RAM) of the processor or present data values available at the terminals of the processor. The filter operates by performing the multiplications and additions required to evaluate the right-hand side of eq. (101) and, thereby, determining each successive value of the response $r(nT)$. The result is sent to the output and also stored in RAM as a new past value of data. When each new value of input sample $s(nT)$ comes in, often at the 8-kHz rate discussed in Section 3.2, the processor performs the arithmetic mentioned above and produces a new value of response $r(nT)$ in a few microseconds. The processor then either waits for a new input sample, or, more commonly, it is assigned other tasks to perform until a new sample arrives.

A flow diagram for the calculations described above is shown in Fig. 8. Note that, although the symbols look like hardware blocks, this is not a hardware block diagram; it is much more closely related to the flow chart used to diagram computer programs. The nodes in this diagram labeled $s[(n-1)T]$, $r[(n-1)T]$, etc., represent RAM storage of past data values, and the blocks labeled D represent time delays of

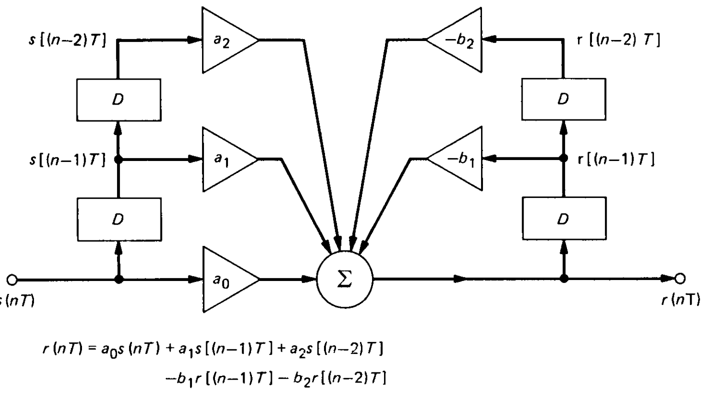


Fig. 8—Flow diagram of the calculations for a biquadratic filter. The symbol D represents a time delay of T units, one sampling interval.

T units, one sampling period. The triangles labeled a_j and b_k represent multiplications of the indicated data values by the filter coefficients.

The computational procedure pictured in Fig. 8 requires four RAM locations for the storage of past data values. This memory requirement can be cut in half by rearranging the computational procedure. One way to develop the modified procedure is to return to the transform relation in eq. (94),

$$R(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} S(z). \qquad (102)$$

This equation can be separated into two parts by defining a new, intermediate, variable given by

$$S_m(z) = \frac{1}{1 + b_1 z^{-1} + b_2 z^{-2}} S(z), \qquad (103)$$

$$= S(z) - b_1 z^{-1} S_m(z) - b_2 z^{-2} S_m(z). \qquad (104)$$

The quantity $S_m(z)$ represents a modified form of the input stimulus. Now substituting eq. (103) into eq. (102) yields

$$R(z) = (a_0 + a_1 z^{-1} + a_2 z^{-2}) S_m(z), \qquad (105)$$

$$= a_0 S_m(z) + a_1 z^{-1} S_m(z) + a_2 z^{-2} S_m(z). \qquad (106)$$

Repeating the procedure followed in eqs. (96) through (101), eqs. (104) and (106) yields the following two simultaneous difference equations for the biquadratic filter section:

$$s_m(nT) = s(nT) - b_1 s_m[(n-1)T] - b_2 s_m[(n-2)T], \qquad (107)$$

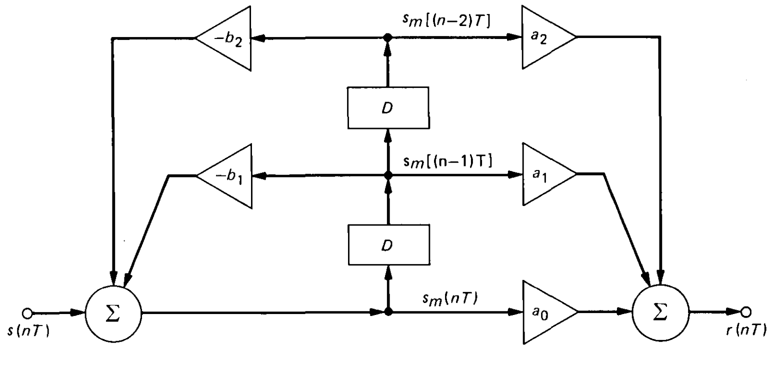
and

$$r(nT) = a_0 s_m(nT) + a_1 s_m[(n-1)T] + a_2 s_m[(n-2)T]. \qquad (108)$$

For each successive sample of input stimulus $s(nT)$, these two equations yield the corresponding sample of response $r(nT)$. Equations (107) and (108) together produce the same result as eq. (101). In this case, however, only two quantities, $s_m[(n-1)T]$ and $s_m[(n-2)T]$, need to be stored in RAM.

A flow diagram for the solution of eqs. (107) and (108) is shown in Fig. 9. As in the case of Fig. 8, this is not a hardware block diagram; instead, it is a flow diagram intended to help the reader visualize the computational steps involved in the solution of eqs. (107) and (108). The symbols used in Fig. 9 are the same as those used in Fig. 8.

The five triangular blocks in Fig. 9 represent five multiplications used to evaluate one sample of the response $r(nT)$. In this connection, it is appropriate to return briefly to eq. (95) and note that, if the scale factor is to be accounted for at a later point, the coefficient a_0 in eq.



$s_m(nT) = s(nT) - b_1 s_m[(n-1)T] - b_2 s_m[(n-2)T]$

$r(nT) = a_0 s_m(nT) + a_1 s_m[(n-1)T] + a_2 s_m[(n-2)T]$

Fig. 9—Flow diagram of the alternative calculations for a biquadratic filter. The symbol D represents a time delay of T units, one sampling interval.

(95) has been normalized to unity. But Fig. 9 shows that if a_0 is unity, then no multiplication by a_0 is needed; a direct transmission of $s_m(nT)$ to the output is sufficient. In this case, the system becomes what is sometimes called a 4-multiply biquadratic section. The 4-multiply section uses less ROM and is somewhat faster than the 5-multiply section.

When power is first applied to the digital filter, the RAM contains random values for the past data values $s_m[(n-1)T]$ and $s_m[(n-2)T]$, and the first few samples of the response $r(nT)$ that are calculated depend on these random initial conditions. However, the effects of the random initial conditions decay with time, and they soon disappear.

VI. CONSTRUCTING THE DIGITAL TRANSFER FUNCTION H(z)

In the preceding sections we showed how the digital filter operates by the numerical evaluation of the appropriate linear difference equation and how the appropriate difference equation can be obtained from the appropriate digital transfer function, a function related to the difference equation by the z transform. The problem that remains, and which the following paragraphs address, is finding the appropriate digital transfer function. No attempt is made to give a complete treatment of this challenging and multifaceted problem. Instead, attention is focused on a single technique that is possibly the most widely used solution. (See Refs. 1 through 3 for further discussions on this technique and others.)

6.1 General considerations

The design of filters, both analog and digital, usually begins with a set of specifications in the frequency domain. These specifications are usually concerned with the passband of the filter, the stopband, attenuation peaks, delay characteristics, and similar frequency-domain considerations. Given this information, there are extensive theoretical tools and computer aids that can be used to obtain a continuous-time analog filter meeting the specifications, provided of course that the specifications are physically realizable. In most cases, the design of discrete-time digital filters takes full advantage of these design aids by first producing a suitable prototype analog filter. Then the s-domain transfer function for the analog filter is transformed into a z-domain transfer function for a digital filter that provides the desired frequency characteristics.

The end result required is a linear difference equation with real coefficients. This result is to be obtained from a digital transfer function by the procedure presented in Section 4.4, eqs. (96) through (101). To obtain the desired result by this method, the transfer function must be a rational function of z, and it must have real coefficients.

The transformation of the prototype analog transfer function into a suitable digital transfer function can be accomplished with the aid of a relation, called a transformation, having the general form

$$s = F(z). \tag{109}$$

When $F(z)$ is properly chosen, substituting it for s in the prototype analog transfer function produces the desired digital transfer function. The problem now is to find a suitable transformation $F(z)$.

But first some related matters need consideration. One of these is the fact that two transformations are now involved in the design of a digital filter. The first is eq. (109), used to obtain a digital transfer function $H(z)$. Then, as shown in connection with eqs. (90) and (92), the frequency characteristics of the digital filter are determined by substituting the transformation

$$z = e^{sT} \tag{110}$$

for z in $H(z)$. Both of these transformations are satisfied independently, and both of them involve the complex-frequency variables s, although each in a different context. In order to avoid confusing these independent uses of s, it is helpful to rewrite eqs. (109) and (110) as

$$s_a = F(z) \tag{111}$$

and

$$z = \exp(s_d T), \tag{112}$$

where the subscripts a and d designate analog and digital, respectively. This notation leads to further symbolism as follows:

$H_a(s_a)$ represents the prototype analog transfer function.
$H(z)$ represents the transformed function, a digital transfer function.
$H_d(s_d)$ represents the transformed function with z replaced by $\exp(s_dT)$. This function gives the frequency characteristics of the digital filter.

An important feature of transformations, such as eqs. (111) and (112), can be illustrated in the following way. Consider first a specific value of the analog frequency

$$s_a = s_{a1}. \tag{113}$$

For this value of s_a, the analog transfer function has a specific value, a complex number,

$$H_a(s_{a1}) = X + jY. \tag{114}$$

The corresponding value of z is obtained by solving eq. (111),

$$s_{a1} = F(z_1). \tag{115}$$

The digital transfer function $H(z)$ is obtained from $H_a(s_{a1})$ by replacing s_{a1} with the equal number $F(z_1)$. Thus,

$$H(z_1) = H_a(s_{a1}) = X + jY. \tag{116}$$

Hence, values of s_a and z that satisfy eq. (111) produce values of $H_a(s_a)$ and $H(z)$ that are equal.

In exactly the same way, values of z and s_d that satisfy eq. (112) produce values of $H(z)$ and $H_d(s_d)$ that are equal, and hence,

$$H_d(s_{d1}) = H(z_1) = X + jY, \tag{117}$$

where X and Y have the same values as in eqs. (114) and (116).

In summary,

$$H_a(s_{a1}) = H(z_1) = H_d(s_{d1}) = X + jY, \tag{118}$$

provided that s_{a1}, z_1, and s_{d1} are related by the transformations of eqs. (111) and (112). Thus, if $H_a(s_a)$ is known for various values of s_a, the the values of $H(z)$ and $H_d(s_d)$ are known for the corresponding values of z and s_d. This fact is important because it relates the frequency characteristics of the digital filter, $H_d(s_d)$, to those of the prototype analog filter, $H_a(s_a)$.

These relations also hold when the transfer functions are infinite; thus, if $H_a(s_a)$ has a pole at s_{a1}, then $H(z)$ and $H_d(s_d)$ have poles at the corresponding points z_1 and s_{d1}.

6.2 The bilinear transformation

With the preliminaries taken care of, it is now appropriate to return to the main problems of identifying a suitable transformation $F(z)$ for use in eq. (111). It seems that the most efficient way of presenting this subject is simply to state at the outset a transformation that does the job well in many cases. The properties of this transformation can then be developed, and the significance of these properties can be examined to gain useful insight into the nature of the problem. In this way, a general understanding of the problem can be gained.

A transformation that does the job well, that is simple, and that is possibly the transformation most widely used in the design of digital filters is one that causes eq. (111) to take the form

$$s_a = \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}, \tag{119}$$

where T is the sampling interval. Since this transformation is the ratio of two linear polynomials in z, it is called a *bilinear* transformation. The multiplier $2/T$ is simply a scale factor, and it proves to be a particularly convenient one. This choice is not a requirement, however, and various writers on this subject use various scale factors. Nevertheless, the final result always comes out the same, for in every case a frequency-scale adjustment, described in Section 6.3, is required, and this adjustment compensates exactly for the various values used for the scale factor in eq. (119).

The use of this transformation always yields a digital filter with an IIR (see Section 4.2). For those cases in which a FIR filter is desired, a different procedure must be used. (See Refs. 1 through 3 for further information on FIR filter design.)

The bilinear transformation has been studied in detail by mathematicians (see Ref. 5), and it has been used to advantage in various applications in physics and engineering. For example, it is the basis for the Smith chart, a valuable tool in the study of transmission lines. Each value of z produces only one value of s_a; therefore, the transformation is single valued. The inverse of the transformation,

$$z = \frac{1 + s_aT/2}{1 - s_aT/2}, \tag{120}$$

is also bilinear, and hence, it is also single valued. This is the most general transformation that is single valued in both directions. Note that the transformation of eq. (112) is multivalued in the inverse direction, as is shown in Section 3.1.

The circle is a characteristic figure for the bilinear transformation. If straight lines are treated as circles of infinite radius, then, under this transformation, all circles in one plane transform into circles in the other plane. For example, the rectangular grid lines for the real and imaginary parts of s_a transform into sets of orthogonal circles in the z

plane with every circle passing through the point $z = -1$. As shown in Section 6.3, one of these grid lines, the $j\omega_a$ axis, transforms onto the unit circle in the z plane. This fact is of considerable importance because the other transformation used in this work, eq. (112), transforms the $j\omega_d$ axis of the s_d plane onto the unit circle in the z plane. Thus, the unit circle in the z plane serves as a bridge between the $j\omega_a$ axis and the $j\omega_d$ axis, and hence it is important in relating the frequency characteristics of the prototype analog filter to those of the digital filter.

The significant properties of the bilinear transformation, developed in Section 6.3, are listed here for immediate use. They are as follows:

(*i*) Rational functions of s_a with real coefficients are transformed into rational functions of z with real coefficients. This property ensures that the $H(z)$ obtained will produce a usable difference equation.

(*ii*) The left half of the s_a plane is mapped into the interior of the unit circle in the z plane. This property ensures that every stable prototype analog filter will lead to a stable digital filter.

(*iii*) The $j\omega_a$ axis of the s_a plane is mapped onto the unit circle in the z plane. This property ensures that the frequency characteristics of the analog filter, $H_a(j\omega_a)$, are preserved in the digital filter, $H_d(j\omega_d)$, although in general there will be some warping of the frequency scale. See eq. (118).

The two transformations involved in the design of digital filters are repeated here for convenience. They are

$$s_a = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}, \tag{121}$$

and

$$z = \exp(s_d T). \tag{122}$$

Consider the stability of the digital filter. It follows from eq. (90) and the related discussion that if the digital filter is to be stable, the poles of the digital transfer function, $H[\exp(s_d T)] = H_d(s_d)$, must lie inside the left half of the s_d plane. Furthermore, as Section 3.1 shows, the transformation given by eq. (122) maps the entire left half of the s_d plane into the interior of the unit circle in the z plane, and it maps the right half of the s_d plane into the exterior of the unit circle. Thus, for a stable digital filter the poles of the corresponding $H(z)$ must lie inside the unit circle. But, as stated in item (*ii*) above, the bilinear transformation in eq. (121) maps the entire left half of the s_a plane into the interior of the unit circle. Therefore, with this transformation every pole in the left half of the s_a plane maps through the unit circle into the left half of the s_d plane, and every stable prototype analog filter leads to a stable digital filter. This is a very desirable feature.

The bilinear transformation maps the poles and zeros anywhere in the left half of the s_a plane, suitable for stable analog filters, into the interior of the unit circle in the z plane, suitable for stable digital filters. The outward manifestation of this change is in the marked difference in the coefficients of the transfer functions $H_a(s_a)$ and $H(z)$. The view can be taken that the transformation transforms a set of filter coefficients useful for analog realization into another set useful for digital realization. In general, a set of filter coefficients suitable for analog realization is not suitable for digital realization.

As shown in Section 3.1, the transformation given by eq. (122) maps the $j\omega_d$ axis of the s_d plane onto the unit circle in the z plane; every point on the $j\omega_d$ axis maps into a point on the unit circle. However, because of the nature of the exponential function in the transformation eq. (122), every point on the unit circle in the z plane maps into infinitely many points on the $j\omega_d$ axis. This matter is discussed in Sections 3.1 and 3.2, where we show that the entire unit circle maps onto the interval $-\omega_s/2$ to $\omega_s/2$ on the $j\omega_d$ axis, where ω_s is the sampling frequency in radians per second. The mapping then repeats itself periodically on the $j\omega_d$ axis as z moves endlessly around the unit circle. (See Fig. 10.) As the mapping repeats itself periodically on the $j\omega_d$ axis, so $H_d(s_d)$ also repeats itself periodically on the axis.

As stated in item (*iii*) the bilinear transformation in eq. (121) maps every point on the $j\omega_a$ axis of the s_a plane onto the unit circle in the z plane. Thus, points on the $j\omega_a$ axis map through the unit circle to points on the $j\omega_d$ axis in the s_d plane. Under these conditions, the values of the prototype analog transfer function H_a on the $j\omega_a$ axis are repeated as values of the digital transfer function H_d on the $j\omega_d$ axis, as indicated by eq. (118), although in general the values are not distributed in the same way along the two $j\omega$ axes. That is, the frequency characteristics of the digital filter are similar to those of the prototype analog filter, but the frequency scale is warped.

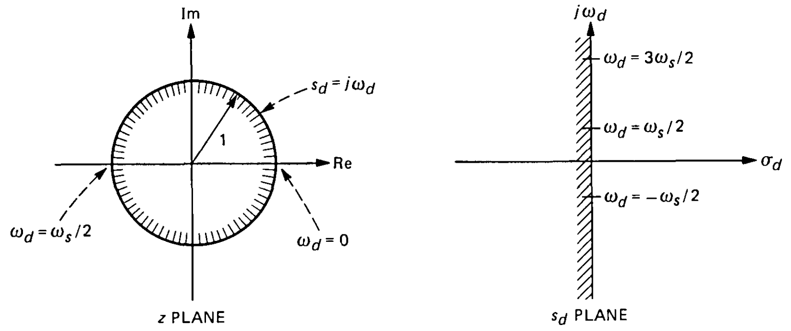


Fig. 10—Relations between the z plane and the s_d plane.

The effectiveness of the bilinear transformation in this application can be highlighted by considering briefly two other transformations that are superficially attractive but that totally fail to produce useful digital filters. The first of these is

$$s_a = \frac{z}{T},$$

where T is the sampling interval. The factor $1/T$ is included as a scale factor and to make the equation dimensionally balanced. This transformation seems attractive because it is so simple. It has the property given in item (*i*) but it does not have the properties of items (*ii*) and (*iii*). Thus, for a given T there are stable analog filters that transform into unstable digital filters. Furthermore, if a stable digital filter is obtained with this transformation, the frequency characteristics of the analog filter are not preserved in the digital filter, and with this transformation it is not possible to obtain a useful digital filter using conventional techniques.

The second transformation to be considered is

$$z = \exp(s_a T)$$

or

$$s_a = \frac{1}{T} \ln z. \tag{123}$$

Section 3.1 shows that this transformation has the properties listed in items (*ii*) and (*iii*) above. Moreover, the transformation in eq. (123) is the exact inverse of the one given by eq. (122), and applying these two transformations in succession produces $H_d(s_d) = H_a(s_a)$, a digital filter identical with the analog prototype. However, transformation eq. (123) does not have the property listed in item (*i*) above, and hence it cannot be used to obtain a difference equation by the method given in Section 4.4. Without a difference equation, the techniques presented here cannot be used to produce a digital filter. It may be of some interest, but of no real significance, to note that the logarithm in eq. (123) can be represented by an infinite series of the form

$$\ln z = 2\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \cdots\right].$$

Approximating the logarithm by the first term in this series and substituting it into eq. (123) yields

$$s_a = \frac{2}{T}\frac{z-1}{z+1},$$

which is exactly the bilinear transformation of eq. (121).

6.3 Details of the bilinear transformation

The properties of the bilinear transformation presented without proof in Section 6.2 are examined in some detail in this section. The discussion is aided by the diagrams shown in Fig. 11. The transformation is given by eq. (119), and its inverse, given by eq. (120), is

$$z = \frac{1 + s_a T/2}{1 - s_a T/2}. \tag{124}$$

Thus, when $s_a = j\omega_a$,

$$z = \frac{1 + j\omega_a T/2}{1 - j\omega_a T/2} = \frac{Me^{j\phi}}{Me^{-j\phi}}$$
$$= e^{j2\phi}, \tag{125}$$

where

$$\phi = \arctan(\omega_a T/2). \tag{126}$$

Thus, when $s_a = j\omega_a$, $|z| = 1$, and hence, the $j\omega_a$ axis in Fig. 11a is mapped by the bilinear transformation onto the unit circle in the z plane as shown in Fig. 11b. Thus, we verify the assertion made to this effect in Section 6.2.

More generally, with $s_a = \sigma_a + j\omega_a$, eq. (124) gives z as

$$z = \frac{1 + \sigma_a T/2 + j\omega_a T/2}{1 - \sigma_a T/2 - j\omega_a T/2} = \frac{N}{D}. \tag{127}$$

Thus,

$$|z| = \left|\frac{N}{D}\right|, \tag{128}$$

with

$$|N| = [(1 + \sigma_a T/2)^2 + (\omega_a T/2)^2]^{1/2}, \tag{129}$$

and

$$|D| = [(1 - \sigma_a T/2)^2 + (\omega_a T/2)^2]^{1/2}. \tag{130}$$

Now, since T is always positive, it follows from eqs. (128), (129), and (130) that

$$|z| < 1 \qquad \text{for all } \sigma_a < 0, \tag{131}$$

and that

$$|z| > 1 \qquad \text{for all } \sigma_a > 0. \tag{132}$$

Thus, the bilinear transformation maps the entire left half of the s_a plane into the interior of the unit circle in the z plane, and it maps the entire right half of the s_a plane into the exterior of the unit circle. Thus, we verify another assertion made in Section 6.2.

After the bilinear transformation has been applied to the prototype analog transfer function $H_a(s_a)$, the transformation given by eq. (122) is used to obtain the digital transfer function $H_d(s_d)$. The results of this transformation are illustrated in Figs. 11b and 11c, and they are

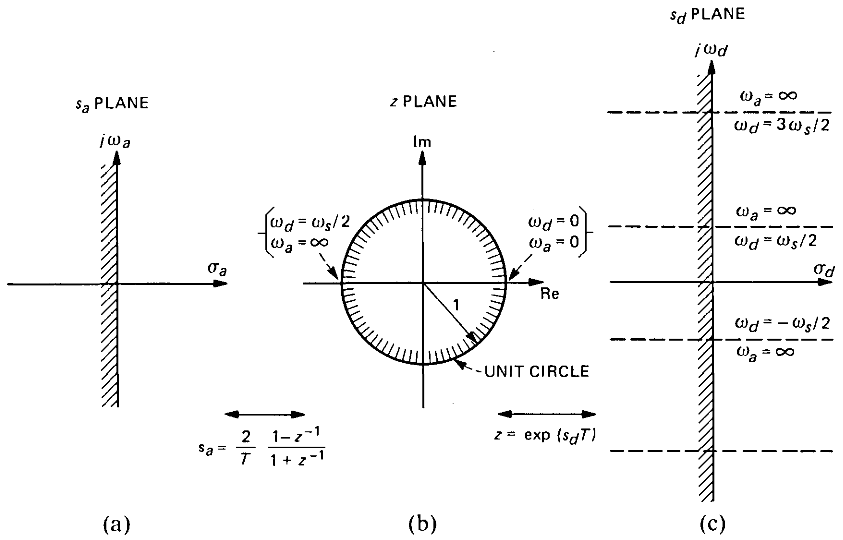


Fig. 11—Transformations and mapping.

discussed in some detail in Section 6.2 in connection with Fig. 10.

Returning to the transformations given by eq. (124),

$$z = \frac{1 + s_a T/2}{1 - s_a T/2}, \tag{133}$$

and eq. (122),

$$z = \exp(s_d T), \tag{134}$$

the following additional information can be deduced. From eq. (133), when

$$s_a = 0, \qquad z = 1, \tag{135}$$

and when

$$s_a = \infty, \qquad z = -1. \tag{136}$$

Similarly, from eq. (134), when

$$s_d = 0, \qquad z = 1, \tag{137}$$

and when

$$s_d = j\omega_s/2, \qquad z = -1, \tag{138}$$

where eq. (138) is worked out in detail in Section 3.1 and ω_s is the sampling frequency in radians per second. These relations are shown in Fig. 11.

As $s_a = j\omega_a$ increases from zero to infinity in Fig. 11a, z moves counterclockwise around the upper half of the unit circle in Fig. 11b from $z = 1$ to $z = -1$. For this movement of z, $s_d = j\omega_d$ increases from zero to $j\omega_s/2$ in Fig. 11c. Thus, as indicated in Fig. 11c, the entire frequency characteristic of the prototype analog filter for all positive values of ω_a is compressed into a segment of the $j\omega_d$ axis between zero and $\omega_s/2$. As indicated by eq. (118), the values of $H_d(j\omega_d)$ in this band vary through exactly the same sequence as the values of $H_a(j\omega_a)$ in the range of ω_a between zero and infinity, but the frequency scale is compressed and distorted. Moreover, the entire frequency characteristic of $H_a(j\omega_a)$ for all frequencies is repeated periodically along the $j\omega_d$ axis, with a warped frequency scale, as indicated in Fig. 11c.

The discussion above shows that when the bilinear transformation is used in the design of a digital filter, there is a close relationship between the frequency characteristics of the prototype analog filter $H_a(j\omega_a)$ and those of the resulting digital filter $H_d(j\omega_d)$. In fact, they are the same except for a warping of the frequency scale. The relationship between the frequency scales can be derived readily from eqs. (125), (126), and (134). Any given value of z lying on the unit circle maps into a point on the $j\omega_a$ axis given by eq. (125) as

$$z = \exp(j2\phi). \tag{139}$$

That same value of z maps into infinitely many points on the $j\omega_d$ axis given by eq. (134) as

$$z = \exp(j\omega_d T \pm 2n\pi), \qquad n = 0, 1, 2, \cdots . \tag{140}$$

Taking the principal value of eq. (140), $n = 0$, and equating it to eq. (139) yields

$$\exp(j\omega_d T) = \exp(j2\phi),$$

or

$$\omega_d T = 2\phi. \tag{141}$$

Substituting eq. (126) into eq. (141) yields

$$\omega_d T = 2 \arctan(\omega_a T/2), \tag{142}$$

and

$$\omega_a = \frac{2}{T} \tan(\omega_d T/2). \tag{143}$$

Using the relation $T = 1/f_s$, where f_s is the sampling frequency, eq. (143) can be rewritten as

$$\omega_a = 2f_s \tan(\omega_d/2f_s).$$

Then,

$$f_a = \frac{f_s}{\pi} \tan\left(\frac{\pi}{f_s} f_d\right) \tag{144}$$

$$= f_d \frac{\tan(\pi f_d/f_s)}{\pi f_d/f_s}. \tag{145}$$

Note that if $f_s \gg f_d$, then

$$f_a \approx f_d.$$

Equation (143), (144), or (145) is used in the design of digital filters when the bilinear transformation is employed. The system designer specifies the desired frequency characteristics in terms of cutoff frequencies, frequencies of attenuation peaks, and the like. These frequencies are critical values of f_d for the digital filter. The critical frequencies are substituted into one of the equations listed above to obtain predistorted critical values of f_a for the prototype analog filter. The analog filter is then designed with these values of f_a, and, because of the predistortion, the digital filter obtained by the bilinear transformation has the desired critical frequencies. Note that every critical frequency contained in the original specifications must be predistorted by one of the equations specified above.

It must also be noted, however, that the bilinear transformation maps the entire frequency characteristic of the prototype analog filter into the range of ω_d lying between zero and $\omega_s/2$. Thus, the designer can control the characteristics of the digital filter only in this range of ω_d, and hence, all of the critical frequencies specified for ω_d must be less than $\omega_s/2$. In this connection, however, it is appropriate to remember that the signal being filtered must be band-limited to this same range of frequencies to avoid distortion as a result of aliasing.

It is clear from eq. (144) that the amount by which the frequency scale is warped depends on the sampling frequency f_s. For a fixed filter design, any change in f_s will cause a shift in the critical frequencies of the digital filter, and if the change is substantial, the result may be an unsatisfactory performance by the filter.

The technique outlined here for the use of the bilinear transformation in the design of digital filters is one possible method that can be used. In addition, several other successful procedures have been developed in detail, and they are described in the literature.[1,3] However, the introductory presentation given here is concerned only with the basic ideas of digital filtering, and hence it makes no attempt to give a complete coverage of all the techniques for designing such filters.

Section 6.2 illustrates that if the digital transfer function $H(z)$ is to be a stable function, then its poles must lie inside the unit circle in the z plane. As we show below, this places a limit on the range of values that the coefficients in $H(z)$ may be permitted to have, and this fact, in turn, may influence the design of the hardware and the supporting software used in realizing digital filters.

As related in Section 4.4, it is a common practice to realize digital filters as a cascade of biquadratic sections used as basic building blocks. The digital transfer function for this building block can be put in the form

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{146}$$

$$= \frac{a_0 z^2 + a_1 z + a_2}{z^2 + b_1 z + b_2}. \tag{147}$$

Note that since the coefficients in $H(z)$ must be real numbers, the poles and zeros of $H(z)$ must either be real or they must occur in complex conjugate pairs. Furthermore, if the two zeros of the denominator are designated z_1 and z_2, then the denominator can be written as

$$D(z) = (z - z_1)(z - z_2) \tag{148}$$

$$= z^2 - (z_1 + z_2)z + z_1 z_2. \tag{149}$$

Comparing this result with the denominator of eq. (147) leads to the following relations:

$$b_1 = -(z_1 + z_2) \qquad \text{and} \qquad b_2 = z_1 z_2. \tag{150}$$

Since for stable digital filters z_1 and z_2 must lie inside the unit circle in the z plane, the following inequalities must be satisfied:

$$-1 < b_2 < 1 \qquad \text{and} \qquad -2 < b_1 < 2. \tag{151}$$

However, it is appropriate to note the fact that even though the inequalities in eq. (151) are satisfied by the coefficients of $H(z)$, this does not guarantee that the filter is stable. The quadratic formula can be used, with a little additional effort, to show that the necessary and sufficient conditions for stability are

$$-1 < b_2 < 1 \qquad \text{and} \qquad -(1 + b_2) < b_1 < (1 + b_2).$$

See Ref. 6 for further details on this subject.

For most digital filters, the zeros of the numerator in eq. (147) are on the unit circle or inside it. In such cases, after the numerator is normalized to make $a_0 = 1$, the same inequalities apply to the coefficients a_1 and a_2. An exception to this rule occurs in the case of delay equalizers (all-pass networks), where the zeros of the numerator must lie outside the unit circle (in the right half of the s_d plane). But even in this case, it is possible to introduce a scale factor to make the numerator coefficients satisfy inequalities like those in eq. (151).

The significance of these facts arises from the additional fact, developed in Section 4.4, that the coefficients in $H(z)$ are also the coefficients in the difference equation that the digital filter must evaluate in order to perform its function. Thus, it follows that the filter hardware and software can be, and sometimes are, designed to work with coefficients limited to the range between 2 and −2.

Example 4.

This example is a simple illustration of the use of the bilinear transformation to obtain a linear difference equation from a given analog transfer function. It is based on the RC filter of Fig. 1, and the results are compared with those obtained in Section II by a different method of analysis. The analog transfer function for this circuit is

$$H_a(s_a) = \frac{1}{1 + RCs_a}. \tag{152}$$

Substituting the bilinear transformation of eq. (121) into eq. (152) yields

$$H(z) = \left(1 + \frac{2RC}{T}\frac{1 - z^{-1}}{1 + z^{-1}}\right)^{-1}. \tag{153}$$

For simplicity, a new symbol is defined as

$$d = \frac{RC}{T} = RCf_s, \tag{154}$$

so that eq. (153) becomes

$$H(z) = \left(1 + 2d\frac{1 - z^{-1}}{1 + z^{-1}}\right)^{-1}.$$

When this equation is rearranged into the standard form, the result is

$$H(z) = \frac{1}{1 + 2d}\,\frac{1 + z^{-1}}{1 + \dfrac{1 - 2d}{1 + 2d}z^{-1}}. \tag{155}$$

This function has a zero at $z = -1$, whereas $H_a(s_a)$ in eq. (152) has no finite zeros. In eq. (155) the zero of $H_a(s_a)$ at infinity has been transformed into a zero of $H(z)$ at $z = -1$, as implied in Fig. 11b by the fact that the point at infinity in the s_a plane maps into the point $z = -1$.

Applying the bilinear transformation to a full biquadratic transfer function involves some tedious algebra, and the tedium is compounded if the required filter consists of several biquadratic sections connected in cascade. Fortunately, computer programs exist that accept the coefficients of the analog transfer function as inputs and deliver the coefficients produced by the bilinear transformation as outputs. (See Ref. 7.)

If the transforms of the input and output voltages in the filter of Fig. 1 are designated, respectively, as $V_1(z)$ and $V_2(z)$, then

$$V_2(z) = H(z)V_1(z), \tag{156}$$

where $H(z)$ is given by eq. (155). To obtain the corresponding difference equation, eq. (155) is substituted into eq. (156), and the result is rearranged to obtain

$$V_2(z) = \frac{1}{1+2d} V_1(z) + \frac{1}{1+2d} z^{-1} V_1(z) - \frac{1-2d}{1+2d} z^{-1} V_2(z). \quad (157)$$

The procedure developed in Section 4.4, eqs. (96) through (101), now yields the desired difference equation,

$$v_2(nT) = \frac{1}{1+2d} v_1(nT) + \frac{1}{1+2d} v_1[(n-1)T] - \frac{1-2d}{1+2d} v_2[(n-1)T]. \quad (158)$$

For comparison, the difference equation obtained by a simpler method in Section II and given by eq. (6) is

$$v_2(nT) = \frac{1}{1+d} v_1(nT) + \frac{d}{1+d} v_2[(n-1)T]. \quad (159)$$

Equations (158) and (159) do not look very much alike. In fact, they do not give similar results except for high sampling frequencies f_s such that $d = RCf_s \gg 1$ and $v_1[(n-1)T] \approx v_1(nT)$. This matter is discussed in the following paragraph.

Curiously enough, the bilinear transformation produces a digital filter having frequency characteristics much closer to those of the prototype analog filter in Fig. 1 than does the seemingly more direct approach used in Section II and represented by eq. (159). The reason for this result can be explained in the following way. The derivation of the difference equation in Section II is equivalent to using the transformation

$$s_a = \frac{1}{T}(1 - z^{-1})$$

in the analog transfer function (eq. 152). Now for the key point: This transformation does not map the $j\omega_a$ axis onto the unit circle in the z plane. Thus, the $j\omega_a$ axis does not map through the z plane onto the $j\omega_d$ axis in the s_d plane, and as a result, the frequency characteristics of the prototype analog filter are not preserved by the transformation. Thus, with this transformation, the digital filter behaves like the analog filter only for high-sampling rates and low-signal frequencies. (See pages 212 through 214 of Ref. 3.)

VII. SUMMARY

In its most usual form the digital filter is a digital machine that performs the filtering process by the numerical evaluation of a linear difference equation in real time under program control.

The z transform, an outgrowth of a special type of Laplace transform, proves to be an especially effective mathematical tool for the analysis and design of digital filters. In fact, the z transform plays a role in the study of linear difference equations that is in many respects comparable to that of the Laplace transform in the study of linear differential equations. In particular, the z transform can be applied to a linear difference equation to obtain an algebraic transfer function. This z-domain transfer function is related to digital filters in the same way that the well-known s-domain transfer function is related to conventional analog filters.

In most cases the design of a digital filter involves determining a digital (z-domain) transfer function having the desired frequency characteristics. This digital transfer function can often be obtained in a straightforward manner from an appropriate analog (s-domain) transfer function. A mathematical relation known as the bilinear transformation proves in many cases to be an especially effective tool for converting an analog transfer function into a useful digital transfer function that can be realized as a digital filter in a straightforward manner.

VIII. ACKNOWLEDGMENTS

The author gratefully acknowledges the assistance of D. L. Favin, P. E. Fleischer, R. N. Gadenz, J. F. Kaiser, and D. C. Stanzione. However, the tutorial aspects of the work necessarily involve many subjective judgments on matters concerning which there are reasonable differences of opinion. As a result, the acknowledgment does not imply complete endorsement of the work by any of the persons named.

REFERENCES

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.
2. A. Peled and B. Liu, *Digital Signal Processing*, New York: John Wiley and Sons, 1976.
3. L. R. Rabiner and B. Gold, *Theory and Applications of Digital Signal Processing*, Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.
4. A. Papoulis, *Signal Analysis*, New York: McGraw-Hill Book Co., 1977.
5. E. G. Phillips, *Functions of a Complex Variable*, New York: Interscience Publishers, Inc., 1949.
6. P. E. Fleischer and K. R. Laker, "A Family of Active Switched Capacitor Biquad Building Blocks", B.S.T.J., *58,* No. 10 (December 1979), pp. 2235–69.
7. Digital Signal Processing Committee, ed., *Programs for Digital Signal Processing*, New York: IEEE Press, 1979.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Adaptive Differential Pulse-Code-Modulation Coding

By J. R. BODDIE, J. D. JOHNSTON, C. A. McGONEGAL, J. W. UPTON, D. A. BERKLEY, R. E. CROCHIERE, and J. L. FLANAGAN

(Manuscript received June 26, 1980)

An adaptive differential pulse-code-modulation technique for encoding and decoding has been implemented using the Bell Laboratories digital signal processor integrated circuit. The encoder/decoder operates in real time and can accommodate 3- or 4-bit (8 kHz) encoding. In this paper, we discuss details of the implementation, the basic algorithm, and the features utilized in the digital signal processor.

I. INTRODUCTION

Adaptive differential pulse-code-modulation (ADPCM) encoding has been shown to be a simple and effective method for digitally encoding speech at bit rates in the range of approximately 24 to 48 kb/s.[1-6] At a rate of 24 kb/s, ADPCM can provide a good quality reproduction of speech that is acceptable for applications such as computer-controlled digital voice response systems.[4] At a rate of 32 kb/s, it can provide essentially a telephone bandwidth "transparent quality" (a quality that is indistinguishable from the original uncoded source) for a single tandem encoding. Adaptive differential pulse-code modulation has been studied for use in some types of transmission systems,[6] and for message storage and retrieval systems in which a reduction by a factor of two in bit rate over that of conventional 64 kb/s μ-law companded PCM is desired.[4]

Various forms of hardware have been suggested for the implementation of ADPCM coders. Some designs are based primarily on analog hardware[1,3] where parameters are pair-wise tuned between transmit-ters and receivers. This results in problems with repeatability and stability in the analog designs. Bates[5] and Adelman, Ching, and Gotz[6] subsequently presented two different methods of designing all-digital ADPCM coders. The Bates approach uses a TTL logic design with a ROM-based look-up table for the adaptive step-size and an up/down counting scheme with digital adders and subtracters to avoid the use of a digital multiplier. The hardware requires a twelve-bit linear PCM input and has a total "package count" of approximately 80 standard TTL logic packages. It was constructed on two wire-wrapped Augat cards (one encoder and one decoder per board). The hardware by Adelman et al., was a ROM-based design that accepted a standard 64-kb/s μ-law companded PCM signal.

Recently, an LSI digital signal processor (DSP) has been developed by Bell Laboratories.[7] The DSP is a programmable processor capable of performing the entire ADPCM algorithm for multiple channels in a single LSI device. The ability of the processor to convert between conventional μ-255 companded PCM and two's complement binary code formats allows the ADPCM algorithm to use either format. The configuration of the ADPCM algorithm that is implemented on the DSP is described in Section II. Section III discusses details of how the algorithm is configured to use DSP features. Sections IV and V describe the hardware configuration and measured performance, respectively.

II. THE ADPCM ALGORITHM

2.1 Overall configuration

Figure 1 illustrates the basic configuration of the DSP implementation of ADPCM. The input analog signal $s(t)$ is sampled and A/D converted to an 8-bit μ-law companded PCM format to produce the sampled data signal $s(n)$, where n is the discrete time index. These operations are done externally to the DSP. Then $s(n)$ is converted in the DSP from μ-law format to a 20-bit linear PCM format for internal processing.

The ADPCM encoding is performed entirely within the transmitter DSP. The output is a 3- or 4-bit codeword, $I(n)$, which can be obtained through the normal output channel of the DSP. In the receiver, a

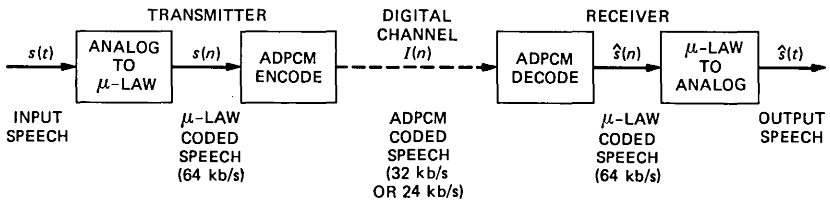


Fig. 1—Overall configuration of the ADPCM encoder/decoder using the DSP.

second DSP is used to decode $I(n)$ back to an 8-bit μ-law format. This signal is then converted to an analog signal with an external D/A converter. The clock and framing signals are carried between transmitter and receiver on separate lines. The transmission protocol for a given application depends on the data channel in which the system is implemented.

2.2 Block diagram of ADPCM

Figure 2 shows the block diagram of the ADPCM algorithm that was implemented on the DSP. The transmitter is shown in Fig. 2a and the receiver is shown in Fig. 2b. We assume that $s(n)$ in Fig. 2 is in a 20-bit linear PCM format, because of a direct 20-bit input to the DSP or because of an 8-bit μ-law to 20-bit linear PCM conversion that has been performed within the DSP.

The ADPCM algorithm can be partitioned into three basic parts (see Fig. 2): the adaptive PCM quantizer, the differential predictor loop, and the adaptive gain (step-size) control for the quantizer. We discuss these operations in the above order since this is the order in which they are computed.

2.3 Adaptive quantizer

A predicted value, $p(n)$, is first subtracted from the input signal $s(n)$ to form the difference signal

$$e(n) = s(n) - p(n). \tag{1}$$

The value of $p(n)$ is obtained from the predictor and is based on computations performed at the previous sample time, $n - 1$.

The difference signal $e(n)$ is then quantized by the adaptive quantizer to produce the ADPCM codeword $I(n)$. This adaptive quantization is achieved by first scaling $e(n)$ to the range of a 3- or 4-bit fixed step-size quantizer (see Fig. 2a). The scaled signal is denoted as

$$e_s(n) = \nabla(n) \cdot e(n), \tag{2}$$

where $\nabla(n)$ is an adaptive scale factor that is also determined from data available at the previous sample time $n - 1$. The combination of scaling by $\nabla(n)$ followed by the fixed step-size quantizer is equivalent to a quantizer with an adaptively varying step-size (which is inversely proportional to $\nabla(n)$).

Figure 3 shows the characteristics for a 3-bit (8 level) fixed step-size quantizer. The input signal $e_s(n)$ (consisting of an integer plus a fractional part) is converted to one of eight quantization levels in the range $-7/2$ to $+7/2$ corresponding respectively to integer codewords, $I(n)$, in the range -4 to $+3$. The output quantized signal is denoted as $\hat{e}_s(n)$ and is related to the code word $I(n)$ by the relation

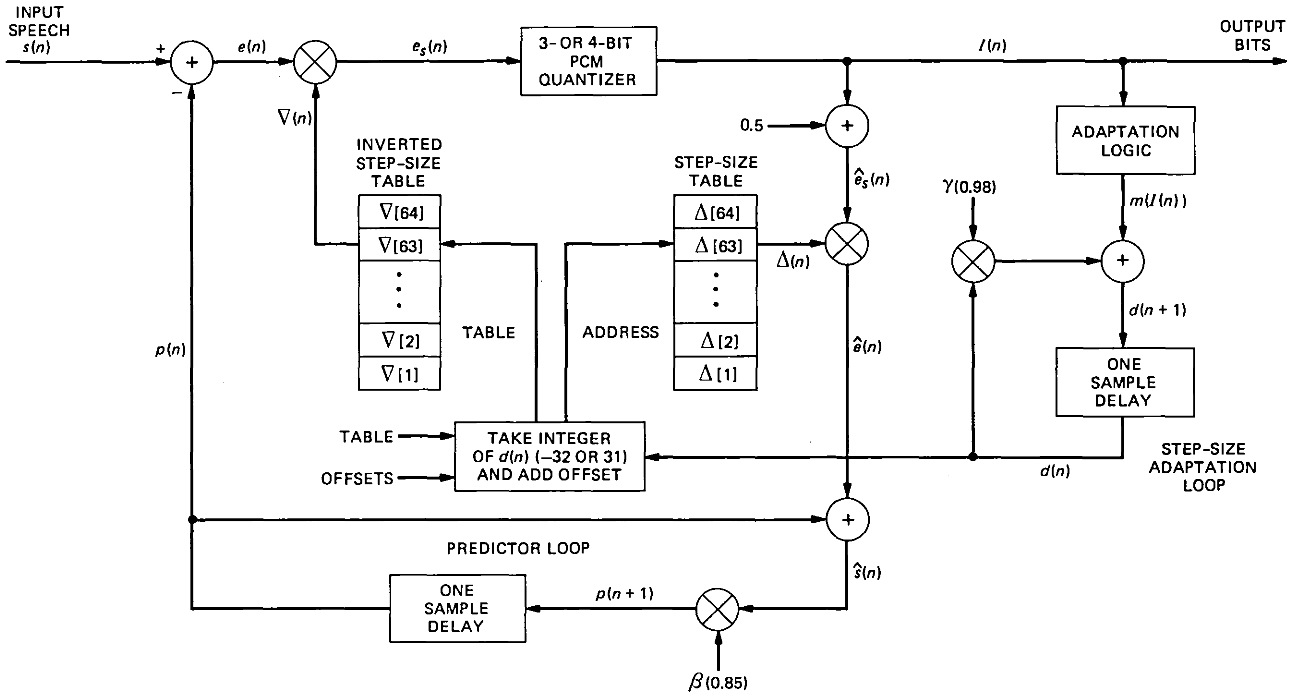


Fig. 2a—Block diagram of the ADPCM encoding algorithm.

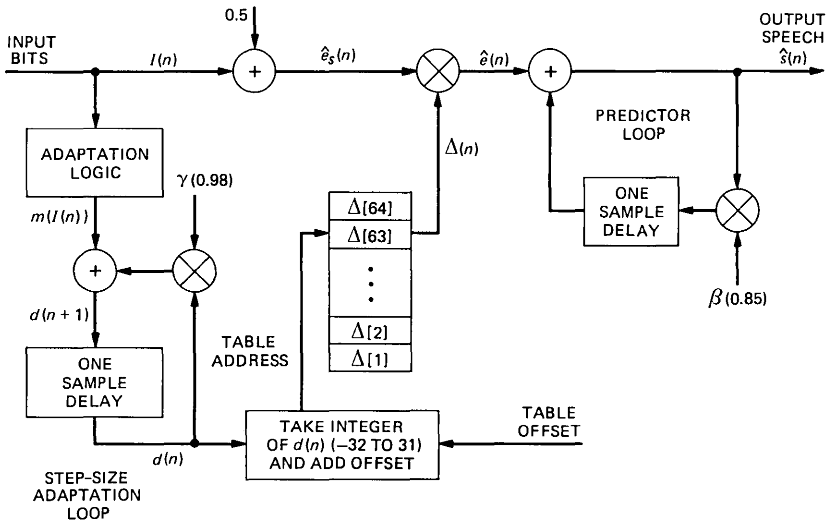


Fig. 2b—Block diagram of the ADPCM decoding algorithm.

$$\hat{e}_s(n) = I(n) + 0.5. \tag{3}$$

A similar quantizer characteristic is used for the 4-bit (16 level) design.

The unscaled, decoded difference signal can be obtained from $\hat{e}_s(n)$ by rescaling it by $\Delta(n)$, where $\Delta(n)$ is inversely related to $\nabla(n)$ and is directly proportional to the "step-size" of the equivalent adaptive quantizer. Thus,

$$\hat{e}(n) = \Delta(n)\cdot\hat{e}_s(n) \tag{4}$$

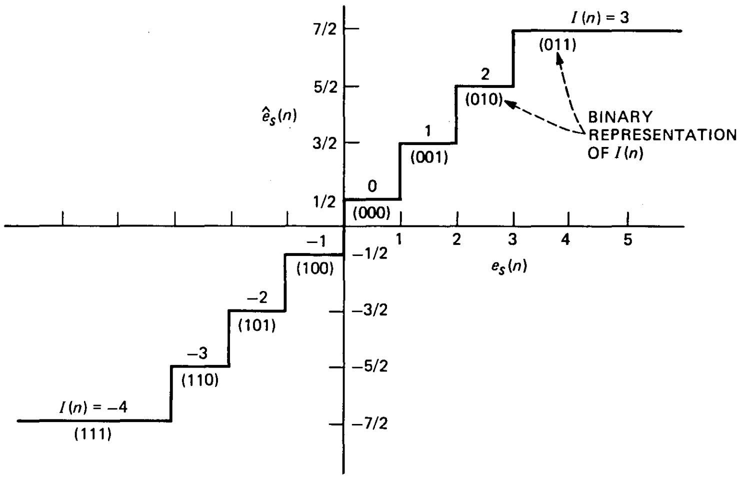


Fig. 3—Quantizer characteristic for the 3-bit PCM quantizer.

is the adaptively quantized representation of the difference signal $e(n)$. This completes the adaptive quantization part of the algorithm.

2.4 First-order predictor

The sum of $p(n)$ and $\hat{e}(n)$ also forms the adaptively quantized representation of the input signal (see Fig. 2a)

$$\hat{s}(n) = p(n) + \hat{e}(n). \tag{5}$$

However, the predictor value $p(n+1)$ and the quantizer scale factors $\nabla(n+1)$ and $\Delta(n+1)$ need to be updated for the next sample time $n+1$.

The new predictor value is computed as β times $\hat{s}(n)$; i.e.,

$$p(n+1) = \beta\cdot\hat{s}(n), \tag{6}$$

where β is the predictor "leak" factor.[2] A value of $\beta = 0.85$ is often used for speech encoding. The value of $p(n+1)$ is then stored for use at the next sample time $n+1$.

2.5 Adaptive step-size control

The computation of the new value of $\Delta(n+1)$ and $\nabla(n+1)$ for the next time sample $n+1$ is more involved. The algorithm used here is based on the robust step-size adaptation approach. The details and advantages of this form of the algorithm are discussed in considerable detail in Refs. 8 to 10. The method is as follows. The step-size referred to above is the effective spacing in the quantizer levels observed in the unscaled quantized signal $\hat{e}(n)$. Since this spacing is proportional to $\Delta(n)$, we will refer to $\Delta(n)$ in the following discussion as the step-size.

