A Custom VLSI Chip Set for Digital Signal Processing in High-Speed Voiceband Modems

SHAHID U. H. QURESHI, SENIOR MEMBER, IEEE, AND HASSAN M. AHMED, MEMBER, IEEE

Abstract—Systems modems intended for use in relatively large private networks are characterized by high performance, reliability and flexibility to support network management, and multiple modes of operation and user features. This paper describes a programmable digital signal processor which is teamed with a 16-bit microprocessor in a dual processor architecture satisfying the requirements of high-speed voiceband systems modems. The architecture of the two custom integrated circuits which form the basis of the signal processor is presented. This processor has novel arithmetic, data structure address generation, and program flow-control capabilities, which result in a high utilization of the arithmetic unit and a low program overhead for housekeeping tasks. Some of these features are illustrated by programming examples.

I. Introduction

VOICEBAND modems operating at bit rates of 4800 bits/s and higher have become one of the most important applications of programmable digital signal processors (DSP) [1]–[3]. A single DSP with adequately high computational throughput can be time-shared to perform all modem signal processing functions such as filtering, modulation, demodulation, and adaptive equalization [2]–[3]. Perhaps the greatest advantage of programmable DSP technology is its flexibility, which permits sophisticated modem structures and techniques to be implemented with ease.

Over the past decade, while modem performance and transmission rate have increased, there has also been a trend toward incorporation of multiple modes of operation and an increasing number of user-oriented features into high-speed modems. Such fully featured "systems" modems are aimed at so-called "premium end-users" who typically require a large but efficient, reliable, and cost-effective data communication network based, at least partially, on leased voiceband telephone links.

In order to understand the features and architectural requirements of systems modems, consider the simplified network configuration of Codex 2600 series modems shown in Fig. 1. The point-to-point modems provide a transmission rate as high as 16.8 kbits/s over conditioned lines. An adaptive rate system implemented in these modems maximizes throughput by continually sensing telephone line parameters and operating at the highest rate which can be supported by channel conditions. Built-in time-division multiplexers allow up to six synchronous or asynchronous data terminal applications to share the single high-speed link.

Multipoint or multidrop links are economically attractive for interactive systems, such as airline reservation or credit card authorization, in which a large database must be accessed for a relatively short time by a number of remote stations. The central site "master" modem transmitter broadcasts inquiries, or "polls," from the communications control unit of a host computer. Addressed to successive remote terminals, these polls are received via "slave" modem receivers. As each remote terminal is polled, it responds via the "slave" modem transmitter which sends a short preamble followed by the data message. Reduction of the length of this preamble is an important factor in determining the link efficiency. Thus the "master" receiver can quickly adapt to receive typically short bursts of data from a number of transmitters over different channel characteristics. Rate information imbedded in the preamble enables different "slave" modems to transmit data at selectable rates of 9600, 7200, and 4800 bits/s. "Slave" modem receivers are able to "train on data," that is, set up without the help of a preamble or training sequence from the "master" modem transmitter. This enables remote locations to be added to a multidrop link without disruption of the system already on line.

A network manager or control system connected to the central site modem through a special terminal interface, enables monitoring, troubleshooting, self-test, and reconfiguration of the network from a central location. Network control information is carried to and from remote modems via a secondary channel in the low-frequency portion of the telephone-line spectrum. Typically, 110 bit/s frequency-shift keying is employed. All modems perform on-line monitoring of the received signal quality and modem-terminal interface signals. Telephone-line parameters are deduced from the received signal and alarms are sent to the central site controller if a particular disturbance

Manuscript received February 1, 1985; revised August 29, 1985. This paper was presented at the Workshop on VLSI Signal Processing, University of Southern California, Los Angeles, November 27–29, 1984.

S. U. H. Qureshi is with the Codex Corporation, Mansfield, MA 02048.
H. W. Ahmed was with the Codex Corporation. He is now with the DSP Division, Analog Devices, Inc., Norwood, MA 02062.

IEEE Log Number 8406183.

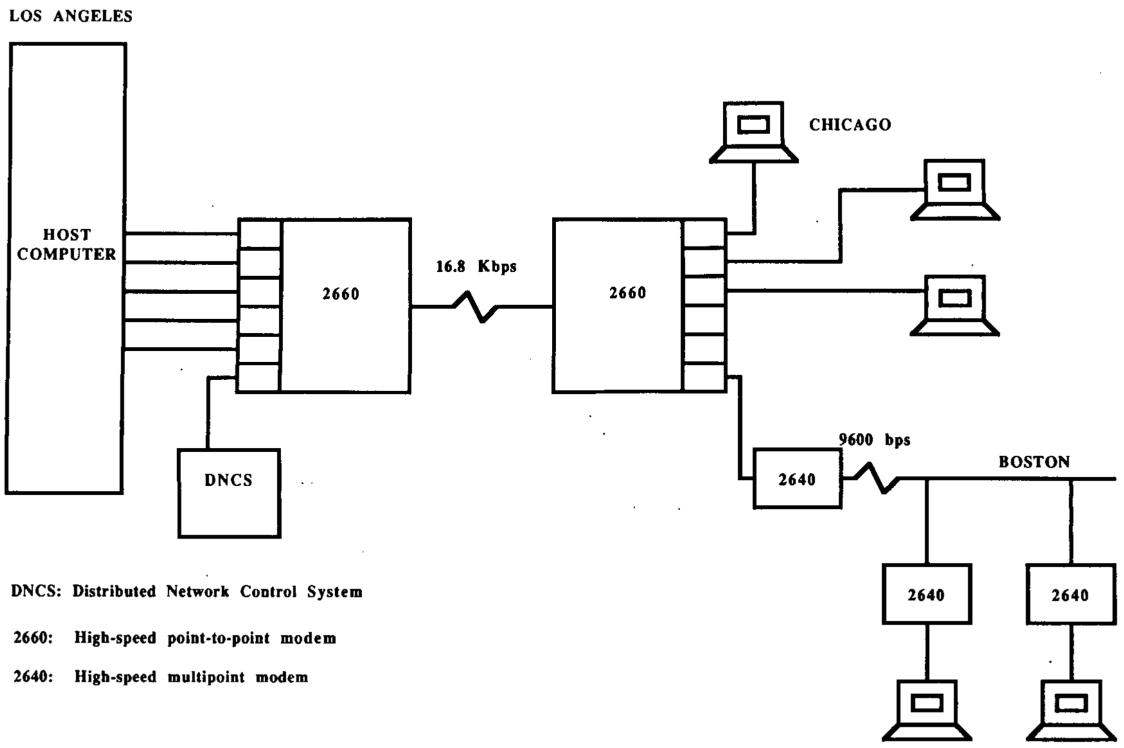

LOS ANGELES
HOST COMPUTER
2660
16.8 Kbps
2660
CHICAGO
DNCS
9600 bps
2640
BOSTON
2640
2640
DNCS: Distributed Network Control System
2660: High-speed point-to-point modem
2640: High-speed multipoint modem

Fig. 1. Simplified data communication network.

exceeds a selected threshold. These and other network management functions built into the modems can be used to rapidly diagnose faults and provide corrective actions from the central site in the network, without requiring operator assistance at the remote locations.

Let us now consider a simplified functional block diagram of a modern high-speed voiceband modem shown in Fig. 2. Such modems almost universally use phase-shift keying (PSK) for lower speeds, e.g., 4800 bits/s and combined phase and amplitude modulation, or equivalently, quadrature amplitude modulation (QAM) [4], for higher speeds from 9600 bits/s to 16 800 bits/s. At the high rates, where noise and other channel distortions become significant, modems using coded forms of QAM such as trellis-coded modulation [5], [6] are being introduced (e.g., in the Codex 2660 shown in Fig. 1) to obtain improved performance.

