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

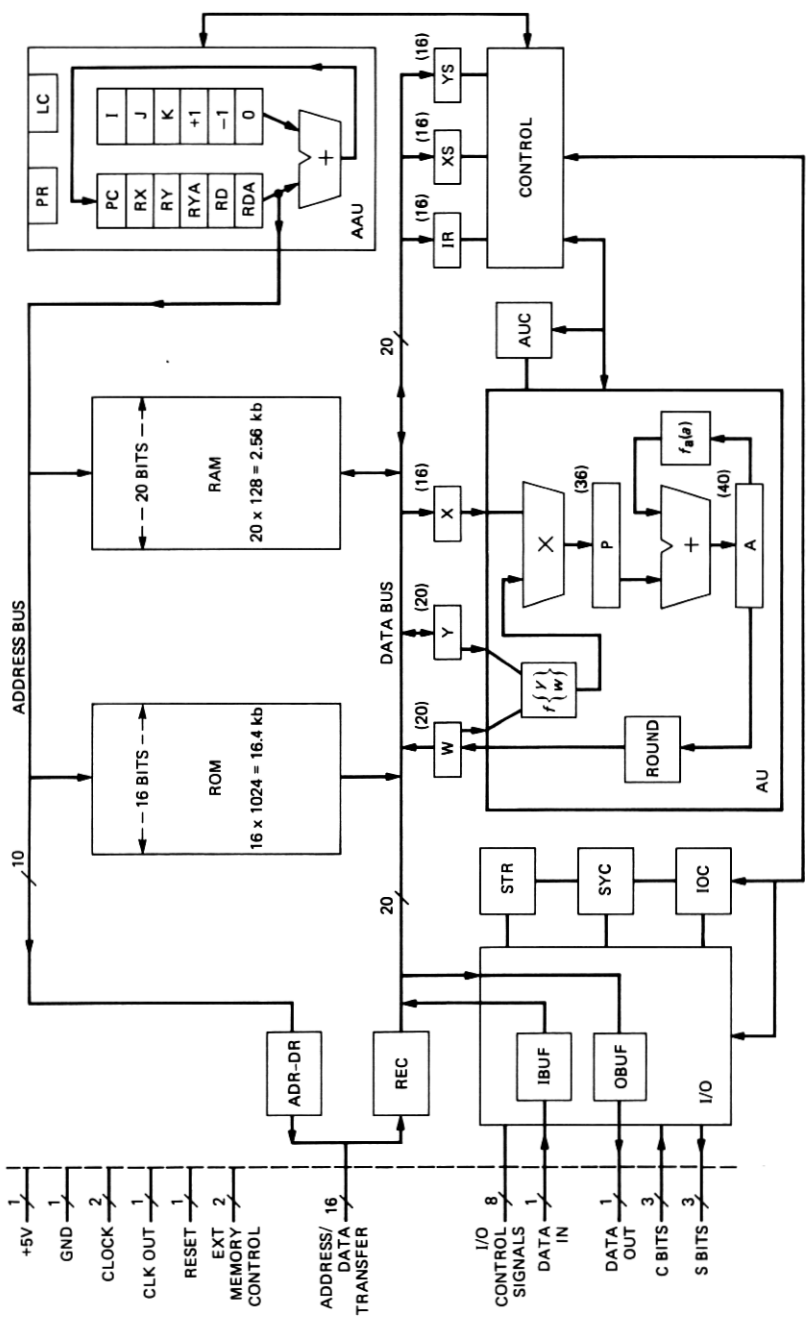

Fig. 1—Digital signal processor architecture.

(*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

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

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-

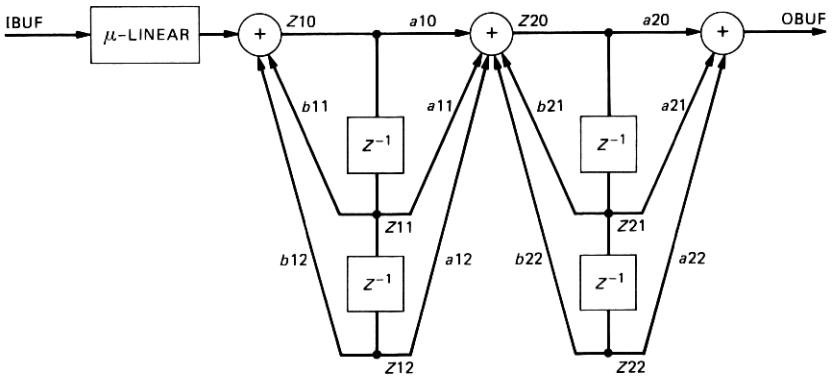

Fig. 2—Fourth-order recursive filter.