In the robust adaptation algorithm, the new step-size $\Delta(n+1)$ is chosen as

$$\Delta(n+1) = (\Delta(n))^{\gamma}\cdot M(I(n)). \tag{7}$$

That is, it is the old step-size $\Delta(n)$ raised to the power γ ($0 < \gamma \le 1$, typically $\gamma = 0.98$) and scaled by a factor $M(\cdot)$ which is a function of the code word $I(n)$. If the code word is one of the upper magnitude levels of the quantizer [e.g., $I(n) = -4, -3, 2,$ or 3 in Fig. 3], a value of $M(\cdot)$ greater than one is used to increase the step-size of the quantizer for the next sample time. If the codeword is one of the lower magnitude levels [e.g., $I(n) = -2, -1, 0, 1$ in Fig. 3], a value of $M(\cdot)$ less than one is used to reduce the quantizer step-size for the next sample time. In this way, the algorithm continually attempts to adapt the step-size such that the dynamic range of the adaptive quantizer is matched to the range of the signal $e(n)$.

A more direct approach for implementing this algorithm is obtained by expressing eq. (7) in logarithmic form. Let

$$d(n) = \log(\Delta(n)), \tag{8}$$

and

$$m(I(n)) = \log(M(I(n))), \tag{9}$$

where the base of the logarithm is determined by the choice of parameters in the coder, as discussed later. Then, eq. (7) becomes

$$d(n+1) = \gamma \cdot d(n) + m(I(n)), \tag{10}$$

and it has the form of a first-order difference equation. This is the equation that is implemented by the upper right-hand part of the block diagram in Fig. 2a.

The driving function $m(I(n))$ is a function of the code word $I(n)$, and it is determined in the adaptation logic algorithm which performs the function

$$m(I(n)) = \begin{cases} 8 \text{ if } |I(n) + 0.5| \geq 2.5, \\ -3 \text{ if } |I(n) + 0.5| < 2.5 \end{cases} \tag{11a}$$

for the 3-bit quantizer and

$$m(I(n)) = \begin{cases} 8 \text{ if } |I(n) + 0.5| \geq 4.5, \\ -3 \text{ if } |I(n) + 0.5| < 4.5 \end{cases} \tag{11b}$$

for the 4-bit quantizer.

The output of the step-size adaptation loop is the signal $d(n)$, which eq. (8) shows as the logarithm of the desired step-size.

Thus, to obtain $\Delta(n)$ (and $\nabla(n)$), we need to implement the relations

$$\Delta(n) = \exp(d(n)) \tag{12a}$$

and

$$\nabla(n) = \frac{1}{\Delta(n)} = \exp(-d(n)), \tag{12b}$$

where, again, the base of the logarithms and exponentials in eqs. (8) to (12) are determined by the choice of parameters of the coder, as discussed later.

The exponentials in eqs. (12a) and (12b) are computed by using look-up tables that are stored in the program ROM as indicated in Fig. 2a. The integer value of $d(n)$ is taken and constrained to the range $-32 \leq [d(n)] < 32$ for a lookup table size of 64. Table offset values are then added to this value to produce the appropriate ROM address locations. The tables are set up so that a value of $d(n) = 0$ points to the center of both tables ($\nabla[32]$ and $\Delta[32]$). Note that the bracketed values $\nabla\,[\cdot]$ and $\Delta\,[\cdot]$ in the tables of Fig. 2a refer to the contents of table locations addressed by $[\cdot]$ and not to sample times (which are referred to by the parenthesis notation $\nabla(n)$ and $\Delta(n)$).

The above algorithm uses 64 different step-size values of $\Delta(n)$ and

$\nabla(n)$. The table values are chosen to span the desired dynamic range of the input signal. If signals of larger or smaller amplitude than this are encountered, the step-size saturates at the upper or lower table values, respectively.

The table address locations are stored for use at the next sample time $n + 1$ to access values $\Delta(n + 1)$ and $\nabla(n + 1)$ as they are needed. This completes the operation of the ADPCM encoder.

2.6 Parameter selection for the step-size tables

The parameters of the step-size table were chosen such that the ratio of the maximum to minimum step-sizes in the table is 256, i.e.,

$$\Delta[64]/\Delta[1] = 256 = 2^8$$

and

$$\nabla[1]/\nabla[64] = 256.$$

This gives a step-size adaptation range of 48 dB (8 bits) that is appropriate for speech coding. Since there are 64 logarithmically spaced step-size values in each table, this corresponds to a 0.75-dB resolution, i.e., step-sizes increase by a ratio of 1.0902 in the table,

$$\Delta[i] = \Delta[1]\cdot(1.0902)^{i-1},$$

$$\nabla[i] = \nabla[1]\cdot(1.0902)^{-i+1}.$$

The maximum step-size is chosen so that the maximum range of $e(n)$ (approximately 17 bits) scales to the maximum quantizer range (3 or 4 bits). This prevents overloading on the high end of the dynamic range. Thus, for a 4-bit quantizer

$$\Delta(n)\,|_{\max} = \Delta[64] = 2^{17-4} = 2^{13}$$

and

$$\Delta(n)\,|_{\min} = \Delta[1] = 2^5.$$

For a 3-bit quantizer, values of $\Delta[\cdot]$ should be increased by two and values of $\nabla[\cdot]$ should be decreased by two for best dynamic range performance.

The manner in which these table values are scaled and stored in the DSP will be explained more fully in Section 3.4.

2.7 Decoder for ADPCM

Figure 2b shows a similar block diagram for the ADPCM decoder. The input code word $I(n)$ is converted to the decoded difference signal $\hat{e}(n)$ by adding 0.5 (see Fig. 3) and scaling the result by $\Delta(n)$. The decoded output signal $\hat{s}(n)$ is then obtained by accumulating values $\hat{e}(n)$ in the predictor loop in the same manner as in the encoder. The new step-size is then computed in an exactly duplicate manner to that in the encoder. Thus, the ADPCM decoder duplicates a subset of the encoder block diagram.

III. PROGRAMMING TECHNIQUES UTILIZED IN THE DSP FOR THE ADPCM ALGORITHM

The ADPCM algorithm, as it is configured above, is designed so that it can be conveniently implemented on the DSP. We have already alluded to ways in which the DSP is used in implementing the algorithm. In this section, we point out some additional aspects of the program and discuss how some of the unique features of the DSP are used. Discussions in this section use the 4-bit algorithm as an example.

3.1 Memory utilization

The encoder program and the step-size tables for $\Delta[\cdot]$ and $\nabla[\cdot]$ are stored in ROM occupying approximately 170 words of memory. Five RAM locations are used for storing the following values: $2(I(n) + 0.5)$, $2\,\hat{s}(n)$, $p(n)$, $d(n)$ and the integer version of $d(n)$. Access to the step-size tables is made by setting the RX register in the DSP to the center address of the desired table. The table address for the appropriate step-size is then obtained by setting the K register to the integer value $[d(n)]$ (limited to the range -32 to 31) and then incrementing RX by the value K using the rxk command. The RX register then contains the ROM address for the appropriate step-size $\nabla[\cdot]$. The DSP instructions that implement this technique are as follows:

```
rya = 5;              "RAM pointer to int [d(n)]"
 rx = &TABLE;         "set rx pointer to ∇[32]"
  k = rym;;;          "set k to int [d(n)]"
  a = p p = rxk*c;    "pointer to appropriate ∇[·]"
```

Note that this technique only works for table sizes up to 256, since the K register is limited to 8 bits.

3.2 Quantization

The computation of the quantizer input, $e_s(n)$, is accomplished in a straightforward manner on the DSP by using a subtraction and a multiplication. The conversion from $e_s(n)$ to a 4-bit integer $I(n)$ (according to the quantizer characteristic in Fig. 3) is done by truncating the fractional part of $e_s(n)$.

Assuming $e(n)$ has been computed and stored in the W register and RX is pointing to the appropriate step size, the following DSP instructions compute $I'(n)$.

```
                  p = 1*c;
a = p             p = rxj*w;       "compute $e_s(n) = \nabla(n) \cdot e(n)$"
a = p + a/2       p = −1*c;        "compute $e_s(n) + 0.5$"
a = p & a;                         "truncate to form $I'(n)$"
```

The truncation is achieved by using the accumulator control statement a = p & a, which performs a bit-by-bit AND operation between the P and A registers in the DSP. $I'(n)$ is stored in the A register and the (binary) number 1111 ⋯ 111.000 ⋯ is stored in the P register by the instruction p = −1*c ($c = 2^{14}$). The operation zeros out the fractional part of $(e_s(n) + 0.5)$ and leaves the integer part untouched. The resulting integer $I'(n)$ is then constrained to the range $-8 \leq I'(n) \leq 7$ (for 4-bit quantization) to form the desired codeword $I(n)$. This is done using the conditional AU operation as shown in the following instructions.

```
                      p = 8*c;
    a = p + a         p = 0*c;
if (a < 0) doau ();   a = p;        "if ($I'(n) < -8$) $I'(n) = -8$"
    a = p + a         p = −15*c;
    a = p + a         p = 0*c;
if (a > 0) doau ();   a = p;        "if ($I'(n) > 7$) $I'(n) = 7$"
    a = p + a         p = 7*c;
    a = p + a                       "4-bit $I(n)$"
```

This yields the 4-bit value $I(n)$ in the accumulator.

3.3 Internal scaling of data

At the output of the quantizer a value of 0.5 is added to $I(n)$ (see Fig. 2) to produce $\hat{e}_s(n)$, which is then multipled by $\Delta(n)$. To accomplish this, $\hat{e}_s(n)$ must be transferred from the A register to the W register and the input of the multiplier. Since this transfer from the A to W registers truncates the fractional part of the value in the A register, it is first scaled by a factor of 2 to avoid the truncation of the fractional part. This scale factor of two is carried through the computation of $\hat{s}(n)$ in the block diagram of Fig. 2.

3.4 Scaling of step-size table values

The values stored in the step-size tables in ROM are more conveniently handled if they are scaled to be less than 2 in magnitude. The internal DSP arithmetic is such that 16-bit values from ROM are assumed to have 14 fractional bits. This is appropriate for the inverse step-size table $\nabla[\cdot]$ in which numbers $(e(n))$ in a 17-bit range are scaled down to a 4-bit range $(e_s(n))$. This table takes on values:

$$\begin{aligned}
\nabla[1] &= 0.031250 \\
\nabla[2] &= 0.028617 \\
&\vdots \\
\nabla[63] &= 0.000133 \\
\nabla[64] &= 0.000122
\end{aligned}$$

which can be directly stored in the ROM.

In the step-size table $\Delta[\cdot]$, the inverse of these numbers must be stored. To accomplish this, the inverses are first scaled by a factor of 2^{-14} and, thus, they take on the fractional range (denoted by primes):

$$\begin{aligned}
\Delta'[1] &= 0.001953 \\
\Delta'[2] &= 0.002133 \\
&\vdots \\
\Delta'[63] &= 0.457650 \\
\Delta'[64] &= 0.499754
\end{aligned}$$

When these table values are used in the multiplication, the resulting product $\Delta'(n)\hat{e}_s(n)$ is scaled back up by a factor 2^{14} using the 14-bit shift option $a = a \ll 14$.

The inverse relationship between the table values requires that

$$\nabla[i] \cdot \Delta'[i] \cdot 2^{14} = 1, \quad i = 1, 2, \cdots, 64. \tag{14}$$

Since $\nabla[i]$ and $\Delta'[i]$ must be quantized to 16-bit numbers (14-bit fractions), for storage in the ROM, the condition in eq. (14) cannot be met exactly. To obtain the greatest accuracy in representing these numbers, the smaller of the two values $\nabla[i]$ or $\Delta'[i]$ for each value of i is quantized first. The reciprocal of this number is then computed (with floating point accuracy) and scaled by 2^{-14}. This value is then quantized to the 16-bit range of the ROM to produce the inverse table value. Thus, the accuracy of the condition in eq. (14) is maintained as closely as possible.

3.5 Control of the address range for the tables

Another unique part of the program involves the control of the range of the table address pointer $d(n)$ to the range −32 to 31 (excluding the table offset). This is accomplished with the aid of the overflow protection feature of the DSP when data is transferred from the A register to the W register. The operations in the step-size control loop are scaled so that $d(n)$ is computed in the upper range of the A register. When this number is transferred to the W register, it is automatically limited to the proper range by the overflow protection option in the DSP. Scaling this number back down to a 6-bit integer range gives the desired range of −32 to 31 for the table location (excluding the table offset).

Specifically, this is accomplished as follows. First the driving function

$$m(I(n)) = \begin{cases} 8\cdot 2^8 \text{ if } |2(I(n) + 0.5)| \geq 9, \\ -3\cdot 2^8 \text{ if } |2(I(\text{n}) + 0.5)| < 9 \end{cases}$$

is computed and stored in the A register. The value $m(I(n))$ is then multiplied by 8 twice and added to $\gamma\cdot d(n)$ to form $d(n + 1)$ scaled by 2^{14}. The value $d(n + 1)$ is limited when it is transferred to the W register, and the table look-up offset, int $(d(n))$, is obtained by multiplying the W register by 2^{10}. This result is directly loaded into the K register as discussed in Section 3.1. Assuming the A register contains the absolute value of 2 $(I(n) + 0.5) - 9$, and the RY register points to $d(n)$, the following DSP instructions compute $d(n + 1)$ and the table offset value, integer $d(n)$.

```
                                          p = 04000*c;
                    if (a < 0) doau ();                       "if (a ≥ 0) m = 8*2^8"
                    a = p                 p = 0176400*c;      "else m = -3*2^8"
                    a = p                 p = 0*c;
                    a = p + 8*a           p = .98*rym;        "compute γ*d(n)"
                    a = p + 8*a           p = 0*c;            "compute d(n + 1)*2^14"
          w = a;                                              "overflow protection"
rdp = w             a = p                 p = 16*w;           "save d(n + 1)"
                    a = p;
          w = a;
rdp = w;                                                      "save int(d(n))"
```

IV. HARDWARE CONFIGURATION

The hardware used to implement the algorithm described in this paper consists of two 16.5- by 11.5-cm wire wrap cards, one for the transmitter and one for the receiver. Both cards are of identical construction with jumper plugs used to determine whether the card will act as a transmitter or a receiver in the configuration shown in Fig. 1.

Figure 4 shows a more detailed block diagram of the cards. Each card contains a DSP, along with a 2048-word by 16-bit PROM holding the program and step-size tables. (For permanent applications the ROM would be integral to the DSP). In addition, there are analog filters, μ-law encoder and decoder chips, clock generators, and synchronization logic.

The ADPCM transmitter card accepts an analog input which is applied to a buffer amplifier and then to a low-pass filter and a μ-law encoder. The sampling rate of this encoder is determined by the repetition period of the sync signal produced by the sync generator. The serial output of the encoder is shifted out by the clock signal. These three

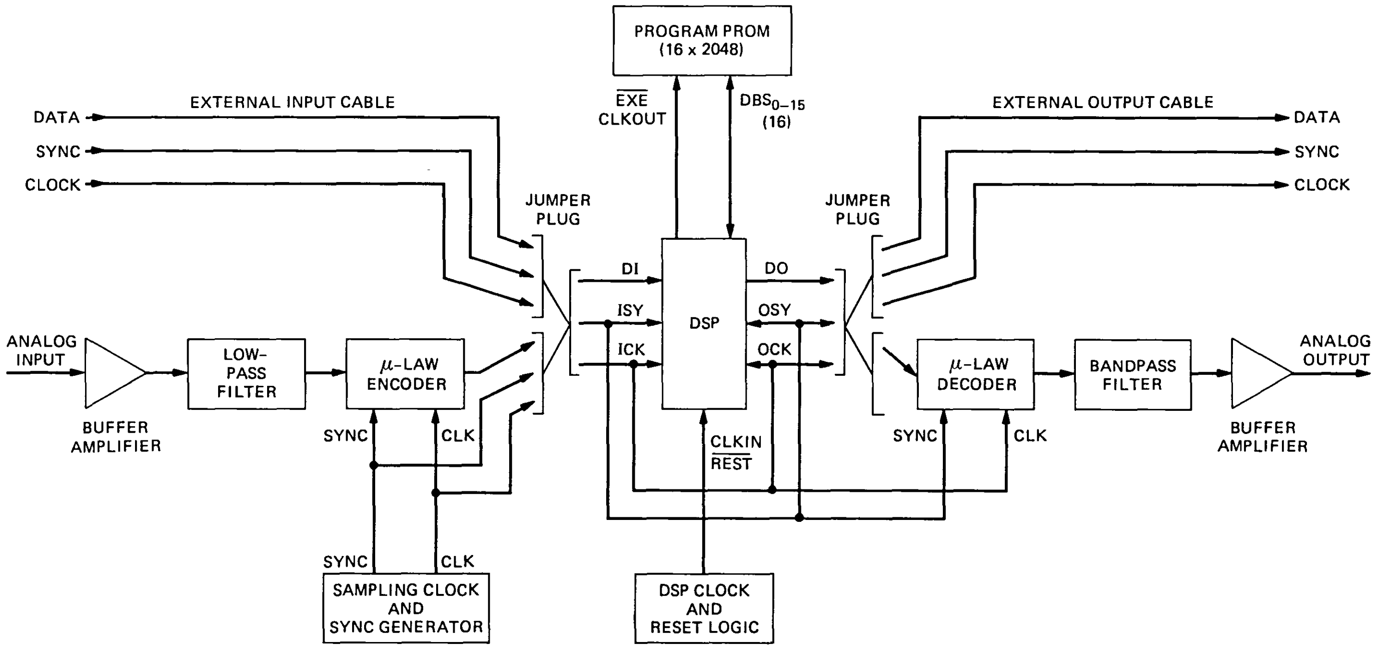


Fig. 4—Hardware configuration for DSP.

signals (data, sync, and clock) are connected through a jumper plug to the data in (DI), input sync (ISY), and input clock (ICK) pins of the DSP. Operating from the encoding program stored in the PROM, the DSP converts the incoming data into ADPCM code words which become available at the data output pin (DO). The shifting of this word out of the DSP is controlled by output sync (OSY) and output clock (OCK) lines. These three signals are connected through another jumper plug to the external output cable which sends them to the receiver card. The μ-law decoder on the transmitter card is not used.

The ADPCM receiver card receives the data, sync, and the clock signals from the transmitter card over the external input cable, which is connected through the first jumper plug directly into the DSP. The μ-law encoder and sync generator on this card are bypassed. The DSP takes the incoming ADPCM code words and converts them to μ-law PCM, according to the instructions of the decoding algorithm in its PROM. The μ-law data words are shifted out of the DSP on the DO line and are connected through the other jumper plug to the input of the μ-law decoder chip. The analog output of the decoder is bandpass filtered and then sent through a buffer amplifier to produce the reconstructed analog output signal.

The cards used here are not limited to only one application of the DSP. By setting the jumper plugs so that audio input and output are both on the same card, different types of filtering algorithms can be tested. Conversely, both the input and output of a card can be connected over the external cables. In this manner, several DSPs can be connected together serially for more complex signal processing.

V. PERFORMANCE

Figure 5 shows the signal-to-noise ratio measured for the 4-bit ADPCM/μ-law coder (Fig. 1) as a function of frequency for input signal

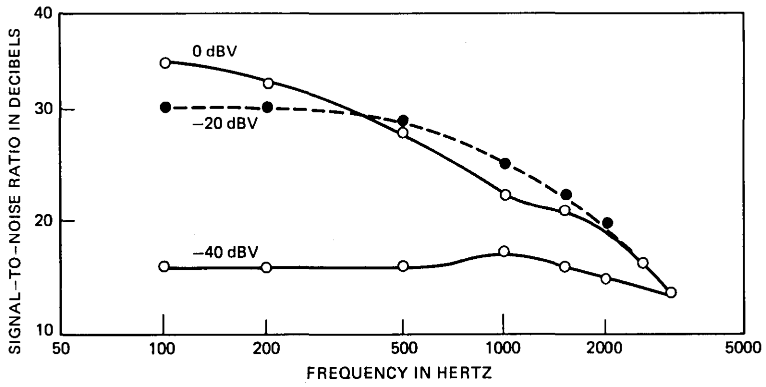


Fig. 5—Signal-to-noise ratio performance of the 4-bit (32 kb/s) ADPCM/μ-law coding configuration of Fig. 1.

levels of 0, −20, and −40 dB of full scale. It corresponds closely to that of the coders in Ref. 3. The dynamic range is limited to about 48 dB because of the range of the step-size tables and the μ-law encoders. This dynamic range performance can be modified (within the limitations of the μ-law encoders) by changing the step-size tables. At low input levels the significant noise is that of the μ-law encoders.

The subjective quality of the 4-bit ADPCM is very similar to 200- to 3200-Hz (telephone band) filtered speech without encoding. This suggests that a single ADPCM link can provide essentially a "transparent" quality for telephone bandwidth speech.

VI. CONCLUSIONS

In this paper, we have discussed the implementation of the ADPCM algorithm on the DSP. The encoder program uses 26 percent of the 8-kHz real-time capability of the DSP running with a 5-MHz clock. It uses 4 percent of the RAM and 17 percent of the ROM. The decoder program uses 22 percent of the real-time capabilities, 4 percent of the RAM and 10 percent of the ROM. This suggests that 4 encoders, 4 decoders, or 2 encoder-decoders could be implemented on a single DSP. The program uses a number of unique features of the DSP to achieve an efficient implementation of the algorithm, and it demonstrates the flexibility of the DSP in doing small-to-medium scale algorithms of this type.

REFERENCES

1. P. Cummiskey, N. S. Jayant, and J. L. Flanagan, "Adaptive Quantization in Differential PCM Coding of Speech," B.S.T.J., *52,* No. 7 (September 1973), pp. 1105–18.
2. N. S. Jayant, "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," Proc. IEEE, *62* (May 1974), pp. 611–32.
3. J. D. Johnston and D. J. Goodman, "Multipurpose Hardware for Digital Coding of Audio Signals," IEEE Trans. Commun., *COM-26,* No. 11 (November 1978), pp. 1785–8.
4. L. H. Rosenthal et al., "A Multiline Computer Voice Response System Utilizing ADPCM Coded Speech," IEEE Trans. Acoust., Speech, Sig. Proc., *ASSP-22,* No. 5 (October 1974), pp. 339–52.
5. S. Bates, "A Hardware Realization of a PCM-ADPCM Code Converter," S. M. Dissertation, Dept. of Electrical Engineering, Massachusetts Institute of Technology, 1976.
6. H. W. Adelmann, Y. C. Ching, and B. Gotz, "An ADPCM Approach to Reduce the Bit Rate of μ-Law Encoded Speech," B.S.T.J., *58,* No. 7 (September 1979), pp. 1659–71.
7. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
8. D. J. Goodman and R. M. Wilkinson, "A Robust Adaptive Quantizer," IEEE Trans. Commun., *COM-23* (November 1975), pp. 1362–5.
9. D. Mitra, "An Almost Linear Relationship Between the Step-Size Behavior and the Input Signal Intensity in Robust Adaptive Quantization," IEEE Trans. Commun., *COM-27* (March 1979), pp. 623–9.
10. J. D. Johnston and R. E. Crochiere, "An All Digital "Commentary Grade" Subband Coder," J. Audio Eng. Soc., *27,* No. 11 (November 1979), pp. 855–65.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Private Communications

By C. A. McGONEGAL, D. A. BERKLEY, and N. S. JAYANT

(Manuscript received August 2, 1980)

Where normal safeguards for message privacy are not adequate, some form of encryption is required. Voice messages, encoded using an adaptive differential pulse-code-modulation encoder such as that described in a companion paper, may be encrypted for privacy (protection against casual eavesdropping) through similar digital signal processor programs with little additional computation. Two methods of implementation are described: The use of U-permutations for temporal scrambling of the transmitted bit stream and the use of bit-masking by stored random numbers. The relative merits of each system are discussed, illustrating both the flexibility and limitations of the digital signal processor for such applications.

I. INTRODUCTION

Situations occur in everyday telephone communication systems where the normal safeguards for message privacy may not be adequate. This could happen, for example, in a radiotelephone system where message contents could be easily intercepted by unauthorized listeners.

In this paper, we discuss two simple methods for ensuring short-term privacy for such telephone systems. These methods are based on the Adaptive Differential Pulse Code Modulation (ADPCM) codec described in a companion paper.[1] Both methods modify the ADPCM transmitted code word in such a way as to randomize the bit pattern. Decoding the resulting randomized code words requires advance knowledge of the randomization key.

The techniques discussed here are non-time-varying and have limited numbers of encryption keys. Thus, the message is only protected from casual listeners. Listeners who possess the necessary equipment can determine the required decoding key. However, the system is designed so that decoding cannot be accomplished in the duration of an average conversation.

Both these methods have been implemented using the Bell Laboratories Digital Signal Processing integrated circuit (DSP)[2] with only a slight increase in processing load relative to that required by non-encrypted ADPCM encoding and decoding. The resulting system should be able to support two or three simultaneous coders or decoders per integrated circuit.

Issues of key distribution and cryptanalysis are outside the scope of this paper.[3] Our purpose, rather, is to demonstrate what can be realized in terms of adapting an existing ADPCM DSP program for a potential privacy application. With ROM and RAM capabilities greater than what are available in the present DSP, levels of privacy can be straightforwardly enhanced—for example, by layering several permutation and masking operations, as in the Digital Encryption Standard.[3]

In the following section, we discuss privacy algorithms. Then, we consider the implementation of each using the DSP. Finally, we discuss the relative merits of each system. This discussion illustrates both the flexibility and limitations of the DSP for such an application.

II. PRIVACY CODING FOR ADPCM TRANSMISSION

There are three major requirements for a simple digital privacy system:

(*i*) It must be possible to generate a "large" number of encryption keys (bit rearrangement or masking patterns) automatically and easily.

(*ii*) The encrypted speech must be unintelligible if decoded by other than the proper key.

(*iii*) The system must be sufficiently tractable so as to be implemented without significant incremental cost.

The two encryption methods we will consider are as follows:

(*i*) U-permutations[4] where the bits in a given block of ADPCM code words are permuted from their normal order, producing a temporal scrambling of the bit stream.

(*ii*) Addition of stored pseudo-random numbers to the ADPCM code words to form randomly "masked" encrypted code words.

Other methods, such as use of linear congruential random number generation[5,6] to form masks, are possible, but methods (*i*) and (*ii*) above are both practical and illustrate the principles of encryption for privacy.

2.1 U-permutations

The class of uniform (or U-permutations on N bits is defined by[4]

$$s = k_1 r (\text{mod } N);\ r,s = 1, 2, \cdots, N, \tag{1}$$

where r is the initial bit position and s is the scrambled bit position in the block of N bits. The encryption key is k_1 and must be prime to N. Unscrambling of the N-bit block is accomplished by another U-permutation

$$r = k_2 s (\text{mod } N), \tag{2}$$

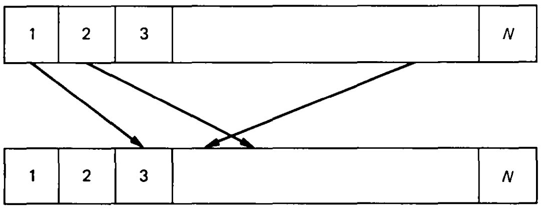
with $k_1 k_2(\text{mod } N) = 1$. Figure 1a illustrates a permutation of bits within a block of N bits, while Fig. 1b shows an example of uniform permutation for $N = 16$, $k_1 = 3$ and $k_2 = 11$.

It has been found that to satisfy the requirement for unintelligibility requires at least $N = 16$ for 24-kb/s ADPCM (8-kHz sampling, 3 bits per sample).[4] We have implemented an $N = 16$ system using 32-kb/s ADPCM (8-kHz sampling and 4 bits/sample to provide telephone quality speech). The scrambled speech is of very low intelligibility with casual listening. However, individual words from a limited vocabulary, such as spoken numbers, may be distinguishable, especially with experienced listening. An implementation with higher N faces some difficulties because of address space limitations. With the current DSP version $N > 32$ is impossible, as will be discussed later.

The number of keys in U-permutation is given by $N \cdot G(N)$, where $G(N)$ is the number of numbers that are prime to N.[4] There are 112 keys available for $N = 16$. For $N = 32$ this increases to 480 keys. The adequacy of a given number of keys depends on the application.

2.2 Random number masking

The random mask method we have considered is basically very simple. In essence, a different random number is added to each ADPCM code word before transmission and that same number is subtracted by



(a)

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s	3	6	9	12	15	2	5	8	11	14	1	4	7	10	13	16

$N = 16 \quad k_1 = 3 \quad k_2 = 11$

(b)

Fig. 1—Example of (a) general bit permutation in a N-bit block and (b) uniform permutation with $N = 16$, $k_1 = 3$, and $k_2 = 11$.

the receiver. A finite table of random numbers is used and synchronization is required between transmitter and receiver.

The encryption key for any given transmission is the starting point in the random number table relative to the block synchronization. Additional keys may be produced by generating the random number mask using multiple table pointers and adding together the random numbers to form the code word mask. Using two pointers, the masking of ADPCM code word C with an L-number random table (4 bits per entry so $L = N/4$) can be written as

$$E = C + R(I + O_1) + R(I + O_2), \tag{3}$$

where $R(I)$ denotes the I-th word of the random table, E is the encrypted code word, $O_{1,2}$ are the table offsets relative to the beginning of the table and $I = 0$ at beginning of a transmission block (synchronization time). Decryption is accomplished by subtracting the same set of random numbers from E. The key words are $O_{1,2}$, leading to a maximum of L^2 possible keys of which $L(L - 1)/2$ are unique. Even for $L = 16$ the encrypted speech is essentially unintelligible and for larger L, the presence of speech is very hard to detect. (The output sounds like continuous white noise at all times.) The table size is limited by DSP ROM size of 1024 words, but $L = 512$ words is certainly practical. Two table pointers then give about 130,000 keys.

III. IMPLEMENTATION USING THE DSP

The basis for both DSP privacy implementations is the ADPCM codec discussed in the companion paper.[1] In both encryption systems the coder or decoder is recast slightly in "subroutine" form which allows more convenient handling of the block synchronization structure. Also, to avoid the problem of two's-complement sign extension, the ADPCM code word is converted to unsigned form; that is, the 4-bit code word, represented as −7 to 8 in the original coder, is offset by 7 and coded as 0 to 15.

3.1 U-permutations

The U-permutation for $N = 16$ is implemented by splitting blocks of four code words (4 bits each) into blocks of 16 one-bit words, rearranging the one-bit words according to the proper permutation and reassembling the permuted block of four words for transmission. One block of four words is being permuted, while a second block is being sent allowing a very simple program organization.

The required modulo N arithmetic is accomplished without any computation being required by overflowing the address register being used as the storage pointer. Thus, for $N = 16$, the disassembled one-bit words are stored at every fourth RAM location. The proper permutation increment is stored in the upper 8 bits of the increment register and the address sequence generated by overflow.

Assuming that the RYA register is pointed to the unsigned form of the ADPCM code word, the following DSP instructions disassemble the code word and store the resulting one-bit words in RAM. The code word is disassembled by loading the bit to be saved in bit 14 of the P register and zeroing all other P register bits. The C register is set to 2^{14}. (Notation for the DSP assembly language is given in a companion paper.[2])

```
                          i = 030;                           "permutation in-
                                                               crement"
                          rd = 0;                            "RAM storage ad-
                                                               dress"
                          a = p      p = 2048*ryz;           "get bit 1 of code
                                                               word"
                          a = p      p = 1*c;                "set up to zero
                                                               other bits"
                          a = p&a    p = 4096*ryz;           "zero other bits;
                                                               get bit 2"
                 w = a    a = p      p = 1*c;                "set up to zero
                                                               other bits"
rdi = w                   a = p&a    p = 8192*ryz;           "save bit 1;
                                                               get bit 3"
                 w = a    a = p      p = 1*c;                "set up to zero
                                                               other bits"
rdi = w                   a = p&a    p = 16384*ryz;          "save bit 2;
                                                               get bit 4"
                 w = a    a = p      p = 1*c;                "set up to zero
                                                               other bits"
rdi = w                   a = p&a;                           "save bit 3; zero
                                                               other bits"
                 w = a;
rdi = w;                                                     "save bit 4"
```

The single bits are reassembled into a code word by shifting and adding the bits using the following instructions:

```
k = 010;                                  "permutation increment"
ry = 160;;;                               "RAM storage address"
a = p              p = 1.*ryk;            "get bit 4"
a = p              p = 1.*ryk;            "get bit 3"
a = p + 2*a        p = 1.*ryk;            "shift and add; get bit 2"
a = p + 2*a        p = 1.*ryk;            "shift and add; get bit 1"
a = p + 2*a;                              "code word reassembled"
```

In this scheme, the RAM values are refreshed at a 500-μs period (for an 8-kHz sampling rate) which is the maximum specified refresh time for the dynamic RAM.[2] To extend the method to $N = 32$, it is necessary to use spare program cycles (of which a sufficient number appears to be available) to supplement the "natural" RAM refresh cycles.

Permutation blocks larger than $N = 32$ bits are not possible using this approach since $2N$ words of RAM are required. Thus, $N = 64$ would fill the 128-word RAM on the DSP allowing no scratch storage as required by the basic ADPCM coder.

The decoder implementation is very similar and has identical limitations. A single DSP μ-law to μ-law codec, using $N = 16$, was implemented successfully and the resulting speech was quite well scrambled although, as mentioned before, some numbers could be distinguished with practice.

The limited number of keys in an $N = 16$ U-permutation system could present a problem in some applications. To increase the number of keys, the easiest route appears to be random number masking, which is discussed next.

3.2 Random number masking

Random 4-bit numbers are stored, 4 bits per word, in a ROM table. The table size is limited only by available ROM. We arbitrarily used a 256-word table for our implementation, but considerably more space is available and can be used if more keys are desired.

In single-pointer masking a pointer into the table is arbitrarily chosen. The DSP automatic (6-bit) loop counter is set to 63 and for each ADPCM code word generated, a random number is fetched, added to the code word, and the pointer incremented. When the loop count is satisfied the pointer is restored to its original value.

If multiple pointers are used, offset values are initialized and each random number is fetched and added to the code word. To avoid additional programs steps for detection of the table end, the pointers are limited to occur no later than 64 locations from the end of the table. The two-pointer version has the following requirement

$$S + \delta + 64 < N, \tag{4}$$

where S is the starting pointer, δ is the offset from S, and N is the mask table size.

All additions are made without attention to overflow out of the 4-bit code word and the least-significant four bits are transmitted. The received word then has identical masks subtracted, without regard to unsigned underflow, and the four least-significant bits are taken as the input to the ADPCM coder. In two's-complement arithmetic the final result is correct, without regard to overflow or underflow, if that result is in the required range. (The decrypted code word must satisfy this condition since the original unsigned word was in the 4-bit range.) An example of masking by this process is shown in Fig. 2.

Assuming the RX register contains the mask pointer, the K register contains the offset value and the RYA register contains a pointer to the unsigned form of the ADPCM code word, the following DSP instructions encrypt the code word:

```
a = p           p = rxk*c;        "get mask word 1"
a = p           p = rxk*c;        "get mask word 2"
a = p + a       p = 1.*w;         "add masks; get code word"
a = p + a       p = 017*c;        "subtract mask from code word"
a = p&a;                          "decrypted 4-bit code word"
```

The code word is decrypted by the following instructions. The RX and K registers contain the table pointer and offset value, respectively. The W register contains the encrypted code word.

```
a = p            p = rxk*c;          "get mask word 1"
a = p            p = rxk*c;          "get mask word 2"
a = p + a        p = 1.*rym;         "add masks; get code word"
a = p − a        p = 017*c;          "add code word"
a = p&a;                             "encrypted 4-bit code word"
```

The two-pointer encrypted codec was implemented on a single DSP with μ-law input and output. Table sizes as small as 16 words, with a single pointer, yield unintelligible scrambled speech.

To examine the synchronization properties of the system, the same

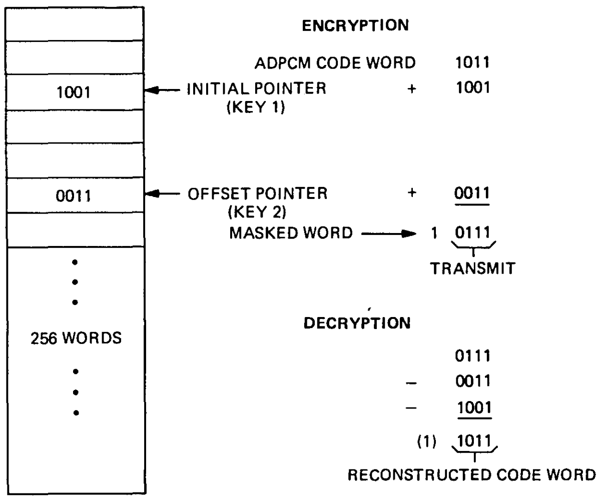


Fig. 2—Example of bit-masking with two-pointer random number encryption and decryption.

codec was also implemented in a two-DSP version. A simplified block diagram is shown for this system in Fig. 3. Digital signal processor 1 provides the encryption and transmits the encrypted bits; DSP 2 performs the decryption. In the absence of appropriate synchronization mentioned below, the output of the receiver digital-to-analog converter is scrambled. This is the same configuration used for the two-DSP codec,[1] except that provision is made for block synchronization. The status and control bits, C0 and S0, provide a "sync" bit for this purpose and, assuming synchronization is recovered externally from the transmission format, are connected separately in parallel with the main serial data-bit stream. Synchronization formatting and recovery is probably also possible within the DSP, but it is beyond the scope of this paper.

Programming the synchronization is very simple. Each time the table pointer is reinitialized the transmitter sends the control signal (S0) using the STR register, and the receiver waits for the control signal (C0) using the SYC register. Digital signal processor instructions for the transmitter and receiver are given below.

```
transmitter
      init:     . . .
                . . .
                lc = 63;          "set up loop counter"
                str = 1;          "send control signal"
                auc = 0x06;       "set c = 2^14, overflow"
                str = 0;          "turn off control signal"
     loop:      . . .
receiver
      init:     . . .
                lc = 63;          "set up loop counter"
                auc = 0x06;       "set c = 2^14, overflow"
                syc = 4           "wait for control signal"
     loop:      . . .
```

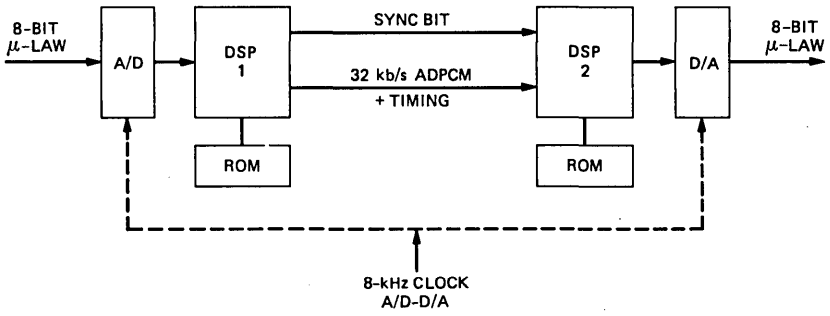


Fig. 3—Simplified block diagram of two-DSP ADPCM codec with block synchronization.

Table I—Memory utilization

Coder Type	Instructions per Sample	Memory ROM	Memory RAM
ADPCM encoder	46	228	5
U-permutation encoder $N = 16$	57	308	37
Random mask encoder $L = 256$	68	552	6
ADPCM decoder	38	156	5
U-permutation decoder $N = 16$	42	214	37
Random mask decoder $L = 256$	55	452	6

As expected, synchronization time is imperceptible and the system sounds exactly the same as does the single DSP codec.

IV. DISCUSSION

Both encrypted codecs provide adequate scrambling in terms of reduced intelligibility, although random number masking is capable of entirely destroying the impression of speech.

Program efficiency, in terms of instructions executed per 125-μs sample, is also similar as shown in Table I. Utilization of the DSP, relative to its maximum execution rate of 156 instructions per 125-s sample, ranges from 24 percent for the ADPCM decoder to 44 percent for the random mask encoder.

Table I also shows memory utilization for the different implementations. (These should be compared to 128 words of available RAM and 1024 words of available ROM.) The ADPCM codec is in subroutine form and savings of about eight instructions can be made by removing this structure at the expense of a considerably more opaque program.

The number of keys required for this type of privacy system has not been studied and other issues, such as transmission of keys, are outside the area of this paper. Clearly, in this implementation random number masking provides for more keys than U-permutations. Also, if greater levels of secrecy are required on a particular transmission link, one can envision a special transmitter/receiver pair with a unique PROM or ROM random number table used externally, thereby achieving a 2^{4k} key system. (This is because there are 2^M binary sequences of length M.) For any single transmission, i.e. a single set of pointer positions, k = 64. However, if one considers other pointer positions this number is greater and is, in general, a function of the random number table length L.

Although the programming was not discussed, the setting of the particular key to be used in a given transmission would require, for example, reading an external switch register during program initialization. Thus, one would need some simple external circuitry to divert the codec input stream at initialization (reset) time and appropriate

DSP programming to handle the input format and store the result in RAM.

V. CONCLUSION

Two privacy encryption systems, based on ADPCM coding of speech, have been discussed. Using the DSP we have implemented both with modest increases in processor load. The U-permutation method makes heavy use of RAM and has limited numbers of encryption keys. Random number masking makes heavy use of ROM and can provide large numbers of keys. Both systems reduce speech intelligibility and could form the basis of an effective privacy system.

VI. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of their associates in the Bell Laboratories Acoustic Research Department, especially that of J. Johnston, in the development of the modulo arithmetic concepts; J. Upton in modification of the DSP hardware for block synchronization; and J. L. Flanagan for his support, encouragement, and discussion of the use of specially prepared ROM for secrecy.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Adaptive Differential Pulse-Code-Modulation Coding," B.S.T.J., this issue.
2. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
3. W. Diffie and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," Proc. IEEE, *67,* No. 1 (March 1979), pp. 397–427.
4. S. C. Kak and N. S. Jayant, "On Speech Encryption Using Waveform Scrambling," B.S.T.J., *56,* No. 5 (May–June 1977), pp. 781–808.
5. D. E. Knuth, *The Art of Computer Programming,* Vol. 2, Massachusetts: Addison-Wesley, 1969, pp. 9–25.
6. M. Buric, J. Kohut, and J. Olive, "Digital Signal Processor: Speech Synthesis," B.S.T.J., this issue.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Receiver for *TOUCH-TONE*® Service

By J. R. BODDIE, N. SACHS, and J. TOW

(Manuscript received July 2, 1980)

This paper describes the design of a single-package, all-digital receiver for TOUCH-TONE® service implemented by using a digital signal processor integrated circuit. The receiver is particularly suited for systems that operate on signals that have been encoded into a digital pulse-code-modulation format. The program contained in the digital signal processor was designed to emulate the signal processing functions of a central office grade receiver. Measurements of performance confirm the equivalency.

I. INTRODUCTION

This paper describes the design of a single-package, all-digital receiver for *TOUCH-TONE*® service implemented by using a digital signal processor (DSP) integrated circuit.[1] The receiver is particularly suited for systems that operate on signals that have been encoded into a digital pulse-code-modulation (PCM) format.

TOUCH-TONE service is a voice frequency signaling system in which any one of 16 digits may be transmitted by simultaneously sending two tones. The frequency of one of the tones may be either 697, 770, 852, or 941 Hz (called the low group) and the frequency of the other tone may be either 1209, 1336, 1477, or 1633 Hz (called the high group). The receiver must tolerate frequency shifts in the transmitter, operate over a wide dynamic range in the presence of noise, and be insensitive to speech (digit simulation).[2]

The program contained in the DSP was designed to emulate the signal processing functions of a central office grade receiver. Measurements of performance confirm the equivalency.

II. DESCRIPTION OF THE RECEIVER

2.1 Receiver architecture

The model for the DSP receiver architecture was the analog type-H service receiver. The type-H receiver is used in many central offices today and has a relatively high sensitivity and noise immunity, as well as good digit simulation performance.

A block diagram of the DSP receiver is shown in Fig. 1. At the input is a filter which reduces interference from power line and precise dial tone components. This filter provides loss at 60, 180, 350, and 440 Hz, and it establishes a passband between 650 and 3000 Hz. It is a fourth-order high-pass filter instead of a sixth-order bandpass filter as in the analog receiver, because the digital receiver does not have to remove cable test tones above 10 kHz. The PCM data have been converted from analog signals that are band-limited to less than 4 kHz.

The output of the input filter is then split by two sixth-order band elimination filters. The low-group band elimination filter (LGBEF) provides loss only in the frequency band from 600 to 1050 Hz. The high-group band elimination filter (HGBEF) provides loss in the high-group frequency passband and gain in the band above 1900 Hz. The use of BEFS is important for good digit simulation performance as discussed later.

Each BEF output is followed by a limiter which serves two functions. First, for signal levels above a minimum value, the limiter gives an output that is independent of the input signal amplitude. The second function of the limiter is to provide digit simulation protection, as discussed later. Each limiter output drives a set of second-order channel filters. The filters are tuned to the signaling frequencies and have a pole-Q of 16.

The channel filters drive detector circuits which sense a signal level that is greater than a fixed threshold. The threshold is set to 2 dB below the peak signal level that would be present if a sine wave of the nominal frequency and amplitude were input to the limiter. If the signal to the detector drops below the threshold, the detector continues to indicate detection for a fixed hold time.

The outputs of the detectors are packed into an 8-bit word and are applied to the routine which implements the digit validation logic. This logic checks for a valid detector output and applies a timing criterion. If the detectors show the presence of a tone pair for a continuous validation interval, the logic indicates a valid digit with the digit present (DP) flag and outputs a code word for the digit. The logic then requires the pair to be continuously absent for the validation interval before indicating no digit. An early detect (ED) flag is also provided to indicate that a tone pair is in the process of being validated.

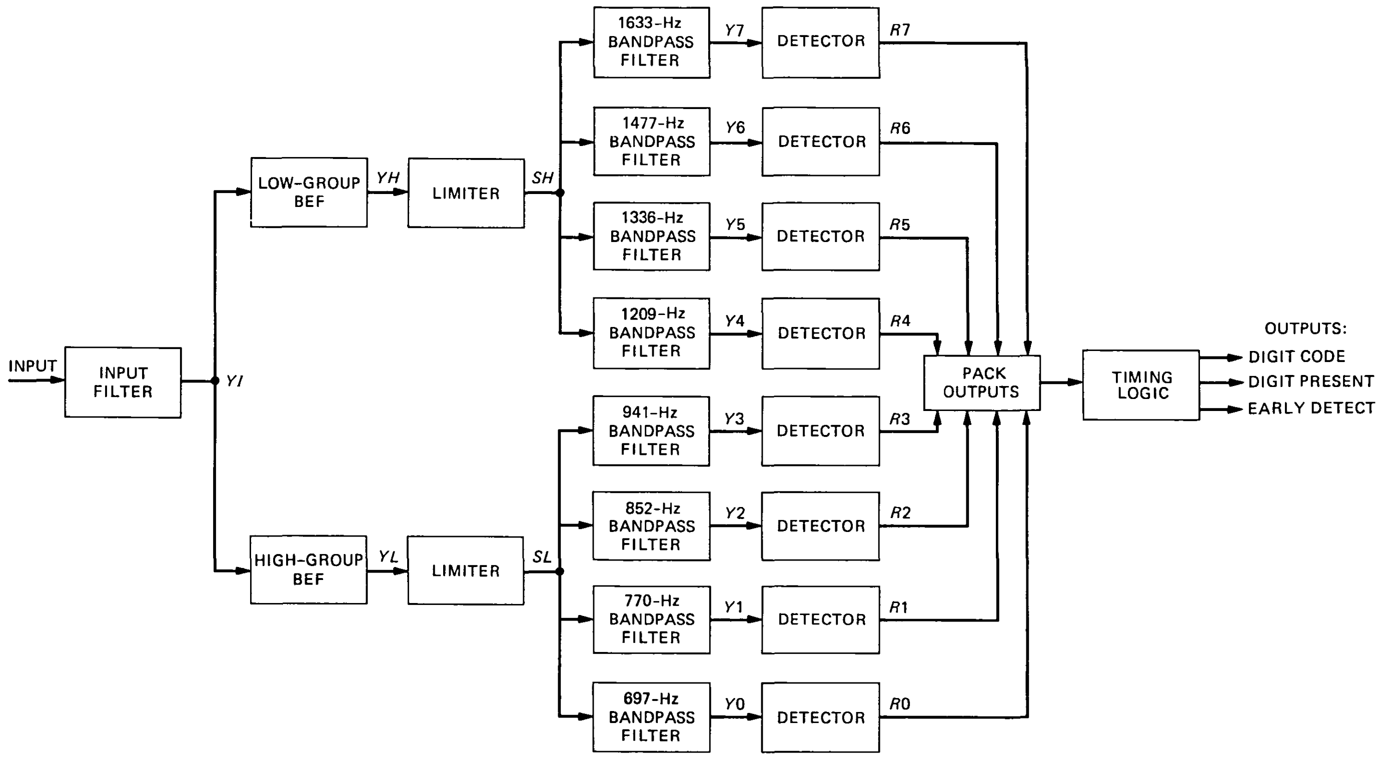


Fig. 1—Receiver architecture.

The digit simulation performance of the receiver relies on limiter guard action. If a signal with two or more frequency components is present at the input of a limiter, the magnitude of each individual signal component at the output of the limiter is reduced to less than it would be if only that component were present at the input. When valid tones are transmitted, each group limiter receives only one signaling component and the magnitude of that component at the limiter output is above a threshold, as measured by the channel filter and detector. When speech is input to the receiver, the group limiters get many frequency components—some of which might be in the range of valid signaling tones. In this case, the limiter guard action reduces the magnitude of any signaling component at the output of the limiter to a value less than the detector threshold. The use of BEFS increases the number of frequency components from speech that would reach the inputs to the limiters.

2.2 Digital signal processor program

Figure 2 shows the organization of the DSP program. After initialization, the DSP executes the main receiver routine which consists of a loop that is traversed once every 250 μs (or two 8-kHz sample periods). The main routine calls a filter subroutine twice per loop so that the digital filtering operations are performed once every 125 μs. The

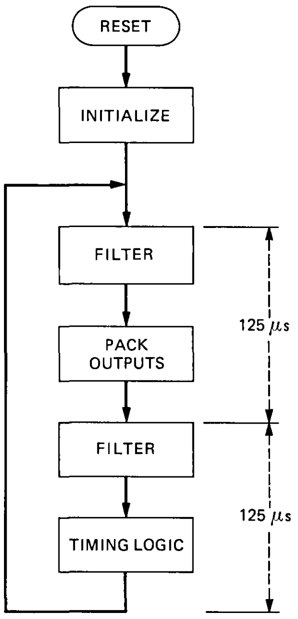


Fig. 2—Program organization.

operations done in the main routine include packing the eight detector outputs into a single word and performing the timing validation logic. The program was organized in this way because all of the filtering and digit validation cannot be done in a single 125 μs interval. However, the digit validation logic does not have to be done at an 8-kHz rate and can be split over two or more intervals.