Referring to Fig. 2, the incoming bit stream is scrambled or randomized to remove repetitive data patterns and spread the transmitted signal power density almost uniformly across the band. A 9.6 kbit/s QAM modem sends 4 bits/symbol interval. The symbol rate is 2400 baud. In each symbol interval, the encoder maps 4 bits into one of 16 possible signal points, or pair of rectangular coordinates, which are then filtered and used to modulate the in-phase and quadrature cosine and sine carrier signals. At the receiver, after filtering, adaptive equalization, and demodulation, a pair of received coordinates is generated. The decision circuit selects the signal point that is closest to the received point. The selected point is then decoded and descrambled to produce a replica of the transmitted bit stream. The decision circuit also generates error signals that are used to update the coefficients of the adaptive equalizer and the demodulator phase angle. Other functional elements of the receiver which involve signal processing are timing recovery, automatic gain control, and received line signal detection. Receiver setup, or initialization of its adaptive elements, is typically accomplished during a training period. A known training signal is transmitted and a synchronized version of this signal is generated in the receiver to acquire information about the channel characteristics.

Note that analog circuits are required only for the telephone line interface, the associated filters, gain adjustment, digital-to-analog and analog-to-digital converters. Virtually all important functional elements are implemented using digital signal processing with a total computational requirement of well over a million arithmetic operations per second.

Recognizing the severe computational burden associated with modems and a number of other signal processing applications, most first-generation commercially available VLSI DSP circuits [7]–[11] provide for very fast multiplication and accumulation. However, these devices generally lack good address-generation capability, a flexible and efficient microprocessor interface, and the ability to connect sufficient external program and data memories [12]. Lacking these capabilities makes these DSP chips less than suitable for systems modem products, which must support multiple modes of operation, built-in network management, circuit quality monitoring, and multiplexing functions.

This paper describes the architecture of a custom DSP chip set, called the signal processing element (SPE), de-

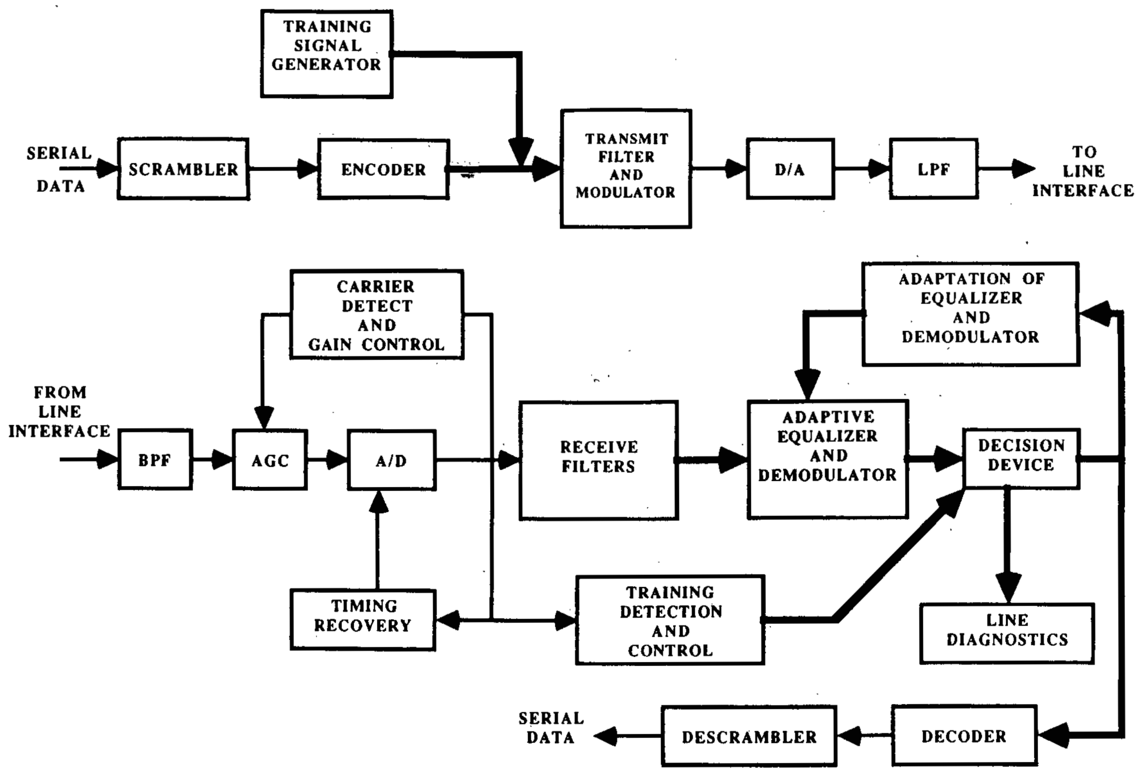

TRAINING SIGNAL GENERATOR
SERIAL DATA
SCRAMBLER
ENCODER
TRANSMIT FILTER AND MODULATOR
D/A
LPF
TO LINE INTERFACE
CARRIER DETECT AND GAIN CONTROL
ADAPTATION OF EQUALIZER AND DEMODULATOR
FROM LINE INTERFACE
BPF
AGC
A/D
RECEIVE FILTERS
ADAPTIVE EQUALIZER AND DEMODULATOR
DECISION DEVICE
TIMING RECOVERY
TRAINING DETECTION AND CONTROL
LINE DIAGNOSTICS
SERIAL DATA
DESCRAMBLER
DECODER

Fig. 2. Modem functional block diagram.

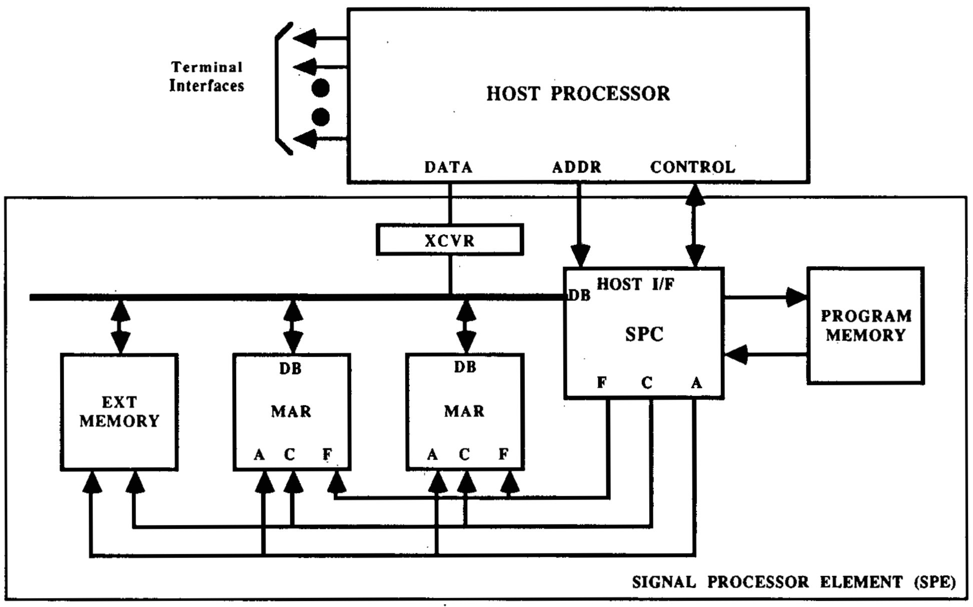


Fig. 3. Dual-processor systems modem architecture.

veloped during the 1978–1982 time period. Teaming the SPE with a Motorola MC68000 16-bit microprocessor results in a powerful and flexible base for the Codex 2600 series of systems modems. The dual-processor loosely-coupled architecture of the modem, shown in Fig. 3, is supported by the flexible microprocessor interface of the SPE. Each processor is assigned only those tasks that are best suited to its capabilities. The microprocessor performs control, management, and data movement functions, and provides for front-panel and terminal interface capabilities, while the SPE executes all the signal processing tasks including extraction of circuit-quality information.