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

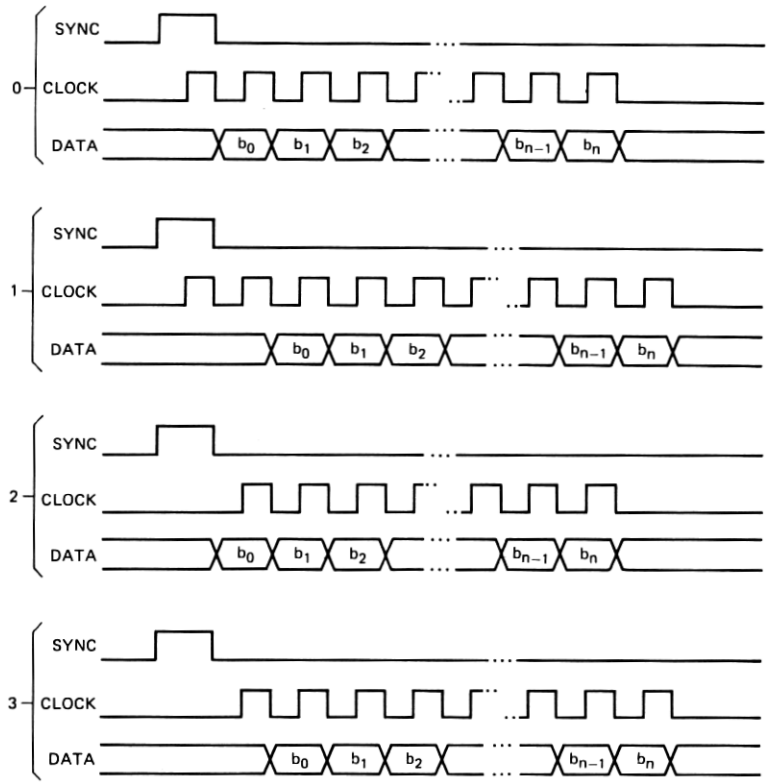

Fig. 3—Input/output active formats.

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,

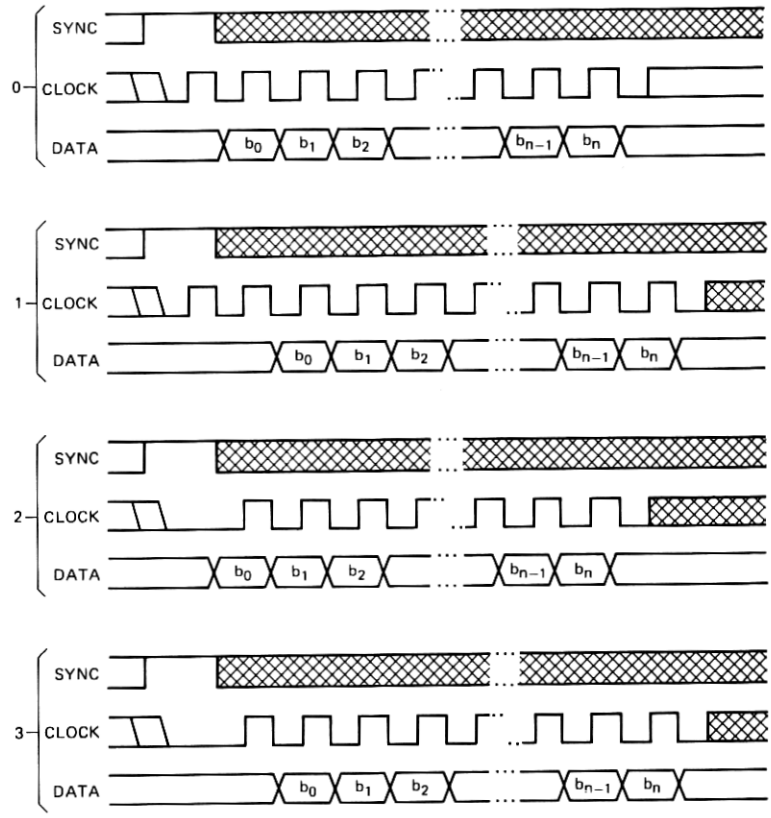

Fig. 4—Input/output passive formats.

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.