Personal Instrument (PI)—A PC-based signal processing system

by G. Shichman

The Personal Computer (PC) technology has seen an enormous growth in the last two years. Although increasingly viewed as a major productivity tool, the PC is likely to be limited for computation-intensive tasks such as telecommunications and improved human-factors I/O. At the same time, there has been another evolving technology—VLSI realization of general-purpose signal processor (SP) engines which are capable of boosting the performance levels of standard PCs by almost two orders of magnitude. With SPs in PCs, we now see tremendous opportunities for distributing computation-intensive tasks away from high-performance mainframe computers; previously formidable tasks such as speech coding and recognition, pattern and scene analysis, spectral analysis, high bit-rate communication, and the like are now all computable by utilizing a single VLSI module embedded in any standard personal computer. This combination of a general-purpose CPU and superfast real-time coprocessor is likely to be key to the future functions and success of advanced workstations. A signal processing subsystem with real-time data acquisition and control capabilities has been developed for the IBM PC at the IBM Thomas J. Watson Research Laboratory and is the topic of this paper.

Introduction

In the high-performance office operating environment, personal computers deliver a high degree of functional capability accessed through an easily understood user interface. Typically, office productivity tasks require up to 1.0 MIPS (millions of instructions per second) of computational power [1] (not including advanced I/O or telephony functions). In the engineering/clinical laboratory and high-speed communication environments, however, there is clearly a need for higher-MIPS-rate computers as well as more powerful I/O handling than that available through conventional PC architectures.

The traditional solution to these issues was a combination of a computer (a minicomputer or a PC) as a controller and an instrument as the front-end processor. As such, computers and instruments had a well-defined relationship; they were connected by the standard serial or bus communication links—the RS-232C or IEEE-488 interface bus, respectively.

It is important to view this relation between instruments and computers by examining their interfaces. The traditional link is the RS-232C **[Figure 1(a)]**, which connects the computer and the instrument over a distance of 50 to 100 feet (enhanced with a modem, connections can be made through the switched telephone network). The IEEE-488 bus, on the other hand, connects as many as 14 individually

addressable instruments to one host controller port **[Figure 1(b)]**. With the aid of bus-extenders, the IEEE-488 bus 60-foot coverage can be extended to the range of the serial RS-232C link.

The Personal Instrument (PI) is different in that both computing power and measurement functions share the same enclosure **[Figure 1(c)]**. (The PC expansion unit falls into this category since it is a direct extension of the PC I/O channel and has the full PC bus bandwidth and control capabilities.) Such a configuration enables use of the full bus bandwidth of the PC as well as allowing, for the first time, real-time control of measurements and processes. The PI, which links different functions over a microcomputer bus, can easily handle real-time interaction, but a system based on the IEEE-488 bus cannot, because the instruments communicate not with each other but with a system controller.

The PC as PI is finding a home in the engineering, design (CAD), and real-time process control environments. A growing number of users have discovered the tremendous flexibility of the PC in engineering applications, provided for the most part by the increasing number of manufacturers producing plug-in boards, add-on modules, and software packages for PCs to turn them into a wide variety of engineering tools. Test and measurement equipment, data acquisition systems, development systems, and design workstations built around the PC are all now available.

The growth and acceptance of PIs may be attributed to three factors:

- Continuing rapid improvement of the price/performance ratio for PCs.
- Industry-levered acceptance and current use of industry standard software.
- Availability of high-grade I/O hardware for PC enhancement.

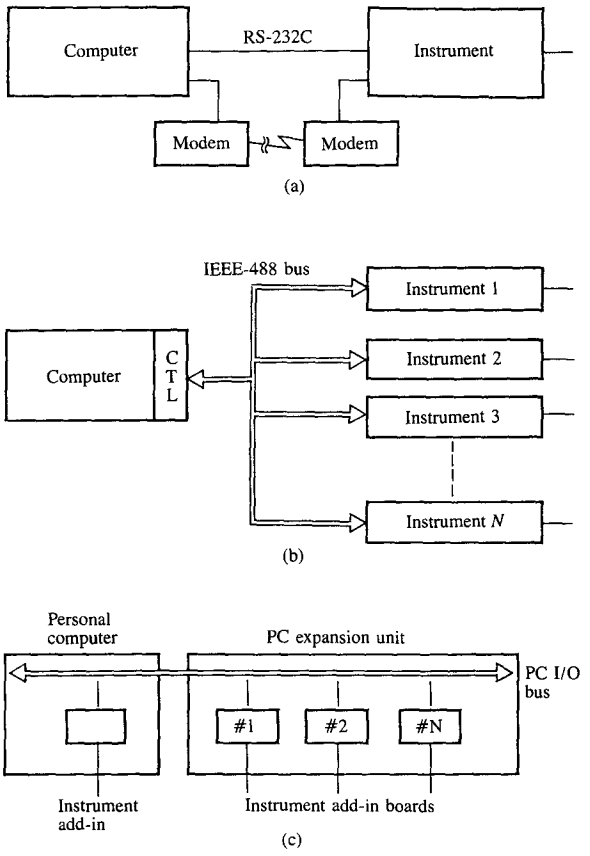

Figure 1

Computer/instrument interface configurations: (a) RS-232C serial link interface; (b) IEEE-488 communication bus interface; (c) Personal Instrumentation interface.