Novel features of the processor in the areas of arithmetic functions, addressing modes, and program flow control are emphasized and illustrated by programming examples. A unique SPE-host microprocessor interface is also described.

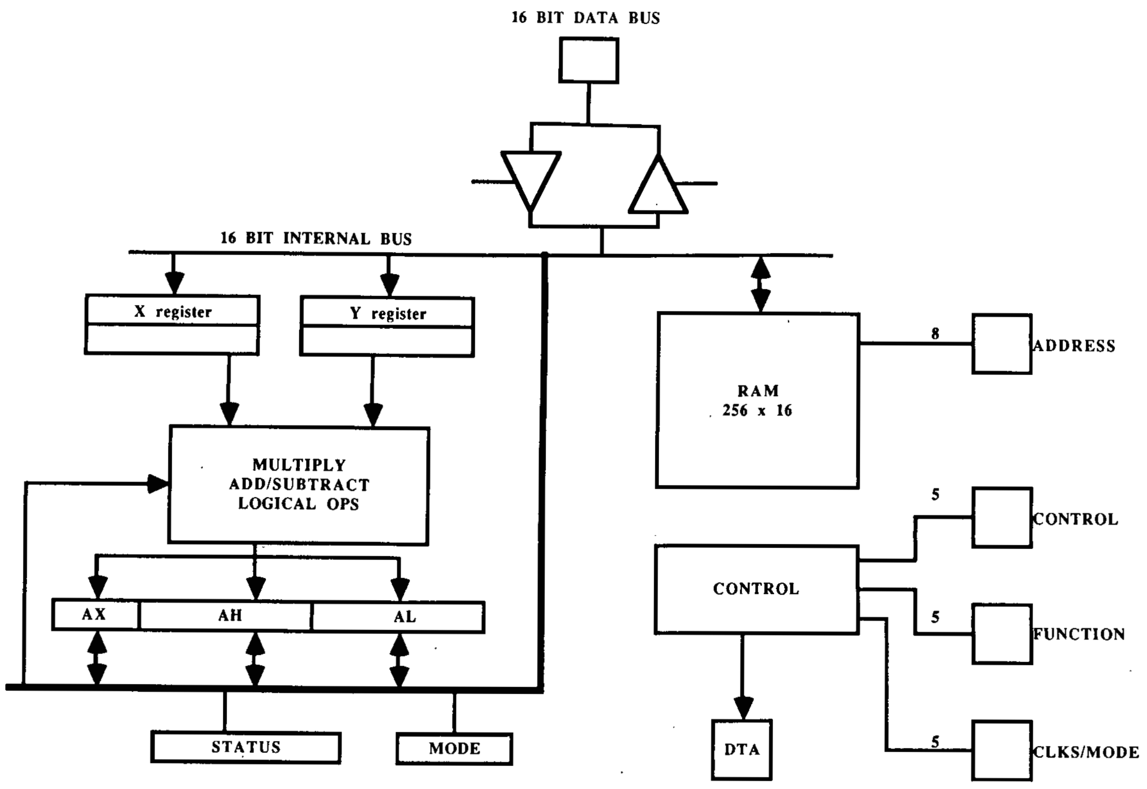


Fig. 4. MAR circuit.

II. The Signal Processing Element (SPE)

The SPE is a custom digital signal processor having its own instruction set and operating its own program. Designed to operate loosely coupled with another processor, the SPE periodically communicates with the host to transfer data and signal processing task information. Interprocessor communication is generally initiated by the host processor through a unique host interface.

The SPE is based on two custom integrated circuits. The first of these is the MAR or "Multiplier-Accumulator-RAM" circuit. The MAR is the arithmetic engine of the SPE having a two-cycle pipelined multiply accumulator and 256 words of on board data memory. It is capable of performing 32 arithmetic functions and has a very flexible data movement facility.

The SPC or signal processing controller circuit generates the necessary signals to control the MAR, external data memory and any input/output devices which may be connected to the SPE data bus. Two MAR's can be simultaneously controlled by a single SPC for complex arithmetic operations. The SPC is also responsible for all data address generation and program flow control.

The basic SPE operations fall into essentially four categories: arithmetic, data-structure control, program-structure control, and host operations. Program control and host operations lie in the domain of the SPC while arithmetic is performed by the MAR. Data-structure control is a shared activity in which the SPC maintains addressing parameters while the data structure resides in the MAR memory.

The remainder of the paper will focus on the architectures of the MAR and SPC chips.

III. The Multiplier-Accumulator-RAM Circuit

The MAR, depicted functionally in Fig. 4, is a single-bus arithmetic engine operating at a 300 ns bus-transfer cycle. It is segmented both functionally and physically into two major components, the RAM and the Arithmetic Unit (AU). A 16×16-bit two-cycle pipelined multiplier with 40 bits of accumulation is at the heart of the AU. Arithmetic operations are determined by a five-bit function field (F0-F4) which is supplied to the MAR by a controller, such as the SPC, on each instruction cycle. The on-board, 256-location RAM (256×16) provides for the storage of operands and results of arithmetic operations. These operands can also reside in the various on-chip registers or be communicated from the external data bus. A six-bit control field (C0-C5) designates the data movement on each instruction cycle. This field must also be supplied by the controller on each instruction cycle. Although the MAR is intended for high-speed synchronous operation with the SPC circuit providing control, it may also be used as an asynchronous peripheral to an MC68000-type microprocessor.

Overall MAR operation may be viewed as follows. During each instruction cycle, the MAR decodes its function and control fields and performs both a data transfer and a function. Data transfers can occur either totally internal to the MAR, e.g., between registers or using the internal memory, or these transfers can involve components exter-

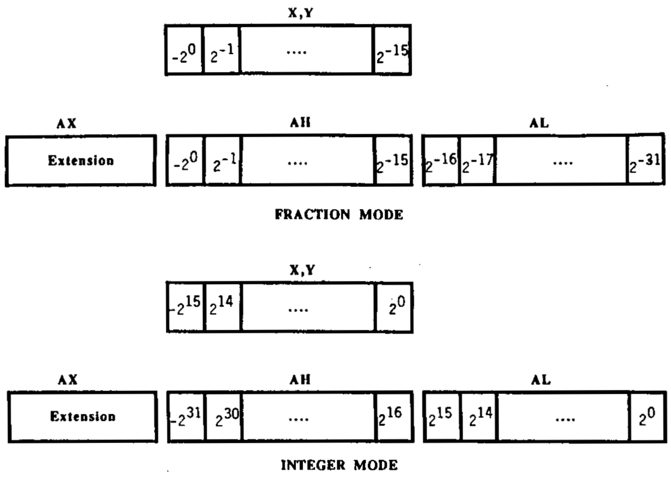
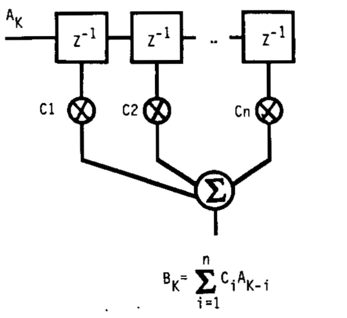


Fig. 5. INTEGER/FRACTION mode data formats.



```
YPP   C1      ; Y<- C1
MPY   A       ; X<-A   , start multiply
YPP   C2      ; Y<- C2
MAC   A       ; X<-A   , start mul-acc
YPP   C3
MAC   A
 .
 .
 .
YPP   Cn
MAC   A
NOP           ; Two NOPs to complete
NOP           ; final multiplication
             ; results are now in  accumulator
```

Fig. 6. FIR filter implementation.

nal to the MAR connected on the data bus (DB0-DB15). Functions can simply set one of the many MAR modes or initiate an arithmetic operation, which is completed during subsequent instruction cycles. The multiple-cycle nature of arithmetic operations does not impose a severe throughput penalty on the SPE because the well-structured computations found in digital signal processing tasks are well-suited to pipelining.