The initialization routine sets the AU control register, AUC, for rounding and overflow protection. The I/O control register, IOC, is set for format 0, 8-bit, passive input and 2.5-Mb/s active output. The RAM is cleared and some register pointers are set.

The first operation of the main routine is to call the filter subroutine. The filter subroutine waits for an input and then converts it from μ-255 to linear. The input and BEFS are shown in a block diagram in Fig. 3. All of the second-order structures use only four multiplies. The signal levels are adjusted at the inputs to the BEFS.

The group limiters are realized by the following code:

```
                                   p = lim**rya;      "rya points to YH"
                    a = p;
                    a = a << 14;
   w = a;
```

The initial multiplication of a fraction (lim) and the output of the BEFS (YH shown here) limits the dynamic range of the receiver. The result in the accumulator is shifted up and transferred to the W register. Signals in the range of 0 to −29 dBm will cause the W register to saturate because of overflow. This limiter action using the overflow protect function is very similar to the behavior of the analog receiver.

The outputs of the limiters are applied to the eight channel filters. These filters are realized as shown in Fig. 4a, which illustrates the 770-Hz BPF.

The detector algorithm for each channel output is shown in Fig. 4b. This is the equivalent of the comparator circuit in the analog receiver. The output is compared with a threshold value (th) for every sample. If the sample is greater than the threshold, the detector output is set to a positive number (CNT) and decremented, otherwise, the detector output is just decremented. The value of CNT is such that when the signal falls below the threshold, the detector output will be a positive number for the detector hold time. The 1633-, 1477-, and 1336-Hz detectors compare the rectified channel filter output with the threshold to improve the detectability of these frequencies. It is possible to sample a sine wave at these frequencies whose amplitude is above the threshold in such a way that none of the positive samples for one cycle

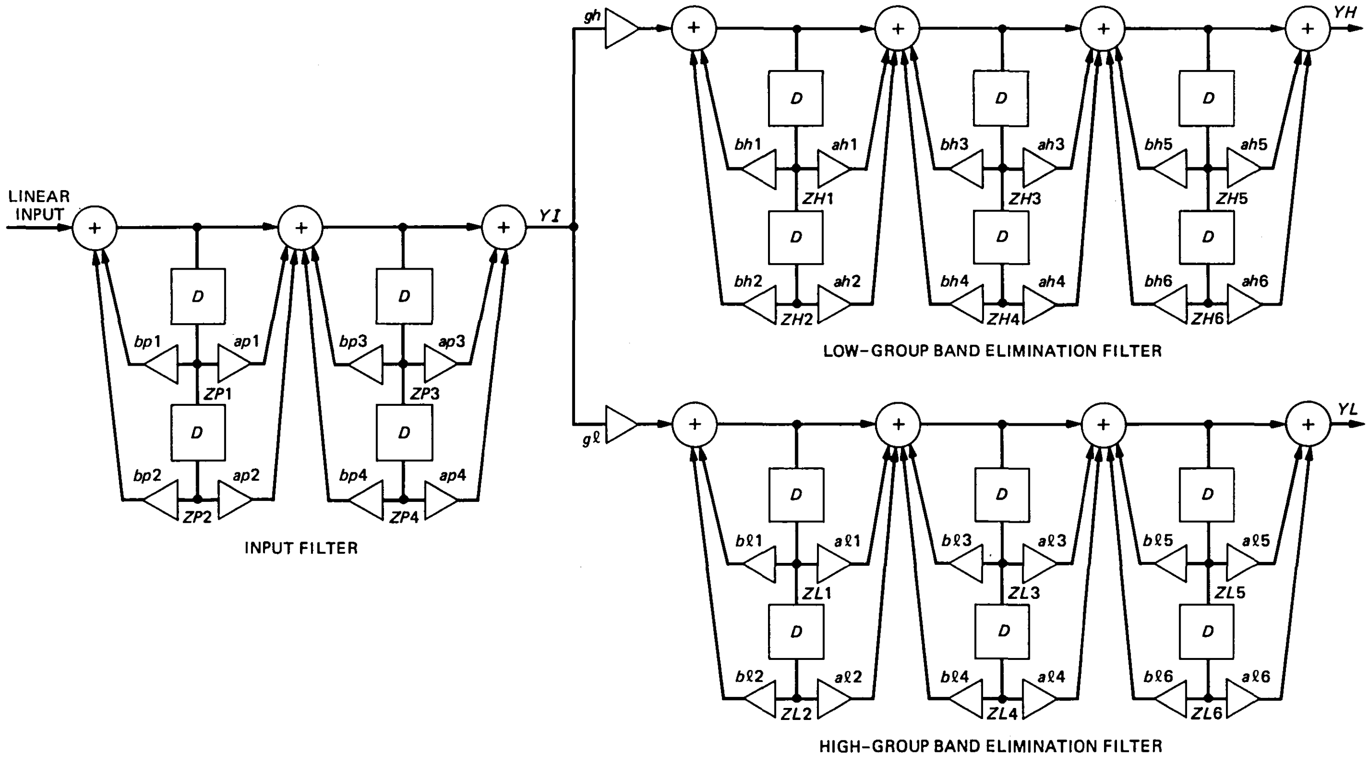
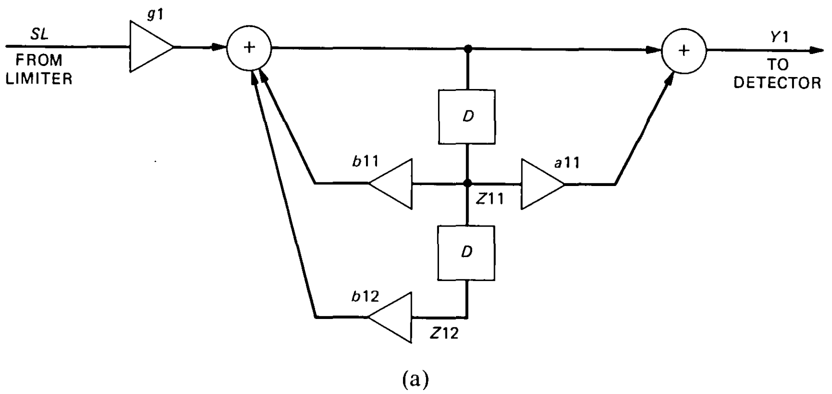


Fig. 3—Input and band splitting filters.



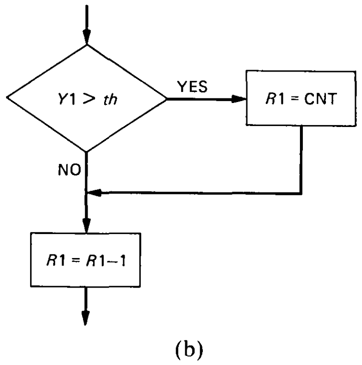


Fig. 4—Channel filter and detector for 770-Hz tone. (a) BPF (770 Hz). (b) Detector (770 Hz).

are above the threshold. The rectification operation guarantees that at least one sample per cycle will be above threshold if the amplitude of the sine wave is 1 dB above the threshold. The filter subroutine then returns to the main routine.

In the main routine, the outputs of the low-group detectors are packed into a single word using the "sgn" function which tests the sign of the detector outputs. The word is tested to see if there is a tone detected in the low group. The high-group detector outputs are packed and tested in a similar fashion. The low- and high-group outputs are then combined to form an address that can be used to access the desired output code for the detected tone pair from a table in ROM.

After calling the filter subroutine again, the main routine does the timing validation function and provides ED and DP flags on the S bits of the DSP. The latest detected tone pair (*NEW*) is compared with the tone pair that is under validation (*OLD*). The result of the validity test can be followed with the flowchart in Fig. 5. If there is no detected tone pair or the detection does not match the previous detected pair,

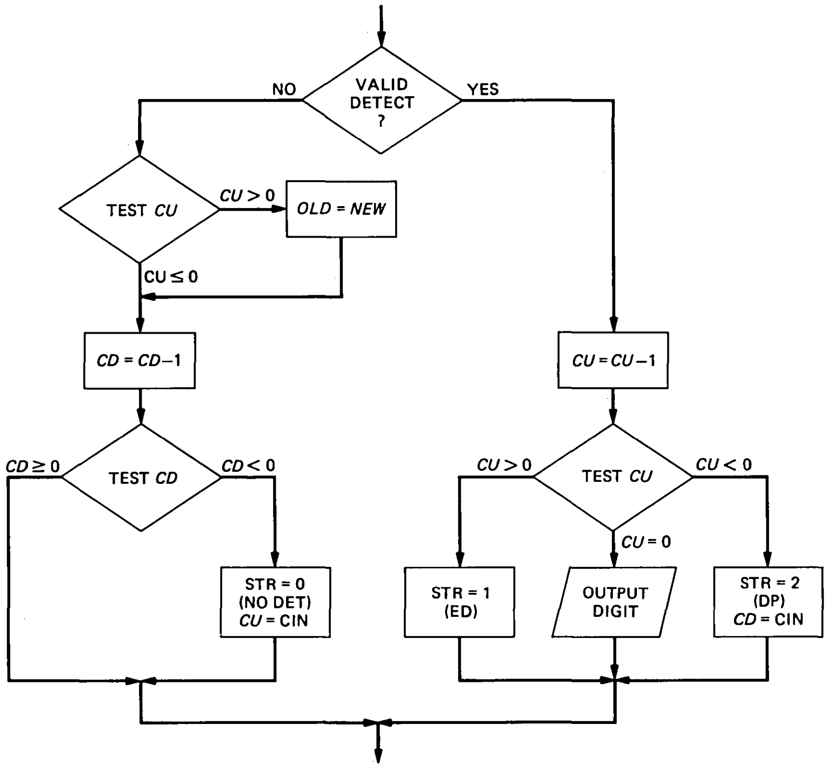


Fig. 5—Timing validation logic.

a variable, *CU*, is tested to see if no digit has been validated. If there is no digit present (positive *CU* value), the *OLD* is updated with the *NEW*. A variable, *CD*, is decremented and tested to see if a tone pair has been continuously absent for the validation interval. If so, *CD* will be negative and the S bits are cleared to indicate "no digit" and the variable *CU* is initialized (CIN). If a valid tone pair is detected, the variable *CU* is decremented and tested to see if the pair has been present for the validation interval. If not, the S bits are set to indicate ED. If *CU* is negative, indicating a continuous detection for the validation interval, the S bits are set for DP and the *CD* variable is initialized (CIN). Finally, if *CU* is exactly zero, the code for the digit pair is written in the output buffer. The program then branches to the beginning of the loop.

As written, the program uses nearly 100 percent of the time available for processing. However, it could be reorganized such that the main routine calls the filter subroutine three or four times per loop in order to do more low-speed operations if necessary. Only 42 percent of the ROM is used for the receiver program. The rest of the ROM space could be used for other programs that are accessed by a conditional branch taken after reset and based on the state of the DSP C bits.

2.3 Hardware and interface considerations

Because the complete receiver is in a single 40-pin package, the hardware requirements are extremely simple. Also, the flexibility of the DSP I/O unit allows a variety of possible interface configurations to be realized. Figure 6 shows an arrangement that is compatible with the programmed I/O control described in the previous section. It is also the circuit that was used to test the performance of the receiver.

A number of connections are independent of the choice of I/O configuration. The DSP must be powered by a +5 volt source, the internal ROM enabled, a 5-MHz clock applied, and reset controlled. The internal ROM is enabled by connecting $\overline{\text{EXM}}$ to +5 volts. A 5-MHz clock may either be applied to the CLKIN pin (as shown) or the on-chip

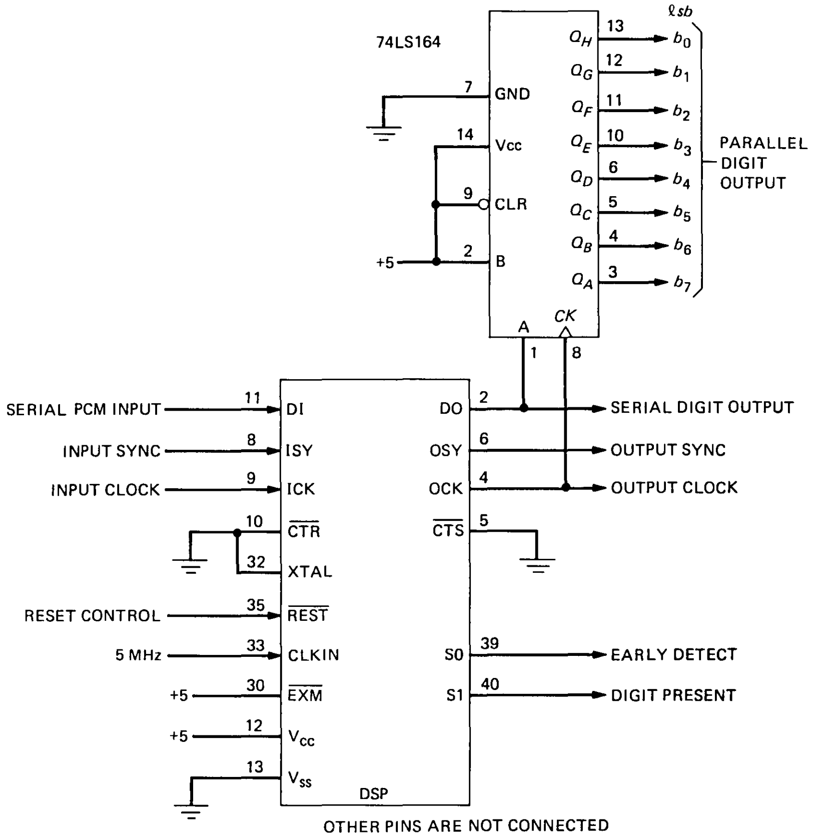


Fig. 6—Receiver hardware.

oscillator may be used with the addition of an external crystal. The reset line should be held low (TTL levels) for at least 600 ns after power up and then held high while the receiver is operating.

The input is set up in the program for passive, 8-bit operation. That is, the DSP-receives clock pulses and synchronization information from the system providing the PCM data. The input is continuously enabled by the grounding of the $\overline{\text{CTR}}$ pin.

The output is programmed for active, 8-bit transmissions at a rate of 2.5 Mb/s. All of the output signals are generated by the DSP and can be used to drive a TTL shift register to provide parallel digit output. The output is also continuously enabled by the grounding of the $\overline{\text{CTS}}$ pin.

The ED and DP flags are provided by the DSP S0 and S1 pins, respectively.

All of the other DSP pins are not used and may be left open.

III. RECEIVER PERFORMANCE

The performance measurements for the receiver were made by interfacing a μ-255 law encoder (analog-to-digital converter) with D-type channel bank filter to the DSP. The receiver could then be checked using analog receiver test facilities. A type-H receiver was tested in parallel.

The signal amplitude sensitivity of the receiver was measured with worst-case parameters. That is, the frequencies of the tones were set to their maximum allowable deviation from nominal, the levels of the two-tone groups were made different, maximum expected dial tone was added, and the duration and interdigit intervals were at their minimum values. The receiver was also tested for detection errors with tones in the presence of gaussian and impulse noise. Finally, digit simulation was tested by applying speech and music to the receiver.

In all tests the DSP receiver performed as well as the type-H receiver.

IV. CONCLUSION

This paper has described a central office quality receiver for *TOUCH-TONE®* service which accepts signals encoded in a digital PCM format. The entire receiver is implemented using the DSP, a single 40-pin dual inline package that is powered by a +5 volt supply. It can easily be customized for different input and output formats, and there is enough spare ROM capacity to implement other functions when the device is not being used as a receiver.

V. ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of J. S. Thompson, who did the preliminary filter designs, and G. T. Kraemer, who provided valuable information about signaling receivers. The circuits used to test the receiver were constructed by C. T. Kirk.

REFERENCES

1. J. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. R. N. Battista, C. G. Morrison, and D. H. Nash, "Signaling System and Receiver for TOUCH-TONE Calling," IEEE Trans. Commun. Electron., *82* (March 1963), pp. 9–17.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Voice-Frequency Transmission Treatment for Special-Service Telephone Circuits

By R. B. BLAKE, A. C. BOLLING, and R. L. FARAH

(Manuscript received July 30, 1980)

This article describes the application of the digital signal processor as a voiceband signal processing element. The application chosen is one of the most stringent voice-frequency signal processing applications in the telephone network—providing transmission treatment (e.g., gain, equalization, and echo control) for special service circuits. A detailed description of a prototype transmission treatment unit, which uses the DSP, *is provided along with descriptions of the digital filter structures and filter synthesis techniques. Measured results for representative extreme cable facility cases are presented, showing that digital signal processing utilizing the* DSP *meets the telephone network transmission requirements for special service circuits.*

I. INTRODUCTION

The first, and still most heavily used, transmission medium in the telecommunications network is copper wire. Twisted pair metallic cables of various gauges, lengths, and sizes make up the bulk of the loop plant and local exchange trunk plant. The application of electronic amplifiers or repeaters to provide gain to compensate for the attenuation of signals on metallic cables was the first large-scale use of electronics in the telecommunications network. Repeaters are still used extensively. By necessity, the systems of the past that provided voice-frequency (VF) transmission treatment on a per-channel basis were analog systems. The large-scale use of sophisticated digital signal processing for these transmission treatment functions was precluded by cost, power consumption, and size of implementation. However, in recent years the capabilities of digital hardware have improved significantly, primarily as a result of the rapid development of integrated circuit technology.

The digital signal processor (DSP),[1] a VLSI device with a large number of logic circuits, is an example of the sophistication that is now possible with digital hardware. The inherent capabilities of the DSP have made it possible to consider the use of digital filters to replace some of the traditional analog network functions in VF transmission systems. This paper reports on the results of an experimental study in which the DSP is used for this purpose. Among the most desirable features of the DSP, and especially important for this study, is the ease with which it can be programmed under computer control to provide transmission treatment functions.

The principal transmission signal processing functions are performed in VF repeaters for metallic transmission systems and in carrier terminal units (CTUs) for transmission systems that contain a metallic-to-carrier interface. These systems provide signal processing for a variety of services. Typically, the most demanding signal processing requirements derive from special services applications.[2] Special service circuits are engineered using all types of transmission media. Special services are a large and rapidly growing part of the telecommunications network.

Performance objectives for a special service circuit are normally specified in terms of 1-kHz loss, attenuation distortion, echo distortion, and various other transmission parameters. If signal processing must be included in a circuit so that transmission objectives for the circuit can be achieved, the circuit is called a treated circuit. The prototype transmission treatment unit, described in this article, which uses the DSP to provide adjustable transmission treatment functions, is referred to as a digital treatment transmission unit (DTTU). The description of the DTTU and its capabilities begins in Section III. First, in Section II, an outline of treated circuits is presented since the performance goals for the DTTU were based on the performance objectives for these circuits.

II. INTRODUCTION TO VOICE-FREQUENCY CIRCUITS

Table I lists some of the applications for metallic treatment systems. Figure 1 illustrates some of the possible circuit arrangements with transmission treatment. Each of the arrangements is assumed to be providing a special service circuit—in this case a foreign exchange trunk. Figure 1a shows a terminal repeater, with a switching system on one side and a 2-wire cable connected to a PBX on the other. Figure 1b is smilar, but with the repeater placed at an intermediate location in the circuit. Figure 1c shows the use of two repeaters in a long circuit.

Finally, Fig. 1d shows a circuit that contains both a carrier link and

Table I—Inserted Connection Loss (ICL) objectives for typical 2-wire switched special services

Switched Special Services	ICL (dB)
PBX-CO trunk	3.5
Foreign exchange trunk	3.5
WATS trunk (to class 5 CO)	3.5
WATS trunk (to class 4 CO)	4.5
Foreign exchange line	3.5
WATS line (to class 5 CO)	3.5
WATS line (to class 4 CO)	4.5

a 2-wire cable facility. This latter configuration, with a carrier link in tandem with a 2-wire metallic extension, is quite common. Carrier systems are designed to be essentially transparent for transmission purposes. Note that if the carrier link is removed in Fig. 1d and the CTUs retained, the resulting circuit arrangement is similar to that shown in Fig. 1a.

The CTUs that provide the 4- to 2-wire interfaces must provide

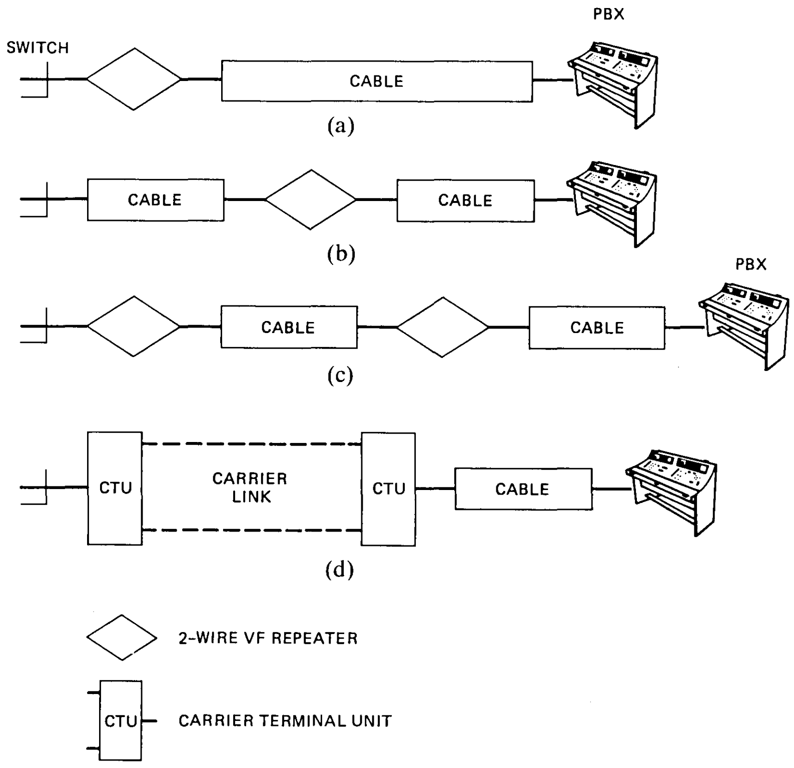


Fig. 1—Some example topologies for a foreign exchange trunk.

transmission treatment for the adjacent metallic cable. Carrier terminal units with analog treatment are currently available for both digital and analog carrier sytstems. A digital carrier system is an obvious candidate for digital transmission treatment since digital encoding and decoding of signals is already required. Digital switching systems, which have transmission characteristics similar to digital carriers, are also strong candidates for digital treatment of special service circuits. This article concentrates on the digital-to-analog 4- to 2-wire interface.

2.1 Transmission treatment objectives

The success of the telecommunications network in providing satisfactory service places requirements or objectives on some of the basic electrical characteristics of the equipment used. The services listed in Table I must be engineered to have carefully controlled transmission properties. The main theme of this study is the use of the DSP to control the loss, attenuation distortion, and echo distortion of treated circuits. This is accomplished by implementing digital filters with the DSP, as is described in detail in the following sections. Although not considered here, the DSP could also be programmed to control delay distortion of a treated circuit.

Of the services listed in Table I, the trunks have the most stringent performance objectives. A typical short haul treated special service trunk must be engineered to have a 1-kHz loss of 3.5 dB to an accuracy of approximately 0.5 dB. Furthermore, the loss of the circuit at 400 and 2800 Hz, relative to the 1-kHz loss, should be within the following limits: at 400 Hz, −1.0 dB to 3.0 dB; at 2800 Hz, −1.0 dB to 4.5 dB. Additionally, the circuit loss should be relatively smooth between these two frequencies. The circuit should also be designed so that its loss response rolls off at the VF band edges to enhance stability margin and echo performance. For the configuration shown in Fig. 1a, crosstalk considerations restrict the gain of the repeater to a maximum of 6 dB at 1 kHz, and the loss of the cable to a maximum of 9 dB at 1 kHz.

Since a 2-wire treatment unit is inherently a feedback device, some means must be provided to "balance" the unit, that is, significantly reduce the feedback, to ensure adequate stability margin and satisfactory echo performance for a treated circuit. In the DTTU, a digital canceler network provides this balance function. Since gain added to a circuit amplifies an echo along with the desired signal, treated circuits must be better balanced than untreated circuits. Poor balance results in listener distortion on short circuits and talker echo on long circuits. At a 4- to 2-wire interface, the equipment should produce at least 15 to 18 dB of loss between the 4-wire ports (e.g., ports 3 and 2 in Fig. 2), including the effects of gain, over most of the voiceband. The loss at the VF band edges may be somewhat less, although adequate stability margin must be maintained.

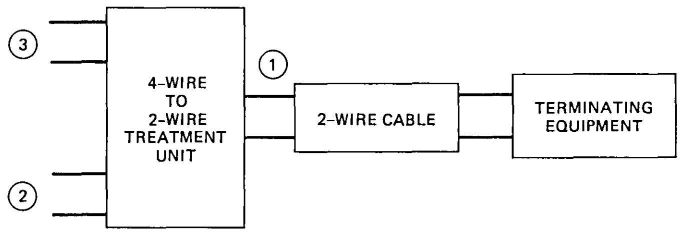


Fig. 2—Balance objectives are achieved if there is at least 15- to 18-dB loss between ports 3 and 2 of the 4- to 2-wire transmission treatment unit.

III. DIGITAL TREATMENT TRANSMISSION UNIT

A block diagram of the DTTU is shown in Fig. 3. The digital ports of the DTTU could interface with a digital carrier or a digital switch, while the 2-wire analog port could interface with loop plant cable or trunk cable. The DTTU consists of a DSP, a line interface unit (LIU), and the appropriate logic devices to interface the DSP to the LIU. All adjustable gain, equalization, and echo cancellation are provided by digital filters implemented with the DSP. The remaining DTTU functions, such as encoding and decoding of signals, are provided in the LIU.

3.1 Line interface unit transmission functions

The experimental version of the LIU contains a commercially available 16-bit, full-linear digital-to-analog converter. Unpublished studies

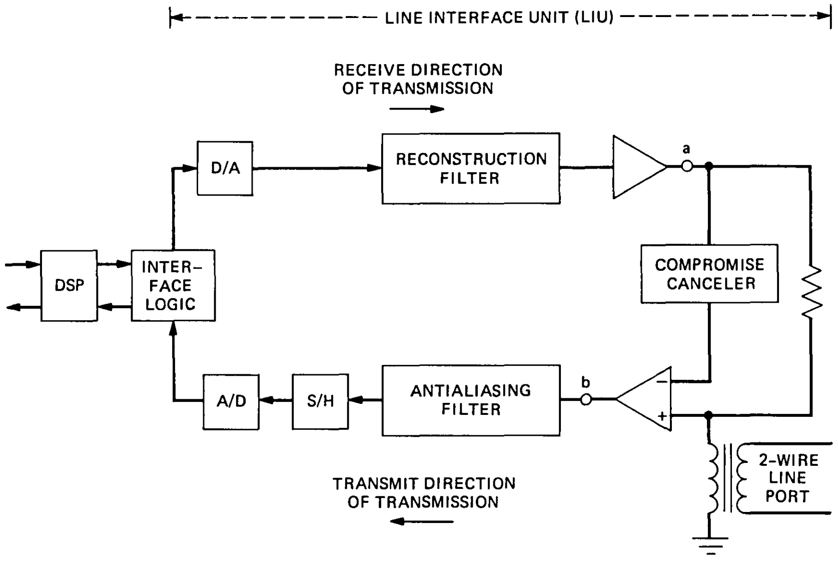


Fig. 3—Block diagram of the digital treatment transmission unit (DTTU).

by J. H. W. Unger have shown that the standard μ-255 encoding does not provide sufficient dynamic range for digital signal processing in some special service applications. The 16-bit linear encoding was chosen to avoid this problem. The converter is followed by a reconstruction filter that removes energy above 4 kHz. The reconstruction filter drives a fixed-gain analog amplifier that, in turn, drives the 2-wire cable through a 900-ohm transformer-coupled output. Additionally, the amplifier drives an analog compromise canceler, a device discussed in more detail in the next paragraph. Input signals from the cable are coupled through the transformer to a differential amplifier. The differential amplifier drives an antialiasing filter that removes energy above 4 kHz and also removes 60-Hz induction. The output of the antialiasing filter is sampled at 8 kHz and then converted to digital form by a 16-bit, full-linear analog-to-digital converter.

The primary purpose of the compromise canceler is to provide some loss (approximately 6 dB or greater) between points "*a*" and "*b*" of the LIU for the universe of cables with which the DTTU is expected to interface. Its transfer function is fixed, with one pole and no zeros. As compared to the DTTU performance in the absence of the compromise canceler, the benefits obtained are twofold:

(*i*) A large signal at point "*a*" is less likely to overload the analog-to-digital converter.

(*ii*) The performance of the digital echo canceler in the DSP is enhanced.

3.2 Digital signal processor transmission functions

As mentioned above, all adjustable gain, equalization, and echo cancellation for the DTTU are provided by digital filters in the DSP. These filters are:

(*i*) an equalizer for the transmit direction of transmission,
(*ii*) an equalizer for the receive direction of transmission, and
(*iii*) a canceler.

The equalizers provide adjustable gain as well as adjustable equalization. The transfer functions of these three filters are dentoed by E_t, E_r, and C, respectively. These filters will be represented in many of the remaining figures in this article by the symbols shown in Fig. 4.

IV. SELECTION OF FILTER FORMS FOR THE EQUALIZERS AND CANCELER

The DTTU must have the capability to provide transmission treatment for a large variety of cable facilities. Laboratory and computer simulations of a representative sample of treated transmission facilities have shown that the DTTU has the required capability if the canceler is a 32-tap transversal filter and each equalizer is a fourth-order

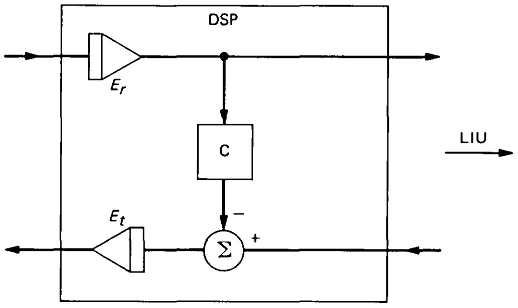


Fig. 4—Symbolic representation of DTTU digital filters.

recursive filter composed of two cascaded biquads. In Z-transform notation, the filter functions are:

$$C = \sum_{n=0}^{31} c_n z^{-n},$$

$$E_t = G_t \prod_{m=0}^{1} \frac{a_{m0t} + a_{m1t}z^{-1} + a_{m2t}z^{-2}}{1 - b_{m1t}z^{-1} - b_{m2t}z^{-2}}, \qquad \text{and} \qquad (1)$$

$$E_r = G_r \prod_{m=0}^{1} \frac{a_{m0r} + a_{m1r}z^{-1} + a_{m2r}z^{-2}}{1 - b_{m1r}z^{-1} - b_{m2r}z^{-2}}. \qquad (2)$$

Symbolic diagrams of these filters are shown in Figs. 5 and 6. (In the figures, x, y, v_1, and v_2 represent signal values, and are used for the discussion in Section V.) The equalizer pre-multipliers, G_t and G_r, allow flexibility in parceling out gain between the filter sections and in

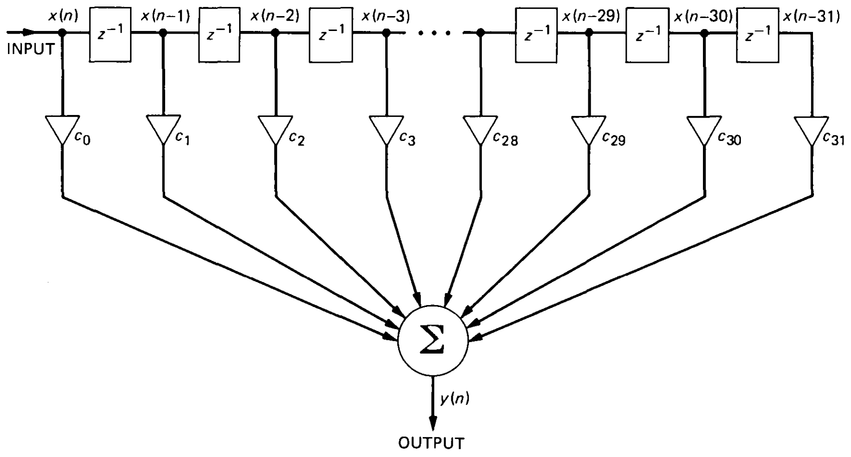


Fig. 5—Block diagram of a 32-tap transversal filter.

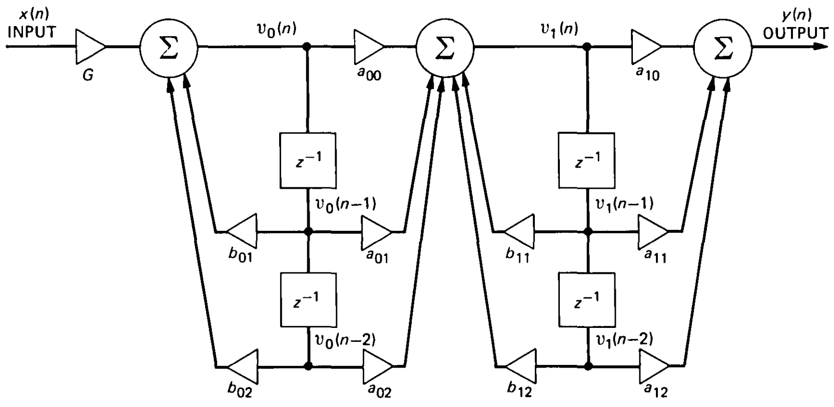


Fig. 6—Block diagram of two cascaded biquads.

controlling coefficient magnitude, which is constrained by the DSP coefficient storage format to be less than 2.0. The coefficients in the three equations above are different for each different cable facility.

A recursive structure, rather than a transversal structure, is used for the equalizers for one primary reason: in addition to providing gain and equalization in the frequency band 400 and 2800 Hz, for 2-wire transmission, it is also necessary that the equalizers provide significant attenuation at the VF band edges to enhance the stability margin performance of the DTTU. This characteristic is more readily obtained with a low-order recursive structure, for the transmission system under consideration, than it is with a reasonable size transversal filter.

A cascaded biquad structure, rather than a direct-form fourth-order structure, is used for the equalizers because the transfer function of the cascaded structure is less sensitive to coefficient quantization. In Refs. 3, 4, and 5 it is shown that if the unquantized pole locations or zero locations lie close to each other, coefficient quantization can cause a large shift in the pole or zero locations. If this shift occurs, the filter response will differ from the desired response. In the direct-form realization, the shift as a function of coefficient quantization is dependent on all the pole or zero locations. However, in the cascade form, the pole or zero shift in one section is independent of the pole or zero locations in the other sections.

V. FILTER IMPLEMENTATION WITH THE DSP

In this section, the implementation of the equalizers and canceler with the DSP is discussed. For a description of the DSP architecture see Ref. 1. There are four topics of interest with regard to the DSP program. These are:

(*i*) location of coefficients and delays in the RAM,
(*ii*) filter implementation,
(*iii*) coefficient loading, and
(*iv*) program storage and execution time.

5.1 Digital signal processor RAM memory map

Since the filter coefficients are different for each different cable facility, they must be stored in the DSP RAM. Also, the delayed signal values must be stored in RAM. Coefficients and delayed signal values are stored as shown in Fig. 7 so as to minimize the number of register sets required to access them.

5.2 Filter implementation

5.2.1 Equalizers

In the DSP program used for the DTTU, the three filtering operations are performed in the following order:
(*i*) E_r,
(*ii*) C,
(*iii*) E_t.
The canceler is discussed below. In this section E_r and E_t are discussed. Since the program steps used to implement E_r and E_t are identical, a general description of the filtering steps in two cascaded biquads is presented.

Figure 6 shows a block diagram of two cascaded biquads. The input signal is labeled x and the output signal is labeled y. Internal signal values are labeled v_0 and v_1, with $v_0(n-1)$, $v_0(n-2)$, $v_1(n-1)$, and $v_1(n-2)$ being delayed signals obtained from preceding filter operations. The general sequence of operations used to filter x and obtain y requires nine additions and eleven multiplications for each value of x input to the filter.

First, $v_0(n)$ is obtained by forming the sum

$$b_{02} * v_0(n-2) + b_{01} * v_0(n-1) + G * x(n)$$

in the accumulator, rounding the result, and storing it in a temporary register. The rounded result is $v_0(n)$. The products are performed in the product register before entering the accumulator. Then, $v_1(n)$ is obtained in an identical manner from the sum

$$a_{02} * v_0(n-2) + a_{01} * v_0(n-1) + a_{00} * v_0(n) + b_{12} * v_1(n-2) + b_{11} * v_1(n-1).$$

Finally, $y(n)$ is obtained from the rounded value of the sum

$$a_{12} * v_1(n-2) + a_{11} * v_1(n-1) + a_{10} * v_1(n).$$

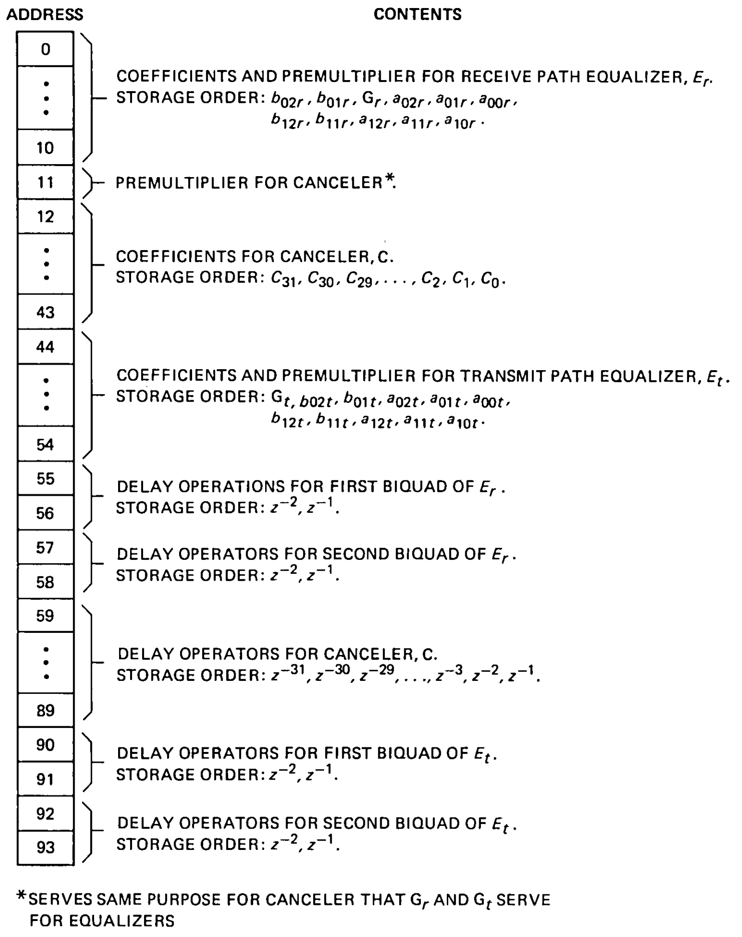

ADDRESS
CONTENTS
0
10
COEFFICIENTS AND PREMULTIPLIER FOR RECEIVE PATH EQUALIZER, E_r.
STORAGE ORDER: $b_{02r}, b_{01r}, G_r, a_{02r}, a_{01r}, a_{00r}, b_{12r}, b_{11r}, a_{12r}, a_{11r}, a_{10r}$.
11
PREMULTIPLIER FOR CANCELER*.
12
43
COEFFICIENTS FOR CANCELER, C.
STORAGE ORDER: $C_{31}, C_{30}, C_{29}, \ldots, C_2, C_1, C_0$.
44
54
COEFFICIENTS AND PREMULTIPLIER FOR TRANSMIT PATH EQUALIZER, E_t.
STORAGE ORDER: $G_t, b_{02t}, b_{01t}, a_{02t}, a_{01t}, a_{00t}, b_{12t}, b_{11t}, a_{12t}, a_{11t}, a_{10t}$.
55
56
DELAY OPERATIONS FOR FIRST BIQUAD OF E_r.
STORAGE ORDER: z^{-2}, z^{-1}.
57
58
DELAY OPERATORS FOR SECOND BIQUAD OF E_r.
STORAGE ORDER: z^{-2}, z^{-1}.
59
89
DELAY OPERATORS FOR CANCELER, C.
STORAGE ORDER: $z^{-31}, z^{-30}, z^{-29}, \ldots, z^{-3}, z^{-2}, z^{-1}$.
90
91
DELAY OPERATORS FOR FIRST BIQUAD OF E_t.
STORAGE ORDER: z^{-2}, z^{-1}.
92
93
DELAY OPERATORS FOR SECOND BIQUAD OF E_t.
STORAGE ORDER: z^{-2}, z^{-1}.
*SERVES SAME PURPOSE FOR CANCELER THAT G_r AND G_t SERVE FOR EQUALIZERS

Fig. 7—Locations of the coefficients and delays in DSP RAM.

Before the next value of x enters the filter, $v_0(n-2)$ is updated by setting it equal to $v_0(n-1)$; $v_0(n-1)$ is updated with $v_0(n)$; $v_1(n-2)$ with $v_1(n-1)$; and $v_1(n-1)$ with $v_1(n)$.

The output of the receive path equalizer is transmitted to the digital-to-analog converter and to the canceler input.

5.2.2 Canceler

Figure 5 shows a block diagram of a 32-tap transversal filter. The basic operation in a transversal filter is the multiplication of a tap coefficient by a delay output, followed by an addition in an accumulator. This operation can be accomplished with a sequence of instructions a = p + a p = axi * ryi;, where a and p are zero at the beginning of the sequence and RX and RY (coefficient and data address registers) are set to point to c_{31} and $x(n - 31)$, respectively. Because the DSP architecture does not permit a write to RAM two instructions before reading a coefficient from RAM, the delay shifts must be done separately from the filtering.

Once the output of the canceler is computed, it is subtracted from the analog-to-digital output in IBUF. The contents of the accumulator are then transferred to the W register, ready for use by the transmit path equalizer.

5.3 Coefficient loading

A two-step procedure is used to load the DSP RAM. In the first step, C0 is set and the address of a RAM location is transferred to the DSP. In the second step, C1 is set and the coefficient is transferred and stored in the RAM location that has the address transferred in step one. Each step takes one frame. The selection of the appropriate step is determined by testing the C0 and C1 control bits.

The procedure is repeated for all 54 coefficients. The coefficients are transmitted in order from address 0 to address 53. Because the DSP RAM is dynamic, and only one memory location is referenced every other frame, it must be refreshed every frame. This is accomplished by sequentially reading all locations.

5.4 Program storage requirements and execution times

The program to implement the filter functions and the coefficient loading routine requires 314 words of memory, with 21 memory locations used for the receive path equalizer, 108 for the canceler, 36 for the transmit path equalizer, 85 for the coefficient loading and refresh, and 64 for miscellaneous operations (PC sets, no-ops, etc.). In the experimental version of the DTTU, code was written to be clear rather than efficient in memory requirement. There are some techniques that in a final product could reduce the amount of memory required. For example, in doing the canceler tap update, the instruction rdi = ryp; was used. This requires two words of storage per "auxiliary" instruction. By using rdi = ryp a = p p = olx * c; the amount of storage for the update sequence is cut in half, since this is a "normal" instruction. The a = p p = olx * c; is just a "fill" and accomplishes no useful operation.

The filtering portion of the program requires 106.40 μs, with 12.8 μs for the receive path equalizer, 56 μs for the canceler, 12.8 μs for the transmit path equalizer, and 24.8 μs for miscellaneous operations. This leaves 18.6 μs for other use. One application might be a self-diagnostic for the chip (e.g., check RAM or the AU).

VI. LOSS DEFINITIONS

A special service circuit must provide a high-quality channel for VF signal transmission. Loss measurements are normally used to assess the performance characteristics of such facilities since many of the performance requirements (or objectives) for the facilities are specified in terms of loss, e.g., return loss (or balance and echo performance), 1-kHz loss, and attenuation distortion. For an all-analog system, well-known definitions and measurement procedures exist for each of these loss types. A service provisioned with the DTTU is not an all-analog system, but is, instead, a mixed analog-digital system. Since measured results of the characteristics of this system are important for all of the remaining material in this article, some of the conventions and definitions used for loss measurements in a mixed analog-digital system are presented in this section to aid the reader in interpreting the measured results.

Two additional, and related, subjects are also discussed in this section. These are: Measurements required to determine filter coefficients and DTTU loss scaling. Loss scaling (signal level control) is necessary for the DTTU since a digital processing system has a limited dynamic range (determined by the 16-bit accuracy of the analog-to-digital and digital-to-analog converters, in this case).

A simplified topology of a typical 2-wire treated facility, with treatment provided by the DTTU, is shown in Fig. 8. The DTTU ports are labeled 1 (2-wire port), 2 (transmit port), and 3 (receive port), with 2 and 3 being digital ports and 1 an analog port. Depending on the service being provided, the terminating equipment at the far end of the cable (port 4) could be a central office switch or various types of customer premises equipment (telset, PBX, etc.).

6.1 Loss in a mixed analog-digital system

The results of three loss measurements, denoted by L_t, L_r, and L_c, are necessary for characterizing the transmission performance of the

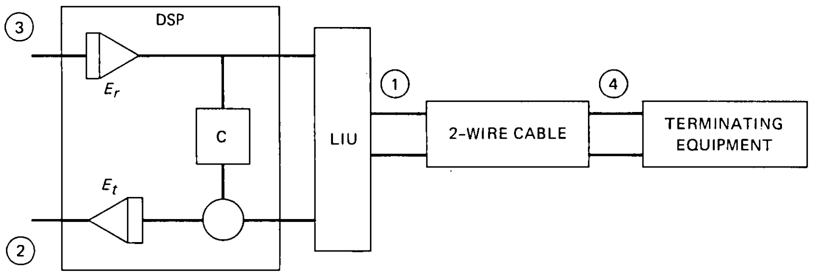


Fig. 8—Simplified topology of a typical 2-wire treated facility.

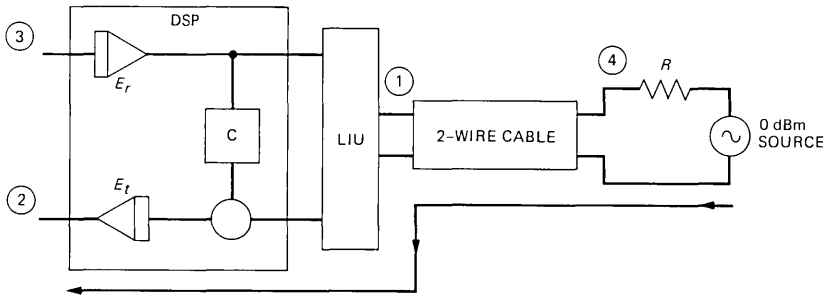


Fig. 9—Transmission path and facility configuration used for measurement of L_t.

facility shown in Fig. 8. The transmission paths associated with L_t, L_r, and L_c are shown in Figs. 9, 10, and 11, where, as the figures show,

L_t = loss from far end of cable to transmit port of the DTTU (analog-digital loss),

L_r = loss from receive port of the DTTU to far end of cable (digital-analog loss), and

L_c = loss from receive port of the DTTU to the transmit port of the DTTU (digital-digital loss).

To define L_t, let a 0-dBm analog sine-wave generator of internal resistance R be attached to the far end of the cable. Internal resistance R is either 900 or 600 ohms, depending on the impedance characteristic of the terminating equipment. At port 2, a sampled representation of this sine-wave will appear, with amplitude and phase characteristics determined by the transfer function of the cable-DTTU system. The information required for determining L_t is the peak of the sine-wave signal at port 2. It is unlikely that the peak of this signal can or will be

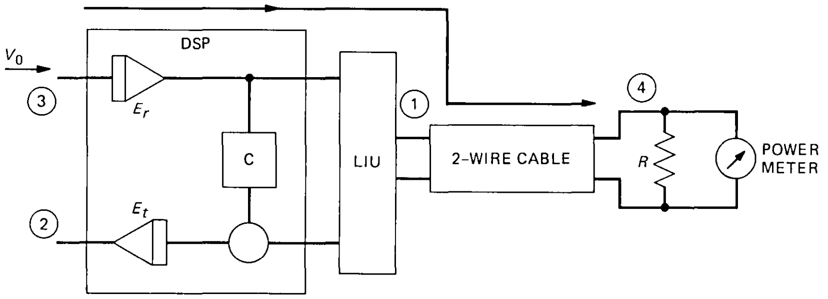


Fig. 10—Transmission path and facility configuration used for measurement of L_r.

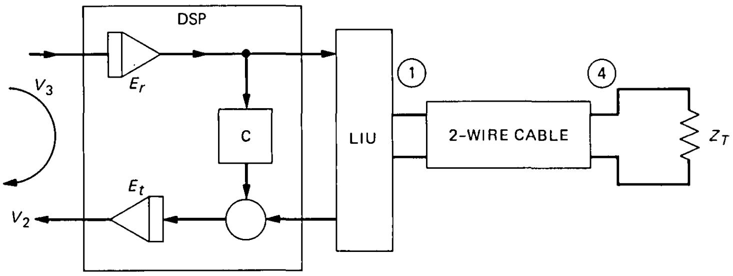


Fig. 11—Transmission path and facility configuration used for measurement of L_c.

encoded; therefore, the samples, which are considered to have integer representations, must be analytically processed to determine the peak.

If the real number representation of the signal peak at port 2, as analytically determined, is called V, then L_t is defined as

$$L_t = 20 \log(V_{fs}/V) - 20 \log(V_{fs}/V_0) = 20 \log(V_0/V),$$

where V_0 is the level representative of 0 dBm at port 2 and V_{fs} is the full-scale level (32767.0 for a 16-bit system); V_0 is less than V_{fs}, and for the system under consideration is defined by the relationship

$$20 \log(V_{fs}/V_0) = 3 \text{ dB}.$$

This definition applies to both the transmit and receive ports and is consistent with the convention used in digital carrier systems where the code is μ-255.

To define L_r, let a digital sine-wave of peak level V_0 be input at port 3. Then, an analog sine-wave will appear at the end of the cable. Its power in dBm, measured across resistance R, is L_r.