Real-time processing is, naturally, the promise of the PI. However, most PC operating systems are not designed to coexist with real-time events. The most important aspect of real-time processing is the real-time response, defined as the time from the sensing of an external event to the moment the real-time process begins. With programs loaded into memory and waiting to respond to an event, real-time response depends only on interrupt latency within the operating system and context-switch time, or the time it takes the operating system to stop one process and begin another. This process is usually time-consuming due to heavy save/restore requirements during the switch. However, operating systems must frequently disable the interrupt system while internal data structures are being modified. Thus, if a real-time device needs processor attention from such a system a few thousands of times per second, the system spends nearly all of its time switching to the real-time process. For high-data-rate tasks, real-time processes must thus be run on a stand-alone processor whose architecture lends itself to efficient interrupt-driven I/O handling.

The major bottleneck in utilizing the PC as PI or as event-driven workstation is its limited computation power and I/O bus bandwidth. For example, acquiring 20K samples per second of speech signal using a plug-in data acquisition board and direct transfer of pulse-coded modulation (PCM) data to storage media (e.g., hard file) can easily block the PC from doing any other task. Moreover, using a standard PC operating system (DOS, CP/M, etc.) disk access method [2], it takes *four minutes* to fill a 10-megabyte hard file. In addition, typical data compression to be performed "on the fly" requires almost two orders of magnitude more computation power than that available from the PC CPU engine [3].

The natural solution to these problems is to augment the PC CPU with a real-time coprocessor which off-loads, from

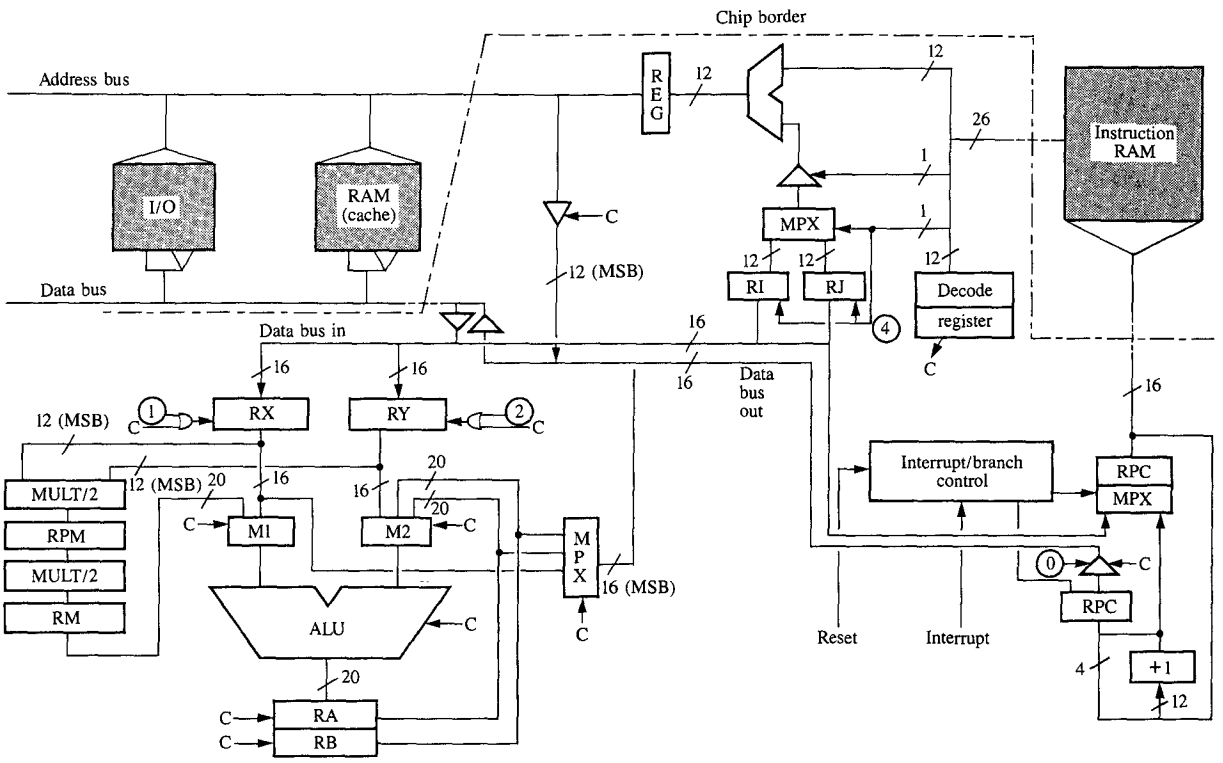

Figure 2

IBM SP block diagram.

the PC, the computation-intensive processing. However, doubling or tripling the PC CPU power is often not nearly enough. There is clearly a need for an engine that can process data (typically using integer arithmetic) in rates of 5 to 20 MIPS and at the same time be able to access a variety of I/O devices in real time [4].