The MAR has several modes that may be programmed into a MODE register and altered dynamically. Two significant modes are INTEGER and FRACTION. In the former case, all operands and results are treated as integer quantities, while in the latter mode, fixed point quantities are assumed. The radix point is implied to the right of the least significant bit of the X, Y, and AL registers in INTEGER mode. Alternately, it is assumed to be to the right of the most significant bit of X, Y, and AH in fractional mode. The data formats and individual bit weights are shown in Fig. 5 for each case. AX serves as an extension for the 32-bit accumulator formed by AH and AL, thus providing a full 40 bits of accumulation. This feature is of great utility in avoiding intermediate overflow during long sequences of accumulations.

The arithmetic unit shown in Fig. 4 embodies a two-cycle radix-4 Booth Multiplier [13]. The choice of two-cycle operation merits some comment since multiplication-accumulation is the most common operation in signal processing and its rapid execution, e.g., in a single cycle, is of utmost importance. First, note that the MAR architecture has only a single bus for data transfer to its operand registers X and Y. This choice was made consciously in order to limit the die size of the chip. Since multiplication is a dyadic operation requiring two operands per operation, two cycles are necessary to set up the operation using the single data bus in the MAR. These two data transfers can occur concurrently with the multiplication in a pipelined fashion without any significant throughput degradation, provided that there are many such operations to perform in sequence. This is often the case of DSP tasks such as filtering. To illustrate this point, consider the execution of a finite-impulse response (FIR) filter.

Fig. 6 depicts an FIR filter and the sequence of MAR operations necessary to realize that filter. The input samples to the filter are denoted $\{A_k\}$, the outputs are $\{B_k\}$ while the coefficient set is $\{C_i\}_{i=1}^{n}$. Inline code with immediate reference to the operands is written to illustrate the operation of the AU, however in practice, the code would simply be a short loop with the addresses of the operands being supplied by the SPC. The instructions are self-explanatory: *YPP* refers to loading the Y register, *MPY* refers to loading the X register and initiating a multiplication ($Acc = X*Y$), and *MAC* means load the X register and initiate a multiply-accumulate ($Acc = Acc + X*Y$). Referring to the code, the second instruction schedules the first multiplication which takes place during the third and fourth instructions and initializes the accumulator. While the multiplier is busy during these two cycles, the next operation is set up by loading the filter coefficient into Y, the sample into X, and initiating a multiply-accumulate. This operation starts at the end of the fourth instruction which is, conveniently, when the multiplier becomes available. The remaining instructions in the filter code are identical to the third and fourth instructions that form the FIR filter kernel. After the last tap of the filter is computed, two cycles are required to allow the final operation to complete. This is the only penalty associated with the pipeline and can often be avoided in a real application by starting another operation. An nth-order FIR filter can be executed in $2n+2$ cycles. The penalty attributed to the two-cycle multiplier is insignificant for large n because the architecture is fundamentally limited by the single data bus.

A rich complement of operations is supported by the MAR AU. These are listed in Table I. Of particular interest to note are the many multiplication modes, including a facility for rounding to enhance numerical stability of various DSP algorithms, and MAX, MIN, and ABS. Space does not permit describing each instruction in detail, so only some of the unique features will be covered.

The FIR filter example indicates that a two-cycle penalty is incurred whenever the multiplication pipeline is disturbed. This penalty was kept to a minimum in the example by ensuring that the pipeline could be maintained throughout the calculation. Additional architectural features have been incorporated in the MAR to ensure that the pipelined operation can be readily realized for other DSP tasks as well. One such innovation is the concept of

TABLE I
MAR OPERATIONS

OPERATION	MNEMONIC	MODE	MATHEMATICAL DESCRIPTION
DIFFERENCE	DIF		$(Y/2 - X/2) \rightarrow Y$ and X
MULTIPLY	MPY, DMP		$\pm X * Y \rightarrow A$
MULTIPLY-ROUND	MPR, DMR	FRAC INTG	$2^{-16} \quad \pm X * Y \rightarrow A$ $\pm X * Y \rightarrow A$
MULTIPLY-ACCUMULATE	MAC, DMA		$A \pm X * Y \rightarrow A$
ADD	ADD		$A + X \rightarrow A$
SUBTRACT	SUB		$A - X \rightarrow A$

COMPARE	CMP	MAG = 0
		MAG = 1
			$Am - X$; sign $\rightarrow S$
			$\lvert Am\rvert - \lvert X\rvert$; sign $\rightarrow S$

MAXIMUM	MAX		$\left\{\begin{array}{l} AX + 1 \rightarrow AX: [\text{COMPARE}] \\ \text{if } S = 1: X \rightarrow Am, AX \rightarrow An \end{array}\right\}$
MINIMUM	MIN		$\left\{\begin{array}{l} AX + 1 \rightarrow AX: [\text{COMPARE}] \\ \text{if } S = 0: X \rightarrow Am, AX \rightarrow An \end{array}\right\}$
NEGATE	NEG		$- X \rightarrow A$

ABSOLUTE	ABS	ACC = 0
		ACC = 1
			$\lvert X\rvert \rightarrow A$
			$A + \lvert X\rvert \rightarrow A$

SIGN MULTIPLY	MSY	ACC = 0
		ACC = 1
			$\pm X * \text{sign}(Y) \rightarrow A$
			$A \pm X * \text{sign}(Y) \rightarrow A$

LOGICAL AND	AND		$X \cdot Am \rightarrow Am$; $0 \rightarrow An$ & AX
EXCLUSIVE OR	EOR		$X \oplus Am \rightarrow Am$; $0 \rightarrow An$ & AX
DIVIDE	DIV	FRAC INTG	$\left\{\begin{array}{c} AH\ (MSB) \rightarrow M \\ 2 * A \pm X + \overline{M} * 2^{-31} \rightarrow A \\ (16 \text{ times}) \end{array}\right\}$ Undefined

±: Except DIVIDE: + in PLUS mode, - in MINUS mode.
DIVIDE: + if M = 1, - if M = 0

Am: AH in FRAC mode, AL in INTG mode
An: AL in FRAC mode, AH in INTG mode

delayed multiplication operations, DMP, DMR, and DMA. These operations are not executed when specified in the program, rather they are triggered by the occurrence of another instruction in the code. Both DMP and DMR are triggered by the storing of the accumulator, while DMA is triggered when the accumulator is loaded. The utility of delayed operations is best demonstrated by example.