Finally, to define L_c, let a digital sine-wave of peak level V_3 be input at port 3. Then, a digital sine-wave of the same frequency will appear at port 2. If the peak level of this signal is V_2, then

$$L_c = 20 \log(V_3/V_2).$$

When a measurement of L_c is made to assess the echo and stability performance of the transmission facility, the cable will normally be terminated in a standard termination that is representative of the impedance of the terminating equipment at the far end of the cable.

6.2 Measurements required to determine filter coefficients

To set the equalizer and canceler filters for treatment of a metallic cable pair, the transmission characteristics (loss and input impedance vs. frequency) of the facility must be known. These may be determined either from a mathematical model of the cable or from measurements of its characteristics. A typical mode of operation for a facility to be provisioned with a DTTU would be one in which the characteristics of the cable are measured. Since the LIU is interposed between the DSP and the cable facility, its transmission characteristics must be included in the measurements.

In the laboratory arrangement used for this study (and in an envisioned practical application), it is most convenient to make these measurements, which are identical to the measurements discussed in Section VI, utilizing digital access points in the DSP. Since measurement access is through the DSP, the equalizers are set to unity and the canceler is set to zero. For this condition, the three loss measurement results are denoted by L_{t0}, L_{r0}, and L_{c0}.

6.3 Digital treatment transmission loss scaling

A transmission treatment unit is required to operate as a low-noise, linear amplifier for a wide range of signals expected to be flowing in the telephone network. Therefore, signal levels in the unit must be controlled so that a large signal is not over-amplified and hence distorted and a small signal is not overly attenuated toward the noise floor of the unit before being amplified, thereby degrading its s/n. In the DSP, one of the most critical interfaces where signal levels must be controlled is the DSP-LIU interface. A large signal incident on the receive port should not be over-amplified by E_r and therefore overflow and be distorted at the DSP-LIU interface. Additionally, a small signal incident on the 2-wire line port should not be overly attenuated in the LIU before being encoded and subsequently amplified by E_t.

To control the signal levels at the DSP-LIU interface, the DTTU is loss-scaled, i.e., signal loss through the unit is adjusted to achieve the desired level control. Fixed-loss (or gain) amplifiers in the LIU are used for this purpose.

The magnitude of the loss scaling used for the experimental version of the DTTU can be exhibited by demonstrating the effect it has on loss measurements performed on a cable-DTTU system when the equalizers are set to unity. In the frequency range 200 to 3400 Hz, where the antialiasing and reconstruction filters introduce negligible frequency shaping,

$$L_{r0} = \text{cable loss} - 7 \text{ dB}$$

and

$$L_{t0} = \text{cable loss} + 3 \text{ dB}.$$

Between 3400 and 4000 Hz, I_{r0} and L_{t0} remain approximately 10 dB apart; however, filter attenuation contributes significantly to their specific values.

VII. EQUALIZER AND CANCELER COEFFICIENTS

This section discusses the procedures for determining equalizer and canceler coefficients from the measured facility transmission characteristics. These procedures are based on minimum-least-square curve fit techniques.

7.1 Overview of the curve fit procedures

The canceler and equalizer coefficients are determined by separate, frequency-domain, minimization algorithms. The goal, in each case, is to minimize a penalty function. These penalty functions are:

$$P_c = \sum_{\{F\}_c} |C(F) - T_c(F)|^2, \tag{3}$$

$$P_t = \sum_{\{F\}_e} (|E_t(F)| - |T_t(F)|)^2, \tag{4}$$

$$P_r = \sum_{\{F\}_e} (|E_r(F)| - |T_r(F)|)^2, \tag{5}$$

where the target functions T_c, T_t, and T_r are dependent on the transmission characteristics of the cable-DTTU system. Functions T_t and T_r are also dependent on the attenuation distortion objectives and 1-kHz loss objective for the facility that is to receive treatment. $\{F\}_c$ denotes a set of frequencies for which P_c is to be minimized, and $\{F\}_e$ denotes a set of frequencies for which P_t and P_r are to be minimized. Extensive study of the results achieved from minimization of the penalty functions listed above, for various frequency sets and for various cables, has shown that satisfactory curve fits can be obtained if

$$\{F\}_c = 100 \text{ to } 3900 \text{ Hz in 100-Hz increments}$$

and

$$\{F\}_e = 100 \text{ to } 3700 \text{ Hz in 300-Hz increments.}$$

In eqs. 3, 4, and 5, the canceler and equalizer functions are expressed in the frequency domain, i.e., the transform variable z has been restricted to the unit circle and is equal to

$$\exp(j2\pi F/F_s),$$

where F_s is the sampling frequency (8 kHz).

As is shown below, minimization of P_c results in a linear solution for the coefficients of C, while equalizer coefficients must be determined with a gradient search technique. Additionally, it is shown that only P_t or P_r need be minimized since, for a 2-wire facility, E_t and E_r differ only by a constant gain factor which is a result of the loss scaling used for the DTTU.

7.2 Canceler target function

In both magnitude and phase, T_c is equal to the transfer function from the receive port to the transmit port of the cable-DTTU system when the equalizers are set to unity and the canceler is set to zero. Using the notation introduced in Section VI, the magnitude of T_c is

$$|T_c| = 10^{-L_{c0}/20}.$$

Figure 12 shows plots of T_c, in dB's, for two example cable cases and in Figs. 13a and 13b the corresponding phase plots are shown.

7.3 Equalizer target functions

Functions T_t and T_r are determined from the following information for the facility that is to receiver treatment:

(*i*) 1-kHz loss objective,

(*ii*) attenuation distortion (AD objectives for the frequency set $\{F\}_e$,

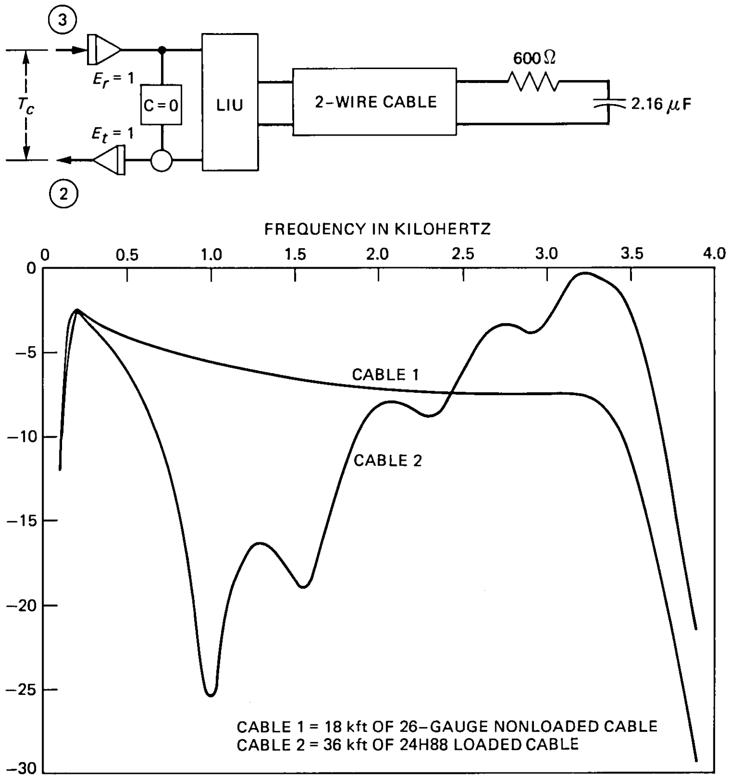


Fig. 12—T_c versus frequency for two cable cases.

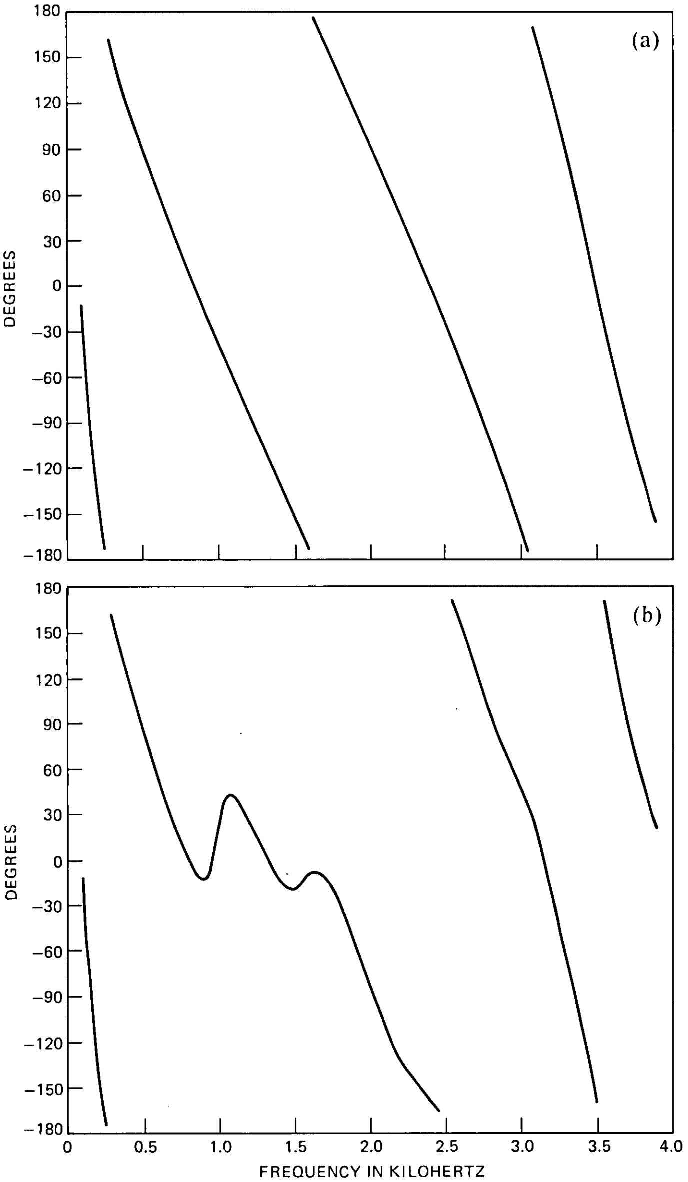


Fig. 13a—Phase plot of T_c for cable 1.

Fig. 13b—Phase plot of T_c for cable 2.

(*iii*) L_{t0} and L_{r0} (see Section VI) for the frequencies from 400 to 2800 Hz in 300-Hz increments.

However, knowing T_t is equivalent to knowing T_r since

(*i*) for all frequencies in the voiceband, for a treated 2-wire facility, the transmit direction loss of the facility should equal the receive direction loss (measured on an end-to-end basis),

(*ii*) L_{t0} and L_{r0} have the same shapes above 200 and below 4000 Hz, but differ in absolute magnitude by a value that is independent of frequency or cable type (see Section VI).

Therefore, T_t and T_r have the same shapes but differ by a constant. The same conclusion can therefore be drawn about E_t and E_r.

Since E_r can be determined from knowledge of E_t and vice versa, the notation distinction between E_t and E_r is dropped for much of the remaining discussion. The following simplified notation will be used:

(*i*) E = equalizer function,

(*ii*) L = loss data used in determining equalizer coefficients (i.e., L_{t0} or L_{r0}),

(*iii*) T = equalizer target function, and

(*iv*) P = equalizer penalty function.

Therefore,

$$P = \sum_{\{F\}_e} (|E(F)| - |T(F)|)^2,$$

where

$$E = G \prod_{m=0}^{1} \frac{a_{m0} + a_{m1}z^{-1} + a_{m2}z^{-2}}{1 - b_{m1}z^{-1} - b_{m2}z^{-2}}.$$

And finally, T will most often be expressed in dB's with the notation

$$D = 20 \log |T|.$$

Now the discussion returns to the main topic, the determination of T (i.e., D). First, the facility objectives are considered.

For 2-wire VF treated services, the loss objective is specified at 1 kHz and the AD objectives are specified at 400 and 2800 Hz. The AD objectives offer a "window" of allowed values. Since the procedure for determining equalizer coefficients is a curve fit, specific AD objectives must be chosen. The choices are:

1 – dB roll-off at 400 Hz

2 – dB roll-off at 2800 Hz.

Additionally, to ensure that the equalizers are well-behaved between 400 and 2800 Hz, it is necessary to specify AD objectives at other frequencies between these endpoint frequencies. The result is an AD objectives "curve" specified at 300 Hz increments between 400 and 2800 Hz. The AD objective at 1 kHz is 0 dB. The AD objective curve used for this study, filled-in between the 300-Hz increments, is shown in Fig. 14. For the remainder of the discussion, the curve depicted in Fig. 14 is called O_a.

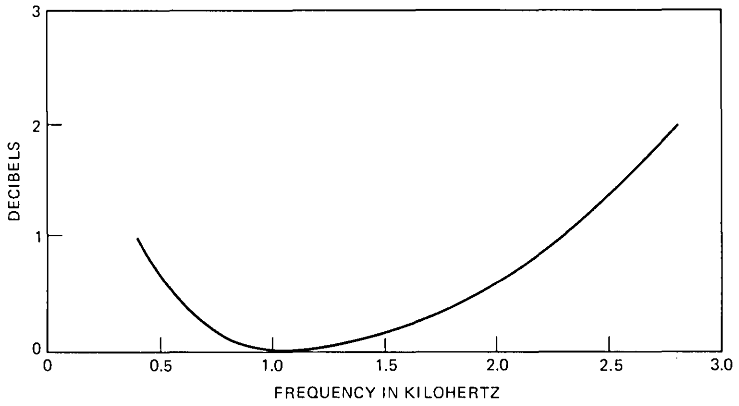


Fig. 14—Attenuation distortion objective curve O_a.

The next quantity needed for determining D is L, which is to be obtained by measurement. It is required that L be known for the set of frequencies from 400 to 2800 Hz in 300-Hz increments. Figure 15 shows plots of L, normalized to 0 dB at 1 kHz, for two different cable facilities.

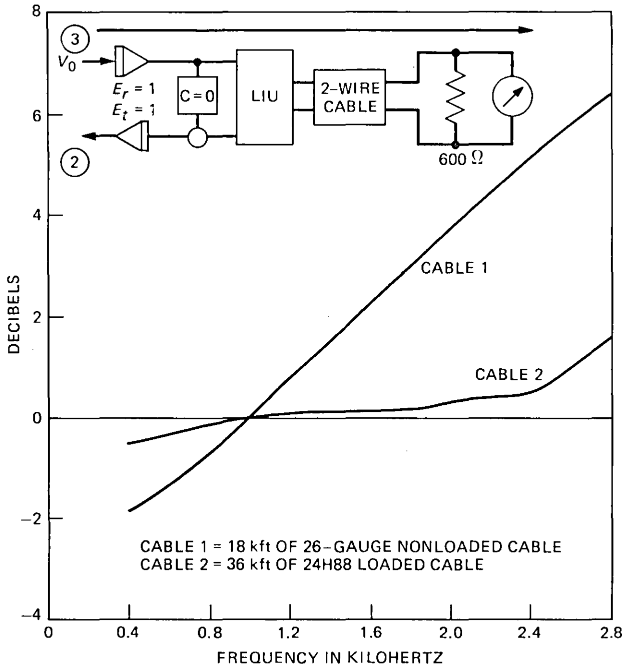


Fig. 15—L versus frequency for two example cables.

The quantities discussed up to this point are sufficient for determining D in the frequency band from 400 to 2800 Hz. In this frequency band,

$$D = L - O_a - K_1,$$

where K_1 is the 1-kHz loss objective for the treated circuit under consideration.

Outside the frequency band from 400 to 2800 Hz, a shape for D is chosen that forces the equalizers to roll off at the VF band edges. Figure 16 shows D curves, normalized to 0 dB at 1 kHz, for two different cable facilities.

7.4 Algorithm for minimizing P_c

The goal of the minimization algorithm for P_c is to obtain a set of

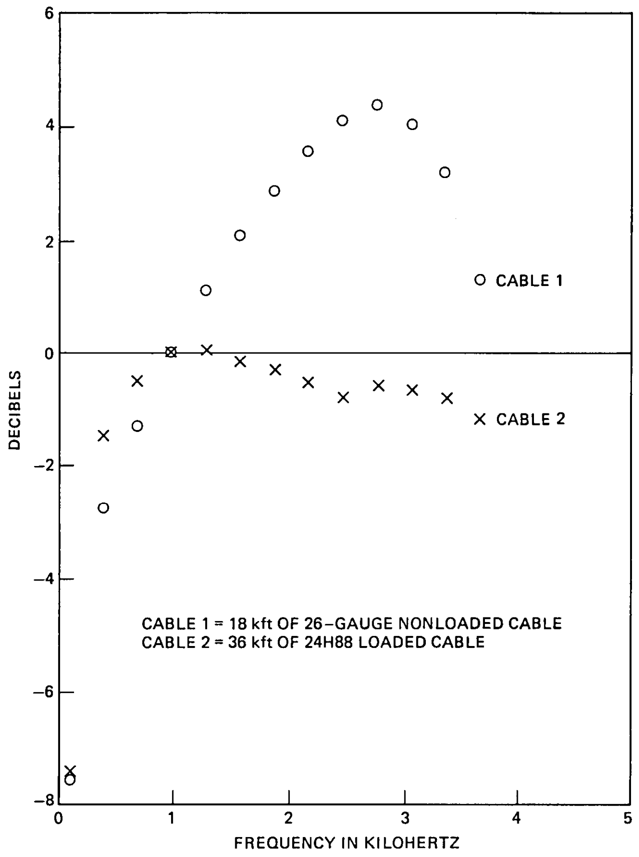


Fig. 16—$D = 20 \text{ Log} |T|$ versus frequency for two example cables.

canceler coefficients that will result in maximum loss from port 3 to port 2 of the DTTU. The transfer function from port 3 to port 2 is

$$\frac{V_2}{V_3} = E^2(T_c - C).$$

Therefore, the function that must be minimized, in a least squares sense, is

$$P_c = \sum_{\{F\}_c} |E|^2 |T_c - C|^2.$$

It is desirable that C not depend on the transfer functions of the equalizers; therefore, all equalizers are set to unity. The resulting function that must be minimized is

$$P_c = \sum_{\{F\}_c} |T_c - C|^2.$$

To minimize P_c, set

$$\frac{\partial P_c}{\partial c_m} = 0, \qquad m = 0, 1, 2, \cdots, 31.$$

The result is a matrix equation

$$\bar{B}\bar{C} = \bar{D},$$

where the elements of $\bar{B}$ are

$$b_{mn} = \sum_{\{F\}_c} \cos[2\pi F(m-n)/F_s],$$

$\bar{C}$ is the vector of tap weights, and $\bar{D}$ is a vector with elements

$$d_m = \sum_{\{F\}_c} \mathrm{Re}[T_c(F)\exp(j2\pi Fm/F_s)].$$

The solution for $\bar{C}$ is

$$\bar{C} = \bar{B}^{-1}\bar{D}.$$

7.5 Algorithm for minimizing P

For VF-treated services, phase equalization is usually not required; therefore, it is necessary only to obtain a best fit for the magnitude of E. In a least squares sense, the function that must be minimized is

$$P = \sum_{i=1}^{13} (|E(F_i)| - |T(F_i)|)^2,$$

where $F_i = (300i - 200)$ Hz. The simple method used for minimizing P_c does not yield a linear system of equations when applied to P. Instead, minimization of P requires a nonlinear optimization procedure. The procedure chosen was developed by Steiglitz.[6] Steiglitz's procedure uses the Fletcher-Powell[7] optimization technique and a novel approach for choosing initial conditions.

To apply this procedure, the equalizer transfer function is first rewritten in the form

$$E(F, A, \bar{X}) = A \frac{1 + x_0 z^{-1} + x_1 z^{-2}}{1 - x_2 z^{-1} - x_3 z^{-2}} \frac{1 + x_4 z^{-1} + x_5 z^{-2}}{1 - x_6 z^{-1} - x_7 z^{-2}} = AH(F, \bar{X}), \quad (6)$$

where

$$\bar{X} = (\, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)'.$$

(The prime denotes transpose.) P can now be written in the form

$$P = \sum_{i=1}^{13} (|AH(F_i, \bar{X})| - |T(F_i)|)^2.$$

Since $|A|$ appears in a linear fashion, it is easy to show that the optimum value of $|A|$ is

$$|A| = \frac{\sum_{i=1}^{13} |H(F_i, \bar{X})|\,|T(F_i)|}{\sum_{i=1}^{13} |H(F_i, \bar{X})|}.$$

The sign of A is irrelevant and is therefore chosen to be positive. Since A can be precisely determined from knowledge of $\bar{X}$ and the 1-kHz gain of the equalizer, the nonlinear optimization algorithm is used only to determine $\bar{X}$.

For most nonlinear optimization techniques, the primary difficulty is in choosing an initial set of coefficients that guarantees convergence in a reasonable number of iterations. The following method for choosing initial coefficients has worked quite well for minimization of P.

The first step is to find an optimum solution for the simpler equalizer function

$$E_1(F, A_1, \bar{Y}) = A_1 \frac{1 + y_0 z^{-1} + y_1 z^{-2}}{1 - y_2 z^{-1} - y_3 z^{-2}} = A_1 H_1(F, \bar{Y}),$$

where

$$\bar{Y} = (y_0, y_1, y_2, y_3)'.$$

The initial values of the coefficients, $\bar{Y}^I$, are chosen to be

$$\bar{Y}^I = (0.0, 0.0, 0.0, 0.0)'.$$

The final values of the coefficients, obtained by minimizing

$$P_1(A_1, \bar{Y}) = \sum_{i=1}^{13} (|A_1 H_1(F_i, \bar{Y})| - |T(F_i)|)^2$$

with the same algorithm that will be used to minimize P, are denoted by

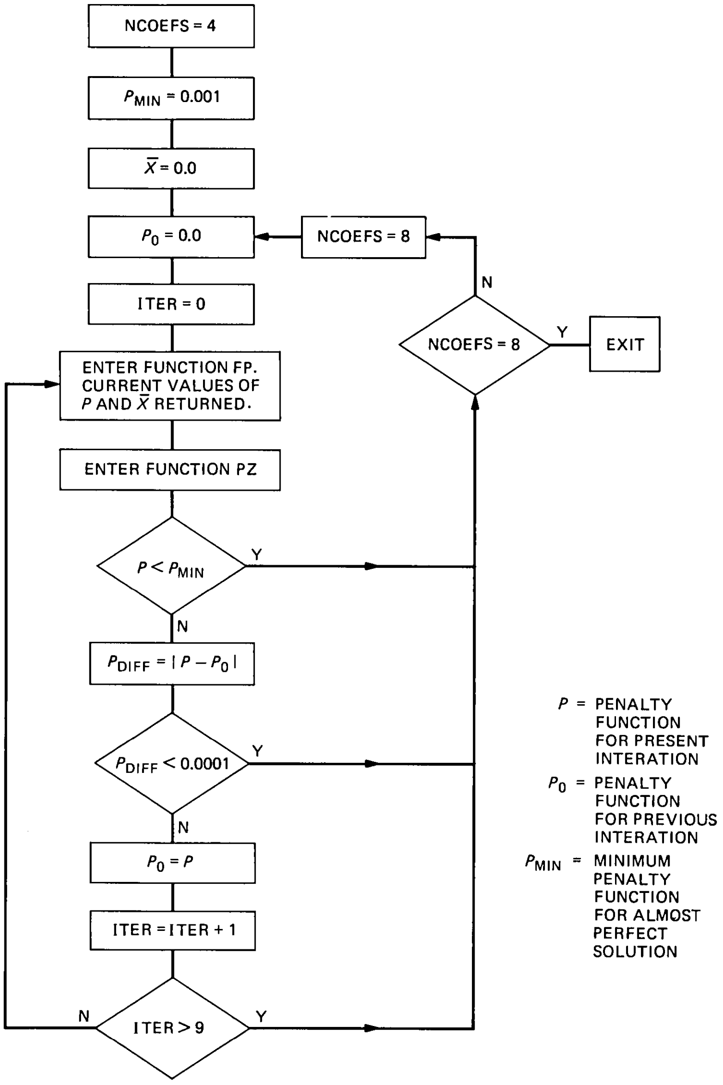
$$\bar{Y}^* = (y_0^*, y_1^* \; y_2^*, y_3^*)'.$$

Following the minimization of P_1, P is minimized, with the initial coefficients being

$$\bar{X}^I = (y_0^*, y_1^*, y_2^*, y_3^*, 0.0, 0.0, 0.0, 0.0)'.$$

The final value of $\bar{X}$ is denoted $\bar{X}^*$.

Figure 17 shows a flow chart of the optimization algorithm used to



NCOEFS = 4
P_{MIN} = 0.001
$\bar{X}$ = 0.0
P_0 = 0.0
NCOEFS = 8
N
ITER = 0
NCOEFS = 8
Y
EXIT
ENTER FUNCTION FP. CURRENT VALUES OF P AND $\bar{X}$ RETURNED.
ENTER FUNCTION PZ
$P < P_{MIN}$
Y
N
$P_{DIFF} = |P - P_0|$
$P_{DIFF} < 0.0001$
Y
N
$P_0 = P$
ITER = ITER + 1
N
ITER > 9
Y
P = PENALTY FUNCTION FOR PRESENT INTERATION
P_0 = PENALTY FUNCTION FOR PREVIOUS INTERATION
P_{MIN} = MINIMUM PENALTY FUNCTION FOR ALMOST PERFECT SOLUTION

Fig. 17—Flowchart of algorithm used to minimize P_1 and P.

minimize P_1 and P. NCOEFS denotes the number of coefficients that are being determined for any one pass through the algorithm, with NCOEFS = 4 for minimization of P_1 and NCOEFS = 8 for minimization of P. In the algorithm, the common notation for both penalty functions is P. The reader should note the calls to functions FP and PZ in the algorithm.

Function FP contains the Fletcher-Powell procedure. The FP returns the current value of P and the corresponding array of coefficients. Within FP there are a maximum of 25 updates to the coefficients. The array $\bar{X}$ is used to store the array $\bar{Y}$ when NCOEFS = 4.

The purpose of function PZ is to invert all poles and zeros that are outside the unit circle to the inside of the unit circle and to move all poles and zeros that are on the unit circle to a small distance inside. If any poles or zeros are outside the unit circle, the actions taken in PZ changes the phase of E, but not the magnitude, which is the important quantity in this case.

The primary justification for the algorithm just described is that it has never failed to yield a suitable minimum for P, for realistic target functions. The algorithm takes 10 to 30 seconds of CPU time, depending on the cable facility, on a DEC PDP 11/70, and is written in the C language.[8]

The output of the algorithm, $\bar{X}^*$, does not contain enough information for a direct determination of the optimum set of equalizer coefficients. The information contained in the vector $\bar{X}^*$ is sufficient only for a determination of the poles and zeros of E. Additionally, the algorithm has not necessarily yielded an ordering of the poles and zeros that is best suited for achieving the noise and distortion objectives for the facility. These issues are now considered. The outcome is equalizer transfer functions in their final form, i.e., the form expected by the DSP program.

7.6 Procedure for putting the equalizer functions in their final form

When the 1-kHz gain of the equalizers is determined, E_t and E_r can be expressed in the form

$$E_t = A_t H(\bar{X}^*), \tag{7}$$

$$E_r = A_r H(\bar{X}^*), \tag{8}$$

where H was defined in eq. 6. The object of the discussion in this section is to outline a procedure for putting eqs. 7 and 8 in the form of eqs. 1 and 2. For the types of signals expected to be flowing through the equalizers, the goals of this procedure are as follows:

(*i*) There should be minimum degradation of the s/n of the signals.

(*ii*) There should be a minimum amount of signal clipping at the critical internal nodes (to be defined below) of the equalizers.

These two goals can operate at odds with each other since the second goal can often be achieved at the expense of the first. In arriving at the filter forms given in eqs. 1 and 2, the magnitude of each coefficient must be constrained to be less than 2.0.

The procedure discussed below does not necessarily have general applicability; instead, it is intended for the specific application described in this article. The justification for the procedure is that it yields the intended goals, as has been born out by experience.

The first stage in the procedure is to select an ordering of the poles and zeros of the equalizer. It should be noted that eqs. 7 and 8 can be expressed in the form

$$E_t = A_t \frac{\prod_{n=1}^{4} (1 - z_n z^{-1})}{\prod_{n=1}^{4} (1 - p_n z^{-1})} = A_t \frac{\prod_{i=0}^{1} N_i}{\prod_{i=0}^{1} D_i}.$$

$$E_r = A_r \frac{\prod_{i=0}^{1} N_i}{\prod_{i=0}^{1} D_i},$$

where N_0, N_1, D_0, D_1 are second order functions of z. If all the z_n's are real, there are six ways of forming N_i, while if at least two of the z_n's are complex, there are only two ways of forming N_i. Obviously, the same statements apply to D_i as to the p_n's.

To select the ordering of the poles and zeros of E_t and E_r, consideration must be given to the transfer functions from the input of the filter to the critical internal nodes[9] of the filter. For two cascaded biquads, there are two critical internal nodes, located at the output of the accumulator and preceding a multiplier (see Fig. 18). At the critical internal nodes, overflow must be prevented for large signals input to the filter, and over-attenuation must be prevented for small signals input to the filter. The reader should note that these goals are identical to those for DTTU loss scaling described in Section VI. The transfer functions to the critical internal nodes of two cascaded biquads are related to the N's and D's as follows:

T_1 is proportional to $1/D_1$,

T_2 is proportional to $N_1/(D_1 D_2)$.

For the system under consideration, these transfer functions are picked according to the following criterion. Each critical node transfer function is chosen such that, in the frequency range 200 to 3400 Hz, the difference between the largest value of the magnitude of the transfer function (in dB's) and the smallest value of the transfer function is as

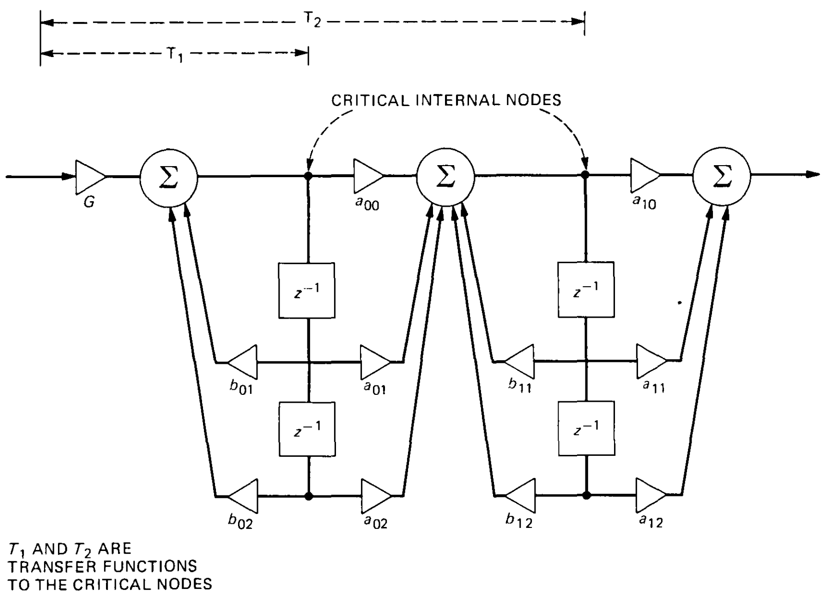


Fig. 18—Critical internal nodes of two cascaded biquads.

small as possible for the choices that are available for the ordering of poles and zeros (i.e., the response is as flat as possible under the constraints imposed).

For either T_1 or T_2 there may be six choices or two choices, depending on the number of complex poles and zeros. The rationale for the criterion given above is that it minimizes the possibility that gain-scaling of the filter sections will result in overflow or a degradation in s/n at any of the frequencies in the voiceband.

When the ordering of poles and zeros has been completed, the next step is to parcel out the gain factors A_t and A_r. The notation used for the equalizer functions is

$$E_t = G_t \frac{G_{0t}N_0}{D_0} \frac{G_{1t}N_1}{D_1} = G_t \prod_{m=0}^{1} \frac{a_{m0t} + a_{m1t}z^{-1} + a_{m2t}z^{-2}}{1 - b_{m1t}z^{-1} - b_{m2t}z^{-2}},$$

$$E_r = G_r \frac{G_{0r}N_0}{D_0} \frac{G_{1r}N_1}{D_1} = G_r \prod_{m=0}^{1} \frac{a_{m0r} + a_{m1r}z^{-1} + a_{m2r}z^{-2}}{1 - b_{m1r}z^{-1} - b_{m2r}z^{-2}},$$

where

$$G_tG_{0t}G_{1t} = A_t,$$

$$G_rG_{0r}G_{1r} = A_r.$$

For the two directions of transmission, different criteria are used to gain-scale the filter sections.

The signals incident on E_t are small, having been attenuated by the cable. Therefore, to ensure that s/n for these signals is not degraded further, G_t and G_{0t} are chosen at their maximum allowed values, determined by the coefficient magnitude constaint. Then, having picked G_t and G_{0t}, G_{1t} is

$$G_{1t} = \frac{A_1}{G_t G_{0t}}.$$

The signals incident on E_r are large compared to those incident on E_t; therefore, to ensure a minimum amount of clipping at the critical internal nodes of the filter, G_r and G_{0r} are chosen to be as small as possible and G_{1r} is chosen at its maximum allowed value. G_{1r} is chosen first. Then G_r and G_{0r} are chosen as follows:

$$G_r = G_{0r} = \sqrt{\frac{A_r}{G_{1r}}}, \tag{9}$$

unless, of course, the resulting G_{0r} is too large (which is very unlikely) to allow $G_{0r}N_0$ to satisfy the coefficient magnitude constraint. If G_{0r} is too large, as defined by eq. 9, it is chosen at its maximum value, with G_r chosen as

$$G_r = \frac{A_r}{G_{0r}G_{1r}}.$$

At this point enough information is available for determination of the final values of the equalizer coefficients as well as the pre-multipliers G_t and G_r. The next important topic is the transmission performance of facilities for which treatment is provided by the DTTU.

VIII. TRANSMISSION TREATMENT CAPABILITIES OF THE DTTU

In this section the capabilities of the DTTU in providing transmission treatment for metallic facilities is presented. Four cable cases are used in the examples. These cables were chosen because they have characteristics, either in loss or impedance, that are representative of "worst case" cables allowed in treated services. The first two cable cases were used for the example measurement results for the discussion in Section VII. For all cases, the cables were simulated in the laboratory with artificial cable kits.

8.1 Equalization

Figs. 19 through 22 show the results of loss measurements performed on a cable-DTTU system for each of the four cables. Each figure contains two plots, labeled "*a*" and "*b*". These plots are of

(*i*) unequalized receive direction loss (*a*) and

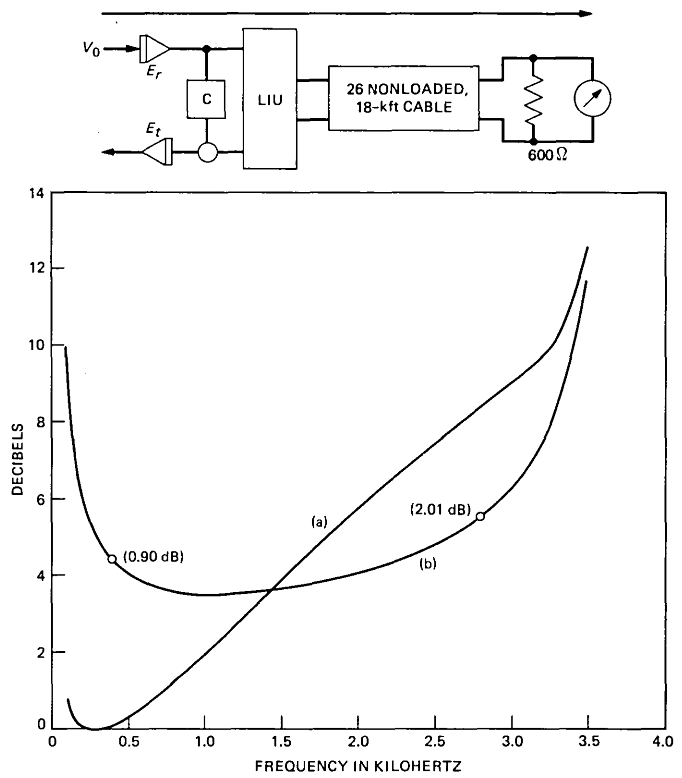


Fig. 19—Loss measurements from receive port to end of cable 1. Plot (a) represents unequalized receive direction loss, and Plot (b) represents equalized receive direction loss.

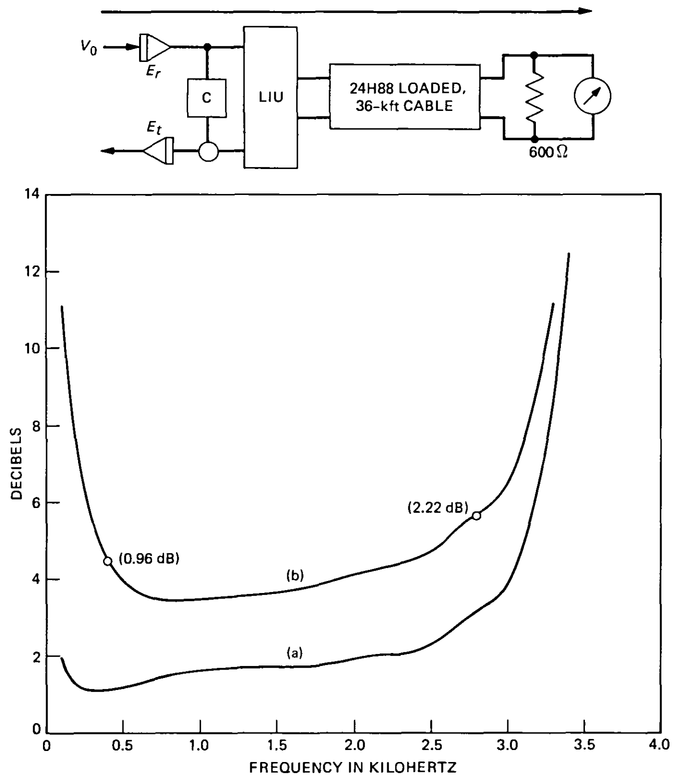


Fig. 20—Loss measurements from receive port to end of cable 2. Plot (a) represents unequalized receive direction loss, and Plot (b) represents equalized receive direction loss.

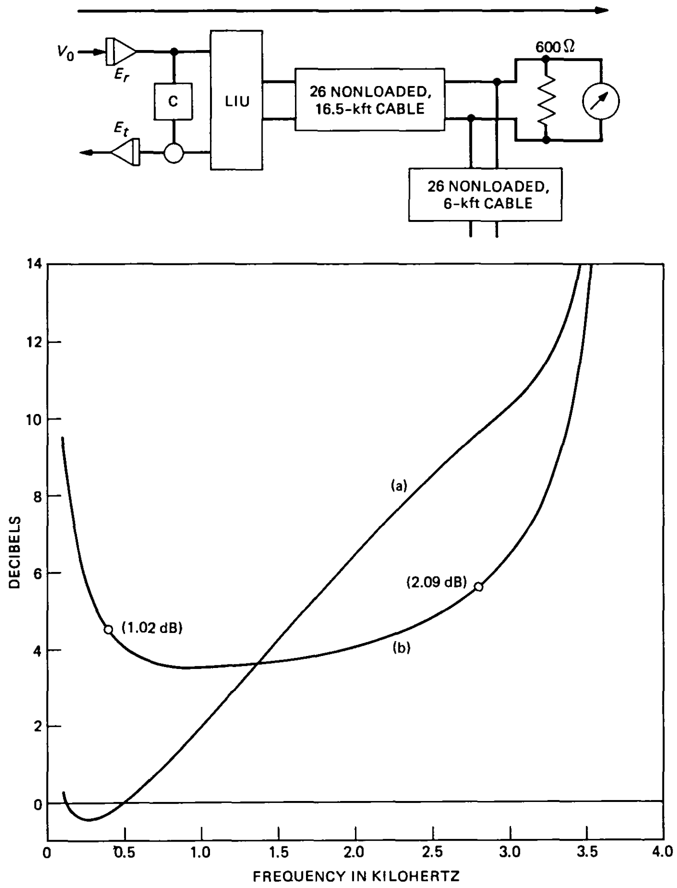


Fig. 21—Loss measurements from receive port to end of cable 3. Plot (a) represents unequalized receive direction loss, and Plot (b) represents equalized receive direction loss.

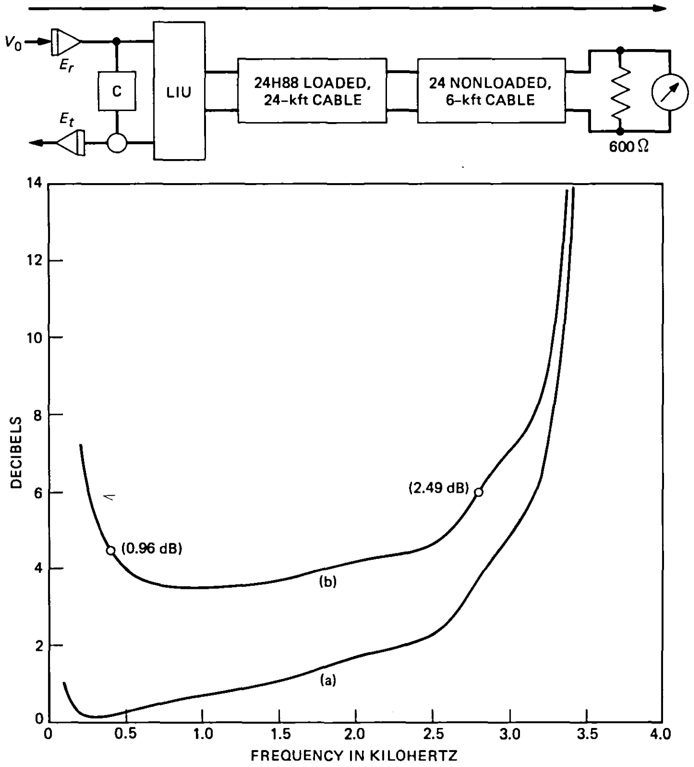


Fig. 22—Loss measurements from receive port to end of cable 4. Plot (a) represents unequalized receive direction loss, and Plot (b) represents equalized receive direction loss.

(*ii*) equalized receive direction loss (*b*).
For each cable, the equalized loss for the transmit direction is identical to the equalized loss for the receive direction, except below 200 Hz where the transmit direction loss is greater because of the low-frequency roll-off of the anti-aliasing filter. The unequalized transmit direction loss is approximately 10 dB greater than curve "*a*" for frequencies above 200 Hz.

For each curve labeled "*b*", the roll-off at 400 and 2800 Hz is displayed. As is evident, the roll-off results are close to the 1-dB objective at 400 Hz and the 2-dB objective at 2800 Hz (see Section VII). The 1-kHz loss objective for each facility is 3.5 dB. In the laboratory, the high-quality equalized performance shown for the four example cables has been consistently achieved for a large selection of other cables.

8.2 Echo cancelation and balance

Figures 23 through 26 each show the results of three separate measurements of loss from port 3 to port 2 of the DTTU, with each of the example cables in turn attached to the 2-wire port of the unit. For all measurements, the cable termination is 600 ohms in series with 2.16 uF, one of the standard terminations used for VF transmission systems. For the measurement results labeled "*a*", the equalizers were set to unity and the canceler was set to zero, leaving only the effect of the fixed compromise canceler. For the measurement results labeled "*b*", the equalizers were also set to unity, but the canceler was set to the result obtained through the minimization algorithm discussed in Section VII. For the measurement results labeled "*c*", equalizers and canceler were set to the results obtained through the minimization algorithms.

In Figs. 23 through 26, note that, on the whole, the effectiveness of the canceler increases with frequency (see curves "*b*"). This type of response counteracts the gain of the equalizers which also increases with frequency for the nonloaded cables. The low-frequency effectiveness of the canceler could be improved by increasing the number of taps. However, increasing the number of taps is not necessary since the roll-off in the equalizers at these low frequencies yields the desired improvement in echo performance. The curves labeled "*c*" illustrate the echo performance of the facility when full treatment is applied. Obviously, the 15- to 18-dB echo objective (see Section II) is easily met for all facilities. The excellent performance shown for the example cables has been achieved for a large number of cables studied in the laboratory.

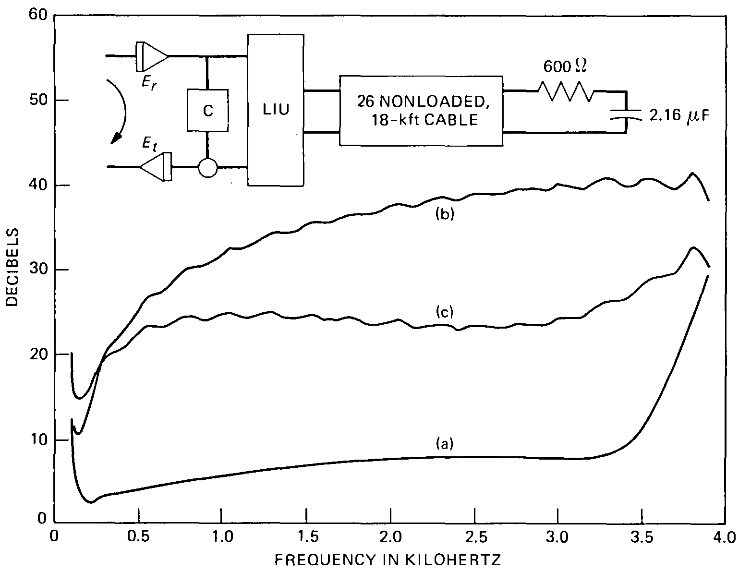


Fig. 23—Loss measurements from receive port to transmit port (cable 1). Plot (a) represents equalizers set to unity, canceler to zero. Plot (b) represents equalizers set to unity, canceler set to result obtained with minimization algorithm. Plot (c) represents equalizers and canceler set to results obtained with minimization algorithms.

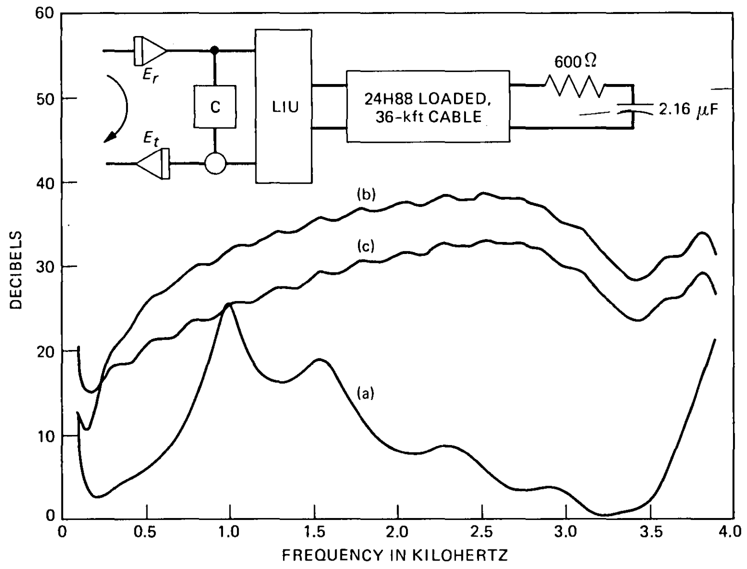


Fig. 24—Loss measurements from receive port to transmit port (cable 2). Plot (a) represents equalizers set to unity, canceler to zero. Plot (b) represents equalizers set to unity, canceler set to result obtained with minimization algorithm. Plot (c) represents equalizers and canceler set to results obtained with minimization algorithms.

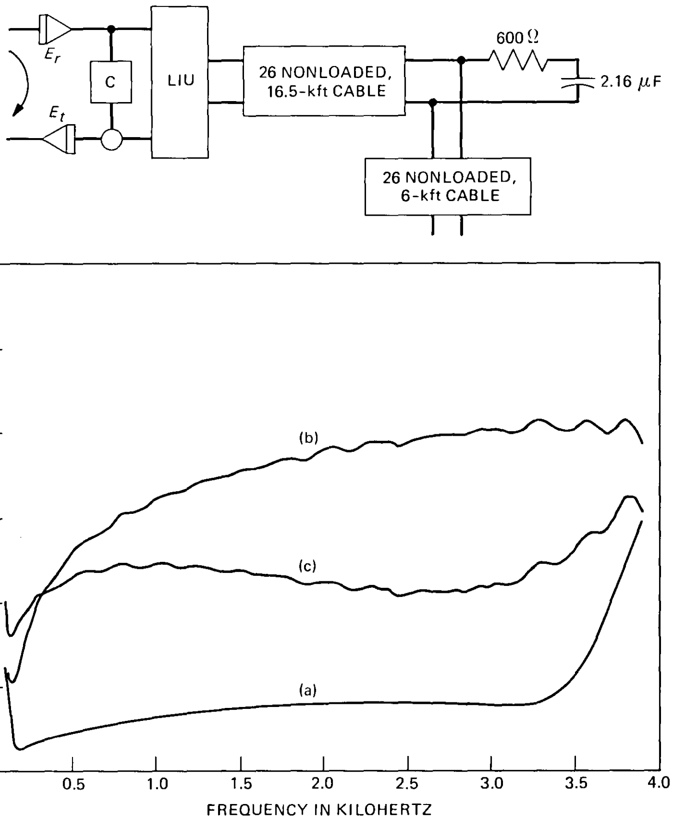


Fig. 25—Loss measurements from receive port to transmit port (cable 3). Plot (a) represents equalizers set to unity, canceler to zero. Plot (b) represents equalizers set to unity, canceler set to result obtained with minimization algorithm. Plot (c) represents equalizers and canceler set to results obtained with minimization algorithms.

IX. CONCLUSIONS

It is clear from the studies reported in this article that the DSP is capable of providing gain, equalization, and echo canceling for VF special services. Techniques have been developed for digital filter synthesis and implementation and for incorporating the DSP into a practical system. Applications to specific systems depend only on the issues of cost and power consumption relative to other techniques.

X. ACKNOWLEDGMENTS

The authors wish to thank the following people for the valuable contributions they made to this project: D. C. Watkins, R. J. Sanferrare, R. J. Gallant, R. L. Overstreet, J. Aagesen, M. R. Aaron, J. R. Boddie, and S. M. Walters. Special thanks go to D. C. Watkins and R.