Consider the Schur product of two n-component vectors, $\boldsymbol{A}$ and $\boldsymbol{B}$, defined as follows:

$$P = \boldsymbol{A}o\boldsymbol{B} = (A_0B_0, A_1B_1, A_2B_2 \cdots A_{n-1}B_{n-1})$$

where

$$\boldsymbol{A} = (A_0, A_1, A_2 \cdots A_{n-1})$$
$$\boldsymbol{B} = (B_0, B_1, B_2 \cdots B_{n-1}).$$

Since each component product of $\boldsymbol{P}$ must be saved, using the normal multiplication instructions would be extremely slow since a two-cycle penalty would be incurred after each component calculation, thus requiring five cycles per component (including the storing of the accumulator). The delayed multiplications proved a convenient mechanism for realizing this vector product without interrupting

```
YPP   A0        ; Y <- A0
MPY   B0        ; X <- B0 and start Multiply
YPP   A1        ; Y<- A1
DMP   B1        ; X<- B1 and set delayed multiply
SAH   P0        ; save P0=A0B0 and start delayed multiply
YPP   A2        ; Y <- A2
DMP   B2
SAH   P1        ; save P1 and start P2 calculation
.
.
YPP   AN
DMP   BN
SAH   PN-1      ; save PN-1 and start final multiply
NOP
NOP
SAH   PN        ; done
```

Fig. 7. Calculation of $\boldsymbol{P} - \boldsymbol{A}o\boldsymbol{B}$ with delay multiplication operation.

the multiplier pipeline. Fig. 7 shows the code for the Schur product. Once again, it is shown inline for simplicity. The first two instructions initiate the pipeline and so a normal multiplication is used to calculate P_0. The next multiplication (for P_1) is set up while the first one is being executed, using the delayed multiplication operation, DMP. This operation is triggered with the storing of P_0 (SAH instruction) and completes during the next two cycles when the calculation for P_2 is being set up. Therefore, instructions three–five form the kernel of the calculation. Again, two NOP cycles are required at the end of the calculation to flush the pipeline, resulting in a total of $3n+2$ cycles for calculating the Schur product. This compares favorably to the $5n$ cycles that would have been necessary without the delayed multiplication feature.

The delayed multiply-accumulate, DMA, instruction is equally valuable in computing

$$Z_i = A_iB_i + W_i \qquad i = 0, 2, \cdots, n-1.$$

The program realizing this calculation is given in Fig. 8. Notice that each product, A_iB_i, is set up using a DMA instruction during the calculation of Z_{i-1}. Upon completion of the latter calculation, Z_{i-1} is stored away and the value of W_i is loaded into the accumulator. This triggers the next computation (Z_i) and the kernel repeats. A total of $4n+2$ cycles are required as compared to the $6n$ that would have been necessary without DMA.

Metrics such as $(X-Y)^2$ and $|X-Y|$ are easily calculated using the difference functions of the MAR. Basically, the YDIF instruction is a load Y register function, however its operation also affects the X register. When the X register is loaded following a YDIF operation, for multiply, absolute value (ABS) or other instruction, the MAR calculates the quantity $(Y-X)/2$ and loads it into both the X and Y registers prior to executing the operation that was specified when X was loaded. Hence, that instruction occurs with the modified values of the X and Y registers.

```
LAH    W0        ; ACC <- W0
YPP    A0        ; Y <- A0
MAC    B0        ; start Z0 calculation
YPP    A1        ; set up Z1
DMA    B1        ; calculation with delayed MAC
SAH    Z0        ; save Z0
LAH    W1        ; ACC <- W1 and trigger Z1 calculation


YPP    AN
DMA    BN
SAH    ZN-1
LAH    WN
NOP
NOP
SAH    ZN        ; done
```

Fig. 8. Calculation of $Z = A o B + W$ with delayed multiply-accumulate operation.

For example, the code

$\mathrm{YDIF}\,A$; $Y \leftarrow A$ and set difference mode
$\mathrm{MPY}\,B$; $X, Y \leftarrow (A - B)/2$ and start multiply
NOP
NOP

calculates $(A - B)^2/4$ and furthermore, this operation may be pipelined to calculate a sequence of such metrics, as would be necessary, for example, in a maximum likelihood detector for a modem. The purpose of scaling the operands is to avoid overflow.

IV. Dual MAR Structure

Two MAR's may be used simultaneously under the control of a single SPC in a manner which is particularily effective for complex valued calculations, such as those occurring in high speed QAM data modems. For reasons that will be apparent, one MAR is called the real MAR, or MAR.R, while the other is referred to as the imaginary MAR, or MAR.I. Since both MAR's can operate concurrently, the SPC specifies on each instruction cycle whether the function is intended for one of the two MAR's or for both. We will utilize instruction qualifiers .R, .I, and .P to indicate whether the function pertains to the real, imaginary, or both MAR's, respectively. In general, the real part of a complex operand is stored in the memory of MAR.R, while the imaginary part is stored at the corresponding location in the memory of MAR.I. Complex valued additions are now easily performed as follows.

LAH.P $MP(a)$;	Load accumulators with first operand
ADD.P $MP(b)$;	Load X regs with 2nd operand and ADD

In this example, the same functions are executed by each MAR. MP refers to the internal memories of the two MAR's. The real MAR uses its memory, (MR), and the

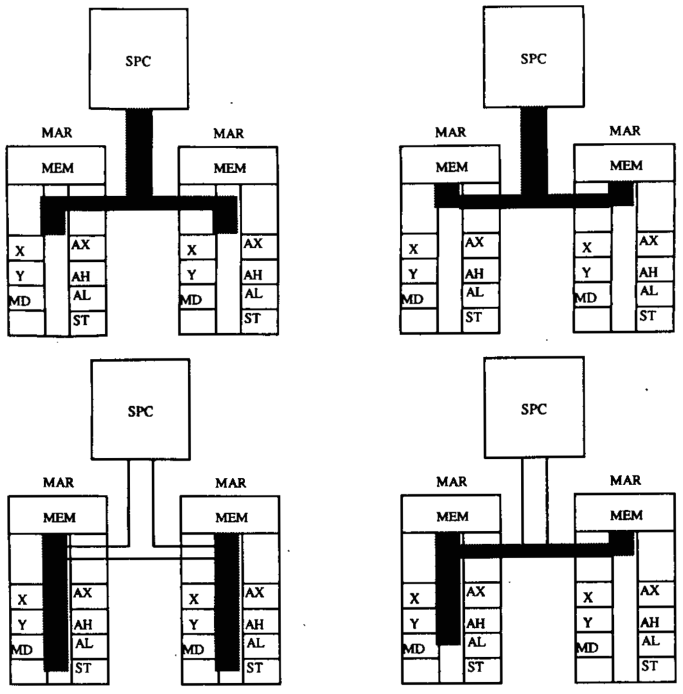


Fig. 9. MAR data movements.

imaginary MAR uses its internal RAM, (MI), (MP is merely a convenient notation for parallel memory). A single address field is supplied by the SPC to both MAR's, hence it is necessary to store the real and imaginary components of the operands at corresponding memory locations. The first instruction loads the real part of "a" into the real accumulator (referred to as AR) and the imaginary part into AI. The second instruction loads b_R into XR and b_I into XI and initiates an addition in each MAR. At the end of the addition, the accumulators AR and AI will, respectively, contain $(a_R + b_R)$ and $(a_I + b_I)$. The accumulators can then be stored into corresponding memory locations in MR and MI with a single instruction.