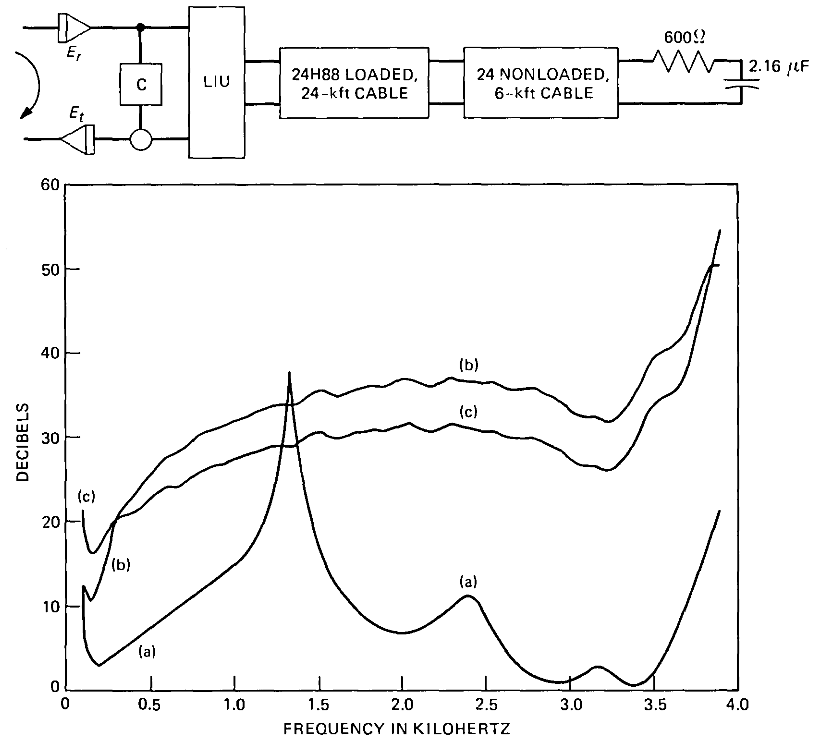


Fig. 26—Loss measurements from receive port to transmit port (cable 4). Plot (a) represents equalizers set to unity, canceler to zero. Plot (b) represents equalizers set to unity, canceler set to result obtained with minimization algorithm. Plot (c) represents equalizers and canceler set to results obtained with minimization algorithms.

J. Sanferrare for their continuing technical and administrative contributions and to R. J. Gallant, who wrote the measurement programs.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. *Telecommunications Transmission Engineering*, Winston-Salem, North Carolina: Western Electric Company, Inc., Technical Publications.
3. J. F. Kaiser, "Digital Filters," *System Analysis by Digital Computers*, F. F. Kuo and J. F. Kaiser, New York: John Wiley, 1966, Chapter 7.
4. J. F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filters," Proc. 3rd Allerton Conf. Circuit System Theory, October 20–22, 1965, pp. 621–33.
5. A. V. Appenheim and R. W. Shafer, *Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1975.
6. K. Steiglitz, "Computer Aided Design of Recursive Digital Filters," *IEEE Trans. Audio Electroacoustics, AU-18* (June 1970), pp. 123–9.
7. R. Fletcher and M. J. D. Powell, "A Rapidly Convergent Descent Method for Minimization," Computer J., *6*, No. 2 (1963), pp. 163–8.

8. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, New Jersey: Prentice-Hall, 1978.
9. L. B. Jackson, "On the Interaction of Roundoff Noise and Dynamic Range in Digital Filters," B.S.T.J. *49*, No. 2 (February 1970), pp. 159–84.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Speech Synthesis

By M. R. BURIC, J. KOHUT, and J. P. OLIVE

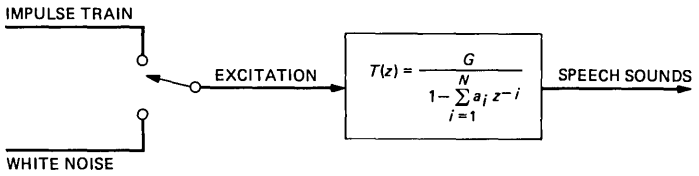
(Manuscript received June 10, 1980)

This paper describes a device that is capable of synthesizing speech in real time and is based on the digital signal processor chip. The device performs a function of a twelfth-order linear prediction coding synthesizer, and as such represents a linear dynamic system approximation of the vocal tract. In this model, short time segments of the speech waveform are derived as output of a system driven by a pseudo-periodic impulse sequence for voiced sounds, or by white noise for unvoiced sounds. The time-varying nature of the system is derived from the input information presented to the device for every new pitch period. Interfacing of the device to standard microprocessors is easy, so that the synthesizer can conveniently be integrated into larger systems.

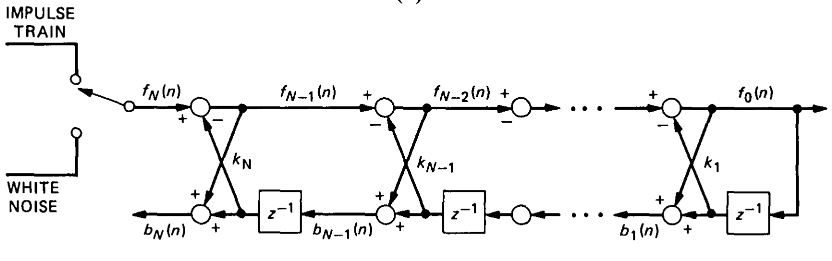
I. INTRODUCTION

Speech synthesis is one of several promising areas of application for the digital signal processor (DSP) chip described in this issue of the *Bell System Technical Journal.* Its computational power, low cost, and easy interfacing are the properties that allow a design of a stand-alone speech synthesizer with very few components outside the DSP chip. Such a synthesizer can be used in a variety of devices intended for providing new services in a business environment, as well as in future residential services.

One of the most successful ways to synthesize speech is based on a linear predictive coding (LPC) model of a vocal tract. See Refs. 1, 2, and 3. In this model, the vocal tract is approximated by a linear dynamic system driven by impulse sequences for voiced sounds, or by white noise for unvoiced sounds (Fig. 1a). The difference equation for such a representation is



(a)



(b)

Fig. 1a—Polynomial form of transfer function.
Fig. 1b—Lattice form of transfer function.

$$s(n) = \sum_{i=1}^{i=N} a_i\, s(n-i) + Gu(n). \tag{1}$$

In this discrete description of a linear system, $s(n)$ is the signal at time instant n, $\{a_i;\ i = 1, 2, \cdots, N\}$ is the set of linear prediction coefficients, $u(n)$ is the system input, and G is the gain coefficient.

A pitch synchronous synthesizer is one in which a new set of coefficients $\{a_i, G\}$ is presented to it for each new pitch period (10 milliseconds average for male voices). The coefficients are held constant for the duration of the pitch interval. This assumes that the system properties are relatively slow-varying with respect to this time scale. The variation is provided by an outside source of information, and it can be obtained at the source either by analyzing real speech in an LPC analyzer, or by complete synthesis based on phonological rules. In the first case, the synthesizer functions as a receiver in a vocoding system, while in the second case, it is a final stage in a speech synthesis system. In addition to the LPC coefficients $\{a_i\}$ and gain coefficient G, the only other variable needed for speech synthesis is the pitch period of the impulse excitation in the case of voiced sounds, or a duration of noise excitation in the case of unvoiced sounds.

The input-output transfer function of a linear system can be preserved under a set of equivalence transformations performed on the $\{a_i\}$ parameters, which yield various representations of the same system. See Refs. 2 and 3. This fact is used to advantage by selecting a representation that is least sensitive to parameter perturbations and errors in finite length arithmetic. In addition, it should provide an easy test of the stability of the system. One such representation is the lattice form of a linear system, described by the following set of difference equations (Fig. 1b),

$$\begin{aligned} f_{m-1}(n) &= f_m(n) - k_m\, b_{m-1}(n-1) \\ b_m(n) &= k_m\, f_{m-1}(n) + b_{m-1}(n-1) \\ m &= N, N-1, \cdots, 1, \end{aligned} \tag{2}$$

where $\{f_i$ and $b_i\}$ are forward and backward signals at an i-th lattice stage, and $\{k_i\}$ are relfection coefficients. The output of the system is $\{f_0(n)\}$, and the input is $\{f_N(n)\}$. Even though this form requires more multiplications per sample output, it has a very important property that all the reflection coefficients belong to the interval $(-1, 1)$ in a stable system.

A device that performs the function of a pitch synchronous synthesizer of the twelfth order in the lattice form has been designed and built around the digital signal processor chip. The device synthesizes speech in real time with an output sampling rate of 10 kHz. It is intended to be used in conjunction with some external device capable of providing the necessary system description for every new pitch period. This information is transmitted to the synthesizer in the form of a fifteen-word message, shown in Fig. 2, consisting of a header word used for synchronization and error recovery, a number representing the pitch period of excitation, an excitation amplitude, and finally the system parameters. The parameters are given as 15-bit reflection coefficients, or as 8-bit log-area parameters.

The basic synthesizer may be interfaced to the outside world in a number of ways, and this will be discussed.

II. DESCRIPTION OF THE SYNTHESIZER

The synthesizer block diagram is shown in Fig. 3. The main components are a digital signal processor chip, an interface for the input

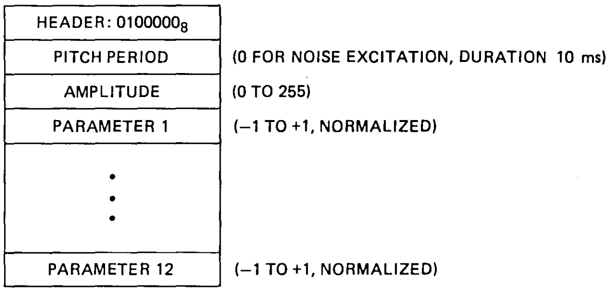


Fig. 2—Message format.

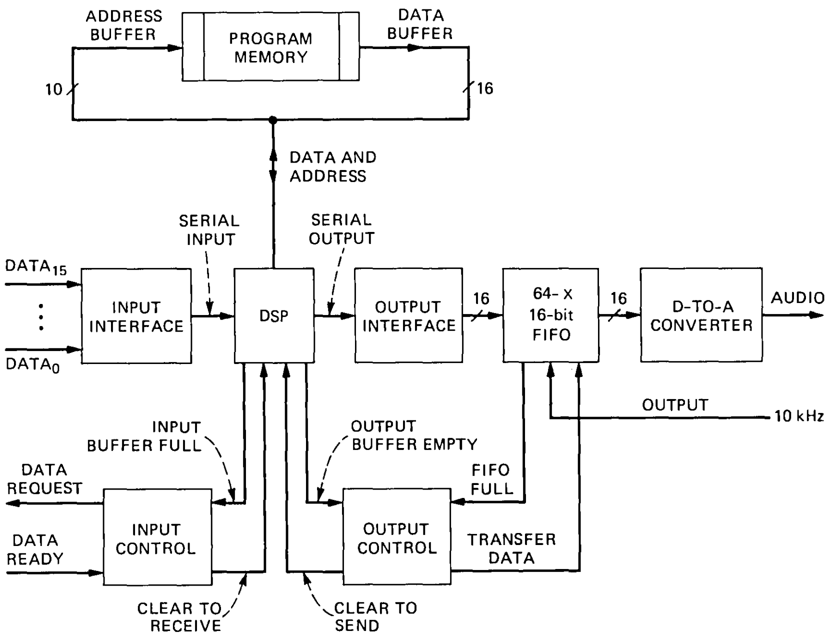


Fig. 3—Synthesizer block diagram.

data, and an output interface which includes a first-in-first-out buffer memory (FIFO). The output of the 64-word FIFO is converted to an analog signal through a digital-to-analog (D-to-A) converter.

The processor derives the synthesis program from a read-only memory (ROM), which is presently external to the processor, but could be contained in the processor's internal memory. The computation within the DSP is done in an arithmetic unit that operates on a 20-bit and a 16-bit operand. Notably, it includes an efficient multiplier and a 40-bit accumulator. These provide a dynamic range which is sufficient for the synthesis application. The parallel and pipelined architecture of the processor maintains a high computational throughput rate.

The processor utilizes a multiplexed address and data bus for accessing instructions/data stored in external memory. To transfer instructions or data from the memory, the processor places an address on the bus during the first half of the bus cycle. An address register is used in the synthesizer to latch the address, which specifies the appropriate memory location. The data from the memory is latched into an external data register, and transferred into the processor during the second half of the bus cycle.

The input interface of the synthesizer consists of a parallel-in serial-out shift register and associated control logic. Data requests from the processor are passed to the outside source of information, and when a word is received, it is transferred to the processor bit-serially.

The output interface contains a serial-to-parallel converter, first-in-first-out memory buffer, and associated control logic. The output of the DSP is a bit-serial stream, which is shifted into the serial-in parallel-out shift register. The sample is then transferred from the shift register to the FIFO, where it joins the queue of previously computed samples. A sample is taken out of the FIFO queue every 100 microseconds under control of a 10-kHz clock, and applied to the digital-to-analog converter.

The role of the FIFO memory is to even out the differences in the three mutually asynchronous processes that are taking place during synthesis. The first process is the input of synthesis information into the DSP, the second is the computation of the speech samples, and the third is the output of the samples to the D-to-A converter. The input process is largely dependent on the host computer and its transmission capabilities. The use of the output FIFO relaxes the transmission requirements. Without the FIFO, the data would be required in a burst mode for each pitch period. Each burst would consist of 15 words transmitted within one sample period of 100 μs. However, since the FIFO queue contains the accumulated output samples for up to 64 sample periods, the input data rate is effectively decreased to 15 words every 6.4 milliseconds. The computational process within the DSP is asynchronous with respect to the output sampling rate of one sample/100 microseconds. The FIFO allows the DSP to compute new samples at full speed by providing the capability of saving the samples until they are needed. This queuing mechanism requires that the average computation time is less than the interval between output requests. The device meets this requirement, and in fact, the DSP performs the computations faster than required most of the time. This implies that there are times when the FIFO becomes full, at which point the computation will be suspended until output permission is granted to the DSP. This happens when a word is taken out of the FIFO.

III. THE SYNTHESIS PROGRAM

From the above description of the synthesizer, it is clear that it can be used for a number of different synthesis schemes, with a variety of speech representation algorithms (LPC, formant, etc.).

The data rate required by the synthesizer is a very important parameter for a practical implementation of a voice response device. The data rate is a function of a speech production model used by the synthesizer, and of a coding scheme used to represent model parameters in a segment of speech. A discussion of quantization schemes of LPC parameters and their effects to the quality of synthesized speech is given in Ref. 2. Even though a significant decrease in the input data rate is possible with these approaches, we will describe an implementation that does not employ parameter quantization or interpolation. Such a scheme is useful when the device is used in a synthesis by rule system. Other applications may require some form of input data compression. The same physical device may be used in such cases, the only difference would be in the DSP program.

So far, we implemented two versions of the synthesis program for the synthesizer. Both of them employ the lattice form of a linear system. They differ only in the input data representation, one of them requires 15-bit reflection coefficients, and the other accepts 8-bit log-area parameters. Clearly, the data rate in the second program is much lower than in the first, with only a slight decrease in speech quality. In this program the input data is converted into reflection coefficients by a table look-up procedure. The relationship between the reflection coefficients and log-area parameters is given by

$$k_i = \frac{2^{A_i - A_{i+1}} - 1}{2^{A_i - A_{i+1}} + 1}, \tag{3}$$

where $\{A_i\}$ is a set of log-area parameters. This relationship is implemented in such a way that for each $A_i - A_{i+1}$ there is a value of k_i in a look-up table. Since the log-area parameters are specified by 8-bit numbers, the table contains 256 entries. Once the conversion is made, both programs function in the same way.

There are three major tasks that the program performs repetitively during the synthesis. The first task is obtaining the data for the synthesis of every new pitch period. The input is handled jointly by the program and the input interface. When the program requests data input, the interface obtains it from the host computer, and transfers it into the DSP. The protocol used by the interface is described in the next section. This procedure is repeated for each data request by the DSP, until all parameters describing the next pitch period have been transferred from the host computer.

Because of possible data transfer errors, a mechanism is provided for error recovery. Each pitch period requires a header word and 14 parameters. The program will proceed with the synthesis only if the header is received. This procedure is a sufficient guard against missed or inserted data words. In the worst case, one pitch frame will be incorrectly synthesized. Without the procedure, any inserted or deleted data word would create a permanent synchronization offset with serious perceptual consequences.

The second task of the synthesis program is to compute the speech samples by utilizing the input information and eq. (2). The program makes a decision to synthesize voiced or unvoiced sounds on the basis of the pitch value. If the pitch period is encoded as a zero, the program simulates a transfer function driven by white noise for 100 samples (10 ms). The noise is computed in the program as a pseudo-random sequence of the length 8192. Otherwise, the input to the transfer function is a single pulse at the beginning of the pitch period, and the number of produced samples is equal to the specified pitch value. The amplitude information is used for scaling the noise value in case of unvoiced sounds, or for scaling the impulse amplitude for voiced sounds.

As soon as a sample is computed, it is output to the FIFO memory, and the program continues with the computation of the next sample. This final task of the program is conditioned upon the state of the FIFO buffer, if the buffer is full the processor waits until a word is removed.

IV. INTERFACE DETAILS

The control signals, which facilitate data transfers between the synthesizer and a host processor, consist of a data request signal generated by the device, and a data ready line activated by the host. These two control signals are sufficient to define a complete communication protocol with the synthesizer, so that the device can easily be integrated into a larger system.

The sequence of events that occurs during the synthesis procedure is shown in Fig. 4. When the DSP requests new data, the interface logic raises the data request signal. The host processor monitors this request, and places a new word on the data lines. When the data is stable, the host processor generates an edge-justified data ready signal. The word is latched in the input shift register 100 ns later. Once the input word is latched, a signal clear to receive is sent to the DSP. The DSP then generates the shift-clock pulses necessary to transfer the word into its input buffer. When the shifting is completed (at 400-ns/bit rate), the data request signal goes low, until the DSP makes another request for a new word.

The output of the DSP is enabled by means of a clear-to-send signal

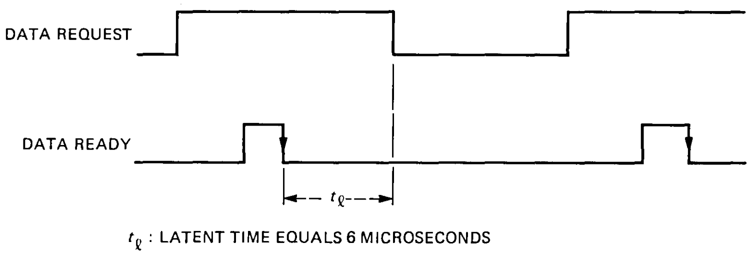


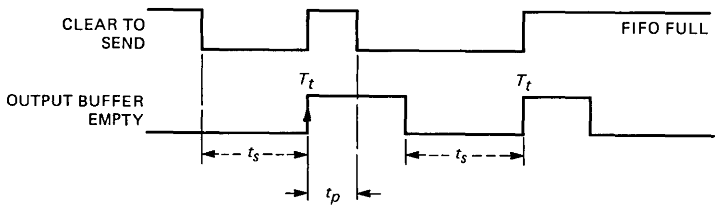
Fig. 4—Input protocol.

(CTS). This signal is granted to the DSP by the FIFO memory. The only times when this signal is not granted are when the FIFO queue is full, and for a short period after the word is placed in the queue. The latter is on the order of 4 microseconds, and is a result of internal data propagation in the FIFO. The output protocol is shown in Fig. 5. If the CTS is granted, the transfer of the data from the DSP to the FIFO memory is done in two phases. In the first phase, the DSP outputs a bit-serial data stream into the output shift register, and in the second phase, the information is transferred from the shift register into the FIFO. This transfer is done upon receipt of a positive transition of the signal Output Buffer Empty, generated by the DSP at the end of output shifting. If the FIFO memory becomes full, the CTS is not granted until a sample has been taken from the head of the queue and placed into the D-to-A converter.

There are two other variations of the basic synthesizer circuit that have been tested also. The first one is a slightly enhanced version of the synthesizer which includes an additional first-in-first-out memory buffer at the input of the system, shown in Fig. 6. The motivation for this input queuing mechanism is again based on the fact that the input process, which feeds the synthesizer with the system coefficients, is asynchronous with respect to the pitch frames. This configuration is especially useful when the host processor has to compute the reflection coefficients needed for synthesis in real-time, while the synthesizer is processing previously obtained parameters. Typically, the time to compute the coefficients has some variance, and this variance is compensated for by the "elasticity" of the input buffer.

The second variation of the basic circuit contains no FIFO buffers. It demonstrates a minimal synthesizer configuration, containing only 15 integrated circuits in addition to the DSP chip. It performs well when the host processor is capable of providing the input data at the required rate.

The synthesizer can easily be connected to standard microcompu-



t_s: SHIFT TIME EQUALS 6 MICROSECONDS
T_t: POINT OF TRANSFER
t_p: PROPAGATION THROUGH FIFO, EQUALS 4 MICROSECONDS

Fig. 5—Output protocol.

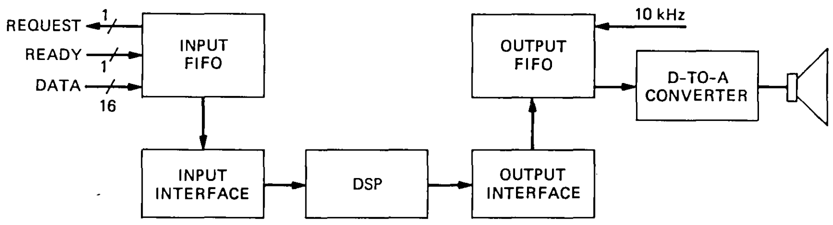


Fig. 6—Synthesizer with input FIFO memory.

ters. Several types of connections have been tried successfully. One of them is shown in Fig. 7, where the synthesizer is contained on a single board interfaced directly to a standard microprocessor bus. The input interface appears as a single memory location in the address space of the processor, and the data is transferred to it by a single "move" instruction. The input FIFO buffer may or may not be implemented. If it is, then the synthesizer interrupts the processor only when the buffer is empty. Without the buffer, the processor is interrupted for each data transfer.

Another type of interface, shown in Fig. 8, contains a synthesizer and direct-memory-access (DMA) circuitry. The iput and output buffers are not needed in this configuration, since the synthesizer obtains the data by accessing the processor memory. The processor sets up a DMA transfer by providing an address and a data count to the synthesizer board. It is interrupted only when the specified number of words have been processed by the synthesizer.

The third way of connecting the synthesizer to a microprocessor is by means of a standard parallel interface board, a standard accessory. In this case the synthesizer is not a part of the microcomputer system, rather it is an outside device.

All of these examples show that the synthesizer may easily be included as a part of intelligent terminals that are usually built around

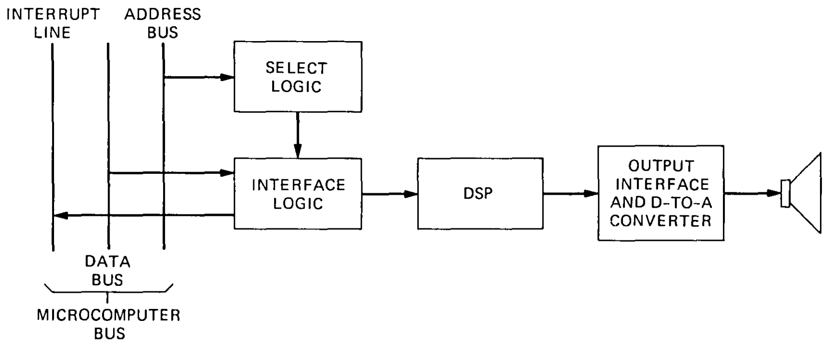


Fig. 7—Interface to microcomputer bus.

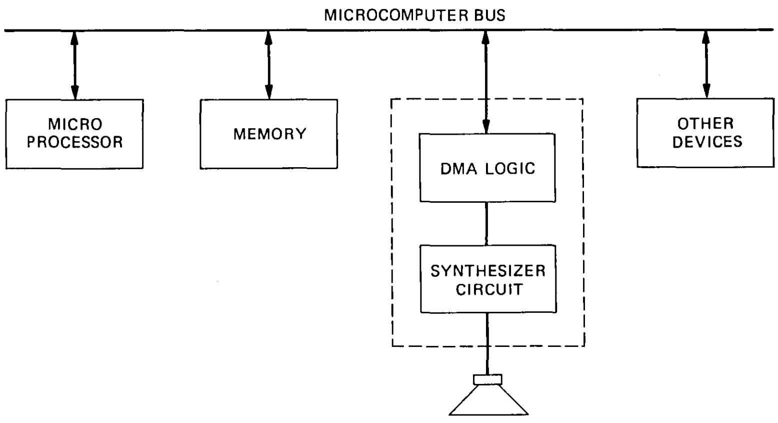


Fig. 8—Interface to microcomputer bus by DMA.

standard microprocessors. A combination of data and voice services can be provided with such configurations. Applications that require low data rate to the synthesizer would use a scheme with quantized LPC parameters, or a synthesis program that provides for interpolation of log-area parameters of longer time intervals.

V. SOFTWARE DRIVERS FOR THE SYNTHESIZER

In order to demonstrate the flexibility of the synthesizer, two types of host computers were used for implementation of the synthesizer drivers.

One is a stand-alone microcomputer, based on an LSI-11 microprocessor. This configuration includes an enhanced mini-*UNIX** operating system, and is intended for real-time speech processing experiments. A parallel interface port is used for driving the synthesizer. The hand-shaking signals required by the parallel port are in agreement with the ones provided by the synthesizer. The data request line interrupts the microprocessor, which then transmits a word to the synthesizer. At the same time a data ready pulse is issued, which is used by the synthesizer to latch the data.

A program that drives the synthesizer with the data from a file containing the speech parameters is used as:

say filename [loop] [frame]

The program reads the file **filename** into a buffer, and transfers it to the synthesizer on the basis of the protocol described earlier. The optional arguments **loop** and **frame** are used for testing purposes. If

* Registered trademark of Bell Laboratories.

a **loop** is present, the driver will synthesize the whole file, go to a mode of repeated transfers of a single pitch frame, whose sequential number is given in the argument **frame**, whose default value is the first pitch period. Once in this mode, the program allows for stepping through successive pitch frames, which is useful for studies of synthesized waveforms, and effects of finite length arithmetic to the stability of the lattice synthesizer. Also, an option is provided for changing the system coefficients during the repetitive pitch frame synthesis.

Another host computer used for driving the synthesizer is an SEL32-75 system. In this configuration, a DMA interface is utilized for transfer of pitch synchronous information to the synthesizer. The hand-shaking protocol between the DMA unit and the synthesizer input logic takes place without an involvement of the computer CPU.

VI. CONCLUSION

A real-time speech synthesizer, and several variations of it, have been designed based on the digital signal processor chip. The DSP has proven to be an important vehicle for digital signal processing applications, and speech processing in particular. A variety of sentences have been synthesized with the device. Informal listening shows no perceptual difference between the speech obtained by the synthesizer and by the general purpose computer using the same algorithm and floating point arithmetic. It has been demonstrated that the synthesizer can be easily integrated into microprocessor based systems for a number of voice response applications.

REFERENCES

1. B. S. Atal and S. L. Hanauer, "Speech Analysis and Synthesis by Prediction of the Speech Wave," J. Acoust. Soc. Amer., *50* (1971), pp 637–55.
2. L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1978.
3. J. D. Markel and A. H. Gray, *Linear Prediction of Speech*, New York: Springer-Verlag, 1976.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Sub-Band Coding

By R. E. CROCHIERE

(Manuscript received July 8, 1980)

This paper explores the use of the Bell Laboratories digital signal processing integrated circuit for digitally encoding speech or audio signals based on the sub-band coding technique. Sub-band coding represents a next level in algorithmic complexity over that of adaptive differential pulse-code modulation, discussed in a companion paper, and it has a corresponding advantage in performance. We discuss the details of a real-time, two-band sub-band coding implementation on the digital signal processor. We then comment on how this approach can be extended to more than two band designs for greater bit rate compression capability. In connection with this, we also consider some general issues involved in implementing multirate signal processing algorithms of this type on the digital signal processor.

I. INTRODUCTION

Digital encoding of speech and audio has been a topic of long-standing interest for purposes of digital communications and digital storage[1-3]. The efficiency of such encoding techniques depends strongly on the degree to which the bit rate can be reduced (compressed) without impairing the quality of the decoded signal. Typically, signals such as speech and audio have a high degree of redundancy that can be used to reduce this bit rate. Also, properties of human perception can be used to reduce the bit rate without impairing the quality of the decoded signal.

To take advantage of these properties, a considerable amount of signal processing is necessary. Thus, in the past many of these techniques have only been implemented by non-real-time computer simulations or with the aid of highly specialized digital hardware. This picture is now rapidly changing, as is exemplified by the recent Bell Laboratories digital signal processing integrated circuit (DSP).[4,5] With this device it is possible to conveniently implement, in real time, signal processing algorithms of low to medium complexity. Thus, a single DSP integrated circuit can be used to implement many of the simpler encoding algorithms and multiple DSPs can be used for some of the more complex algorithms.

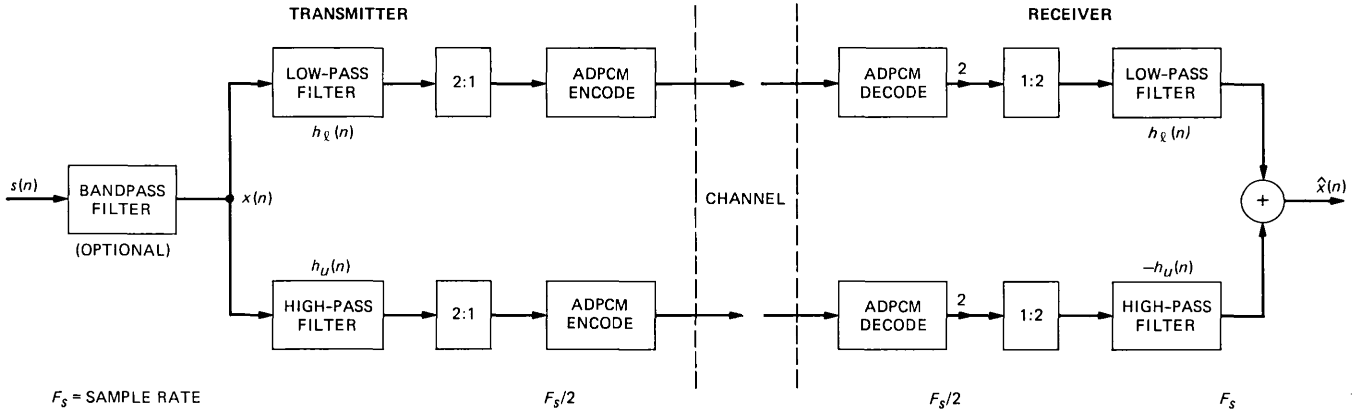
In a companion paper,[6] it is shown that the ADPCM (adaptive differential PCM) encoding algorithm, which offers a bit rate reduction factor of approximately two over conventional logarithmic companded PCM encoding (for speech), can be efficiently implemented on the DSP, and that it uses only about one-quarter of the processing capability of the device. In this paper, we report on continuing efforts towards a next level of complexity of encoding techniques on the DSP. In particular, we discuss the technique of sub-band coding (SBC).[3,7,8] Our efforts focus primarily on a two-band sub-band coder design which demonstrates the capability of the DSP for this class of algorithms. By extension of these same techniques, it is expected that more complex SBC designs on the DSP (e.g., four or more bands) with greater bit-rate compression capability will also be possible and efforts are continuing in this direction.

II. THE SUB-BAND CODER ALGORITHM

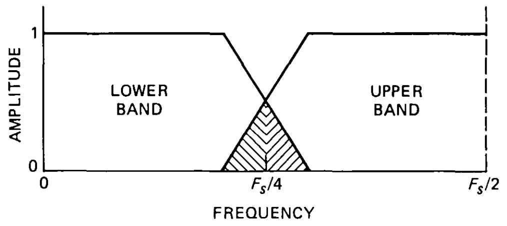
Figure 1 reviews the basic conceptual configuration for a two-band SBC design. The input signals $s(n)$ is assumed to be in digital (linear PCM) form and it may be (optionally) filtered with a bandpass prefilter for reasons to be discussed later. The output signal $x(n)$ is then divided into two equally spaced frequency bands by low-pass and high-pass filters, $h_l(n)$ and $h_u(n)$, respectively. Each sub-band signal is reduced in sampling rate by a factor of two, i.e. if F_s is the sampling rate of the input signal, $F_s/2$ is the sampling rate of the sub-band signals. The sub-band signals are then encoded with ADPCM encoders and the output bits are multiplexed for storage or transmission.

In the receiver, the sub-band signals are decoded and interpolated back to their original sampling rates with the aid of similar low-pass and high-pass filters. The sum of the two interpolated sub-band signals, $\hat{x}(n)$, is the reconstructed version of the input signal, $x(n)$, (see Fig. 1).

This process of dividing the signal into sub-bands permits each band to be encoded with a different number of bits per sample and with a independent adaptive step-size in order to obtain a better perceived quality. For telephone band speech (200 to 3200 Hz) sampled at 8 kHz, the two-band technique provides about a 3- or 4-kb/s advantage over ADPCM in the bit-rate range of 24 kb/s.[9] In another study,[3,10] it has been shown that the two-band SBC design is useful for encoding of



(a)



(b)

Fig. 1—(a) General block diagram of a two-band sub-band coder. (b) A spectral description of the sub-bands.

wider bandwidth (7 kHz) signals at bit rates which are commensurate with digital transmission rates commonly used for telephony (56 or 64 kb/s). The "commentary quality" obtained from this design is suitable for applications such as broadcast services for news, correspondence, sports, and AM radio music transmission.

Greater bit-rate compression, (or higher quality performance for the same bit rate) over that of a two-band scheme is possible with more bands.[7,8] This can be accomplished, for example, by further subdividing each of the two sub-band signals into two more sub-bands at one-quarter of the original sampling rate to produce a four-band SBC design (with equally spaced frequency bands). Alternatively, designs with octavely spaced bands are possible by successively subdividing the lower bands in a "tree structure."[11] Such designs are a logical extension of the two-band approach, and they can have a performance advantage of up to about 8 kb/s over ADPCM (in the range of 16 to 24 kb/s) for speech.[7-9,11]

In the following sections, we discuss in detail an implementation of the two-band SBC design on the DSP. In Section III, we first review some of the theoretical aspects of the design, particularly the structure of the quadrature mirror approach to the filter bank. Then, in Section IV, we consider some of the programming aspects of the design on the DSP and show how features of the DSP are used in the implementation.

III. DESIGN CONSIDERATIONS

3.1 The quadrature mirror filter bank

An important and critical aspect of the SBC design is that of the filter bank and its interaction with the sampling rate reduction (decimation) and the subsequent sampling rate increase (interpolation) of the sub-band signals. The approach used in this design is that of the quadrature mirror filter bank (QMFB).[12] In this section, we will summarize the basic concepts of the QMFB and consider a practical filter design. Later, in Section 4.4, we will discuss the implementation of the QMFB on the DSP.

The reduction of the sub-band sampling rates is necessary in order to maintain a minimal overall bit-rate in encoding these signals. This sampling rate reduction introduces aliasing terms in each of the sub-band signals. For example, in the lower band the signal energy in the frequency range above $F_s/4$ is folded down into the range 0 to $F_s/4$ and appears as aliasing in this signal, as illustrated by the shaded region in Fig. 1b. Similarly, for the upper band any signal energy in the frequency range below $F_s/4$ is folded upward into its Nyquist band $F_s/4$ to $F_s/2$. This mutual aliasing of signal energy between the upper and lower sub-bands is sometimes called interband "leakage." The amount of leakage that occurs between sub-bands is directly dependent on the degree to which the filters $h_l(n)$ and $h_u(n)$ approximate ideal low-pass and high-pass filters, respectively.

In the reconstruction process the sub-band sampling rates are increased by filling in zero valued samples between each pair of sub-band samples. This introduces a periodic repetition of the signal spectra in the sub-band. For example, in the lower band the signal energy from 0 to $F_s/4$ is symmetrically folded around the frequency $F_s/4$ into the range of the upper band. This unwanted signal energy, referred to as an "image" is filtered out by the low-pass filter $h_l(n)$ in the receiver. This filtering operation effectively interpolates the zero valued samples that have been inserted between the sub-band signals to values that appropriately represent the desired waveform.[13] Similarly, in the upper sub-band signal an image is reflected to the lower sub-band and filtered out by the filter $-h_u(n)$.

The degree to which the above images are removed by the filters $h_l(n)$ and $-h_u(n)$ is determined by the degree to which they approximate ideal low-pass and high-pass filters. Because of the quadrature relationship of the sub-band signals in the QMFB the remaining components of the images can be *exactly* canceled by the aliasing terms introduced in the analysis (in the absence of coding errors). In practice, this cancellation is obtained down to the level of the quantization noise of the coders.

To obtain this cancellation property in the QMFB, the filters $h_l(n)$ and $h_u(n)$ must be symmetrical finite impulse response (FIR) designs[14] with even numbers of taps, i.e.,

$$h_l(n) = h_u(n) = 0 \qquad \begin{array}{l}\text{for } n < 0,\\ \text{and } n \geq N\end{array} \tag{1}$$

where N, even, is the number of taps. The symmetry property implies that

$$h_l(n) = h_l(N-1-n), \qquad n = 0, 1, 2, \cdots, \quad \frac{N}{2} - 1, \qquad \text{and} \tag{2a}$$

$$h_u(n) = -h_u(N-1-n), \qquad n = 0, 1, 2, \cdots, \quad \frac{N}{2} - 1. \tag{2b}$$

The QMFB further requires that the filter in Fig. 1a satisfy the condition.[12]

$$h_u(n) = (-1)^n\, h_l(n) \qquad n = 0, 1, \cdots N-1, \tag{3}$$

which is the mirror image relationship of the filters.

With the above constraints, the aliasing cancellation property of the QMFB can be easily verified.[12] A derivation is given in the Appendix. As seen from this derivation, the filters $h_l(n)$ and $\mathrm{h}_u(n)$ must also ideally satisfy the condition

$$|H_l(e^{j\omega})|^2 + |H_u(e^{j\omega})|^2 = 1, \tag{4}$$

where $H_l(e^{j\omega})$ and $H_u(e^{j\omega})$ are the Fourier transforms of $h_l(n)$ and $h_u(n)$, respectively.

The above filter requirement of eq. (4) cannot be met exactly except when $N = 2$ and when N approaches infinity. However, it can be very closely approximated for modest values of N. Filter designs which satisfy eq. (2a) and approximate the condition of eq. (4) and the lowpass characteristic can be obtained with the aid of an optimization program. Reference 15 describes a procedure based on the Hooke and Jeeves optimization algorithm and presents a set of filter designs for values of $N = 8, 12, 16, 24, 32, 48$, and 64. Also, useful but less optimal designs can be obtained from conventional Hanning window designs.[3]

Figure 2 shows the frequency response characteristics for an $N =$ 32-tap filter design that was used in the DSP implementation and Table I gives the filter coefficients. Fig. 2a shows the magnitude of $H_l(e^{j\omega})$ and $H_u(e^{j\omega})$ expressed in dB as a function of ω and Fig. 2b shows the magnitude of the expression

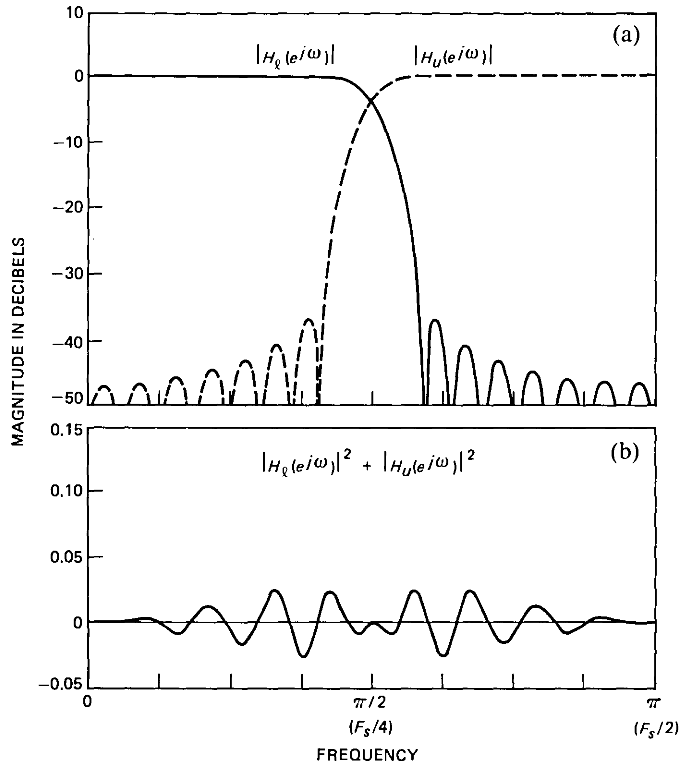


Fig. 2—Frequency response for a 32-tap quadrature mirror filter design. (a) Magnitude responses of the individual filters. (b) Magnitude response of the composite system.

Table I—Coefficients for 32-tap FIR quadrature mirror filter

$h_l(0) =$	0.002245139	$= h_l(31)$	$h_l(8) =$	−0.007961731	$= h_l(23)$
$h_l(1) =$	−0.003971152	$= h_l(30)$	$h_l(9) =$	−0.034964400	$= h_l(22)$
$h_l(2) =$	−0.001969672	$= h_l(29)$	$h_l(10) =$	0.019472180	$= h_l(21)$
$h_l(3) =$	0.008181941	$= h_l(28)$	$h_l(11) =$	0.054812130	$= h_l(20)$
$h_l(4) =$	0.000842683	$= h_l(27)$	$h_l(12) =$	−0.044524230	$= h_l(19)$
$h_l(5) =$	−0.014228990	$= h_l(26)$	$h_l(13) =$	−0.099338590	$= h_l(18)$
$h_l(6) =$	0.002069470	$= h_l(25)$	$h_l(14) =$	0.132972500	$= h_l(17)$
$h_l(7) =$	0.022704150	$= h_l(24)$	$h_l(15) =$	0.463674100	$= h_l(16)$

$$|H_l(e^{j\omega})|^2 + |H_u(e^{j\omega})|^2$$

expressed in dB as a function of ω. As can be seen from Fig. 2b, the requirement of eq. (4) is satisfied to within ± 0.025 dB which is more than satisfactory for good SBC performance. The above filter design is based on the "32 D" design.[15]

This concludes our discussion of the QMFB conditions and the filter design. In Section 4.4 we discuss how the mirror image relationship of eq. (3) is used to advantage in the DSP implementation.

3.2 The ADPCM coders

The adaptive differential PCM (ADPCM) coders in the two-band SBC are based on the algorithm by Cummiskey, Jayant, and Flanagan[16] and they use the robust form of the step-size adaptation by Goodman and Wilkinson.[17]

A detailed description of this algorithm is given in a companion paper.[6] Therefore, in this section we will only briefly outline the form of the algorithm to identify relevent parameters and refer the reader to Ref. 6 for specifics.

Figure 3a shows a simplified block diagram of the ADPCM algorithm. The input (decimated) sub-band signal is denoted as $y(n)$. A predicted estimate of this signal, $p(n)$, is subtracted from $y(n)$ to produce the difference signal

$$e(n) = y(n) - p(n). \tag{5}$$

This difference signal is then quantized with an adaptive step-size quantizer to produce the code word $I(n)$ and the decoded difference signal $\hat{e}(n)$.

The step-size of the quantizer $\Delta(n)$ is adaptively varied according to the relation

$$\Delta(n) = (\Delta(n-1))^{\gamma} \cdot M(I(n-1)), \tag{6}$$

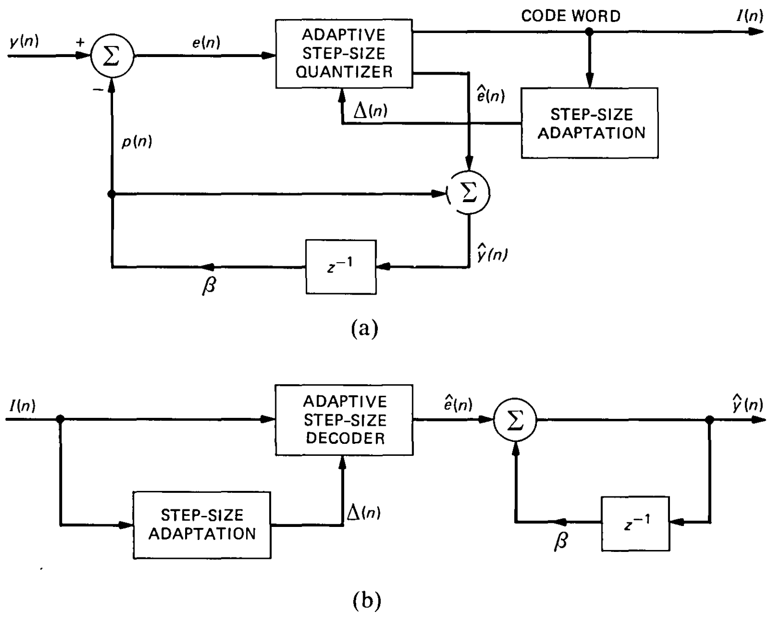


Fig. 3—General block diagram of the ADPCM coders. (a) Encoder. (b) Decoder.

where $\Delta(n-1)$ is the step-size and $I(n-1)$ is the code word at the previous sample time $n-1$. The parameter γ is a number in the range.

$$0 \le \gamma \le 1, \tag{7}$$

and, typically, has a value of $\gamma = 0.98$. It is used to introduce a limited memory to the step-size adaptation algorithm to mitigate the effects of channel errors.[17] The scale factor $M(I(n))$ is a number that depends on the code word $I(n)$. If an outermost positive or negative quantizer level is used at time $n-1$ a value of $M(\cdot)$ greater than one (typically $M(\cdot) = 2$) is used to increase the step-size for the sample time n. If a lower quantizer magnitude level is used at time $n-1$, a value of $M(\cdot)$ less than one (typically $M(\cdot) = 0.77$) is used to reduce the step-size at time n. In this way, the step-size is dynamically varied in an attempt to match the center of the quantizer characteristic to that of the rms level of the difference signal $e(n)$. The values of $M(\cdot)$ can be tailored to modify the adaptation characteristics of the quantizer. Typically, a faster attack (step-size increase) and a slower decay (step-size decrease) is preferred[10,16] for best subjective performance.

The sum of the decoded difference signal $\hat{e}(n)$ and the predictor signal $p(n)$ gives the decoded version of the input signal, denoted as $\hat{y}(n)$, i.e.

$$\hat{y}(n) = \hat{e}(n) + p(n). \tag{8}$$

This is used in the ADPCM receiver, (see Fig. 3b) to produce the decoded output signal. It is also used in the ADPCM transmitter (see Fig. 3a) to generate the predictor signal according to the relation

$$p(n) = \beta \cdot \hat{y}(n-1). \tag{9}$$

The parameter β determines the fraction of the signal $\hat{y}(n-1)$ that is used to predict the next incoming sample $y(n)$. Ideally it should be equal to the sample-to-sample correlation that exists in the signal $y(n)$.[16]

For speech, sampled at 8 kHz, it has been suggested that values of $\beta_l = 0.7$ (lower band) and $\beta_u = -0.45$ (upper band) are appropriate.[9] The negative correlation in the upper sub-band is because the frequency scale of the spectrum is inverted in the decimation process of the QMFB. For audio signals, sampled at 14 kHz, values of $\beta_l = 0.16$ and $\beta_u = -0.82$ have been suggested[3] (note that β_l and β_u are referred to as α_1 and α_2 in Ref. 3).

The number of bits per sample used to encode each sub-band is dependent on the overall bit rate of the coder. For speech, sampled at 8 kHz, a choice of 4 bits/sample for the low band and 2 bits/sample for the upper band leads to a 24-kb/s design. For audio, sampled at 14 kHz, a choice of 4 bits/sample was used in each sub-band for the 56-kb/s commentary grade coder.[3]

3.3 Prefiltering

It is sometimes desirable in SBC coding to band-limit the input signal prior to encoding. For example, in speech a substantial amount of signal energy may be present in the frequency range from 0 to 200 Hz. This energy contributes to an increased step-size and, therefore, more quantization noise in the lower sub-band. If telephone band speech (200 to 3200 Hz) is of interest, then band-limiting the input signal to this range before encoding removes the signal energy below 200 Hz and above 3200 Hz. This permits the use of a lower step-size in the bottom band and, therefore, produces less quantization noise. For audio, a similar advantage is gained from prefiltering by removing low frequency hum and turntable rumble components in the signal prior to encoding.

Figure 4 shows an example of a cascade filter structure for a sixth-order infinite impulse response (IIR) filter[14] that was used in the DSP implementation for this purpose. The coefficients for a 200- to 3200-Hz bandpass elliptic filter design (assuming an 8-kHz sampling rate) are given in Table II and the frequency response for this design is shown in Fig. 5.

In the design of IIR filters, some caution must be observed in minimizing the effects of roundoff noise, limit cycles, and dynamic

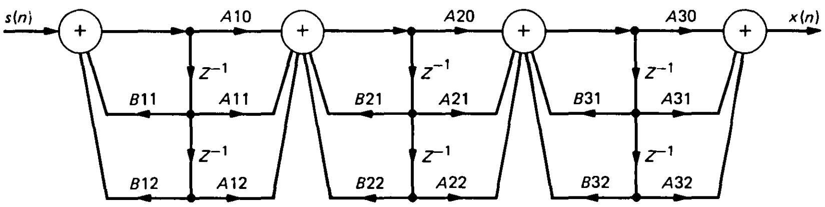


Fig. 4—Block diagram of the cascade structure for the IIR prefilter.

range constraints within the filter. This can be accomplished by observing several rules of thumb for pairing and ordering the arrangement of poles and zeroes within the cascade filter structure and also by appropriately scaling the signal from section to section within the filter structure to control the internal dynamic range. Further information on this subject can be found in Refs. 14 and 18. In general, these procedures are more critical for high-order high Q filters and less critical for low-order low Q designs.

IV. IMPLEMENTING THE SBC ALGORITHM ON THE DSP

4.1 Some general programming considerations

As seen from the above discussion there are a number of different aspects to consider in the implementation of an algorithm such as SBC on the DSP. In this section, we discuss some of these issues and point out some general programming techniques that were used. Principles such as modularization, stream processing, block processing, and double buffering will be introduced. In Sections 4.2 to 4.5 we discuss more specifically how these principles are used in the SBC software. We will assume in the following discussion that the reader is generally familiar with the DSP software.

The software development for the SBC and similar signal processing algorithms is greatly simplified by recognizing the fact that there are several well-defined operations that are being performed in the algorithm, such as filtering, coding, and sampling rate conversion. By identifying these operations and modularizing the software around them, the problem can be subdivided into a series of smaller problems.

Table II—Coefficients for sixth-order IIR bandpass elliptic filter
(200- to 3200-Hz BW, 8000-Hz sampling rate)

A10	1.0	B21	−1.44837372
A11	−1.99730706	B22	−0.746318392
A12	1.0	A30	1.0
B11	0.470839023	A31	1.95459080
B12	0.2281607545	A32	1.0
A20	0.459738676	B31	1.9053369
A21	0.0	B32	−0.92630251
A22	−0.459738676		

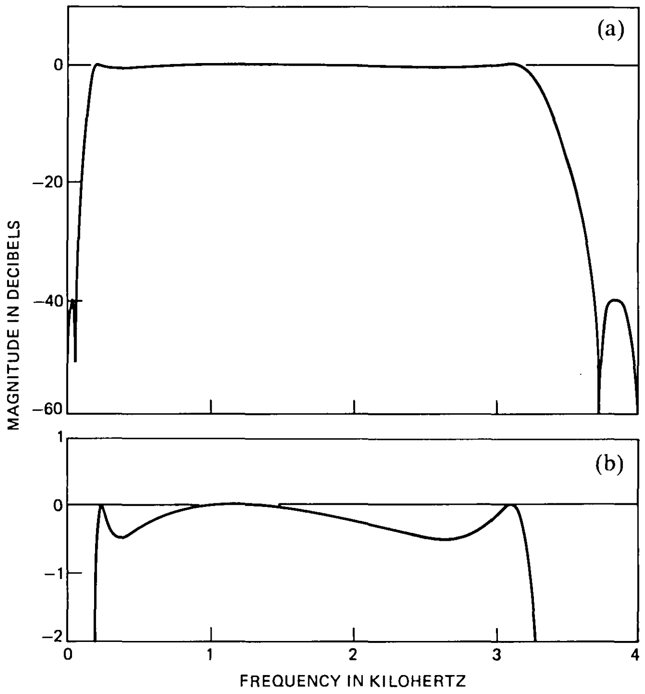


Fig. 5—Frequency response of the 200- to 3200-Hz (8000-Hz sampling rate) bandpass elliptic filter. (a) Overall response. (b) Expanded view of the passband ripples.

This modularization process consists of assigning and labeling separate parts of the DSP RAM memory for each block in the algorithm. For example, in the IIR filter the six internal (state) variables necessary for the filter, denoted as BPI[0], BPI[1], ⋯ , BPI[5], can be assigned to RAM locations using the RAM-variable definition statement

ram BPI[6].

At the beginning of each module, the address pointers of the DSP can then be initialized for that module. For example, the statement

rya = &BPI[0];

sets the DSP address pointer RYA to the first state variable in the filter. The automatic increment or decrement feature of the address pointers in the DSP can then be used to step RYA through the state variables within the module. Although it may cost a few extra lines of code, this simple, perhaps obvious, principle goes a long way toward simplifying the programming and debugging process in the DSP by unlinking the address connections between modules and generally producing more readable code.

Signal processing operations, such as filtering and coding, generally require a single input sample and produce a single output sample for each sample time. They also perform essentially the same signal processing operations for each sample time. Such operations will be referred to as stream processing operations and they can be conveniently defined and implemented in the modular fashion discussed above.

When more than one sampling rate is involved in an algorithm, the operations to be performed within a module, or the entire module itself, may be different from sample to sample. For example, if there is a 2-to-1 difference in sampling rate within the algorithm it may be necessary to perform one set of operations at the odd sample times (cycle zero) and a different set of operations (cycle one) at the even sample times in certain parts of the algorithm. For this, it is necessary to introduce the concepts of block processing in which different program modules are used for different cycles. Furthermore, interfacing stream processing modules with block processing modules may require double buffering operations and care must be used to distribute the amount of processing performed in each cycle to avoid I/O synchronization problems. These concepts will become more clear when we discuss the implementation of the QMFB and sampling rate conversion in Sections 4.4 and 4.5.

We first consider the implementation of the IIR prefilter and the ADPCM coder module. Then, we show how these modules fit within the general multirate processing framework of the SBC coder.

4.2 IIR prefilter

The IIR filter can be implemented in a straightforward stream processing manner on the DSP. Beause the DSP is especially suited for linear filtering algorithms of this type, the filter structure of Fig. 4 can be very efficiently realized with approximately one DSP instruction for each combination of a shift, multiply, and add in the structure.

Assuming that the internal state variables are stored in RAM locations BPI[0], BPI[1], $\cdots$, BPI[5], and the input signal $s(n)$ is in the P register, the following DSP instructions compute $x(n)$ according to Fig. 4 (see Table II for the filter coefficients).

```
                   rya = &BPI[0];
                   rda = &BPI[0];;
                                     a = p      p = B12**rya++;
                                     a = p + a  p = B11**rya--;
                                     a = p + a  p = A12**rya++;
*rda++ = y         w = a             a = p      p = A11**rya++;
*rda++ = w                           a = p + a  p = A10*w;
                                     a = p + a  p = B22**rya++;
```

```
                                    a = p + a   p = B21**rya--;
                                    a = p + a   p = A22**rya++;
*rda++ = y      w = a               a = p       p = A21**rya++;
*rda++ = w                          a = p + a   p = A20*w;
                                    a = p + a   p = B32**rya++;
                                    a = p + a   p = B31**rya--;
                                    a = p + a   p = A32**rya++;
*rda++ = y      w = a               a = p       p = A31**rya++;
*rda++ = w                          a = p + a   p = A30*w;
                                    a = p + a;
```

The output signal $s(n)$ appears in the A register. The filter coefficients are stored in the beginning of the program using # define statements and are inserted into the code by the assembler.

4.3 ADPCM encoder and decoders

The ADPCM encoders and decoders are also stream processing algorithms in the sense that they receive a single input sample and produce a single output sample for each sample time. However, as seen in Fig. 1, the sampling rate at which they operate in the SBC algorithm is one half of the input sampling rate. As will be discussed in the next section, this can be accomplished by a two-cycle computational structure in which the encoder and decoder for the lower sub-band are computed in one cycle time (cycle 0) and the encoder and decoder for the upper sub-band are computed in the second cycle time (cycle 1). Since each cycle time is associated with one-half of the input sampling rate, the ADPCM coders operate in a stream processing manner within this framework.

See Ref. 6 for a detailed discussion of the ADPCM algorithm and its implementation on the DSP, since essentially the same design and code have been used for the SBC algorithm.

4.4 Quadrature mirror filter bank and the multirate computational structure

Perhaps the most subtle aspect of the SBC algorithm is that it is a multirate system;[13] i.e., it has more than one sampling rate. This imposes a block processing framework on the computational structure of the system. In the next two sections, we will discuss these issues in more detail and present a computational structure for the two-band SBC. First, we will discuss the polyphase structure for the QMFB which takes advantage of its mirror image and multirate properties and results in a more efficient realization than the one implied by Fig. 1.

From the mirror image property of the QMFB described by eq. (3), note that the coefficients used for the upper and lower sub-band filters are identical, except for the signs of the odd-numbered coefficients.

This property can be used to save a factor of two in computation by sharing the computation between the filters in the manner described in Fig. 6.[12] The partial sums of products are accumulated separately for the even- and odd-filter coefficient values. The sum of these two partial sums then gives the lower sub-band signal, and their difference produces the upper sub-band signal. Since the sampling rates are one-half of the input sampling rate, an additional factor of two is gained by computing the sums of products indicated in Fig. 6 once for every other input sample. Thus, each sample is shifted two delays in the shift register of Fig. 6 before being used.

Because of this sample rate reduction, the filter structure of Fig. 6 can be divided into two parts as shown in Fig. 7. This structure is a two-band version of a more general class of multirate structures sometimes referred to as polyphase structures.[13,19] As Fig. 7 shows, the input signal is separated into two sets by a commutator. Assuming that the commutator is in the upper position at time $n = -1$, the upper branch receives odd values of $x(n)$, i.e. $x(-1)$, $x(1)$, $x(3)$, $x(5)$ $\cdots$, and the lower branch receives even values of $x(n)$, i.e. $x(0)$, $x(2)$, $x(4)$, $\cdots$. Both branches now operate at one-half of the original sampling rate. Odd values of $x(n)$ are filtered at odd sample times in the upper branch with an $N/2$ tap filter of odd valued filter coefficients. Similarly, even valued samples of $x(n)$ are filtered in the lower branch with an $N/2$ tap filter of even filter coefficients.

At the end of the even sample times, the sums and differences of the two filter outputs are taken to produce the (decimated) lower and upper sub-band signals respectively. This sum and difference amounts to a two-point DFT (a discrete Fourier transform butterfly) in the two-band polyphase framework. The purpose of the double buffer will be discussed in more detail in connection with the timing and control structure in the next section.

By careful analysis of the receiver structure of Fig. 1 a similar efficient polyphase structure can be generated for the QMFB synthesis. Alternatively, it can be generated by applying concepts of multirate

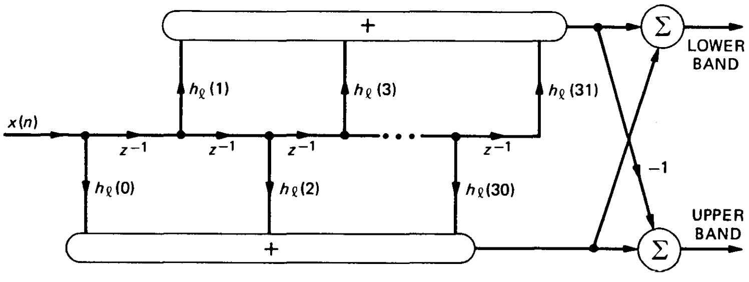


Fig. 6—Quadrature mirror filter bank structure that shares computation between upper and lower filters.

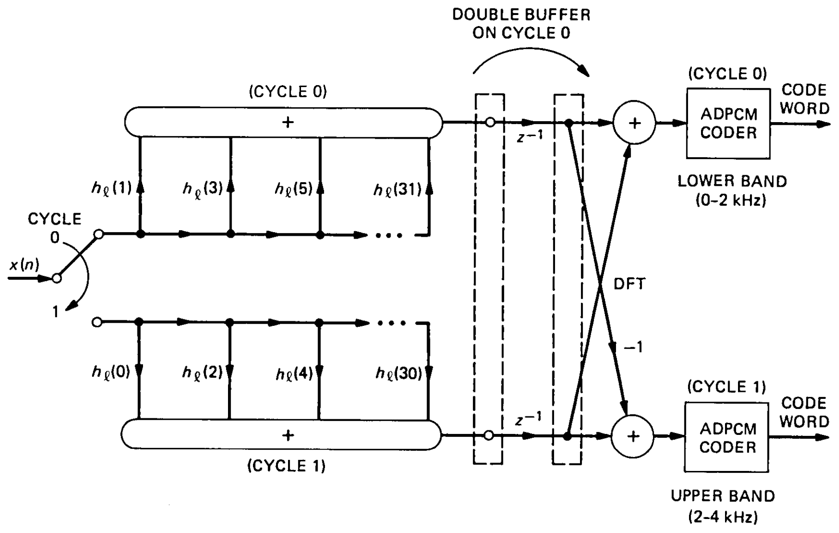


Fig. 7—Sub-band coding transmitter structure using a polyphase QMFB.

network transposition[20] to the structure of Fig. 7. The resulting structure, with either approach is shown in Fig. 8. The sum and difference (an inverse DFT) of the ADPCM decoder output signals are first computed to produce inputs for the odd and even FIR filters, respectively. At odd sample times (cycle 0) the even FIR filter coefficients (upper branch) are used to compute odd sample values of the output $\hat{x}(n)$, i.e.

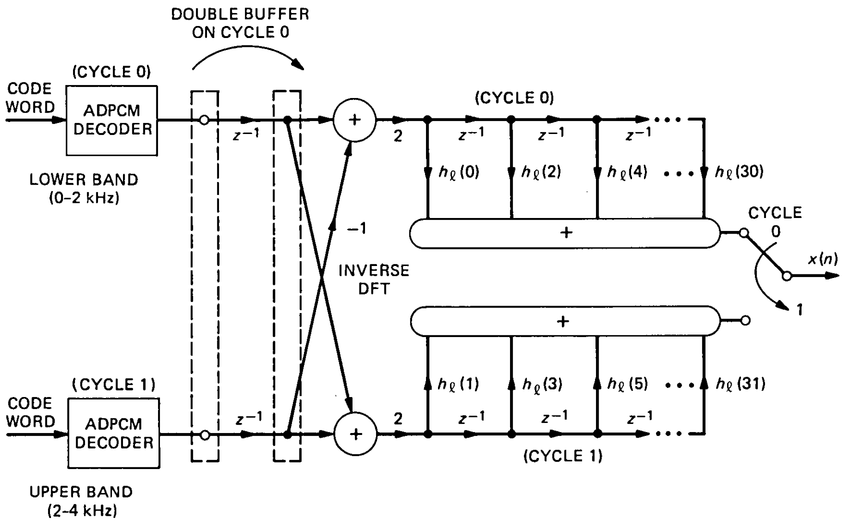


Fig. 8—Sub-band coding receiver structure using a polyphase QMFB for synthesis.

$\hat{x}(-1)$, $\hat{x}(1)$, $\hat{x}(3)$, $\cdots$. At even sample times (cycle 1), the odd FIR filter coefficients (lower branch) are used to compute even samples of the output $\hat{x}(n)$, i.e. $\hat{x}(0)$, $\hat{x}(2)$, $\hat{x}(4)$, $\cdots$.

As in the analysis structure of Fig. 7, note that one-half of the filter computation is performed at even sample times and the other half is performed at the odd sample times. Thus, the computational load is evenly distributed between even and odd time cycles. This will be discussed in more detail in the next section on the control structure.

The DSP can be very efficiently used to perform the above operations. For example, the FIR filters can be implemented with essentially one line of code per tap in the filter, plus a few setup instructions. Assuming that the state variables of the 16-tap odd coefficient filter in the upper branch of Fig. 7 are stored in RAM locations $XO[0]$, $XO[1]$, $\cdots$, $XO[15]$, and the filter input is stored in the A register, the following DSP instructions compute the filter output.

```
            rya = &XO[15];
            rda = &XO[15];;
            w = a         a = p        p = H31**rya--;
*rda-- = y                a = p        p = H29**rya--;
*rda-- = y                a = p + a    p = H27**rya--;
*rda-- = y                a = p + a    p = H25**rya--;
*rda-- = y                a = p + a    p = H23**rya--;
*rda-- = y                a = p + a    p = H21**rya--;
*rda-- = y                a = p + a    p = H19**rya--;
*rda-- = y                a = p + a    p = H17**rya--;
*rda-- = y                a = p + a    p = H15**rya--;
*rda-- = y                a = p + a    p = H13**rya--;
*rda-- = y                a = p + a    p = H11**rya--;
*rda-- = y                a = p + a    p = H9**rya--;
*rda-- = y                a = p + a    p = H7**rya--;
*rda-- = y                a = p + a    p = H5**rya--;
*rda-- = y                a = p + a    p = H3**rya--;
*rda = w                  a = p + a    p = H1*w;
                          a = p + a;
```

The coefficients $H1$, $H3$, $\cdots$, $H31$ correspond to the filter coefficients $h_l(1)$, $h_l(3)$, $\cdots$ $h_l(31)$, respectively in Table I. They are stored at the beginning of the program using # define statements and inserted into the code using the assembler. Note that the filter state variables are addressed in reverse order, i.e. $XO[15]$, $XO[14]$, $\cdots$, $XO[2]$, $XO[1]$, $XO[0]$ and that the filter input is held in the W register until coefficient $H1$ is reached, and then it is stored in location $XO[0]$. The output of the filter appears in the A register.

4.5 Control and data flow structure

So far we have discussed block diagrams and programs for individual modules within the SBC code. In this section, we consider the overall control structure and flow of data within the program. Because the SBC algorithm is a multirate system with a sampling rate ratio of two, it requires a two-cycle control structure (for a sampling rate ratio of 4, a four-cycle control structure would be needed, ect). Cycle zero is associated with odd sample times $n = 1, 1, 3, 5, \cdots$, and cycle 1 is associated with even sample times $n = 0, 2, 4, \cdots$. As seen in Figs. 7 and 8, most of the operations in the upper branches of these structures are computed in cycle 0 and most of the operations in the lower branches are computed in cycle 1. Thus, the ADPCM coder and decoder for the lower sub-band are computed in cycle 0 and the ADPCM coder and decoder for the upper sub-band are computed in cycle 1. In this way, the computational load is shared equally between both cycles. Note that the bandpass prefilter must be computed in both cycles since it is implemented at the high (input) sampling rate.

Double buffering is required in the multirate structure when data computed in one cycle is required in another cycle. For example, in Fig. 7 the outputs of the even and odd filters are computed and stored in the left buffer for cycles 0 and 1, while the DFT uses available data from the right buffer which was computed in the previous 0 and 1 cycles. At the beginning of cycle 0 (or the end of the last cycle) the data is transferred from the left buffer to the right buffer. This transfer can be accomplished with essentially no extra overhead by using the features of the DSP.

For example the DFT sum in the upper branch of Fig. 7 is computed in cycle 0 (the difference is computed in cycle 1). This is accomplished by reading data from the left buffer (RAM memory $XB[0]$ and $XB[1]$) and simultaneously transfering it to the right buffer (RAM memory $XBB[0]$ and $XBB[1]$), while the DFT sum is being performed. The following DSP instructions perform this operation:

```
                    rya = &XB[0];
                    rya = &XBB[0];;
*rda++ = y                                   p = *rya++;
*rda = y                            a = p    p = *rya;
                                    a = p + a;
```

The DFT sum appears in the A register. In cycle 1, the DFT difference output is computed from data in the left buffer (read in reverse order to accommodate the difference operation in the DSP).

Table III summarizes the sequence of operations that are performed in cycle 0 and cycle 1 of the two-band SBC software. Both the SBC transmitter and receiver are implemented in the same DSP for the sake

Table III—Control structure for two-band SBC

I. Cycle 0	A. Transmitter
	1. Double buffer
	2. DFT sum
	3. ADPCM encoder (lower band)
	4. Output (or store) code word
	5. Input one sample of $x(n)$
	6. BP prefilter
	7. FIR filter (upper branch)
	B. Receiver
	1. Double buffer
	2. IDFT sum
	3. FIR filter (upper branch)
	4. Output one sample of $\hat{x}(n)$
	5. Input code word
	6. ADPCM decode (lower band)
II. Cycle 1	A. Transmitter
	1. DFT difference
	2. ADPCM encoder (upper band)
	3. Output (or store) code word
	4. Input one sample of $x(n)$
	5. BP prefilter
	6. FIR filter (lower branch)
	B. Receiver
	1. IDFT difference
	2. FIR filter (lower branch)
	3. Output one sample of $\hat{x}(n)$
	4. Input code word
	5. ADPCM decode (upper band)
Return to I. Cycle 0.	

of demonstration and the code words are simply stored in memory and read back in the receiver. One input sample of the signal $x(n)$ is used in each cycle and one output sample of $\hat{x}(n)$ is generated in each cycle. Although the code for the transmitter and receiver are interlaced in this control structure it can be easily separated, because of the modular design, so that the transmitter and receiver can be realized in separate DSPS.

V. DISCUSSION

We have discussed an implementation of a two-band sub-band coder using the DSP. Both the transmitter and receiver have been implemented on the same DSP including a sixth-order bandpass prefilter. The algorithm was implemented according to the signal processing structures in Figs. 4, 7, 8, and the ADPCM encoder and decoder structures discussed in Ref. 6. Table III outlines the overall control structure for the algorithm.

The program uses approximately 98 percent of the 8-kHz real-time capability of the DSP running with a 5-MHz clock. It uses approximately 78 percent of the RAM and 73 percent of the ROM. If the transmitter and receiver are separated to two DSPs (in a practical situation) approximately one-half of the above processing capability and memory will be required for each DSP.

Based on the above figures the two-band commentary grade coder for audio ecoding at a 14-kHz sample rate[3] is possible using a single DSP for the transmitter and another DSP for the receiver.

Also these figures suggest that a four-band sub-band coder design which further subdivides each of the above two sub-bands into two more sub-bands by similar quadrature mirror filter techniques is realizable using one DSP for the transmitter and a second DSP for the receiver. Such designs offer greater bit-rate compression capability for speech than the two-band SBC algorithm discussed here.[7-9,11] Efforts are continuing in this direction

VI. ACKNOWLEDGMENTS

The author wishes to thank C. A. McGonegal, J. D. Johnston, R. V. Cox, and J. W. Upton for their comments and discussion in the course of this work.

APPENDIX

Aliasing Cancellation Property of the Quadrature Mirror Filter Bank[12]

Let $X_l(e^{j\omega})$ and $X_u(e^{j\omega})$ be the Fourier transforms of the lower and upper sub-band signals, respectively before decimation and $X(e^{j\omega})$ be the transform of $x(n)$. Then

$$X_l(e^{j\omega}) = X(e^{j\omega})\, H_l(e^{j\omega}), \qquad \text{and} \tag{10}$$

$$X_u(e^{j\omega}) = X(e^{j\omega})\, H_u(e^{j\omega}), \tag{11}$$

where $H_l(e^{j\omega})$ and $H_u(e^{j\omega})$ are the Fourier transforms of $h_l(n)$ and $h_u(n)$, respectively. After decimation the lower and upper sub-band signals will be defined as $Y_l(e^{j\omega})$ and $Y_u(e^{j\omega})$, respectively and can be expressed as[13]

$$Y_l(e^{j\omega}) = \tfrac{1}{2}\,[X_l(e^{j\omega/2}) + X_l(e^{j(\omega+2\pi)/2})], \qquad \text{and} \tag{12}$$

$$Y_u(e^{j\omega}) = \tfrac{1}{2}\,[X_u(e^{j\omega/2}) + X_u(e^{j(\omega+2\pi)/2})]. \tag{13}$$

Letting $U_l(e^{j\omega})$ and $U_u(e^{j\omega})$ be the interpolated lower and upper sub-band signals, respectively in the receiver, and ignoring effects of quantization because of the coders we get

$$U_l(e^{j\omega}) = 2\, Y_l(e^{j2\omega})H_l(e^{j\omega}), \qquad \text{and} \tag{14}$$

$$U_u(e^{j\omega}) = -2\, Y_u(e^{j2\omega})H_u(e^{j\omega}). \tag{15}$$

Finally the output signal $\hat{X}(e^{j\omega})$, the transform of $\hat{x}(n)$ in Fig. 1a, can be expressed as

$$\hat{X}(e^{j\omega}) = U_l(e^{j\omega}) + U_u(e^{j\omega}). \tag{16}$$

Combining eqs. (10) to (16) gives the input to output relation

$$\hat{X}(e^{j\omega}) = X(e^{j\omega})[H_l^2(e^{j\omega}) - H_u^2(e^{j\omega})]$$
$$+ X(e^{j(\omega+\pi)})[H_l(e^{j\omega})H_l(e^{j(\omega+\pi)}) - H_u(e^{j\omega})H_u(e^{j(\omega+\pi)})]. \quad (17)$$

The first term in this expression expresses the desired signal component of $\hat{X}(e^{j\omega})$ and the second term expresses the undesired aliasing component. The cancellation of this aliasing component can be observed by transforming eq. (3) to get

$$H_l(e^{j\omega}) = H_u(e^{j(\omega+\pi)}), \quad (18)$$

and applying this condition to eq. (17). It can be easily verified that the second term cancels leaving

$$\hat{X}(e^{j\omega}) = X(e^{j\omega})[H_l^2(e^{j\omega}) - H_l^2(e^{j(\omega+\pi)})]. \quad (19)$$

From the symmetry property in eq. (2), it can be shown that the frequency response of $H_l(e^{j\omega})$ can be expressed in the form

$$H_l(e^{j\omega}) = |H_l(e^{j\omega})|e^{j\omega(N-1)/2}. \quad (20)$$

Recalling that N is even and applying this condition to eq. (19), leads to the expression

$$\hat{X}(e^{j\omega}) = X(e^{j\omega})[|H_l(e^{j\omega})|^2 + |H_l(e^{j(\omega+\pi)})|^2]e^{j\omega(N-1)}. \quad (21)$$

In the above expression, the term $e^{j\omega(N-1)}$ implies that there is an $N-1$ sample delay between $\hat{x}(n)$ and $x(n)$. Furthermore, it can be seen from eq. (21) that if $\hat{x}(n)$ is to be a (delayed) replica of $x(n)$ then $H_l(e^{j\omega})$ must satisfy the requirement that

$$|H_l(e^{j\omega})|^2 + |H_l(e^{j(\omega+\pi)})|^2 = 1, \quad (22)$$

or equivalently

$$|H_l(e^{j\omega})|^2 + |H_u(e^{j\omega})|^2 = 1. \quad (23)$$

REFERENCES

1. J. L. Flanagan et al., "Speech Coding," IEEE Trans. Commun., *COM-27,* No. 4 (April 1979), pp. 710–37.
2. J. M. Tribolet and R. E. Crochiere, "Frequency Domain Coding of Speech," IEEE Trans. ASSP, *ASSP-27,* No. 5 (October 1979), pp. 512–30.
3. J. D. Johnston and R. E. Crochiere, "An All Digital Commentary Grade Sub-band Coder," J. of the Audio Eng. Society, *27,* No. 11 (November 1979), pp. 855–65.
4. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
5. J. S. Thompson and J. R. Boddie, "An LSI Digital Signal Processor," Proc. 1980 IEEE Int. Conf. ASSP (April 1980), pp. 383–5.
6. J. R. Boddie et al., "Digital Signal Pressure: ADPCM Coding," B.S.T.J., this issue.
7. R. E. Crochiere, S. A. Webber, and J. L. Flanagan, "Digital Coding of Speech in Sub-Bands," B.S.T.J., *55,* No. 7 (October 1976), pp. 1069–85.
8. R. E. Crochiere, "On the Design of Sub-Band Coders for Low-Bit-Rate Speech Communications," B.S.T.J., *56,* No. 5 (May–June 1977), pp. 747–70.

9. R. V. Cox, "A Comparison of Three Coders to be Implemented on the Digital Signal Processor," B.S.T.J., this issue, Part 2.
10. J. D. Johnston and D. J. Goodman, "Digital Transmission of Commentary-Grade (7 kHz) Audio at 56 or 64 kb/s," Proc. IEEE Int. Conf. ASSP. Proc. (1979), pp. 442–4.
11. A. J. Barabell and R. E. Crochiere, "Sub-band Coder Design Incorporating Quadrature Filters and Pitch Prediction," in Proc. IEEE Int. Conf. ASSP (1979), pp. 530–3.
12. D. Esteban and C. Galand, "Application of Quadrature Mirror Filters to Split Band Voice Coding Schemes," Proc. IEEE Int. Conf. ASSP (1977), pp. 191–5.
13. R. E. Crochiere and L. R. Rabiner, "Interpolation and Decimation of Digital Signals—A Tutorial Review," Proc. IEEE, *69,* No. 3 (March 1981), pp. 300–31.
14. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing,* New York: Prentice Hall Inc., 1975.
15. J. D. Johnston, "A Filter Family Designed for use in Quadrature Mirror Filter Banks," Proc. IEEE Int. Conf. ASSP (April 1980), pp. 291–4.
16. P. Cummiskey, N. S. Jayant, and J. L. Flanagan, "Adaptive Quantization in Differential PCM Coding of Speech," B.S.T.J., *52,* No. 7 (September 1973), pp. 1105–18.
17. D. J. Goodman and R. M. Wilkinson, "A Robust Adaptive Quantizer," IEEE Trans. Commun., *COM-23* (November 1975), pp. 1362–5.
18. L. B. Jackson, "Roundoff-Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," IEEE Trans. Audio Electroacoust., *AU-18* (June 1970), pp. 107–22.
19. M. G. Bellanger, G. Bonnerot, and M. Coudreuse, "Digital Filtering by Polyphase Network: Application to Sample Rate Alteration of Filter Banks," IEEE Trans. ASSP, *ASSP-24,* No. 2 (April 1976), pp. 109–14.
20. T. A. C. M. Claasen and W. F. G. Mecklenbrauker, "On the Transposition of Linear Time-Varying Discrete-Time Networks and its Application to Multirate Digital Systems," Philips J. Res. *23* (1978), pp. 78–102.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Tone Generation

By D. L. FAVIN

(Manuscript received September 10, 1980)

Two programs have been written for the recently developed single-chip digital signal processor (DSP) integrated circuit that enable it to function as a tone generator in testing transmission systems. One program is based on the table look-up method and the other on the Maclaurin expansion method. A DSP tone generator based on the look-up method can generate up to 12 components and is suitable for all transmission testing applications. A generator based on the Maclaurin expansion method is limited to less than four components and is particularly applicable in two-tone testing.

I. INTRODUCTION

A tone generator is required for a number of transmission system tests. The tones required include *TOUCH-TONE*® signaling, multi-frequency (MF) signaling, a milliwatt source, centralized automatic reporting on trunks (CAROT) responses and CAROT test tones. In addition, a tone consisting of 21 components each having a settable phase and level is required for a fast Fourier transform (FFT)-based system. In all cases, the tone is to be transmitted on a 4-kHz digital channel, with a sample every 125 μs (Nyquist sampling rate of 8 kHz). Programs have been written for the DSP that enable it to function as a tone generator. The following two methods are used: (*i*) table look-up, and (*ii*) Maclaurin expansion.

The table look-up method consists of storing in read only memory (ROM) the trigometric values of $\sin(n\theta)$, where $0 \le n \le N$, and N is determined by both frequency granularity requirements and harmonic distortion considerations. At each sampling instant, the value of the sample is taken from the appropriate location of the ROM table and scaled for its desired level. If more than one tone is desired, each component is independently determined and then all component values added together to form the sample value.

The Maclaurin expansion,

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots,$$

can be used to determine successive sample values by direct calculation. The number of terms to be considered is a function of desired harmonic purity and the time allowed for the computation.

II. TABLE LOOK-UP METHOD

A program has been written for the DSP that uses the table look-up method to implement a tone generator. It produces up to six independently specified components in the 0- to 4-kHz range with an 8-bit, μ-255 encoded output. A diagnostic is included that checks most of the DSP features used by this routine. The program accepts successive 16-bit serial input instructions that:

(*i*) Specify tone generation or a diagnostic.

(*ii*) Specify the number of components, from 1 to 6, that comprise the tone. When the number of components specified is 0, quiet tone is generated. When 7 components are specified, milliwatt tone is generated.

(*iii*) Specify the frequency of a component, from 0 to 4 kHz, in 1/8-Hz steps.

(*iv*) Specify the phase of a component from 0 to 360° in 1/8° steps.

(*v*) Specify the level of a component as a fraction of full output, from 0 to 1 in steps as fine as 2^{-15}.

(*vi*) Steps 3, 4, and 5 are repeated for each component comprising the tone.

Tone generation starts with a reset, except when phase continuity is required. The output may be changed to a new frequency without loss of samples or phase continuity when the information about the new frequency is presented to the input buffer.

The diagnostic ends with the S0-bit set to 1, if all tests pass.

2.1 Algorithm

A full 2π sine table, with 512 16-bit entries is used. A $\pi/2$ sine table could be used but would require keeping track of quadrants and signs, which takes time and reduces the number of tones that could be generated.

The table entries are so arranged that the entry for 0° is at an address called &TABLE. Two hundred fifty-five successive locations above this contain the positive half of the sine wave trignometric entries. Two hundred and fifty-six entries below this address contain the negative going entries. Two additional entries, corresponding to the first nonzero positive entry, at location (&TABLE − 257) and 0 at (&TABLE + 256) are also included. Hence, the ROM table is as shown in Fig. 1.

Table accesses above and below the normal range may occur because of the effect of rounding. Electing truncation in the DSP in the initialization of this process would avoid the need for adding these memory locations, but their presence assures continuity of sample values.

The frequency of a component is determined by the number of table entries $\Delta\phi$ stepped per sample, if the samples being generated at a particular frequency are

$$y(nT) = A \sin(2\pi f nT),$$

where f is the desired frequency, T is the sampling interval, and n takes on integer values, then the phase increment between successive values is

$$2\pi f T = 2\pi \frac{f}{f_s},$$

where $f_s = \frac{1}{T} = 8000$ Hz is the sampling frequency.

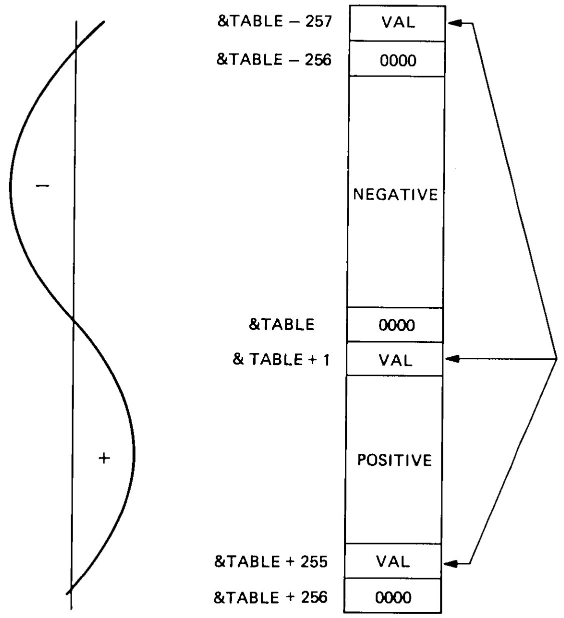


Fig. 1—Read only memory table.

Since a full 2π sine wave table is present in ROM and is represented by 512 entries, the phase difference between successive entries is $2\pi/512$. The number of entries, $\Delta\phi$, to be stepped per sample is, therefore,

$$\Delta\phi = \frac{2\pi f/8000}{2\pi/512} = 0.064f.$$

To minimize processing time, modulo 512 arithmetic is used so that no sign or table limit checking is required. Incrementing around the table is automatic. To accomplish this, the table is entered at &TABLE + N_ϕ, where N_ϕ is modulo 512 and is stored in register Y as shown in Fig. 2. As desired, N_ϕ is never larger than 255 and its sign alternates. Assuming an initial phase of 0, N_ϕ starts at zero and is then incremented by $\Delta\phi$. Thus, for p passes

$$N_\phi = \sum_{l=0}^{p-1} l\Delta\phi.$$

Given f, $\Delta\phi$ is obtained in modulo 512 by transferring as shown in Fig. 3.

To generate a frequency close to the desired value, it is important to have as much precision as possible in $\Delta\phi$. In this case, after W register truncation, it is 11 bits. The precision also has a marked influence on harmonic distortion.* The harmonic distortion products are shown to be 50 dB below the fundamental for this arrangement.

2.2 Phase continuity

To change the frequency, $\Delta\phi$ must be changed, but phase continuity can be preserved. In one sample interval, the new frequency is fetched and the new $\Delta\phi$, subsequently calculated, is placed into the appropriate random access memory (RAM) location. The calculations then proceed from the previously accumulated phase.

2.3 Flow chart

A flow chart of the DSP program is given in Fig. 4. In essence, the overall flow breaks down into the following areas shown in the chart:

(1a) Input a data control word, the frequencies, phases and levels, or

(1b) Input a diagnostic control word and run. The output is a pass or fail indication.

Because of the dynamic nature of the DSP RAM, it is necessary during the input data routine, to refresh the RAM locations.

Once the input data process is completed, one of three paths is possible:

(2a) Calculate sample values for $1 \leq n \leq 6$ tones.

* W. N. Fabricius, unpublished work.

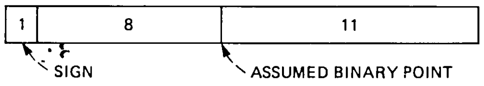


Fig. 2—Modulo 512 stored in Y register.

(2b) Calculate samples for quiet tone, $n = 0$.
(2c) Calculate samples for a 0 dBm, milliwatt, $n = 7$.
(3a) Continue calculating with phase continuity.
(3b) Continue with old input data.

2.4 Experimental results

The spectrum of the DSP output signal after decoding is given in Figs. 5, 6, and 7. Calculations, based on the work of Fabricius, indicate that the harmonic production is mainly because of the μ-255 format.

Some synthesized waveforms are represented by the waveforms in Figs. 8 and 9.

The measured phase jitter is 0.65° at 1000 Hz and 0.75° at 1004 Hz. The AM jitter experienced is less than 0.1 dB.

The linearity of the output signal fell well within the μ-255 format requirements. (For this experiment, a linear output was delivered by the program.)

2.5 Extensions

The program, without μ-255 encoding, has generated 13 components within a 125-μs sampling interval. A total of 21 components could be generated by synchronizing two DSPs, one producing 10 and the other

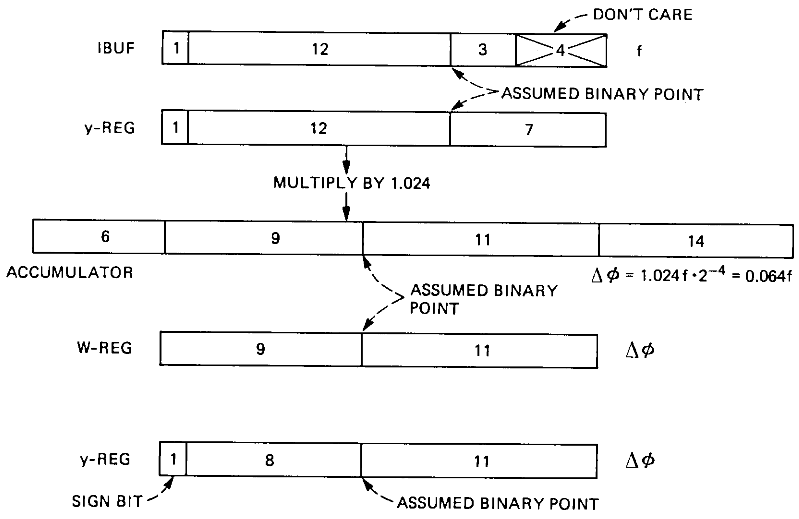
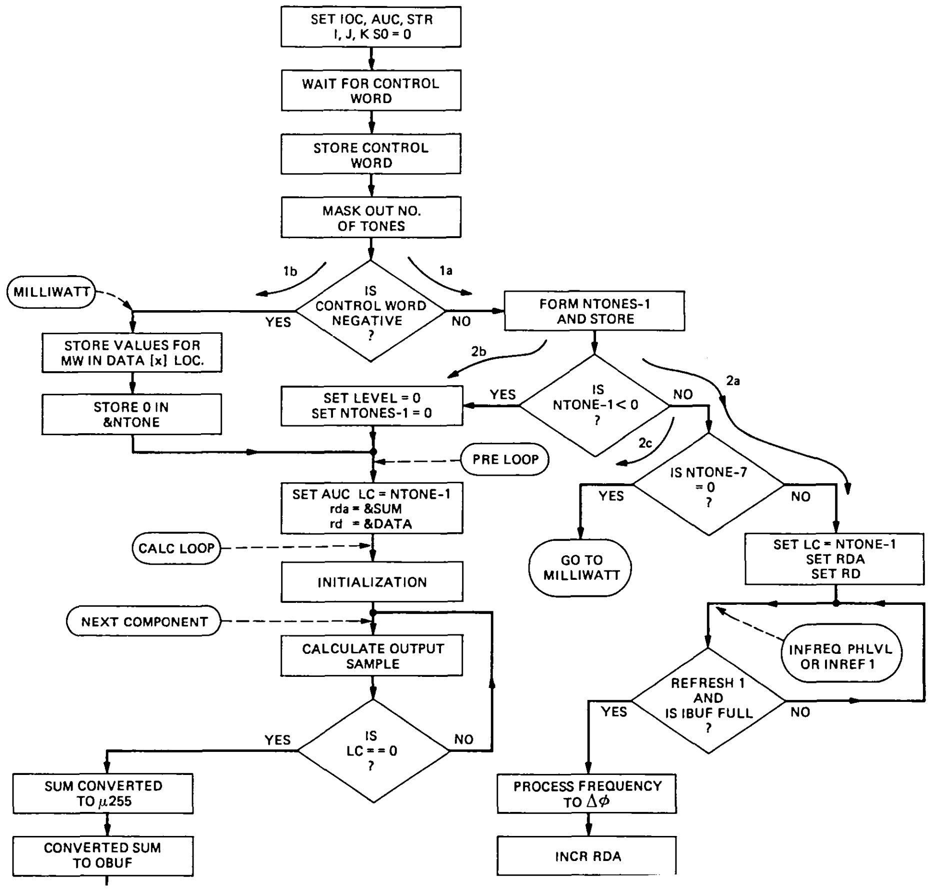


Fig. 3—Register manipulation for modulo 512.


SET IOC, AUC, STR I, J, K S0 = 0
WAIT FOR CONTROL WORD
STORE CONTROL WORD
MASK OUT NO. OF TONES
1b
1a
MILLIWATT
IS CONTROL WORD NEGATIVE ?
YES
NO
FORM NTONES-1 AND STORE
STORE VALUES FOR MW IN DATA [x] LOC.
2b
2a
SET LEVEL = 0 SET NTONES-1 = 0
YES
IS NTONE-1 < 0 ?
NO
STORE 0 IN &NTONE
2c
PRE LOOP
IS NTONE-7 = 0 ?
YES
NO
SET AUC LC = NTONE-1 rda = &SUM rd = &DATA
CALC LOOP
GO TO MILLIWATT
SET LC = NTONE-1 SET RDA SET RD
INITIALIZATION
NEXT COMPONENT
INFREQ PHLVL OR INREF 1
CALCULATE OUTPUT SAMPLE
REFRESH 1 AND IS IBUF FULL ?
YES
NO
IS LC = = 0 ?
YES
NO
SUM CONVERTED TO μ255
PROCESS FREQUENCY TO Δφ
CONVERTED SUM TO OBUF
INCR RDA

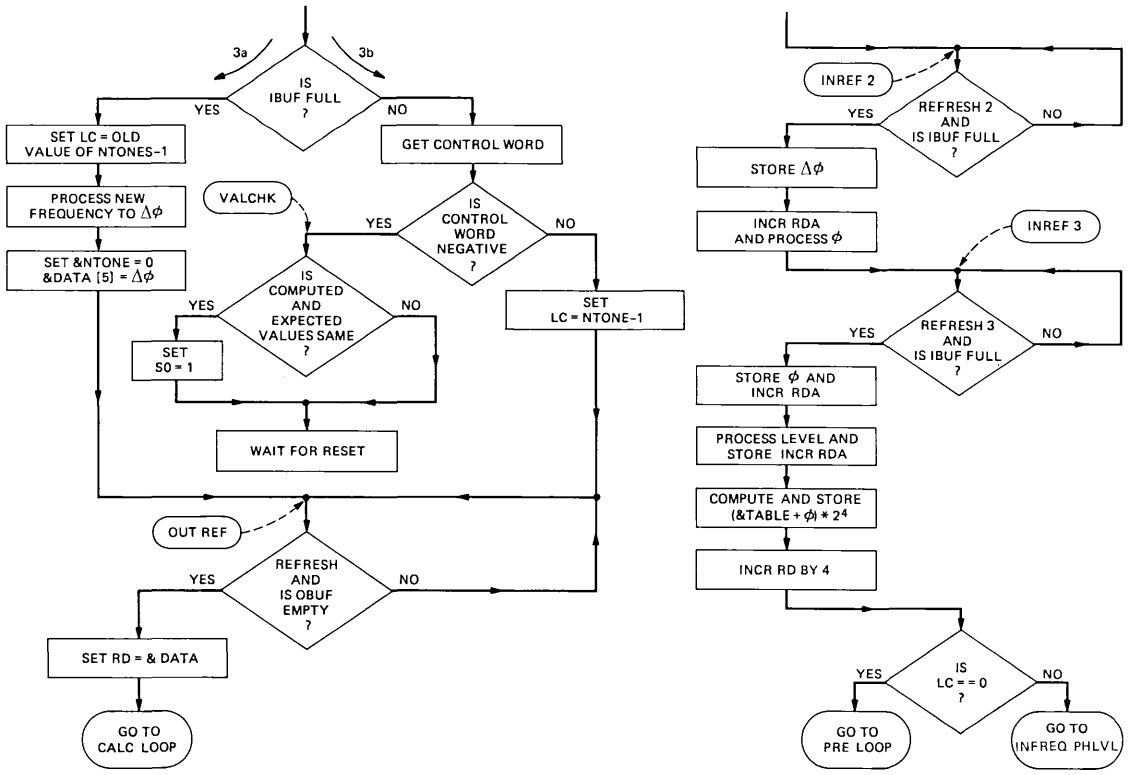

3a
3b
IS IBUF FULL ?
YES
NO
SET LC = OLD VALUE OF NTONES-1
GET CONTROL WORD
PROCESS NEW FREQUENCY TO Δφ
VALCHK
IS CONTROL WORD NEGATIVE ?
YES
NO
SET &NTONE = 0 &DATA [5] = Δφ
IS COMPUTED AND EXPECTED VALUES SAME ?
YES
NO
SET LC = NTONE-1
SET S0 = 1
WAIT FOR RESET
OUT REF
REFRESH AND IS OBUF EMPTY ?
YES
NO
SET RD = & DATA
GO TO CALC LOOP
INREF 2
REFRESH 2 AND IS IBUF FULL ?
YES
NO
STORE Δφ
INCR RDA AND PROCESS φ
INREF 3
REFRESH 3 AND IS IBUF FULL ?
YES
NO
STORE φ AND INCR RDA
PROCESS LEVEL AND STORE INCR RDA
COMPUTE AND STORE (&TABLE + φ) * 2⁴
INCR RD BY 4
IS LC = = 0 ?
YES
NO
GO TO PRE LOOP
GO TO INFREQ PHLVL

Fig. 4—Flow diagram for DSP tone-generator program.

11. The first DSP would serially transmit its 20-bit sample value, after generation, to the second DSP. This would take 8 μs at a 2.5-Mb/s rate, and could be accomplished during the time the second DSP was computing its 11th component. The second DSP would then take this value in its input buffer and add it to its sample value, convert the result to μ-255, and transmit it.

III. MACLAURIN EXPANSION

A program has been written for the DSP that can evaluate a truncated Maclaurin expansion in less than 30 μs. Frequency resolution is not limited by quantizing as in the table look-up method, but by the accumulator and the product registers in the DSP.

3.1 Algorithm

Each component is generated by a truncated Maclaurin series expansion. The choice of the number of terms included in the approxi-

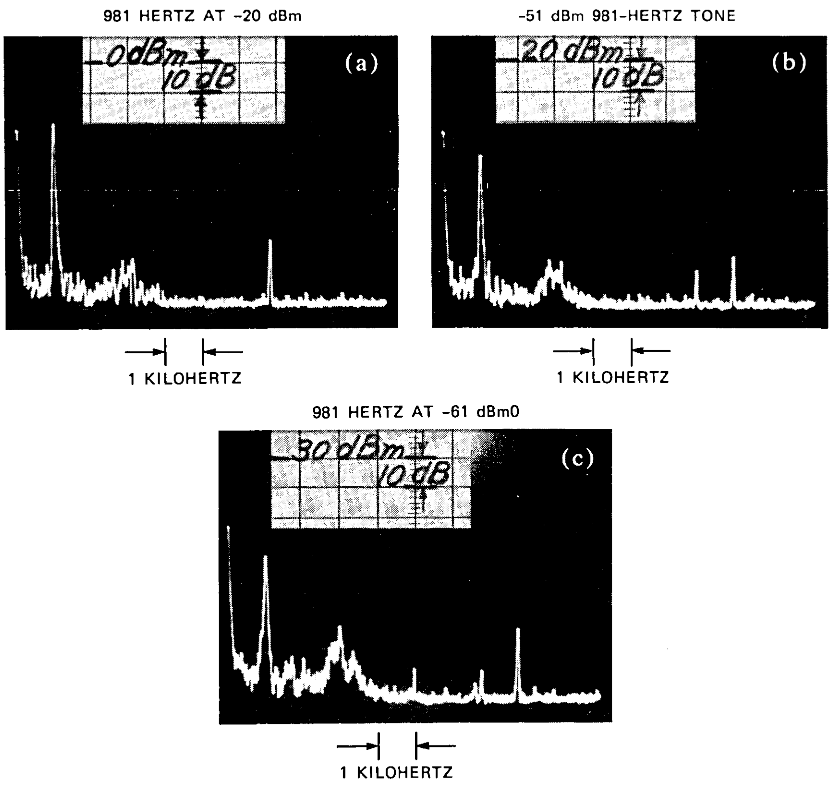


Fig. 5—Tone spectrum—low level (981 Hertz).

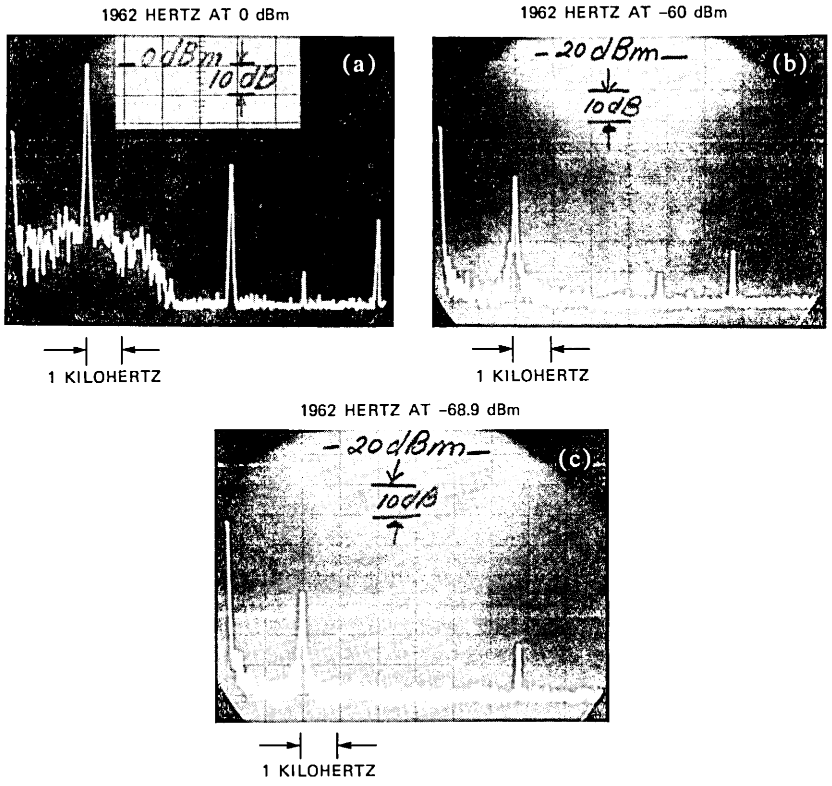


Fig. 6—Tone spectrum at 1962 Hertz.

mation and the point about which the series is expanded all affect the harmonic distortion.