Complex multiplication is somewhat more complicated because it is necessary to exchange data between the two MAR's. The rich complement of data movements afforded by the control field of the MAR provides the needed utility to accomplish this task. Fig. 9 indicates some of the possible data movements that are available in a two-MAR structure being controlled by an SPC. There are actually more than 30 different data movement possibilities. Notice that inter-MAR communication as well as data exchange with the SPC is supported. In order to explain how to perform complex multiplication, it is also necessary to introduce another load Y instruction, which is useful for parallel operations. Remember that only a single function can go to the two MAR's on each cycle, e.g., both MAR's can be made to add. However, as with complex multiplication, it is often necessary to make one MAR add while the other one subtracts. In effect, both MAR's execute the same operation but with different signs. The YPM.P instruction is helpful here because, while it loads both YR and YI simultaneously, it tags one of them with a minus sign (the data are not changed however). Thus, executing

```
YPP.P   MR(a_r)        ; YR,YI <- a_r
MPY.P   MP(b_r,b_i)    ; AR <- a_r b_r   AI <- a_r b_i
YPM.P   MI(a_i)        ; YR,YI <- a_i except YR tagged negative
MAC.R   MI(b_i)        ; AR <- a_r b_r - a_i b_i
MAC.I   MR(b_r)        ; AI <- a_r b_i + a_i b_r
NOP                    ; results will be valid
NOP                    ; after last NOP
SAH.P   MP(result)     ; save in parallel
```

Legend:

- MR, MI refer to the internal memories of the real and imaginary MARs respectively while MP refers to each MAR accessing its own memory simultaneously

- MR(a) refers to the contents of MR at address 'a'

Fig. 10. Complex multiplication with two MAR's.

MAC will result in addition in one of the MAR's while a subtraction actually occurs in the other. Fig. 10 shows the code for the single complex multiplication

$$(a_R + ia_I)(b_R + ib_I) = (a_R b_R - a_I b_I) + i(a_R b_I + a_I b_R).$$

Notice how MAC is used together with YPM to compute the sums in the components of the complex product. Further note that the MAR's can use operands from each other's memories as well as they do out of their own because of the control field. Complex multiplication can be readily pipelined to realize an nth-order complex FIR filter in $5n+2$ cycles. Such filters are common in QAM modems [14] since the received signal is separated into its inphase (real) and quadrature (imaginary) components.

The MAR is a very powerful arithmetic engine and is mated with an equally powerful controller which we shall now describe.

V. The Signal Processing Controller (SPC) Circuit

The SPC, the second custom chip of the SPE, is a controller that executes a program written in its own powerful instruction set. Apart from the control of the two MAR's via the F, C, and A lines, the SPC is responsible for data structure maintainence, program control, peripheral control, and communication with the host processor. The architectural philiosophy employed in the design of the SPC is simple to state: Arithmetic capability is of relatively little value if the overhead tasks involved in data structure maintenance, program flow, etc., hamper the ability to keep the arithmetic engines busy. Hence, the SPC has a large part of its real estate dedicated to rapidly performing housekeeping chores in order to keep the MAR's busy. This philosophy sets the SPE apart from commercially available DSP chips and also accounts for the high efficiency of its architecture in actual modem applications (by efficiency, we mean the ratio of the achieved multiplication rate in an application versus the raw multiplication rate of the chips).

Fig. 11 provides a simplified block diagram of the SPC. It essentially has three external interfaces—the program memory, SPE, and the host interface. The program memory interface is a dedicated parallel path to the SPE program store. A program address (PA0-PA12) is output on each cycle and an instruction is retrieved from the program memory. The SPE interface provides for data and control information within the SPE. A 16-bit data bus (DB0-DB15) is the major data path to which the MAR's, external memory, and external devices are connected. Control of these devices is provided on each instruction cycle via the F, C, and A lines. The former two are the function and control lines of the MAR's and also carry information related to external memory and devices, while the A field provides the addresses for both internal and external memories. Two edge-triggered interrupts into the SPC allow for exception processing. The final interface is a rather unique host interface that provides for a low-overhead asynchronous communication path with the host processor. The host can send both data and instructions to the SPC without the need for any special hardware such as dual-port RAM's. This facilitates SPE program and data control from the host.

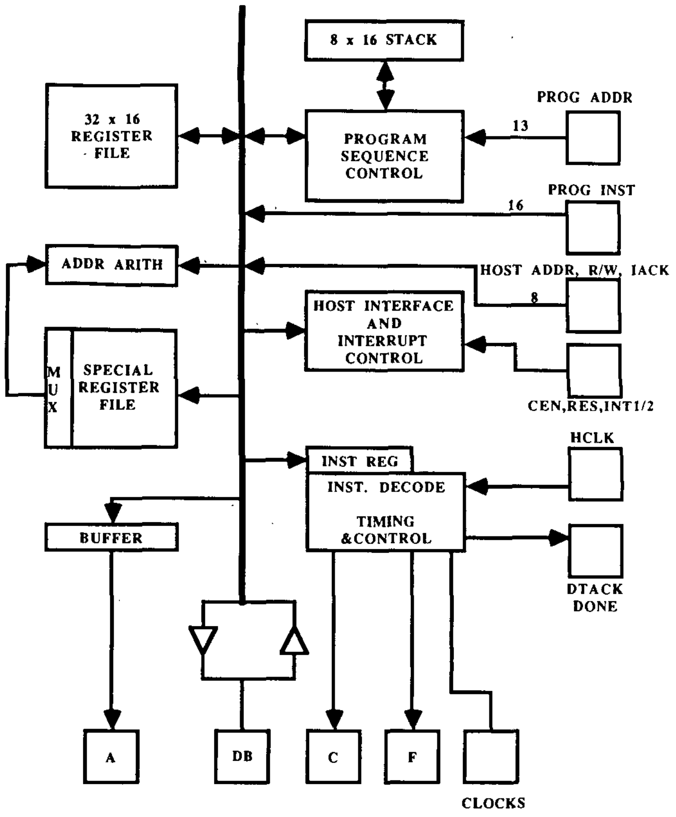


Fig. 11. SPC circuit.

It is difficult, in the limited space, to describe all the intricacies of the SPC instruction set or of its functional blocks. We will, therefore, highlight some of the more unique addressing modes and program flow constructs. The host interface will also be briefly described. These terse descriptions will serve to demonstrate the ability of the SPE to efficiently handle the aforementioned housekeeping chores thereby enhancing processor throughput through architectural innovation rather than circuit speed.

A. Program Flow Control

Program flow control is a major source of overhead in programmable machines. The SPC reduces this overhead through various program control structures available to the programmer (to be described). In addition, program instruction fetch, decode, and execution are pipelined for throughput enhancement. This works well for most DSP tasks where the code is inline, but conditional branching, however infrequent, disturbs the pipeline. The SPC branch instructions are delayed by one cycle, thus allowing the formation of condition codes that influence the branch. Any single cycle instruction may be inserted in the code during that cycle, otherwise an NOP is used and the cycle is wasted. The burden of managing the pipe resides with the programmer. Undelayed jump instructions are also available for unconditional branches and situations where future results do not affect the branch.

To demonstrate some of the SPC's program flow control capability, consider the overhead associated with looping structures and procedure calls. Loops are implemented in conventional Von-Neumann architectures by decrementing a loop counter and testing for zero at the end of every iteration of the loop body. These calculations often utilize the arithmetic facilities of the processor, thus preventing the occurence of any useful calculations. The time spent in maintaining the loop variable is thus overhead. Similarily, procedure calls and returns entail overhead in updating the program addresses. Unfortunately, these program control mechanisms (as well as others) cannot be dispensed with since they are invaluable in conserving code space and generally expedite program development. The SPC provides hardware support of these and other program control structures. Hence, flow control variables such as loop pointers and counters can be updated concurrently with useful calculations, thereby avoiding any speed degradation. Basically, the SPC trades hardware for throughput.