Consider the following Maclaurin series expansions around zero,

$$\sin(\theta) = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

$$\cos(\theta) = 1 - \frac{\theta^2}{2!} - \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots.$$

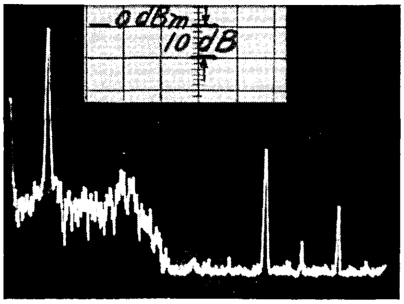
The accuracy of the approximation, of course, becomes better when more and more terms of the expansion are used. The harmonic distortion because of series truncation of the cosine series is examined in the Appendix. Let $P(\theta)$ represent the portion of the series that is retained, and $R(\theta)$ represent the remaining terms so that

$$\cos\theta = P(\theta) + R(\theta),$$

where both of these functions are considered defined *only* in the interval

$$-\pi/2 \leq \theta \leq \pi/2.$$

981 HERTZ AT 0 dBm



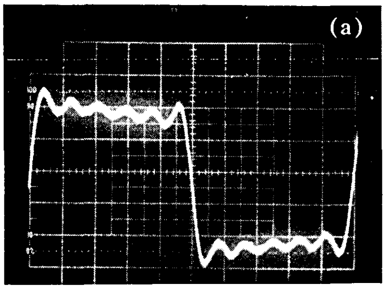
1 KILOHERTZ

Fig. 7—Tone spectrum—high level (981 Hertz).

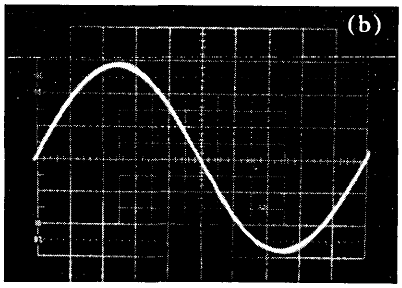
They are replicated, with sign reversals, in successive intervals of π forming a wave periodic in the interval 2π. An upper bound on the harmonic distortion, because of truncation after the nth term of a Maclaurin expansion, is

$$\text{Distortion} \leq = 20 \log \left[\frac{(2n)!}{(\pi/2)^{2n}} \right].$$

100-HERTZ SQUARE WAVE (6-TONE COMPONENTS)



981-HERTZ WAVEFORM (0 dBm)



3-TONE WAVEFORM

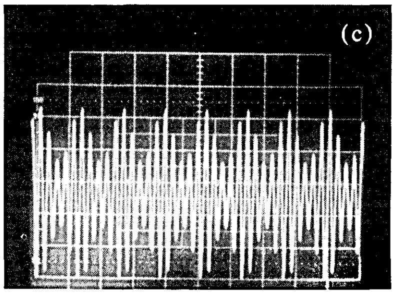


Fig. 8—Time domain waveforms.

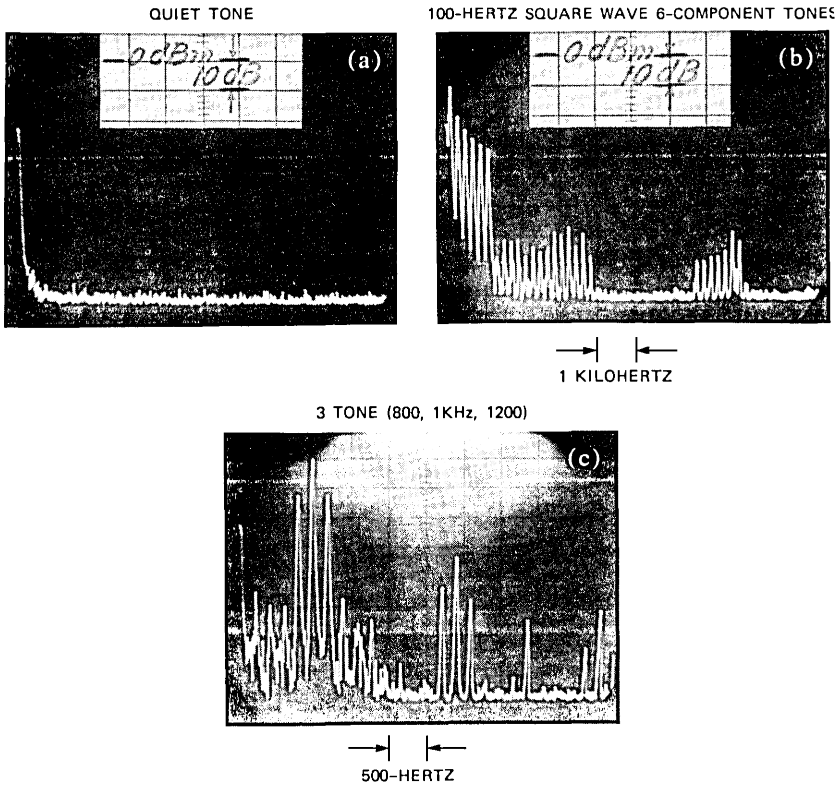


Fig. 9—Spectrum of time waveforms.

Table I lists this upper bound for values of n. This is a mathematical upper bound and ignores the effects of quantizing distortion.

The harmonics generated due to the μ-255 format generally are in the −42 dB range. The total signal to distortion is in the 36-dB range. Thus, $n = 4$ should be used for the Maclaurin expansion since it produces distortion components less than those generated by the μ-255 format.

By a similar analysis, it can be shown that the sine series should also be terminated after four terms so that the harmonic distortion upper bound is −78.9 dB. The Maclaurin cosine series is chosen instead

Table I—Harmonic distortion

n	Upper Bound
3	−33.61 dB
4	−60.73 dB
5	−91.97 dB

of the sine series, since its highest term contains θ^6 instead of θ^7 and, thus, less programming is required.

The program starts at $-\pi/2$ and increments in steps of $\phi = 2\pi f T$, where f is the desired frequency and T is the interval between samples. The computed function should remain positive until after n samples, when the accrued phase θ_n is such that $\theta_n = -\pi/2 + n\phi$ and exceeds $\pi/2$. At this time, π is subtracted from the accrued phase yielding a new value θ_n such that $-\pi/2 < \theta_n < 0$ and the process starts again. The computed value is now made negative. The negative sign prefixes each sample until, again, the accrued phase exceeds $+\pi/2$. Once again, π is subtracted from the accrued phase. The sign of each computed point becomes positive, and the algorithm cycle is complete.

An expansion around $\pi/4$ would result in less error in the resulting approximation at the end points, but it would require more programming to fix the sign. The errors for an expansion around zero are less than those attributable to the μ-255 operation and, thus, it is not important to reduce them.

3.2 Flow diagram

A simplified diagram, Fig. 10, covers the generation of one component. Note that the loop counter (LC) serves as a flip-flop for determining the sign of the result. Suppose LC = 0 and $\pi/2$ has been exceeded. The right-hand branch decision (LC$\neq$0) point in Fig. 8 causes LC to be decremented to -1. During the next pass when the left-hand decision (LC$\neq$0) is reached, the sign of the result is changed. This is maintained until the accumulated phase would again exceed $+\pi/2$, when traversing the right-hand decision point forces LC = 0. Now, the computed result is positive and the process repeats.

3.3 Experimental results

Figure 11 shows the spectrum of the DSP output after decoding. The harmonics are approximately 42 dB below the fundamental as predicted for a μ-255 decoder. Note, however, the differences in the spectrums for frequencies of 500 and 502 Hz. Since 500 Hz is a rational submultiple of 8 kHz, all samples per sine wave cycle are repeated in subsequent cycles. There is essentially no quantizing noise, i.e., the sample values are completely periodic. The pronounced spectral lines demonstrate this purity. For the 502-Hz case, there is no periodicity in the samples for successive cycles and, hence, quantizing noise exists. The peak value of the spectral harmonics are reduced and the noise floor is raised.

IV. CONCLUSIONS

A DSP tone generator using the table look-up method is able to generate up to 12 components (or 21 components for 2-coupled gen-

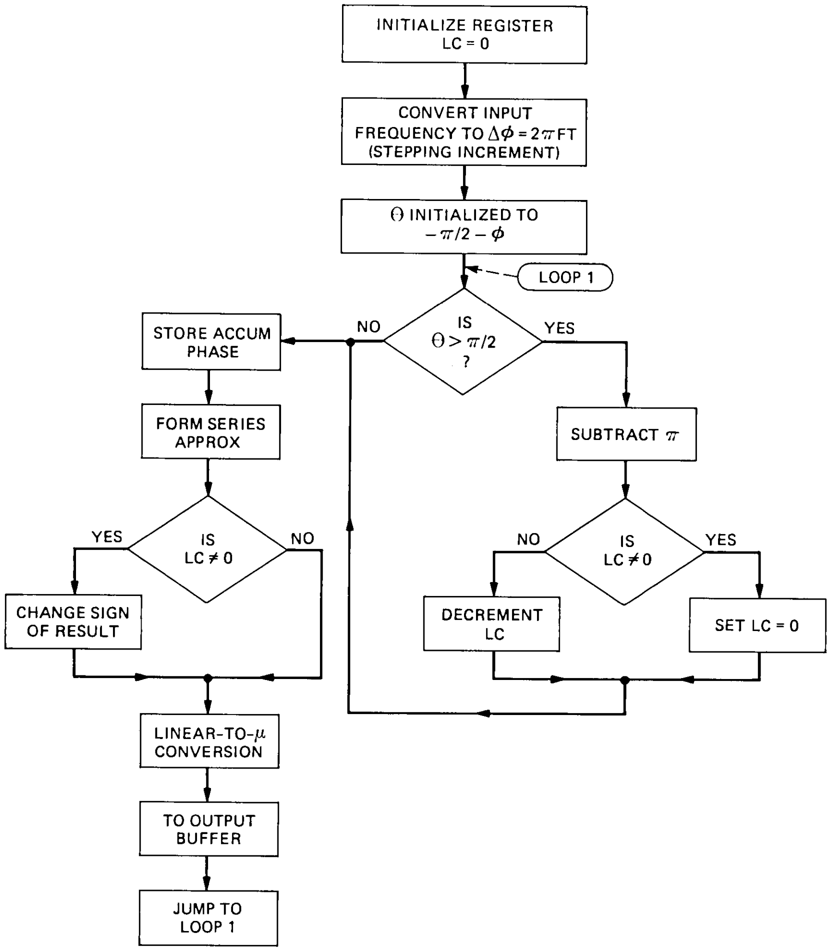


Fig. 10—Maclaurin expansion flow diagram.

erators) assuming a 4-kHz, μ-255 channel with an output sample every 125 μs. Harmonic distortion is substantially below that inherent in the channel. Such a versatile generator is suitable for all transmission system testing applications.

Given a suitable decoder, a DSP tone generator can also be used to create any analog wave representable by 13 spectral lines restricted to the 4-kHz band. A sine wave component can be generated every 17.75 μs and a 26-kHz sine wave can be generated. By suitable program changes, it seems possible to produce a 50-kHz sine wave.

A DSP tone generator using the Maclaurin expansion method is limited to less than four components. It requires less ROM than the table look-up method and, hence, is applicable where such a program

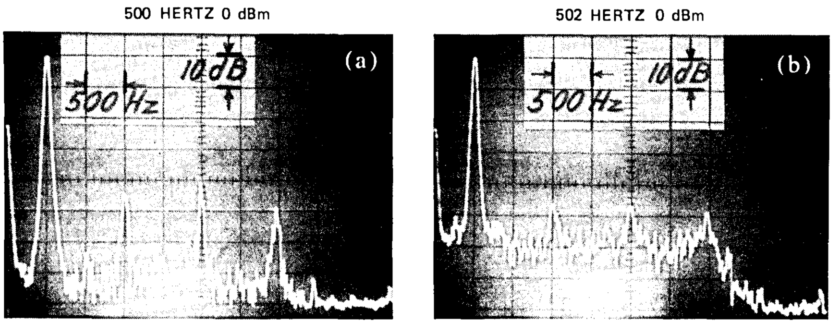


Fig. 11—Maclaurin spectrums.

would be co-resident with other measurement routines in a single DSP. Where 2-tone testing is required, e.g., envelope delay distortion, slope-sag intermodulation distortion, etc., such a generator is especially applicable.

V. ACKNOWLEDGMENTS

Grateful thanks to E. J. Angelo and J. R. Boddie for their general assistance; to D. P. Yorkgitis for providing the diagnostic and refresh routines for the table look-up method; to Basil Papatrefon for testing the programs and deriving all experimental results; and to Jack Salz for offering his suggestions concerning the mathematics bounding the harmonic distortion for the Maclaurin expansion approach.

APPENDIX

This analysis is directed to specifying an upper bound on the harmonic content of a truncated Maclaurin expansion.

Given the Maclaurin expansion

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \frac{\theta^8}{8!} - + \frac{\theta^{2(n-1)}}{[2(n-1)]!} \cdots, \qquad (1)$$

consider it as being made up of a finite polynomial $P(\theta)$ consisting of the first n terms of the above expansion and the remaining portion of the series $R(\theta)$, i.e.,

$$\cos\theta = P(\theta) + R(\theta).$$

Consider now that a periodic function be generated such that

$$P_N(\theta) = P(\theta) \quad \text{for} \quad -\frac{\pi}{2} \le \theta < \frac{\pi}{2},$$

and

$$P_N(\theta) = 0 \quad \text{elsewhere.} \qquad (2)$$

Similarly,

$$R_N(\theta) = R(\theta) \quad \text{for} \quad -\frac{\pi}{2} \leq \theta < \frac{\pi}{2},$$

and

$$R_N(\theta) = 0 \quad \text{elsewhere.} \tag{2a}$$

By a replicating procedure, then

$$\cos\theta = \sum_{n=-\infty}^{\infty} (-1)^n P_N(\theta - n\pi) + \sum_{n=-\infty}^{\infty} (-1)^n R_n(\theta - n\pi). \tag{3}$$

The portion of eq. 3,

$$\cos\theta \approx \sum_{n=-\infty}^{\infty} (-1)^n P_N(\theta - n\pi),$$

is the approximation used for sample generation in the Maclaurin series program given in this paper.

Because of the definitions in eqs. (2) and (2a), each sum specified in eq. 3 is periodic with period 2π and, hence, can be expanded into a Fourier series. Let p_k and r_k be the Fourier series coefficients defined by

$$\sum_{n=-\infty}^{\infty} (-1)^n P_N(\theta - n\pi) = \sum_{k=-\infty}^{\infty} p_k e^{ik\theta}, \tag{4}$$

$$\sum_{n=-\infty}^{\infty} (-1)^n R_N(\theta - n\pi) = \sum_{k=-\infty}^{\infty} r_k e^{ik\theta} \tag{5}$$

and

$$\cos\theta = \sum_{k=-\infty}^{\infty} p_k e^{ik\theta} + \sum_{k=-\infty}^{\infty} r_k e^{ik\theta}, \tag{6}$$

where

$$p_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} \sum_{n=-\infty}^{\infty} (-1)^n P_N(\theta - n\pi) e^{-ik\theta} d\theta \tag{7}$$

and

$$r_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} \sum_{n=-\infty}^{\infty} (-1)^n R_N(\theta - n\pi) e^{-ik\theta} d\theta. \tag{8}$$

Interchanging the order of integration and summation in eq. 7, then

$$p_k = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} (-1)^n \int_{-\pi}^{\pi} P_N(\theta - n\pi) e^{-ik\theta} d\theta. \tag{9}$$

Changing the variable of integration $y = \theta - n\pi$ modifies this equation to be

$$p_k = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} (-1)^{n+kn} \int_{-\pi(n+1)}^{-\pi(n-1)} P_N(y) e^{-iky} dy, \tag{10}$$

and further simplifies to

$$p_k = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} (-1)^{n(1+k)} \int_{-\pi(n+1)}^{-\pi(n-1)} P_N(y) e^{-iky} dy. \tag{11}$$

Because of the definition of $P_N(y)$, the integral only exists for $n = 1, -1, 0$;

$$p_k = \frac{1}{2\pi} \left[\int_{-\pi/2}^{\pi/2} P_N(y) e^{-iky} dy + (-1)^{(1+k)} \int_{-\pi/2}^{0} P_N(y) e^{-iky} dy \right.$$
$$\left. + (-1)^{-(1+k)} \int_{0}^{\pi/2} P_N(y) e^{-iky} dy \right]. \tag{12}$$

Combining these terms results in

$$p_k = \frac{1}{2\pi} [1 + (-1)^{(1+k)}] \int_{-\pi/2}^{\pi/2} P_N(y) e^{-iky} dy. \tag{13}$$

When k is even, $p_k = 0$, therefore, only odd harmonics exist in the periodic function given by eq. 4. Since $P_N(\theta)$ is an even function, it follows that

$$p_k = \frac{2}{\pi} \int_0^{\pi/2} P_N(\theta) \cos k\theta d\theta \qquad k \text{ odd.} \tag{14}$$

By a completely similar development for r_k, and substituting into eq. 6,

$$\cos\theta = 2 \left[\sum_{\substack{k=1 \\ k\text{odd}}}^{\infty} p_k \cos k\theta + \sum_{\substack{k=1 \\ k\text{odd}}}^{\infty} r_k \cos k\theta \right]. \tag{15}$$

From this equation, it is apparent that

$$p_k = -r_k \quad \text{and} \quad k \neq 1 \tag{16}$$

since only the fundamental exists on the left-hand side of the equation.

Given the properties of $R(\theta)$ such that each term is positive and smaller than the preceding term, then one can state that the magnitude of the first term of $R(\theta)$ is greater than $|R(\theta)|$. Hence, if $P(\theta)$ contains

n terms then

$$\frac{\theta^{2n}}{2n!} \geq |R(\theta)|. \tag{17}$$

The stipulation that the terms are successively smaller implies for the m^{th} and $m^{\text{th}} + 1$ term that

$$1 \leq [\theta^{2(m-1)}/(2(m-1))!]/[\theta^{2m}/(2m!)] = \frac{2m(2m-1)}{\theta^2}.$$

Since this must hold for all $-\pi/2 \leq \theta \leq \pi/2$, then

$$2m(2m-1) \geq \left(\frac{\pi}{2}\right)^2$$

and, hence, for this inequality to hold

$$m > 1 \quad \text{applies.}$$

Equation 17, therefore, applies for maximum θ, after the second term, and certainly then

$$[(\pi/2)^{2n}/(2n)!] \geq |R(\theta)| \quad \text{for} \quad n \geq 2. \tag{18}$$

Parsevals theorem states that

$$\frac{1}{2\pi}\int_{-\pi}^{\pi}\left[\sum_{n=-1}^{\infty}(-1)^n R_N(\theta - n\pi)\right]^2 d\theta = \sum_{\substack{k=1\\ k\text{odd}}}^{\infty} r_K^2. \tag{19}$$

Since $R(\theta)$, and hence, $R_N(\theta)$, are bounded, as has been stated in eq. (18), then certainly

$$[(\pi/2)^{2n}/(2n)!]^2 \geq \sum_{\substack{k=3\\ k\text{odd}}} r_k^2 = \sum_{\substack{k=3\\ k\text{odd}}} p_k^2. \tag{20}$$

Therefore, the first term of $R(\theta)$, evaluated at $\pi/2$, represents an upper bound on any harmonic component of the replicated $P_N(\theta)$ i.e., the truncated Maclaurin expansion. The harmonics of

$$(\text{Replicated} \quad P_N(\theta)) < 20 \log [(2n)!/(\pi/2)^{2n}]. \tag{21}$$

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Power Measurements

By S. CORDRAY, D. L. FAVIN, and D. P. YORKGITIS

(Manuscript received January 7, 1981)

Power measurement is fundamental to transmission quality testing. This measurement need is extended to signals represented by digital bit streams. Accurate and precise measurements over a 60-dB range have been made using the digital signal processor. One algorithm that has been used measures the power of fixed-length sample sequences. A second algorithm yields periodically updated power measurements of infinitely long sample sequences, but with slightly increased measurement ripple and frequency restriction. Theoretical expectations for measurement variation in the fixed-length measurements of noise power are also discussed.

I. INTRODUCTION

The measurement of power is fundamental to transmission quality testing. Power measurements of single tones, such as the milliwatt standard, are used to adjust transmission levels. Multiple-tone power measurements are used in nonlinear distortion testing. Examples of power measurements of band-limited noise are return-loss and *C*-message weighted noise measurements.

This paper gives an overview, discusses the theoretical accuracy and precision of digital noise power measurements, and presents some results using the digital signal processor (DSP) A3990 for making power measurements.

II. OVERVIEW

The measurement of power will be presented following the block diagram of Fig. 1. Since the incoming signal is generally encoded for bit compression, the signal samples first must be decoded to linear

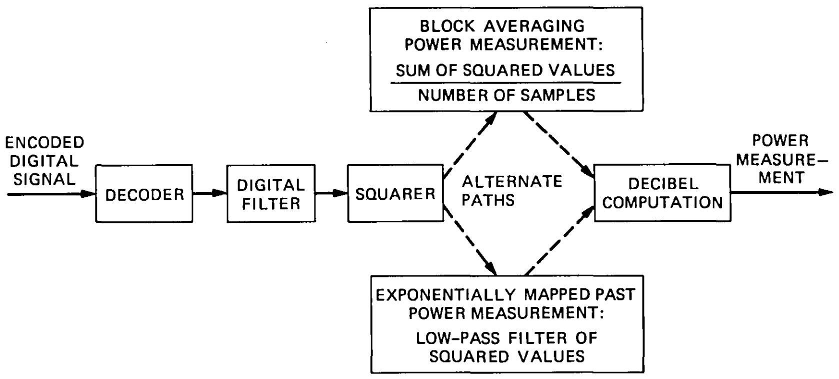

BLOCK AVERAGING POWER MEASUREMENT:
SUM OF SQUARED VALUES / NUMBER OF SAMPLES
ENCODED DIGITAL SIGNAL
DECODER
DIGITAL FILTER
SQUARER
ALTERNATE PATHS
DECIBEL COMPUTATION
POWER MEASUREMENT
EXPONENTIALLY MAPPED PAST POWER MEASUREMENT:
LOW-PASS FILTER OF SQUARED VALUES

Fig. 1—Time domain technique of power measurement.

samples, as indicated in the first block of the figure. For μ-255 encoding and decoding, the DSP has a dedicated instruction set and associated circuitry.

The decoded digital signal is scaled and passed through a digital filter. This paper discusses measurements with flat weighting, with *C*-message weighting, and through a *C*-notched filter.

Note that Fig. 1 depicts essentially a time-domain approach, which should be contrasted with the frequency-domain approach depicted in Fig. 2. These two approaches are tied together by Parseval's Theorem:

$$\text{Signal Power} = \lim_{\mathscr{T}\to\infty} \frac{1}{\mathscr{T}} \int_0^{\mathscr{T}} y^2(t)\, dt = \int_0^{\infty} S(w)\, dw = R(0), \qquad (1)$$

where

$\mathscr{T}$ = averaging interval,
$y(t)$ = analog signal,
$S(w)$ = the power-density spectrum, and
$R(0)$ = autocorrelation function at zero lag, i.e., the dc component of $y^2(t)$.

The first integral in eq. (1) is implemented as in Fig. 1; the second, as in Fig. 2. Implementation of the latter is not discussed in this paper.

To compute the first integral, the sample values must be squared, as indicated in Fig. 1. Two different methods of determining the power from the squared signal were used.

The first method is the straightforward approach. If the incoming

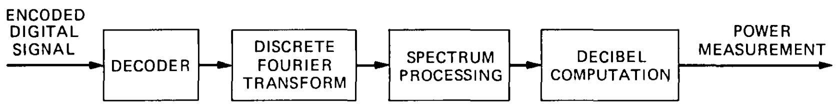


Fig. 2—Frequency domain technique of power measurement.

signal is considered stationary, the power can be approximated from the first integral of eq. (1) as follows:

$$R(0) \cong \frac{1}{\mathscr{T}} \int_0^{\mathscr{T}} y^2(t)\, dt \cong \frac{1}{NT} \sum_{i=0}^{N-1} y_i^2 T = \frac{1}{N} \sum_{i=0}^{N-1} y_i^2, \tag{2}$$

where

N = the number of samples,
y_i = ith signal sample, and
T = interval between the samples ($T = 125\ \mu$s in most voice telephone applications).

This approach is termed block averaging (BA) in this paper.

To use BA, N must be chosen large enough so that the measurement variation is within the required tolerance. In the next section, the probability of measuring noise power within certain confidence levels for different values of N is derived.

The second method for extracting the dc component employs a convolution of the squared sample values with the infinite impulse response of a first-order, low-pass filter. This algorithm, described in Appendix B, has been termed exponentially mapped past (EMP).

The EMP is not as applicable as BA because of the extra frequency components generated by squaring a signal. Because of the sampled nature of the signals, some components can be aliased into the passband of the EMP low-pass filter and, thus, yield measurement ripples. For example, the EMP parameters discussed in this paper yield a ripple of ±0.4 dB for a 15-Hz tone and a ripple of less than ±0.1 dB for an 80-Hz tone.

Once the dc component has been extracted, it can be converted to a dB measurement before it is reported. A method for calculating the required logarithms with the DSP is described in Appendix C.

Currently, the BA program reports a dB computation once per block, i.e., once every N samples, where N = 4096. However, when EMP is used, linear-to-dB conversions can be made more frequently. After a conversion is made in the current EMP program, the next conversion can be made after another three samples have been read from the input buffer. The rate with which conversions can be made and reported during EMP power measurements depends on several variables, as explained in Appendix D.

For display or printout routines, a binary number representing a dB level can be converted to binary-coded decimal (BCD). A BCD routine was used to yield the BA signal-to-quantizing noise ratio measurements described below. This routine is not discussed in this paper.

III. THEORETICAL PRECISION OF NOISE POWER MEASUREMENTS

In this section the noise power measurement precision that is theoretically possible by digital power measurements is presented.[1]

Consider the finite set of noise samples $n_0, n_1, \ldots n_{N-1}$. The ac power of these N samples is expressed as

$$P_n(N) = \frac{1}{N} \sum_{i=0}^{N-1} (n_i - \bar{n})^2, \tag{3}$$

where

$$\bar{n} = \frac{1}{N} \sum_{j=0}^{N-1} n_j$$

is the dc component of the noise.

How large should N be in order that the estimated power $P_n(N)$ be within plus or minus some δ_n (in dB) of the actual power? That is, for a given N, what is the probability $P\{\ \}$ that

$$\left| 10 \log \frac{P_n(N)}{P_{\text{ref}}} - \lim_{M\to\infty} 10 \log \frac{P_n(M)}{P_{\text{ref}}} \right| < \delta_n, \tag{4}$$

where P_{ref} is any reference power? The following analysis assumes that the noise is Gaussian, but results are in most cases applicable to other types of noise (e.g., quantizing or coding noise).

Assume the noise sample n_i to be an independent, zero-mean, Gaussian random variable with finite variance σ^2. Then $P_n(N)$ is an estimate of σ^2, with expected value $= \dfrac{N-1}{N}\,\sigma^2$ (see Ref. 2). To the accuracy required for the BA program, $P_n(N)$ is effectively an unbiased estimate for $N > 100$. For a given δ_n, eq. (4) becomes

$$\left| 10 \log \frac{P_n(N)}{P_{\text{ref}}} - 10 \log \frac{\sigma^2}{P_{\text{ref}}} \right| < \delta_n \tag{5}$$

or

$$\sigma^2 10^{-\delta_n/10} < P_n(N) < \sigma^2 10^{\delta_n/10}. \tag{6}$$

Now if the probability density function $f_n(\alpha)$ of $P_n(N)$ can be found, then the probability $P\{\ \}$ of satisfying the inequality (5) is

$$P\left\{ \left| 10 \log \frac{P_n(N)}{P_{\text{ref}}} - 10 \log \frac{\sigma^2}{P_{\text{ref}}} \right| < \delta_n \right\} = \int_{\sigma^2 10^{-\delta_n/10}}^{\sigma^2 10^{\delta_n/10}} f_n(\alpha)\, d\alpha. \tag{7}$$

In Appendix A the density function is derived along with the probability. The result is

$$P\{|\mathcal{N}_m - \mathcal{N}_a| < \delta_n\} \cong \int_{l_1}^{l_2} \frac{e^{-t^2/2}}{\sqrt{2\pi}}\, dt, \tag{8}$$

where

$$\mathcal{N}_m = 10 \log \frac{P_n(N)}{P_{\text{ref}}}, \qquad \mathcal{N}_a = 10 \log \frac{\sigma^2}{P_{\text{ref}}}, \qquad N > 30,$$

$$l_2 = \sqrt{2(10)^{\delta_n/10}N} - \sqrt{2N-3},$$

and

$$l_1 = \sqrt{2(10)^{-\delta_n/10}N} - \sqrt{2N-3}.$$

For a graphical representation of eq. (8), see Fig. 3. For example, with 500 samples the probability that the noise power measurement precision is within ±0.2 dB is 53 percent. To meet standard specifications for noise and signal power measurements,[3] N was chosen to be 4096, yielding a precision of ±0.5 dB.

IV. TEST RESULTS

The ability to measure power precisely with both the BA and EMP schemes can be seen by comparing BA and EMP measurements with the actual signal-to-quantizing noise ratios (SNRS) of ideally-encoded sine waves at levels ranging from 3 to −64 dBm. Such encoded sine waves have inherent quantizing noise frequency components across the voiceband range of frequencies. To make SNR measurements, the C-notched and C-message filters are generally used.

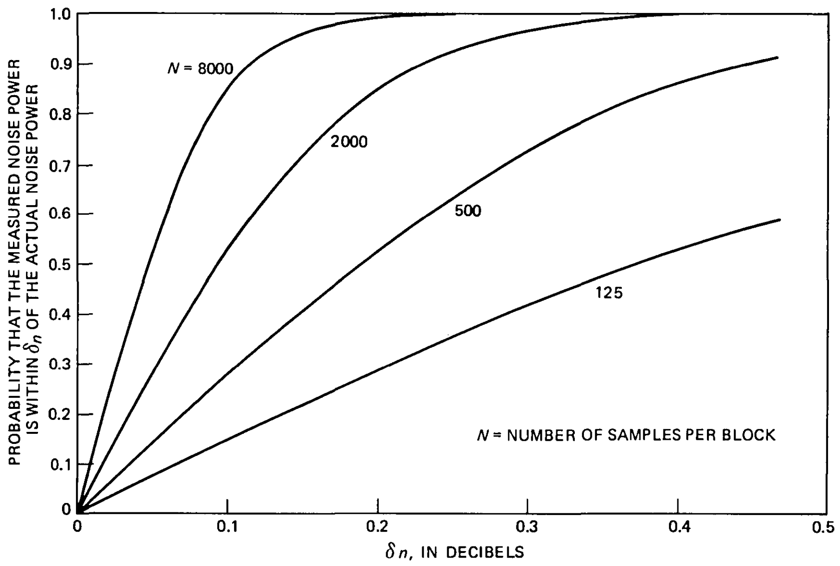


Fig. 3—Theoretical noise power measurement precision.

The C-notched filter is actually two cascaded filters, each implemented with three cascaded, second-order sections. The first filter, the C-message filter, has approximately unity gain from 1000 to 2500 Hz, and its attenuation increases gradually on either side of the passband to 54.7 dB at 60 Hz and to 13.7 dB at 3900 Hz. The second filter, the notch filter, is designed for attenuation of more than 50 dB from 989 to 1020 Hz. The frequency response of the digital C-notched filter was measured by both the EMP measurement routine and by an analog power meter. The results, limited by quantizing noise, appear in Fig. 4.

To determine the SNR of an encoded signal between 1004 and 1020 Hz, the C-message weighted measurement must be subtracted from the C-notched measurement. By means of a real-time developmental tool, the DSPMATE, SNR measurements were made with the BA and EMP programs using ideal, encoded sine waves at 1015.625 Hz. The BA and EMP measurements yielded a range of ±0.5 dB, which was within the theoretical noise power measurement precision. In Fig. 5, the BA derived SNRS are plotted against the actual SNRS.

In order to retain significant bits at low power levels, the programs were modified when the test signal powers were below −27 dBm. After μ-to-linear conversion, sample values from signals above −27 dBm in power were divided by 4, while sample values from signals below −27 dBm in power were multiplied by 8. The DSP can be programmed to choose the appropriate scaling.

Because of quantizing noise, the maximum SNR is approximately 40 dB. Thus, the maximum SNR in Fig. 5 is comparable to the quantizing noise floor of Fig. 4.

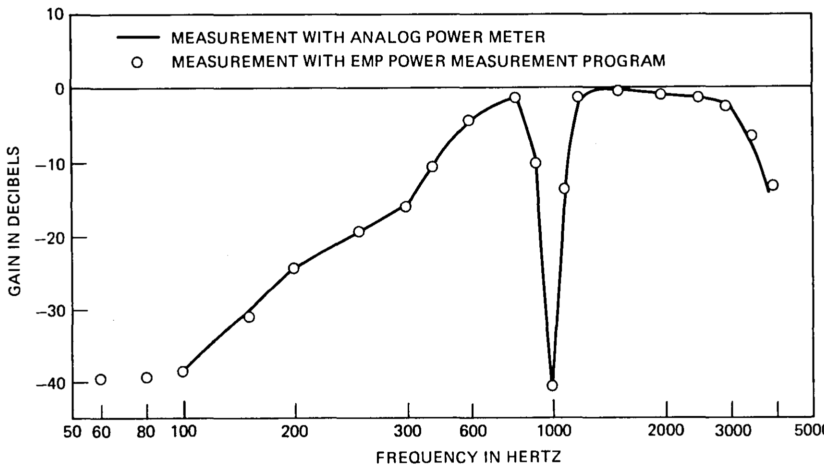


Fig. 4—Frequency response of a digital C-notched filter. (Measured signals are μ-255 coded.)

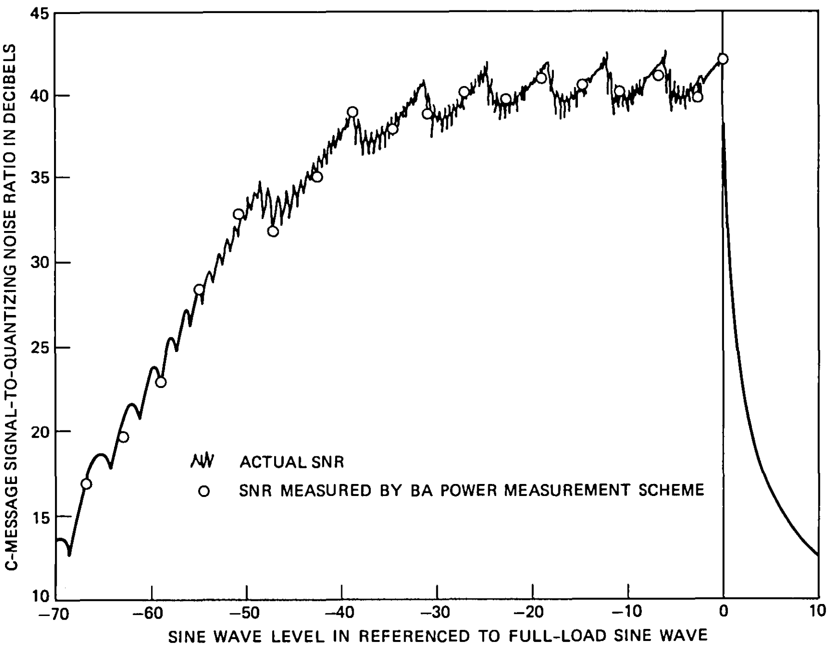


Fig. 5—Measured versus actual SNRs for μ-255 encoding. (Zero dB full-load sine wave = 3.17 dBm.)

V. CONCLUSION

Two reliable methods for measuring power with the DSP have been presented: the BA approach, and the EMP approach. Both of the approaches, when examined mathematically, have no significant bias in their expected values. The accuracy, therefore, is as good as the digital samples representing the signals being measured.

The precision of signal power, noise power, and SNR measurements were investigated. Digital signal processing measurements of these parameters showed that:

(*i*) In general, signal power measurements were precise to within 0.1 dB.

(*ii*) With appropriate scaling, SNR measurements were precise to within 0.5 dB over a 60-dB range.

(*iii*) Block-averaging noise power measurements all fell within theoretical limits.

In the EMP program, frequencies in the range 80 to 3920 Hz yielded measurement ripples less than ±0.1 dB, and frequencies in the range 30 to 3970 Hz yielded ripples less than ±0.2 dB. These measurements were made with an EMP convolution having a 3-dB cutoff at 2.489 Hz. However, these ripples were about twice as large as the measurement ripples from the BA program.

Two advantages of EMP over BA are compactness of code and ability to update a measurement 1365 times as often. Frequent updating may aid in identifying particular types of problems and, hence, aid in problem sectionalization.

APPENDIX A

Derivation of Noise Power Measurement Precision

In the following analysis, the probability density function of $P_n(N)$ is presented and used to find, in computable form, the probability that the ac noise power $P_n(N)$ is within some δ_n (in dB) of the noise variance σ^2.

Recall the definition of ac noise power:

$$P_n(N) = \frac{1}{N} \sum_{i=0}^{N-1} (n_i - \bar{n})^2.$$

As a result of assumptions that n_i is a zero-mean, Gaussian random variable with finite variance σ^2, the probability density function of $P_n(N)$ (see Ref. 4) is

$$f_n(\alpha) = \frac{\alpha^{(N-3)/2} e^{-\alpha N/2\sigma^2}}{2^{(N-1)/2}(\sigma/\sqrt{N})^{N-1}\Gamma[(N-1)/2]}, \tag{9}$$

which is a chi-square density. Hence, the probability $P\{\ \}$ that $P_n(N)$ is within δ_n of the actual noise power is

$$P\left\{\left|10 \log \frac{P_n(N)}{P_{\text{ref}}} - 10 \log \frac{\sigma^2}{P_{\text{ref}}}\right| < \delta_n\right\} = \int_{\sigma^2 10^{-\delta_n/10}}^{\sigma^2 10^{\delta_n/10}} f_n(\alpha)\, d\alpha. \tag{10}$$

Using the substitution $t = \alpha N/\sigma^2$ yields

$$P\left\{\left|10 \log \frac{P_n(N)}{P_{\text{ref}}} - 10 \log \frac{\sigma^2}{P_{\text{ref}}}\right| < \delta_n\right\}$$
$$= \int_{10^{-\delta_n/10}N}^{10^{\delta_n/10}N} \frac{t^{(N-3)/2} e^{-t/2}\, dt}{2^{(N-1)/2}\Gamma[(N-1)/2]}. \tag{11}$$

For $N > 30$, this probability can be expressed in terms of the normal probability integral:[5]

$$P\left\{\left|10 \log \frac{P_n(N)}{P_{\text{ref}}} - 10 \log \frac{\sigma^2}{P_{\text{ref}}}\right| < \delta_n\right\} \cong \int_{l_1}^{l_2} \frac{e^{-t^2/2}}{\sqrt{2\pi}}\, dt, \tag{12}$$

where

$$l_2 = \sqrt{2(10)^{\delta_n/10}N} - \sqrt{2N-3}$$

and

$$l_1 = \sqrt{2(10)^{-\delta_n/10}N} - \sqrt{2N-3}.$$

Eq. (8) and Fig. 3 in the text follow.

APPENDIX B

Derivation of EMP

The result of passing an analog signal $y^2(t)$ through an analog filter with impulse response $g(t)$ is $P(t)$, where

$$P(t) = \int_{-\infty}^{\infty} y^2(u)g(t-u)\,du. \tag{13}$$

Let T be the sampling period of the digital signal $y^2(nT)$, or $y^2(n)$ for short. The result of passing $y^2(n)$ through a digital filter with impulse response $h(n)$, is $P(n)$, where

$$P(n) = \sum_{k=-\infty}^{\infty} y^2(k)h(n-k). \tag{14}$$

An analog, first-order, low-pass filter uses

$$g(t) = \begin{cases} A\, e^{-t/\tau}, & t \geq 0 \\ 0, & t < 0, \end{cases} \tag{15}$$

where τ is the time constant of the filter. Such a filter has a 3-dB cutoff at $(2\pi\tau)^{-1}$ Hz.

If the impulse invariance $h(n) = g(nT)$ is used to form an equivalent digital filter, the corresponding impulse response is

$$h(n) = \begin{cases} A\, e^{-nT/\tau}, & n \geq 0 \\ 0, & n < 0, \end{cases} \tag{16}$$

where T is the sampling period. Because of sampling, aliasing is introduced, but the effects of the aliased components are negligible if the cutoff frequency is low.

For ease in notation, let $m = T/\tau$. From eqs. (14) and (16),

$$P(n) = \sum_{k=-\infty}^{\infty} y^2(k)h(n-k),$$

or

$$P(n) = \sum_{k=-\infty}^{n} y^2(k)A\, e^{-m(n-k)}. \tag{17}$$

From this, a simple recursive relationship for $P(n)$ can be developed:

$$\begin{aligned} P(n+1) &= \sum_{k=-\infty}^{n+1} y^2(k)A\, e^{-m(n+1-k)} \\ &= e^{-m} \sum_{k=-\infty}^{n+1} y^2(k)A\, e^{-m(n-k)} \\ &= e^{-m}\, P(n) + e^{-m}y^2(n+1)A\, e^{m} \end{aligned} \tag{18}$$

or

$$P(n+1) = e^{-m}\, P(n) + Ay^2(n+1). \tag{19}$$

In eq. (19), A should be chosen to ensure unity gain at dc. To determine A, let

$$y^2(n) = \begin{cases} L, & n \geq 0 \\ 0, & n < 0 \end{cases} \quad \text{and} \quad P(-1) = 0. \tag{20}$$

Then,

$$P(n+1) = e^{-m}P(n) + AL, \tag{21}$$

which implies that $P(n)$ is a geometric series:

$$\begin{aligned} P(n) &= AL \sum_{k=0}^{n} e^{-mk} \\ &= AL \frac{1 - e^{-m(n+1)}}{1 - e^{-m}}, \end{aligned} \tag{22}$$

which approaches L for n approaching infinity if $A = 1 - e^{-m}$.

The EMP power measurement $P(n)$ can, thus, be obtained by the recursion formula

$$P(n) = e^{-m}P(n-1) + (1 - e^{-m})y^2(n), \tag{23}$$

where $m = T/\tau$.

If the range of the signal power to be measured is large and no automatic gain control is to be used, then some double-precision arithmetic has to be done to save the least-significant bits resulting from sums and products. In particular, the result of squaring the input samples nearly doubles the number of significant bits in the accumulator. Therefore, all bits resulting from squaring should be saved. In addition, EMP, which follows the squaring, should be implemented with double precision.

This implementation is facilitated by representing e^{-m}, which is nearly unity for a low cutoff frequency, by $1 - 2^{-R}$, where R is an integer. The time constant τ is

$$\tau = -T/\ln(1 - 2^{-R}). \tag{24}$$

For $R = 9$, $\tau = 63.9$ ms; and the 3-dB cutoff is 2.489 Hz, which yields power measurements with ripples of less than ±0.1 dB at frequencies between 80 and 3920 Hz.

To implement EMP in double precision, the stored value $P(n - 1)$ must occupy two storage locations. Conveniently, one location could contain its integral part, while the other, its fractional part.

APPENDIX C

Using the DSP to Compute the Natural Logarithm of a Number

Using the DSP, the logarithm of up to a 40-bit number P can be computed to an accuracy of 0.001. Suppose that

$$\begin{aligned} \ln P &= \ln[\mathscr{M}(2^{\mathscr{E}})] \\ &= \ln \mathscr{M} + \mathscr{E} \ln 2, \end{aligned} \tag{25}$$

where $\mathscr{M}$ is a real number and $\mathscr{E}$ is an integer. Then, $\ln P$ can be computed from

(*i*) a series expansion on $\mathscr{M}$ plus

(*ii*) a table of multiples of $\mathscr{E}\ln 2$.

If $0 < \mathscr{M} \le 2$, the following expansion can be used:

$$\ln \mathscr{M} = (\mathscr{M} - 1) - \frac{(\mathscr{M} - 1)^2}{2} + \frac{(\mathscr{M} - 1)^3}{3} - \frac{(\mathscr{M} - 1)^4}{4} + \cdots \tag{26}$$

As indicated in Fig. 6, if $0.68 \le \mathscr{M} \le 1.36$, then $\ln \mathscr{M}$ can be computed to an accuracy of ±0.001 with only four terms. Since the upper bound on $\mathscr{M}$ is twice its lower bound, $\mathscr{E}$ can be determined by repeated scalings of P by 2.

APPENDIX D

Rate of Linear-to-dB Conversions in the EMP Program

In the current EMP program, a linear-to-dB conversion of the power measurement is made after every third sample. Although every sample is used to update the linear power measurement, there is insufficient time to make a conversion to dB after every update.

The number of samples, S, that must be used to update the linear power measurement S times and to make one linear-to-dB conversion can be determined from the following relation:

$$S(R + U) + C < ST, \tag{27}$$

where

R = the time to check whether the input buffer is full and, if so, to read it,

U = the time to update the linear power measurement using the new sample,

C = the time to make a linear-to-dB conversion of the power measurement, and
T = the sampling interval.

Then S can be any integer greater than $C/T - R - U$. In Fig. 7, the case for $S = 3$ is depicted. In this figure, L is the time necessary to load the input buffer.

As shown in Fig. 7, the second sample in each group of three must be read before the third sample begins to be loaded. Expressed symbolically, this means

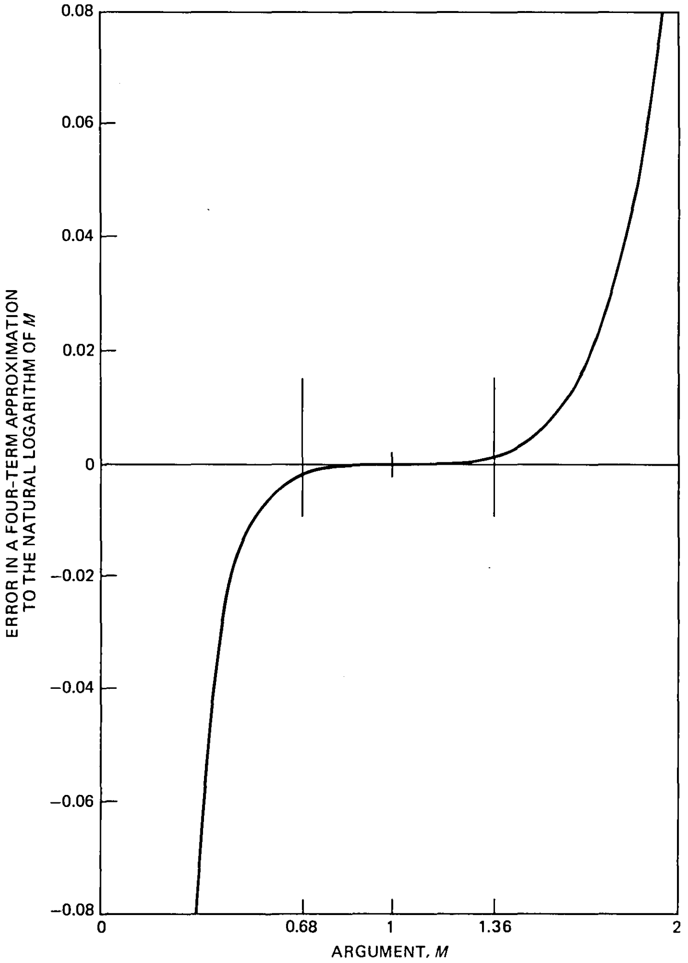


Fig. 6—Performance of a four-term polynomial approximation to the natural logarithm.

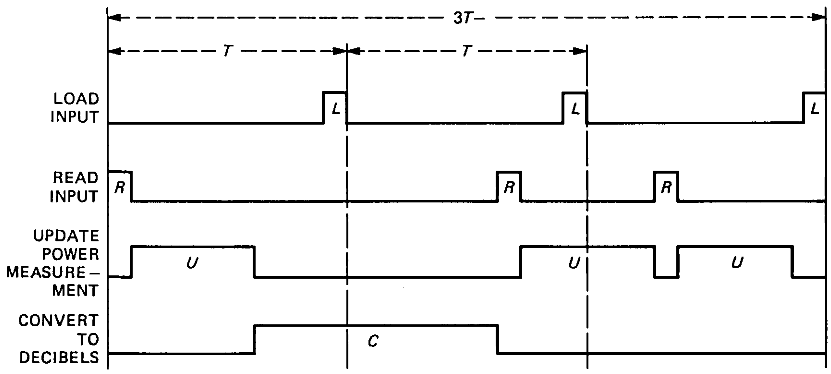


Fig. 7—Timing diagram showing the rate of linear-to-dB conversions in the EMP program.

$$R + U + C + R < T + T - L,$$

or

$$C < 2T - L - 2R - U. \tag{28}$$

If eq. (28) were not satisfied by C, then the second sample in each group of three would have to be stored during the linear-to-dB conversion and then the sample value returned after the conversion was complete.

In the current EMP program, the upper bound on C was sufficiently high that, with $S = 3$, each linear-to-dB conversion could be followed by a conversion to BCD.

V. ACKNOWLEDGMENT

R. W. Kolor's detailed critique of this paper is appreciated.

REFERENCES

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, New York: Prentice-Hall, Inc., 1975, Ch. 11.
2. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, New York: Prentice-Hall, Inc., 1975, p. 537.
3. Bell System Technical Reference, "Transmission Parameters Affecting Voiceband Data Transmission Measuring Techniques," AT&T Co., PUB 41009.
4. A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill Book Co., 1965, p. 250.
5. J. R. Blum and J. I. Rosenblatt, *Probability and Statistics*, Philadelphia: W. B. Saunders Co., 1972, p. 252.

Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 7, September 1981
Printed in U.S.A.

Digital Signal Processor:

Tone Detection for CCITT No. 5 Transceiver

By R. N. GADENZ

(Manuscript received July 17, 1980)

This paper describes the application of a recently developed large-scale-integration digital signal processor, the DSP, *to tone detection in a proposed digital* CCITT *No. 5 signaling unit. The design of the digital filters required in the receiver is discussed and an algorithm presented for tone detection. Two channels per* DSP *can be accommodated by using a sampling frequency of 8 kHz and a* DSP *clock of 5 MHz.*

I. INTRODUCTION

The application of a recently developed LSI digital signal processor, the DSP,[1] to tone detection in a proposed digital CCITT No. 5 signaling unit is described. This unit is to be part of an echo canceler terminal for No. 4 ESS international switching centers. At present, an analog configuration, which includes an analog transceiver, terminates trunks with CCITT No. 5 signaling.

Line signaling information in the CCITT No. 5 signaling system is transmitted via 2400- and 2600-Hz tones used either separately or in combination. The block diagram of Fig. 1 shows the section of the proposed digital receiver to be implemented using the DSP. The band-pass filters (BPFs) detect energy at one of the two signaling frequencies. The band elimination filter (BEF) serves both as a guard filter, to detect energy other than at the signaling frequencies, and as an attenuator of any signaling energy present in the input signal. The detector compares the outputs of the three filters and determines if either, or both, of the tones are present. This information is then used by the time validation circuit that follows. The circuit, called control and output logic in Fig. 1, determines if the tone(s) is(are) present for

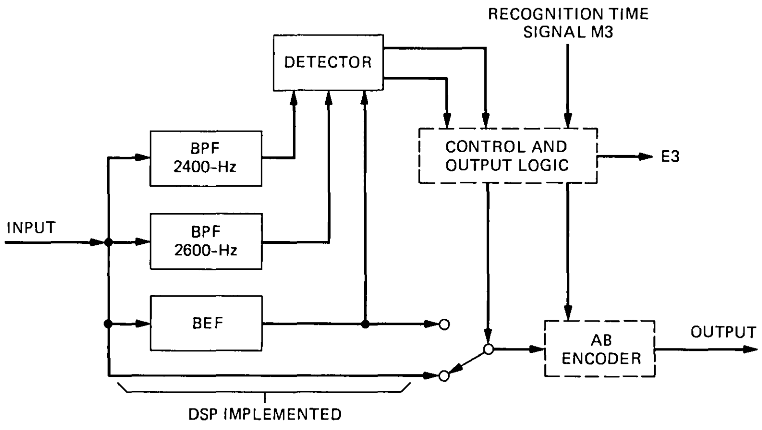


Fig. 1—Receiver.

the prescribed time, as determined by the M3 signal, and also controls the switch that selects either the input or the BEF output as the signal to be sent to the AB encoder and the output. The DSP implements the filtering and detection functions, excluding time validation.

II. FILTER DESIGN

2.1 Bandpass filters

The design of the digital BPFs was based on the performance of the corresponding filters presently used in the analog CCITT No. 5 signaling transceiver. For each filter, the transfer function in the z-domain, $T(z)$, was obtained from the transfer function in the s-domain, $T(s)$, via the bilinear transformation. A sampling frequency of 8 kHz was assumed. The transformation was accomplished using one of the interactive computer programs available, e.g., FILSYN.[2]

Optimization in the z-domain can then be used to correct for the warping effect of the bilinear transformation. A program exists for this purpose,[3] and it usually requires very few iterations to achieve excellent matching between the actual and the desired responses. Figure 2 illustrates the loss response, after optimization, for the 2400-Hz BPF, and Fig. 3, for the 2600-Hz BPF. Figures 2 and 3 also show the specifications, as derived from the analog filters mentioned above. The response of each digital filter is slightly better than that of the corresponding analog design.

Before implementing each transfer function, another step is required—the pairing of the poles and the zeros, and the ordering and scaling of the sections to be connected in cascade. This is done to avoid overflow and to reduce the quantization noise (error) due to rounding or truncation at the output of the DSP arithmetic unit (AU). In-house programs developed by K. Mina are available to assist in these operations.

The rounding (or truncation) of the transfer function coefficients also introduces a deviation in the frequency response which could be corrected by optimization. However, this was not necessary here,

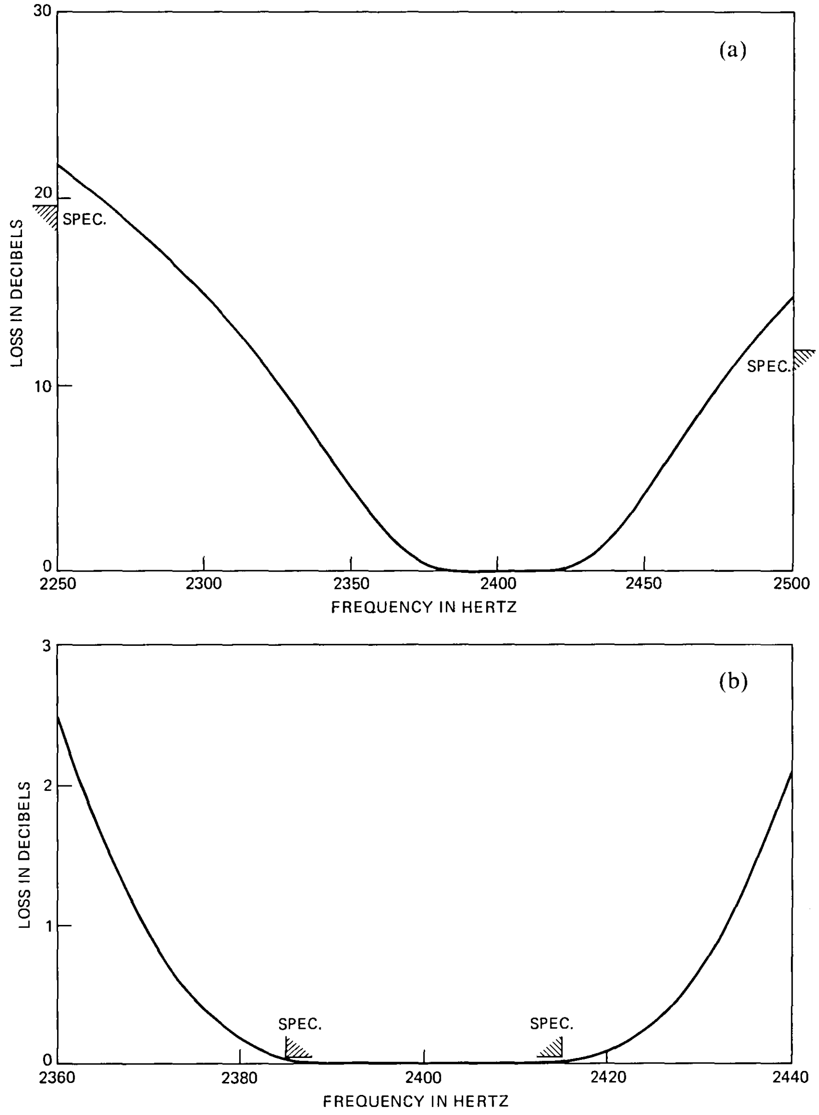
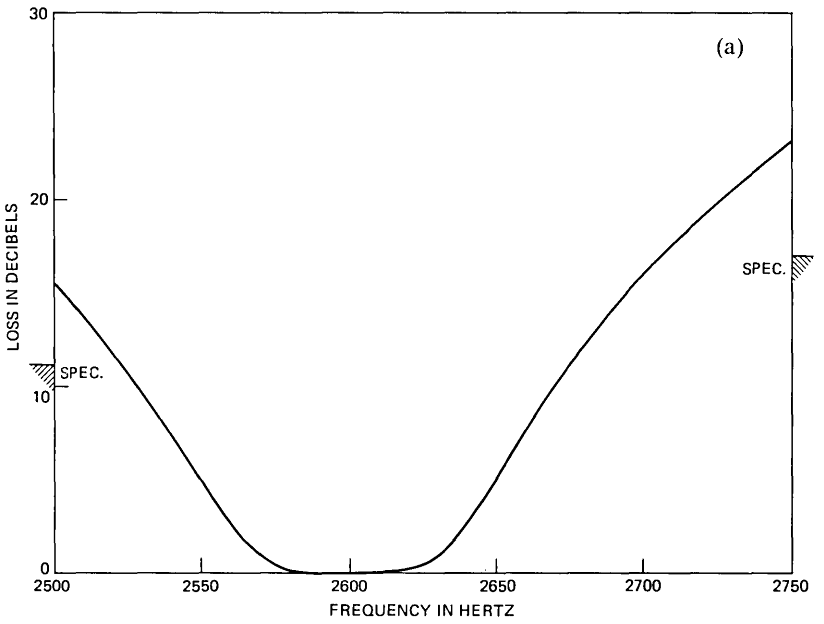


Fig. 2—Bandpass filter—2400 Hz. (a) Loss response. (b) Passband response—expanded scale.



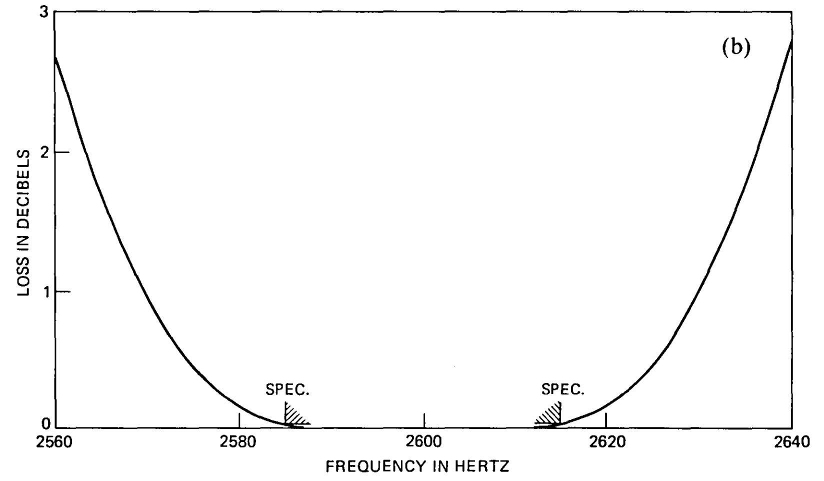


Fig. 3—Bandpass filter—2600 Hz. (a) Loss response. (b) Passband response—expanded scale.

because the DSP allows 16-bit coefficients and, thus, the distortion introduced in the loss response is negligible.

2.2 Band elimination filter

The analog signaling transceiver uses a two-section guard filter to detect energy other than at the signaling frequencies, and a separate three section BEF to remove tones, if present, from the input signal. In the digital signaling transceiver, only one BEF is to be used to perform both functions, as indicated in Fig. 1. Thus, the design of the digital BEF was not based on either of the two analog BEFs. Rather, a new analog BEF was designed to give the proper loss with only two second-order sections. For the same passband ripple, as in the 3-section BEF mentioned above, the passbands are now slightly reduced, but this is acceptable. The saving of one second-order section is crucial for the implementation of more than one channel per DSP, as discussed later.

The transfer function of this new analog BEF was then transformed into the z-domain and processed in a similar fashion as described before for the BPFs. Note that, in this case, instead of correcting the warping effect by optimization, a prewarped analog design could be used. The two approaches were in fact compared with very similar results. The optimization program in Ref. 3 is very efficient and, thus, attractive. The loss response for the digital BEF is shown in Fig. 4.

III. DETECTOR DESIGN

To perform the detection function, the outputs of the three filters are first rectified and then smoothed with a low-pass filter (LPF) (see Fig. 5). The signals P1, P2, and G then go to a threshold detector where the presence or absence of signaling tones is established.

One first-order section is used for each LPF. The corresponding loss and step responses are illustrated in Figs. 6 and 7, respectively.

The presence of tones is determined by the following criteria:

(*i*) $f_1 = 2400$ Hz is present if

$$t_l \leq P_1 \leq t_h \quad \text{and} \quad \frac{P_1}{G} > t_G$$

(*ii*) $f_2 = 2600$ Hz is present if

$$t_l \leq P_2 \leq t_h \quad \text{and} \quad \frac{P_2}{G} > t_G,$$

where the limits t_l, t_h, and t_G are determined from the CCITT recommendations for the receiver performance. The upper limit t_h should be 5 dB below the maximum amplitude of the tone (3 dBm0), and the lower limit t_l should be 19 dB below the maximum amplitude. Then, considering that P_1, P_2, and G are approximately the average values of the respective signals, and that for a tone, the average value is $2/\pi$ times the amplitude, we get:

$$t_h = 0.3579976064 \cdot A$$

$$t_l = 0.0714299132 \cdot A,$$

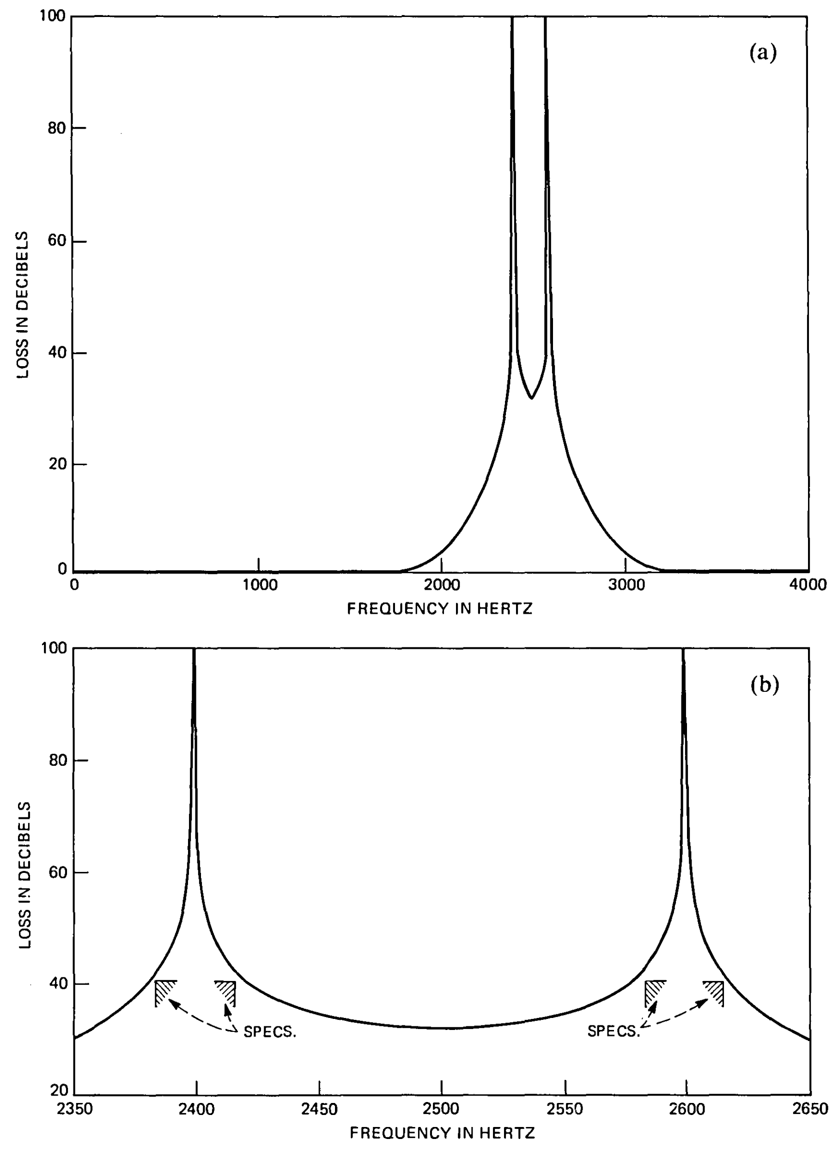


Fig. 4—Band elimination filter. (a) Loss response. (b) Stopband response—expanded scale.

where A is the maximum amplitude of either tone. The value of T_G is 5.

IV. THE DSP ALGORITHM

The filtering and rectification functions are easily implemented

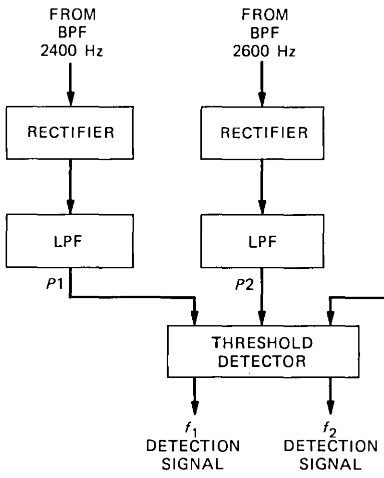


Fig. 5—Details of detector function.

using the DSP.[1] The order in which these operations are executed, as well as the way in which the threshold detector is implemented, are described below.

The s0 and s1 pins of the DSP are used to output the information on the presence of tones, thus reserving the normal (serial) DSP output for

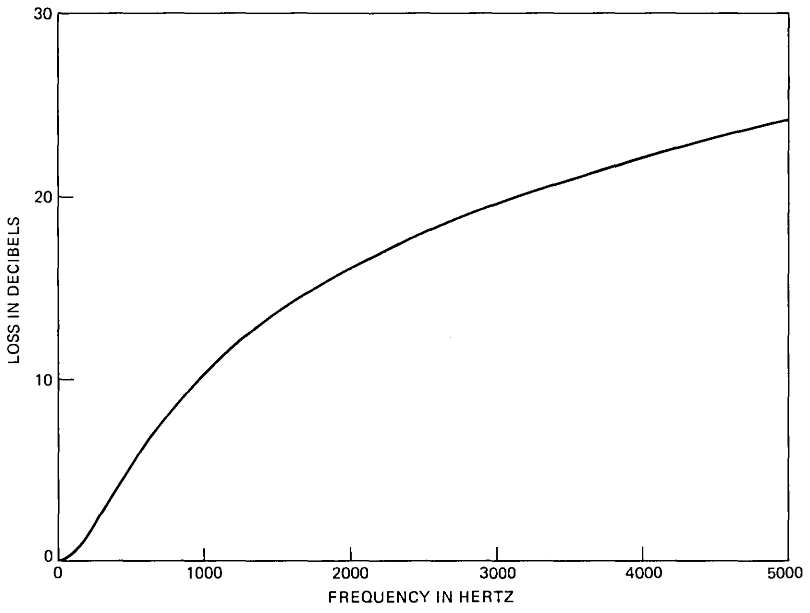


Fig. 6—Low-pass filter—loss response.

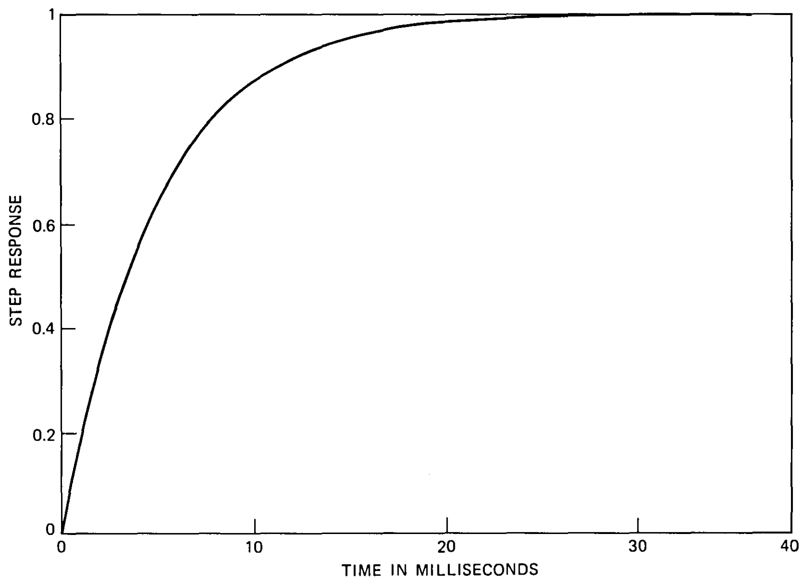


Fig. 7—Low-pass filter—step response.

the output signal of the BEF. The setting of S pins is determined as follows: If any condition required for tone f_1 or f_2 to be present is violated, a negative number is produced whose sign causes the corresponding pin (s0 or s1, respectively) to be set to zero. A logical one on either pin indicates that the corresponding tone is present, i.e., has been detected.

An alternative approach could be based on the following: If any condition required for tones f_1 or f_2 to be present is violated, a negative number is stored in memory; otherwise, a positive number is stored. The signs of these stored numbers could then be used to construct a 2-bit output word in which each bit indicates whether the corresponding tone is present or not. (Actually an 8-bit word would have to be output, but the remaining 6-bits would be irrelevant.) This approach would use the normal DSP output for both the tone information and the output of the BEF, which was not desirable in this application. However, a few instructions would be saved, thus, freeing up some processing time.

In more detail, the algorithm used here is as follows (see Fig. 8):

(a) Set the AU and I/O control registers, clear the RAM, and initialize the memory pointers.

(b) Read a μ-law PCM sample from the input buffer, convert it to linear format, and save the result s in RAM location 127.

(c) Process the linear sample s through the BEF and save the result r in the output register of the DSP AU.

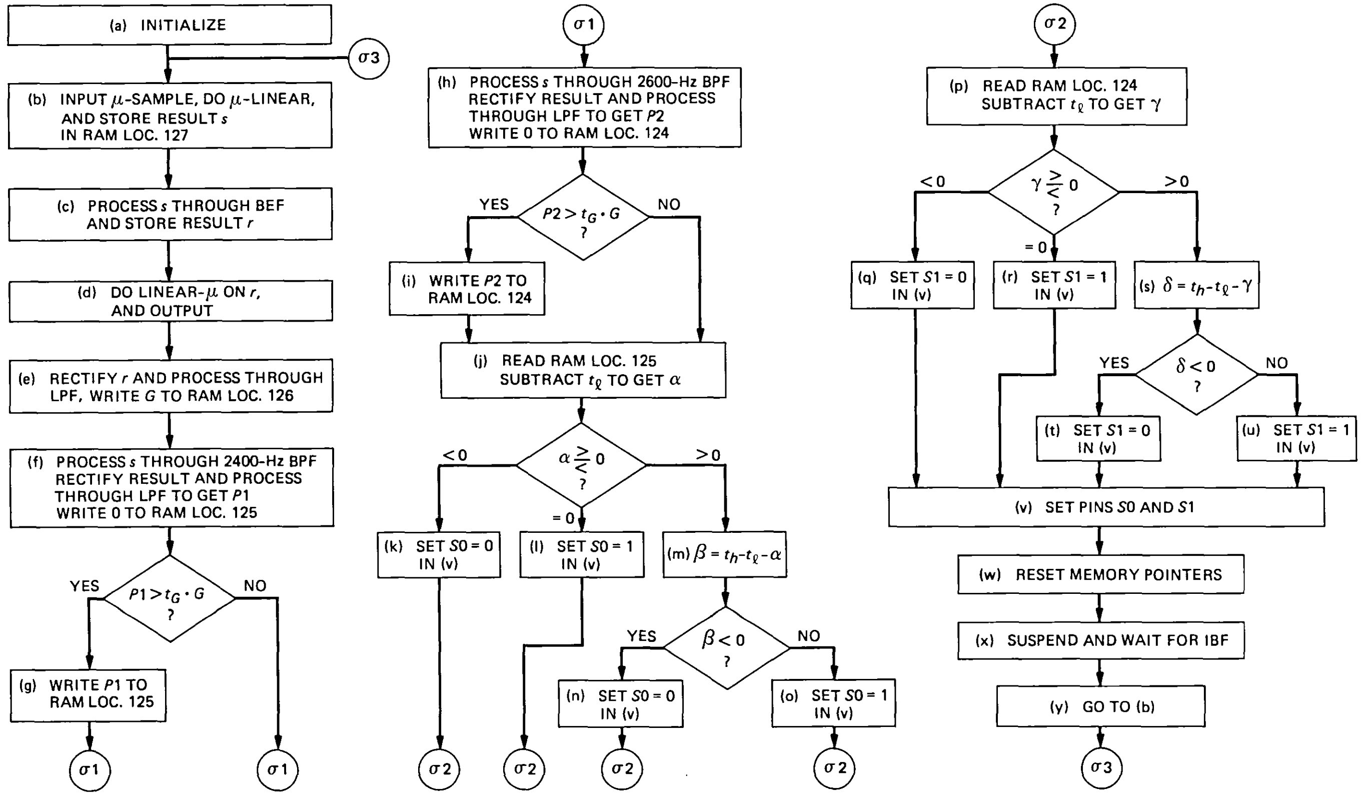

(a) INITIALIZE
σ3
(b) INPUT μ-SAMPLE, DO μ-LINEAR, AND STORE RESULT s IN RAM LOC. 127
(c) PROCESS s THROUGH BEF AND STORE RESULT r
(d) DO LINEAR-μ ON r, AND OUTPUT
(e) RECTIFY r AND PROCESS THROUGH LPF, WRITE G TO RAM LOC. 126
(f) PROCESS s THROUGH 2400-Hz BPF RECTIFY RESULT AND PROCESS THROUGH LPF TO GET P1 WRITE 0 TO RAM LOC. 125
YES
P1 > t_G · G ?
NO
(g) WRITE P1 TO RAM LOC. 125
σ1
σ1
σ1
(h) PROCESS s THROUGH 2600-Hz BPF RECTIFY RESULT AND PROCESS THROUGH LPF TO GET P2 WRITE 0 TO RAM LOC. 124
YES
P2 > t_G · G ?
NO
(i) WRITE P2 TO RAM LOC. 124
(j) READ RAM LOC. 125 SUBTRACT t_ℓ TO GET α
< 0
α ≷ 0 ?
> 0
= 0
(k) SET S0 = 0 IN (v)
(l) SET S0 = 1 IN (v)
(m) β = t_h - t_ℓ - α
YES
β < 0 ?
NO
(n) SET S0 = 0 IN (v)
(o) SET S0 = 1 IN (v)
σ2
σ2
σ2
σ2
σ2
(p) READ RAM LOC. 124 SUBTRACT t_ℓ TO GET γ
< 0
γ ≷ 0 ?
> 0
= 0
(q) SET S1 = 0 IN (v)
(r) SET S1 = 1 IN (v)
(s) δ = t_h - t_ℓ - γ
YES
δ < 0 ?
NO
(t) SET S1 = 0 IN (v)
(u) SET S1 = 1 IN (v)
(v) SET PINS S0 AND S1
(w) RESET MEMORY POINTERS
(x) SUSPEND AND WAIT FOR IBF
(y) GO TO (b)
σ3

Fig. 8—Flow chart for the DSP algorithm.

(d) Convert the output r of the BEF to μ-law format and output it.

(e) Rectify the output r of the BEF and process it through the LPF. Save the result G in RAM location 126.

(f) Process the linear sample s through the 2400-Hz BPF, rectify the result and process it through the LPF. The output is P_1. Write zero to RAM location 125.

(g) Compare P_1 with $t_G \cdot G$. If $P_1 > t_G \cdot G$, write P_1 into RAM location 125. Otherwise, a zero remains in that RAM location, which was written during the execution of step (f).

(h) Process the linear sample s through the 2600-Hz BPF, rectify the result and process it through the LPF. The output is P_2. Write zero to RAM location 124.

(i) Compare P_2 with $t_G \cdot G$. If $P_2 > t_G \cdot G$, write P_2 into RAM location 124. Otherwise, a zero remains in that RAM location, which was written during the execution of (h).

(j) Read RAM location 125. If the write operation in (g) did not occur because $P_1 \leq t_G \cdot G$, zero is obtained; otherwise, P_1 is read. Subtract t_l to get α.

(k) If the result α in (j) is negative, no f_1 tone is present and s0 will be set to zero in (v) at the end of the program; continue with (p).

(l) If the result α in (j) is zero, $P_1 = t_l$; for reasons of compactness in the DSP code, $t_h - t_l$ is added and subtracted; the result is still zero and its sign is also zero, as for a positive number; this causes s0 to be set to one in (v) at the end of the program, indicating that the f_1 tone is present; continue with (p).

(m) If the result α in (j) is positive, subtract it from $t_h - t_l$ to get β.

(n) If the result β obtained in (m) is negative, $P_1 > t_h$ and no f_1 tone is present; s0 will be set to zero in (v) at the end of the program; continue with (p).

(o) If the result β obtained in (m) is non-negative, $P_1 \leq t_h$ and the f_1 tone is present; s0 will be set to one in (v) at the end of the program; continue with (p).

(p) Read RAM location 124. If the write operation in (i) did not occur because $P_2 \leq t_G \cdot G$, zero is obtained; otherwise, $P2$ is read. Subtract t_l to get γ.

(q) If the result γ in (p) is negative, no f_2 tone is present and s1 will set to zero in (v); continue with step (v).

(r) If the result γ in (p) is zero, $P_2 = t_l$; for reasons of compactness in the DSP code, $t_h - t_l$ is added and subtracted; the result is still zero and its sign is also zero, as for a positive number; this causes s1 to be set to one in (v), indicating that the f_2 tone is present; continue with (v).

(s) If the result γ in (p) is positive, subtract it from $t_h - t_l$ to obtain δ.

(t) If the result δ obtained in (s) is negative, $P_2 > t_h$ and no f_2 tone is present; s1 will be set to zero in (v); continue with step (v).

(u) If the result δ obtained in (s) is non-negative, $P_2 \leq t_h$ and the f_2 tone is present; s1 will be set to one in (v).

(v) Set the s0 and s1 pins to the proper values.

(w) Reset the memory pointers.

(x) Suspend the DSP operation and wait for the input buffer to be filled with another sample. Execution resumes when the IBF flag goes high.

(y) Loop back to step (b).

The RAM is organized as follows:

Location		Contents
0	y_1	State variables for BEF
1	x_1	
2	y_2	
3	x_2	
4	x	State variable for LPF following the BEF
5	y_1	State variables for 2400-Hz BPF
6	x_1	
7	y_2	
8	x_2	
9	x	State variable for LPF following the 2400-Hz BPF
10	y_1	State variables for 2600-Hz BPF
11	x_1	
12	y_2	
13	x_2	
14	x	State variable for LPF following the 2600-Hz BPF
...	...	
124	P_2	See Fig. 5
125	P_1	
126	G	
127	s	Input sample in linear format.

Note that, for one channel, only the first 15 and the last four RAM locations are used.

This algorithm can be easily translated into the corresponding DSP code, which is then assembled[4] and stored in the DSP ROM. The filter coefficients are also stored in ROM, in line with the code. In the program, the quantities $-(1/8)t_G$, $-t_l/A$, and $(t_h - t_l)/A$ are used instead of the quantities t_G, t_l, and t_h, because they are more convenient.

The loop in the program has 77 instructions, independent of the path followed. Then, the code for two channels will have a loop of 154 instructions. With a 5-MHz clock, the instruction cycle is 800 ns. This implies that for a sampling frequency of 8 kHz, 156 instructions can be accommodated in a period (125 μs) between samples. Therefore, with the algorithm given here, two channels can be implemented per DSP, and 12 DSPs are needed per digroup of 24 channels.

The program has been tested both in software (using the DSP simulator[5]) and in hardware (using the DSP device with external ROM[6]), and was found to perform as expected.

V. CONCLUSION

Use of the DSP in implementing filtering and tone detection functions in the receiver of a proposed digital CCITT No. 5 signaling unit has been shown. Characteristics of the required digital filters have been described, along with a procedure for designing the filters. A way to realize the detector has been illustrated and the criteria used to determine the presence of tones have been presented. Finally, an algorithm has been given which, for a sampling frequency of 8 kHz and a DSP clock of 5 MHz, allows the implementation of two channels per DSP.

VI. ACKNOWLEDGMENTS

The author is grateful to D. M. Brady, A. M. Gupta, K. Mina, D. C. Stanzione, H. C. Kirsch, and J. R. Boddie for their support and suggestions.

REFERENCES

1. J. R. Boddie et al., "Digital Signal Processor: Architecture and Performance," B.S.T.J., this issue.
2. G. Szentirmai, "FILSYN—A General Purpose Filter Synthesis Program," Proc. of the IEEE, *65,* No. 10 (October 1977), pp. 1443–58.
3. M. T. Dolan and J. F. Kaiser, "An Optimization Program for the Design of Digital Filter Transfer Functions," in *Programs for Digital Signal Processing,* IEEE Press Book, 1979.
4. C. L. Semmelman, "Digital Signal Processor: Design of the Assembler," B.S.T.J., this issue.
5. J. Aagesen, "Digital Signal Processor: "Software Simulator," B.S.T.J., this issue.
6. J. R. Boddie, "Digital Signal Processor: Overview: The Device, Support Facilities, and Applications," B.S.T.J., this issue.

ACRONYMS AND ABBREVIATIONS

AAU	address arithmetic unit
AB	address bus
A/D	analog-to-digital conversion
ADPCM	adaptive differential pulse-code modulation
ASCII	American Standard code for information interchange
ATG	automatic test generation
ATP	all tests passed
AU	arithmetic unit
AUC	arithmetic unit control register
BA	block averaging
BCD	binary coded decimal
BEF	band elimination filter
BP	bandpass
BPF	bandpass filter
CAD	computer-aided design
CAROT	centralized automatic reporting on trunks
CCITT	International Telegraph and Telephone Consultative Committee (Comite Consultatif International Telegraphique et Telephonique)
CLKIN	clock input (also pin)
CLKOUT	clock output (also pin)
CPU	central processing unit
CST	control input (also pin)
CTR	clear to read (pin)
CTS	clear to send (pin)
CTU	carrier terminal units
CU	control unit
D/A	digital-to-analog conversion
DBS	multiplexed address and data bus (pins)
DFT	discrete Fourier transform
DIP	dual in-line package
DI	data input (pin)
DMA	direct memory access
DO	data output (pin)
DSP	digital signal processor
DSPMATE	hardware development system for DSP
dspsim	digital signal processor simulator
DTTU	digital treatment transmission unit
ESS	electronic switching system
$\overline{\text{EXM}}$	external memory control (pin)
$\overline{\text{EXE}}$	external memory control (pin)

FFT	fast Fourier transform
FIFO	first-in-first-out
FIR	finite impulse response
FSK	frequency shift keying
HGBEF	high-group band elimination filter
IBF	input buffer full (pin)
IBUF	input buffer register
ICK	input clock (pin)
IDFT	inverse discrete Fourier transform
IIR	infinite impulse response
I/O	input/output
IOAC	I/O, AAU, and control sections of DSP
IOC	register for DSP
IR	instruction register
IS	input stack
ISY	input synchronization (pin)
LAMP	logic analyzer for maintenance planning
LC	loop counter
LGBEF	low-group band elimination filter
LIU	line interface unit
LPC	linear predictive coding
LPF	low-pass filter
LSI	large-scale integration
LSL(LOCAL)	logic simulation language
LTX	computer-aided design system
MF	multifrequency
MOSFET	metal-oxide-semiconductor field effect transistor
MOS	metal-oxide semiconductor
MSI	medium-scale integration
NMOS	N-channel metal-oxide semiconductor
OBE	output buffer empty (pin)
OBUF	output buffer register
OCK	output clock (pin)
OS	output stack
OSY	output synchronization (pin)
PBX	private branch exchange
PC	program counter
PCM	pulse-code modulation
PR	program return
PROM	programmable read-only memory
QMFB	quadrature mirror filter bank
RAM	random access memory
RD	digital signal processor destination address register

RDA	digital signal processor auxiliary destination address register
REST	digital signal processor reset pin
ROM	read only memory
RX	digital signal processor X source address register
RY	digital signal processor Y source address register
RYA	digital signal processor auxiliary Y source address register
SBC	sub-band coding
SNR	signal-to-noise ratio
SOS	second-order section
SPICE	simulation program with integrated circuit emphasis
SSI	small-scale integration
STB	synchronization pulse (pin)
STR	digital signal processor status register
SYC	digital signal processor synchronization register
TTL	transistor-transistor logic
VCC	power supply (pin)
VF	voice frequency
VLSI	very large-scale integration
VSS	ground (pin)
XTAL	crystal connection (pin)

CONTRIBUTORS TO THIS ISSUE

John Aagesen, Civilingeniør (Electrical Engineering), 1956, Technical University of Denmark; M.A.Sc. (Electrical Engineering), 1959, University of Toronto; Bell Laboratories, 1959—. Mr. Aagesen was involved in early work on media characterization for a millimeter waveguide communications project. Later he worked in the area of computerized data acquisition and analysis. Other activities include characterization and fault location techniques for coaxial cable. Most recently he has been engaged in software development for microcomputer-controlled facility terminal systems.

E. James Angelo, Jr., B.S.E.E., 1939, North Carolina State University; S.M., 1948, Sc.D., 1952, Massachusetts Institute of Technology; Polytechnic Institute of Brooklyn, 1953-1968; Bell Laboratories, 1968—. Throughout his career, Mr. Angelo has been concerned primarily with the analysis and design of electronic circuits. In recent years, he has been engaged in the preparation of tutorial documentation for integrated electronic circuits of various kinds, both analog and digital, and their applications. Member, Tau Beta Pi, Eta Kappa Nu, Sigma Xi; Senior Member, IEEE.

Frank E. Barber, B.S., 1974, M.S. (Electrical Engineering), 1976, Lehigh University; Bell Laboratories 1976—. At Bell Laboratories Mr. Barbar has designed metal-oxide semiconductor static and dynamic memories. He has also designed memory sections for microprocessor integrated circuits, and is currently designing an MOS static, dual-port RAM. Member, Tau Beta Pi, Eta Kappa Nu.

Thomas J. Bartoli, B.S. (Electrical Engineering), 1974, Lafayette College; Bell Laboratories, 1966—. Since joining Bell Laboratories, Mr. Bartoli has been involved in the design of digital bipolar integrated circuits. For the past four years, he has been involved in the design of digital metal-oxide semiconductor very-large-scale-integration circuits for digital signal processing applications.

David A. Berkley, B.E.E., 1961, and Ph.D. (Applied Physics), 1966, Cornell University; Bell Laboratories, 1968—. Since 1975, Mr. Berkley has been supervisor of Electroacoustics and Acoustic Signal Processing in the Acoustics Research Department. His research has included work on nonlinear speech processing, hearing, echo suppression, and hands-free telephone conferencing.

Roy B. Blake, Jr., B.S.E.E., 1960, North Carolina State University, M.S.E.E., 1963, Duke University; Bell Laboratories, 1960—. In his early work Mr. Blake was engaged in the design of various high-performance digital magnetic recording devices. He then spent several years engaged in maintenance and diagnostic software design for large computer systems. He is currently supervisor of a group responsible for the analysis and design of both voice and data transmission systems over metallic cables. Member, Eta Kappa Nu, Tau Beta Pi.

James R. Boddie, B.S.E.E., 1971, Auburn University; S.M. and E.E., 1973, Massachusetts Institute of Technology; Ph.D. (Electrical Engineering), 1976, Auburn University; Bell Laboratories, 1976—. Mr. Boddie joined Bell Laboratories as a Post-Doctoral Fellow in the Acoustics Research Department where he implemented a dereverberation algorithm on an array processor. In 1977, he became a member of the technical staff in the Signal Processing and Integrated Circuit Design Department. He was an architect and circuit designer for the digital signal processor integrated circuit. In February, 1980, he became supervisor of the Digital Signal Processing Group.

Anthony C. Bolling, A.B. (Physics and Mathematics), 1965, Pfeiffer College; Ph.D. (Physics), 1970, Virginia Polytechnic Institute; Bell Laboratories, 1970—. Mr. Bolling is a member of the Facility Terminal Exploratory Development Department. His early work concerned analytical studies of voice-frequency transmission systems. Since 1978, his interest has been in the exploratory development of new types of voice-frequency transmission systems.

Milorad R. Buric, Dipl. Eng., 1971, University of Belgrade, Yugoslavia; M.S.E.E., 1975, Ph.D., 1978, University of Minnesota; Bell Laboratories, 1978—. Mr. Buric has done research in nonlinear system theory, digital signal processing, and computer architectures for real-time speech processing. His current interests include algorithms for very-large-scale-integration structures in speech processing and pattern recognition.

Steven P. Cordray, B.S. (Engineering Physics), 1977, University of Oklahoma; M.S. (Electrical Engineering), 1979, California Institute of Technology; Bell Laboratories, 1978–1981. Mr. Cordray has completed work on a microprocessor-controlled test set. The test set is designed to access digital pulse-code modulated signals on digital transmission facilities. He is now employed by Schlumberger. Member, Tau Beta Pi.

Ronald E. Crochiere, B.S. (Electrical Engineering), 1967, Milwaukee School of Engineering; M.S., 1968, and Ph.D. (Electrical Engineering), 1974, Massachusetts Institute of Technology; Bell Laboratories, 1974.— Mr. Crochiere was employed by Raytheon Company from 1968 to 1970. In 1970, he returned to the Massachusetts Institute of Technology to continue graduate studies for the doctorate and, at the same time, he became a member of the Research Laboratory of Electronics. At Bell Laboratories, he joined the Acoustics Research Department where he has been involved in research in decimation and interpolation, sub-band and transform coding of speech, and the measurement of digital speech quality. In 1976, he received the IEEE Acoustics, Speech, and Signal Processing (ASSP) Award for his paper on decimation and interpolation of digital signals. Mr. Crochiere is Secretary-Treasurer of ASSP's Advisory Committee and a member of its Technical Committee on Digital Signal Processing. He served for two years as technical editor on digital signal processing for ASSP Transactions.

Gobind T. Daryanani, B.S.E.E., 1963, Calcutta University; M.S.E.E., 1965, Virginia Polytechnic Institute; Ph.D.E.E., 1968, Michigan State University; Bell Laboratories, 1969—. Mr. Daryanani has worked on active filters, digital signal processors, and lightwave communications systems. He is currently supervisor of the Lightwave Circuits Group and is responsible for the development of regenerators and optical test sets.

Ismail I. Eldumiati, B.S.E.E., 1962, Alexandria University, Egypt; M.S.E.E., 1966; M.S., 1968, Ph.D. (Physics), 1970, University of Michigan; Bell Laboratories, 1972—. Initially, Mr. Eldumiati worked on the fault-locating system of the T4M high-speed digital transmission system. From 1974 to 1976, he worked on high-speed bipolar integrated circuits. Between 1977 and 1978, he was an architect and circuit designer for the BELLMAC-4^{TM} microcomputer. In July, 1978, he joined the Digital Signal Processing Group, where he worked on the design of the digital signal processor family. He is currently supervisor of the Digital MOS Circuits Group.

Robert L. Farah, B.S.E.E., M.S.E.E., 1977, Polytechnic Institute of New York; Bell Laboratories, 1977—. Mr. Farah initially worked on the design of digital filters for use as equalizers and echo cancelers for metallic special service circuits. Subsequently, he developed laboratory test systems for the study of digital and sampled data filters. Presently, he is engaged in low-bit-rate voice studies. Member, Tau Beta Pi, Eta Kappa Nu.

David L. Favin, S.M.E.E., 1952, Massachusetts Institute of Technology; B.S.E.E., 1950, University of Pennsylvania; Bell Telephone Laboratories 1952—. Mr. Favin has designed transmission measuring equipment including microwave sweepers; envelope-delay distortion measuring sets; impulse noise measuring sets; and microprocessor controlled, FFT-based measuring systems. He holds 22 patents. Member, Eta Kappa Nu, Sigma Psi, Tau Beta Pi.

James L. Flanagan, B.S. (Electrical Engineering), 1948, Mississippi State University; M.S., 1950, Sc.D., 1955, Massachusetts Institute of Technology; Bell Laboratories, 1957—. Mr. Flanagan is Head, Acoustics Research Department. He has project responsibilities for digital voice encoding, speech recognition and synthesis, electroacoustic systems, and transducers. Member, National Academy of Engineering.

Ronald L. Freyman, A.E.T. (Associate Electrical Technology), 1962, Pennsylvania State University; Astro-Electronics Division of RCA Corporation, 1962–1968; Bell Laboratories, 1968—. Mr. Freyman worked in metal-oxide semiconductor design and customer service organizations before he was assigned to the digital signal processor project.

Renato N. Gadenz, Ingeniero Civil Electricista, 1960, Universidad de Chile; M.S.E.E., 1962, University of Pittsburgh; Ph.D. (Electrical Engineering), 1972, University of California at Los Angeles. Mr. Gadenz has done research and teaching on network theory, automatic control as applied to electric machines, and computer-aided design. After joining Bell Laboratories in 1973, he was engaged in sensitivity analysis and design of active filters, and the development of software for testing microprocessor systems. More recently, he became interested in digital signal processing and was a member of the team that conceived, designed, and tested the digital signal processor. Mr. Gadenz is presently a member of technical staff in the Signal Processing and Integrated Circuit Design Department. Member, IEEE, IEEE CAS Society.

Jack A. Grant, Certificate (Electronics Industrial Technology), 1968, Ward Technical Institute; A.A.S. (Electronics Engineering Technology), 1974, County College of Morris, B.S. (Electronics Engineering Technology), 1977, Trenton State College; Bell Laboratories, 1968—. Mr. Grant initially worked in the military systems area until 1969. He then served four years in the U.S.A.F., Minuteman I.C.B.M. electronic maintenance. Returning to Bell Laboratories in 1972, he became involved in digital integrated circuits including evaluation, design, testing, test system design, and logic and timing simulation. Member, IEEE.

Nuggehally S. Jayant, B.Sc. (Physics and Mathematics), 1962, Mysore University; B.E., 1965, and Ph.D. (Electrical Communication Engineering), 1970, Indian Institute of Science, Bangalore; Research Associate at Stanford University, 1967-1968; Bell Laboratories, 1968—. Mr. Jayant was a visiting scientist at the Indian Institute of Science in 1972 and 1975. He has worked in the field of digital coding and transmission of waveforms, with special reference to robust speech communications. He is also editor of the IEEE Reprint Book, *Waveform Quantization and Coding.*

James D. Johnston, B.S. (Electrical Engineering), 1975, and M.S. (Electrical Engineering), 1976, Carnegie-Mellon University; Bell Laboratories, 1976—. Mr. Johnston is a member of the Acoustics Research Department. His research interests include wide and narrow bandwidth waveform coding techniques, fast small-scale digital processors, analog-to-digital and digital-to-analog techniques, the behavior of adaptation mechanisms in adaptive pulse-code modulation and adaptive differential pulse-code modulation, and analog circuit design. He has published in IEEE Trans. Commun., in the J. of Audio Eng. Soc., in conference records of the Int. Conf. on Commun., and Int. Conf. on Acoustics, Speech, and Signal Processing, and in IEE Electron. Lett. Member, Audio Eng. Soc.

Jack Kane, B.S. (Engineering), 1968, University of California, Los Angeles; M.S. (Electrical Engineering), 1969, Stanford University; Bell Laboratories, 1968—. From 1969 to 1971 Mr. Kane worked with the U.S. Public Health Service, National Institute of Health, Bethesda, MD. In 1971 he returned to Bell Laboratories, where he was involved in bipolar medium-scale integration and large-scale integration design until 1976. From 1976 to 1978, Mr. Kane was involved in catalog metal-oxide semiconductor (MOS) memory design. Since 1978, he has been involved in custom MOS LSI design. Presently, Mr. Kane is supervisor of an LSI design group. Member, Tau Beta Pi, IEEE.

Robert N. Kershaw, A.T. (Electronic Technology), 1963, Temple University; B.S. (Electrical Engineering), 1969, Lafayette College; M.S. (Electrical Engineering), 1972, Lehigh University; Bell Laboratories, 1963—. In his early work, Mr. Kershaw was involved with developing magnetic materials and magnetic memory devices. From 1969-1970, Mr. Kershaw was with Leeds and Northrup where he designed instrumentation systems. He returned to Bell Laboratories in 1970 and designed digital bipolar custom integrated circuits, as well as digital MOS integrated circuits for the digital signal processor. He is currently a supervisor in the Silicon Integrated Circuit Design Department. Member, Eta Kappa Nu, Tau Beta Pi, Phi Beta Kappa.

Joseph Kohut, A.A.Sc, 1962, Capitol Institute of Technology; Fairleigh-Dickinson University 1965-1967; Bell Laboratories, 1952—. At Bell Laboratories, Mr. Kohut has worked on underwater systems and in television systems. He is presently working in acoustics research in the design of hardware for synthesizing speech and music.

Carol A. McGonegal, B.A. (Mathematics), 1974, Fairleigh Dickinson University; M.S. (Computer Science), 1977, Stevens Institute of Technology; Bell Laboratories, 1967—. Ms. McGonegal is a member of the Acoustics Research Department where she has worked on problems in digital filter design, digital speech processing, computer voice response, and speaker verification.

Joseph P. Olive, B.S. and M.S. (Physics), 1964, University of Chicago; Ph.D. (Physics), 1969, University of Chicago; M.A. (Music Composition), 1969, University of Chicago; Bell Laboratories, 1969—. J. P. Olive's work at Bell Laboratories has centered on various problems of speech synthesis.

Nadia Sachs, B.S. (Electrical Engineering), 1978, Polytechnic Institute of New York; M.S.E. (Electrical Engineering), 1979, Princeton University; Bell Laboratories, 1978—. Mrs. Sachs has worked on various applications of digital signal processing.

Charles L. Semmelman, B.E.E., B.Sc. (Physics), 1939, Ohio State University; Bell Telephone Laboratories, 1940-1942 and 1946—. Signal Corps Engineering Laboratories, 1942-1946. Mr. Semmelman has been engaged in the design of filters and equalizers, and supervised a group which developed computer programs for the analysis, synthesis, and optimization of transmission networks and systems. He is currently engaged in high-level simulation and compiler development. Associate, Sigma Xi; Member, Tau Beta Pi, Eta Kappa Nu, Pi Mu Epsilon, Sigma Pi Sigma.

John S. Thompson, B.S.E.E., 1962, B.S.E.P. 1963, Lehigh University; M.S.E.E., 1965, Ph.D., 1967, University of Rochester; Bell Laboratories, 1967—. Mr. Thompson has been involved in research on algorithms and architectures for digital signal processing and on problems in digital switching and signal encoding. He is currently engaged in studies of digital techniques in small business communication systems. Member, Eta Kappa Nu, Tau Beta Pi, Phi Beta Kappa, Sigma Xi, IEEE.

James Tow, B.S., 1960, M.S. (Electrical Engineering), 1962, and Ph.D. (Electrical Engineering) 1966, University of California, Berkeley; Bell Laboratories, 1966—. Mr. Tow has been concerned with computer-aided network analysis and design and with the implementation of practical active filters for telecommunication systems. He is currently engaged in the application of digital signal processing techniques to signaling systems. Member, IEEE, Eta Kappa Nu, Phi Beta Kappa.

John W. Upton, B.S. (Electrical Engineering), 1971, Carnegie-Mellon University; Western Electric, 1971–1972; Bell Laboratories, 1972—. At Western Electric, Mr. Upton worked on the development of the processor unit of the CLC-1 computer used by the Safeguard System. Since joining the Acoustics Research Department at Bell Laboratories he has designed computer interfaces and support equipment. His present work involves the design of microprocessor-based hardware for digital signal processing. Member, IEEE.

Stephen M. Walters, B.E.E., 1974, Auburn University; M.S., 1976, Ph.D. (Electrical Engineering), 1977, Virginia Polytechnic Institute and State University; Bell Laboratories, 1977—. At Bell Laboratories, Mr. Walters has been engaged in design and development of the digital signal processor (DSP) and its supporting development system, DSPMATE. He is presently supervisor of the Peripheral Control Group, concerned with digital terminals for the No. 4 ESS. Member, IEEE, Phi Kappa Phi, Eta Kappa Nu, Sigma Xi.

David P. Yorkgitis, B.S. (Mathematics), 1977, Carnegie-Mellon University; M.S. (Electrical-Biomedical Engineering), 1979, Carnegie-Mellon University; Bell Laboratories, 1979—. Mr. Yorkgitis has worked on digital signal processor programs to perform signal processing functions in transmission systems.

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