Signal processing algorithms are very computation intensive and generally consist of small kernels of code that are executed repetitively, i.e., the algorithms are best implemented as a set of nested loops that occupy most of the processor's time. The SPC provides a single-cycle LOOP instruction that efficiently implements a looping construct. At the beginning of a loop, the starting address is pushed onto the SPC stack. One of the register file (RF) registers (Fig. 11) serves as the loop counter and is initialized to the number of iterations (minus 1). Every time the LOOP instruction is encountered, the register is automatically decremented and tested and if the loop has not expired, a branch is generated to the loop start address stored on the stack.

The innermost loop of a nested set of loops is executed most frequently so any overhead in this loop generally has a substantial performance impact on the overall program. Recognizing this, a single-level full hardware looping facility for the inner loop was added to the SPC. Depicted in Fig. 12, it consists of two dedicated registers. One of the registers serves as the loop counter just as an RF register

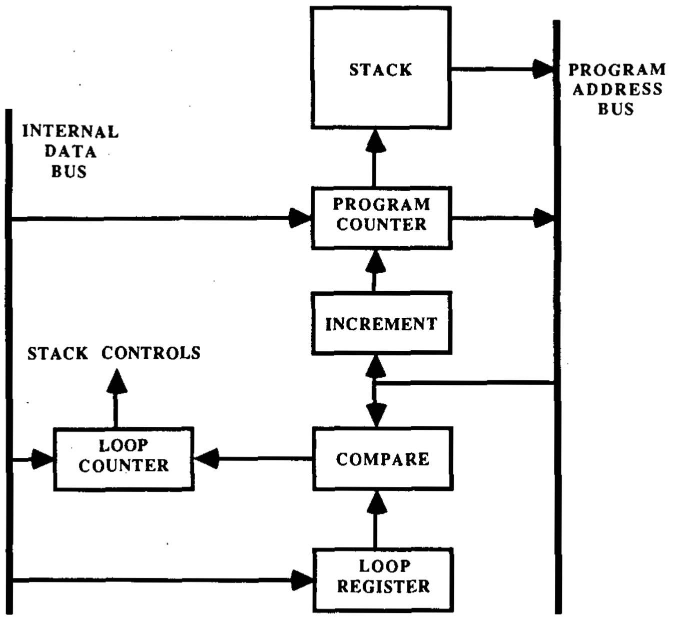


Fig. 12. DO LOOP hardware architecture.

did previously. To set up the loop, the second register shown in Fig. 12 is loaded with the address of the *last* instruction in the loop, and the start address of the loop is stored on the stack. The program counter is constantly compared to this latter register via a hardware comparator. When the last loop address is encountered, the loop counter is decremented and the program counter is appropriately adjusted, thus avoiding the need for an explicit LOOP instruction.

The SPC looping constructs incur some overhead to initialize the registers prior to commencing a loop, however they avoid the much greater overhead of updating and checking the register values at each iteration.

B. Data Structure Maintenance

We have identified program flow control as a major source of overhead. Data structure maintenance represents another large housekeeping task that detracts from processor throughput. The SPC is tasked with maintaining pointers to all the data structures that reside either in the internal memories of the MAR's, or in the SPE external memory. These pointers are updated according to some predetermined rule every time the data structure is accessed. As with program flow variables, updating pointers detracts from the time that would ordinarily be spent performing useful calculations unless the updates can be performed transparently. The SPC has special circuitry to provide for pointer modification concurrently with data structure access, thus avoiding much of the overhead. Furthermore, the SPC supports a rich set of update options that cover the majority of addressing mechanisms used in signal processing.

An increasing number of addressing modes have become common in modern computers owing primarily to the need for efficiently handling the many data structures common to file maintenance and operating system needs. Similarly, DSP algorithms are supported by data structures that demand efficient access and address maintenance that is made possible through efficient addressing modes. Generally, the prevalent modes in general-purpose computers, i.e., base addressing, indirect addressing with post- or pre-modification, immediate addressing, etc., are also common to DSP tasks (since some of the data structures are the same as occur in general-purpose computing). However, DSP algorithms also have some special needs that are not met by limited post-modification strategies found on general-purpose processors. For example, circular buffers are very common data structures in signal processing. Conventional addressing modes cannot efficiently support this structure without an explicit magnitude comparison and branch at each access. The SPC handles circular buffers very effectively with its modulo addressing mode.

A circular buffer is implemented as a contiguous block of memory locations. Data are entered into the buffer at sequential addresses until the end of the block is reached, at which point the pointer is reset to the buffer start address. A second pointer operating in a similar manner may be used to unload data from the buffer. The modulo addressing scheme implements a standard post-increment or decrement by one on each access, followed by a hardware comparison to the buffer-end address. If the end of buffer is reached, then the pointer is reset.

The update is simply stated as

$$\text{new\_address} = \left(\text{old\_address} - \text{base\_addr} \pm 1\right) \bmod M + \text{base\_addr}$$

where M is the buffer length and base_addr is the base address of the buffer. The buffer lengths must be stored in one of four available modulo registers that are part of the special register file (SRF) (Fig. 11). Hence, buffers of different lengths may be realized. The base address of the buffer is restricted to 2^k boundaries, such that $k > M$, in order to avoid the need for explicitly storing the base address, and to facilitate the address update. Once again, hardware has been traded for throughput by performing housekeeping chores concurrently with useful tasks.

Another example of an uncommon address update mechanism can be found in the need to perform decimation filtering. A contiguous block of memory locations once again serve as the data structure, however every Nth location is read on successive accesses instead of every consecutive one. The postincrement/decrement by N mode handles this case transparently. Several values of N may be stored in the offset register of the SRF.

C. SPE-Host Communications

As described earlier, delivering data between a terminal and the telephone line requires much processing which is segmented among the two processors shown in the systems modem architecture of Fig. 3. The host microprocessor maintains overall control, assigns specific tasks to the SPE, supplies data, and collects results.

The following attributes of the interface between the two processors are desirable in order to take maximum advantage of the dual-processor architecture: a) permit parallel processing, i.e., both processors should be able to perform their functions simultaneously with low overhead for interprocessor communication, b) flexibility with regard to the number of tasks that may be assigned and the number of data transfers that may be performed, c) accessibility of the signal processor data memories to the host processor for diagnostics, and d) low complexity and cost of interface circuitry.

Traditional techniques for interprocessor communication include the use of interrupts with data latches or first-in-first-out (FIFO) buffers and dual-port memories with semaphore registers. These techniques satisfy some of the desirable interface attributes listed above, but typically either the processor overhead, or the flexibility, or the complexity is unfavorable.

SPE-host communication is realized without the need for external FIFO buffers or dual-port memories. The host and SPE data buses are connected via a tristate-bus transceiver as shown in Fig. 3. This transceiver, which is mapped into the SPE I/O device address space, may be enabled by the SPC to effect a data transfer between the two processors in either direction. Apart from this data-bus connection, the SPC receives address and control signals from the host MPU (see Fig. 3) which allow the host to address the SPC at one of 64 addresses. Each address invokes the execution of a different host command by stealing an SPE instruction cycle at the end of the instruction currently in process. The SPE program execution is resumed upon completion of the host command.

Generally, host commands facilitate the manipulation of data structures for passing data and tasks between the two processors. Since the host exercises overall modem control, it maintains software-implemented state machines according to which, specific signal processing tasks are passed to the SPE to perform different algorithms. In effect, signal processing functions appear as high-level macros. Tasks and data from the host can be stored in input queues, and results of the tasks performed by the SPE can be collected in output queues. Host commands permit asynchronous access by the host to these circular queues, which reside in SPE data memories and are easily maintained using the address registers and addressing modes of the SPC. This extremely versatile interface, established using a minimum of external circuitry, allows both processors to be utilized in parallel to nearly their full capacity.

The 64-host command supported by the SPE can be divided into three groups. The first group enables data transfers between the host and MAR memories, or external memory, using one of eight SPC RF registers for indirect addressing with optional post incrementing. The second group of host commands permit the host to read or write one of eight registers in the SPC register file. The last group of host commands, such as test and set, read program counter (PC) and halt, write PC and execute (jump), read program memory and increment PC, single step, etc., have been included to aid in SPE program development, diagnostics, and monitoring.

VI. Conclusions

We have described two custom chips, the MAR and SPC, which have been designed to facilitate complex signal processing tasks prevalent in high-speed telephone-line modems. Our approach has been to divide the modem tasks between two loosely coupled processors: the SPE is devoted to the computationally intensive signal processing algorithms, while the host performs control tasks. A flexible host interface provides an efficient communications link between the processors.

The MAR architecture uses instruction pipelining and a powerful data movement structure to overcome some of the inefficiencies of a single-bus architecture. Several special instructions, e.g., delayed multiplication, are key to maintaining a full pipeline during several common signal processing tasks. Throughput is enhanced by providing powerful instructions such as MAX that perform operations traditionally requiring several instructions. Furthermore, two MAR's may be conveniently connected in parallel to perform complex arithmetic.

The SPC complements the MAR arithmetic capability with a powerful instruction set that facilitates the implementation of data and program control structures. Together, the SPC and MAR exploit several architectural innovations that improve the throughput of the signal processor. These features are essential in a programmable environment since instruction and control overhead often limit the utility of even the fastest arithmetic units. The SPE averages 2 million delivered operations (e.g., multiplication) per second for modem algorithms, which compares favorably to the raw rate of 3.33 million/s possible with the arithmetic unit. Our conclusion, therefore, is that signal processors benefit from more architectural features than simply a fast multiplication facility. Furthermore, the SPE can be substantially enhanced with a single-cycle multiplier supported by two operand buses afforded by modern process technology.

Acknowledgment

The new Codex premium end-user modem product line, of which these chips are a part, is the result of the vision and efforts of many people both technical and managerial —we are merely the scribes. This paper is dedicated to the memory of our colleague, G. Chamberlin.

References

[1] K. Murano, Y. Mochida, F. Amano, and T. Kinoshita, "Multiprocessor architecture for voiceband data processing (application to 9600 bps modem)," in *Proc. IEEE Int. Conf. Commun.*, June 1979, pp. 37.3.1–37.3.5.

[2] T. Tsuda, Y. Mochida, K. Murano, S. Unagami, H. Gambe, T. Ikezawa, H. Kikuchi, and S. Fujii, "A high-performance LSI digital signal processor for communication," in *Proc. 1983 IEEE Int. Conf. Commun.*, June 1983, pp. A5.6.1–A5.6.5.

[3] G. Ungerboeck, D. Maiwald, H. P. Kaiser, P. R. Chevillat, and J. P. Beraud, "Architecture of a digital signal processor," *IBM. J. Res. Develop.*, vol. 29, no. 2, pp. 132–139, Mar. 1985.

[4] R. W. Lucky, J. Salz, and E. J. Weldon, Jr., *Principles of Data Communication.* New York: McGraw-Hill, 1968.

[5] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 56–67, Jan. 1982.

[6] G. D. Forney, Jr., R. G. Gallager, G. R. Lang, F. M. Longstaff, and S. U. Qureshi, "Efficient modulation for band-limited channels," *IEEE J. Select. Areas Commun.*, vol. SAC-2, pp. 632–647, Sept. 1984.

[7] W. E. Nicholson, R. W. Blasco, and K. R. Reddy, "The S2811 signal processing peripheral," in *Proc. WESCON*, 1978, pp. 1–12.

[8] T. Nishitani, R. Maruta, Y. Kawakami, and H. Goto, "A single-chip digital signal processor for telecommunications applications," *IEEE J. Solid-State Circuits*, vol. SC-16, pp. 372–376, 1981.

[9] E. R. Caudel and R. K. Hester, "A chip set for audio frequency digital signal processing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, 1982, pp. 1065–1068.

[10] S. S. Magar, E. R. Caudel, and A. W. Leigh, "A microcomputer with digital signal processing capability," *IEEE Int. Solid-State Circuits Conf., Digest Tech. Papers*, Feb. 1982, pp. 32–33 and 284–285.

[11] M. Kikuchi, T. Inaba, Y. Kubono, H. Hambe, and T. Ikesawa, "A 23 K gate CMOS DSP with 100 ns multiplication," in *IEEE Int. Solid-State Circuits Conf., Digest Tech. Papers*, Feb. 1983, pp. 128–129.

[12] R. E. Owen, "VLSI architectures for digital signal processing," *VLSI Design*, vol. 5, pp. 20–28, 1984.

[13] A. D. Booth, "A signal binary multiplication technique," *Quart. J. Mech. Appl. Math., Part 2*, vol. 4, pp. 236–240, 1951.

[14] S. U. H. Qureshi, "Adaptive equalization," *Proc. IEEE*, vol. 73, no. 9, pp. 1349–1387, Sept. 1985.

Shahid U. H. Qureshi (M'73–SM'81) was born in Peshawar, Pakistan, on September 22, 1945. He received the B.Sc. degree from the University of Engineering and Technology, Lahore, Pakistan, in 1967, the M.Sc. degree from the University of Alberta, Edmonton, Canada, in 1970, and the Ph.D. degree from the University of Toronto, Toronto, Ont., Canada, in 1973, all in electrical engineering.

In 1967 and 1968, he was a Lecturer with the Department of Electrical Engineering, University of Engineering and Technology, Lahore, Pakistan. He held the Canadian Commonwealth Scholarship from 1968 to 1972. Since 1973, he has been with Codex Corporation, Mansfield, MA, where he is currently Senior Director of Research in Transmission Products. His interests include data communication, computer architecture for signal processing, and the application of digital signal processing to communication. In 1984, he was named a Motorola Dan Noble Fellow in recognition of leadership in modem research and development.

Hassan M. Ahmed (M'82) received the B.E. degree in electrical engineering and the M.E. degree in aeronautical engineering from Carleton University, Ottawa, Canada, in 1977 and 1978, respectively, and the Ph.D. degree from Stanford University, Stanford, CA, in 1982.

During 1977–1978, he was a project engineer with Miller Communications Systems Ltd., Kanata, Ontario, where he was responsible for the design of multiple microprocessor-based digital signal processors for communications. During 1979–1982 he consulted for Codex Corporation, Mansfield, MA, and later joined them as Director of VLSI Systems. At Codex, he was responsible for the design of custom VLSI chips for voiceband signal processing. Since January 1985, he has been with Analog Devices Incorporated where he is the Engineering Manager of the Digital Signal Processing Divison, a semiconductor division that designs VLSI chips for the DSP markets. His research interests include digital signal processing, computer architecture, VLSI, and unsteady fluid mechanics.

Dr. Ahmed is a member of Sigma Xi.