In the sections that follow we discuss the motivation and detailed architectural aspects of a PC-based co-signal-processing subsystem aimed at providing the PC with the capabilities mentioned above, namely 10 MIPS of 16-bit arithmetic and real-time I/O handling.

IBM SP—Architectural overview

A detailed description of the IBM SP is given elsewhere in this issue [5]; below, we briefly discuss some of its basic concepts and performance.

The IBM SP is a high-performance digital signal processor with an architecture optimized for signal processing and coding/decoding applications [6]. It is a realization of the SP12/16 signal processor designed in the IBM Zurich Research Laboratory [7]. Because a high degree of parallelism is necessary to obtain the required processing throughput, separate instruction and data storage have been incorporated to permit pipelined instruction fetching and execution **(Figure 2).** A fast hardwired multiplier is used in addition to the standard ALU. With such a parallel approach, multiplication operations are not the limiting factor in processor performance. Rather, performance is limited by the data transfer capabilities of the processor (i.e., bus bandwidth to/from storage). For simplicity, a single data bus structure is utilized to connect processor registers and accumulators, storage, and I/O devices. In order to maximize throughput over this bus, data transfers and ALU/multiplier operations are performed independently of one another.

The SP instruction format supports two independent operation codes: The first, transfer code, specifies, in conjunction with the index enable index select and the operand, the data transfer over the data bus; the second, computation code, defines the ALU operation performed on the contents of two previously loaded registers. The two codes are used in a combined fashion for conditional branching, whereby the computation code establishes the branch condition [5].

For the greatest efficiency in multiply/accumulate signal processing applications, it is imperative that the multiplier be

able to accept a new operand or operand pair every instruction cycle. This is achieved by using a parallel multiplier. For further simplicity, the SP accepts a single interrupt and employs a memory-mapped I/O scheme. The PI SP currently operates at 100 ns per cycle, or 10 MIPS. Because two independent operations and two memory fetches are executed during one machine cycle (ALU and multiply, data or I/O access and instruction prefetch), the effective machine throughput is 20 million operations per second.

PI

• *Application environments*

A Personal Instrument (PI), or PC-based co-signal-processor, can be particularly useful in applications which exhibit some of the following characteristics:

- Repetitive arithmetic processing.
- Many input data elements.
- Computation-intensive processing.
- Minimal real-time response latency.

Application areas that may be addressed with the PI include the following:

1. Telecommunication
   - High-speed modems (synchronous, full-duplex, high-line coverage, etc.).
   - Adaptive equalizers.
   - Data compression (e.g., digital PBX interface).
   - Synthesis, emulation, and analysis of communication signals.
2. Signal processing
   - Digital filtering.
   - Adaptive filtering, adaptive noise canceling.
   - Fast Fourier, Hilbert, and other computation-intensive transforms.
   - Spectral analysis.
   - Waveform generation for signal analysis.
   - Real-time seismic analysis.
3. Speech processing
   - Speech recognition.
   - Speech coding (vocoders).
   - Speech compression, speech store-forward.
   - Speech synthesis (text to speech).
4. Laboratory instrumentation
   - Spectrum analysis.
   - Phase lock loops and averaging.
   - Signal capture and transient analysis.
   - Arbitrary waveform generation and recording.
   - Digital scopes and multimeters.
   - Automated IC/board testing (ATE).
   - Gas and liquid chromatography (GC, LC).
     a. Continuous flow colorimetry.
     b. Spectroscopy.
     c. Thermal analysis.
5. Control and industrial applications
   - Vibration and structural analysis.
   - Digital servo control.
   - Robot-arm joint sense and control.
   - Process control and data logging.
   - Power transient monitoring.
   - Machine vision.
   - Energy management and control.
6. Image processing
   - Pattern recognition.
   - Handwriting and printed character recognition.
   - Calculation, reconstruction, and enhancement of images.
   - Image interpolation (zooming, rotation, etc.).
   - Image compression.
   - Signature and fingerprint analysis.
   - QC inspection, noncontact measurements.
   - Thresholding and halftoning of images.
7. Biomedical clinical/diagnostics applications
   - EKG monitoring and analysis.
   - Sonography and echo-cardiography.
   - EEG data acquisition and analysis.
   - Evoked potentials analysis.
   - Digital radiography.
   - Ultrasound imaging.
   - Microwave hyperthermia temperature distribution measurements.
   - Ambulatory blood pressure monitoring.
   - EMG data recording and analysis.
   - Biofeedback systems.

• *Architectural overview*

The PI, based on an IBM SP as a co-signal-processor to the PC CPU, originated from research aimed at developing speech signal vector quantizers for the Yorktown Speech Recognition System [8]. A collaboration with the Processor/Communications group of the IBM Zurich Research Laboratory [9] resulted in an early architectural definition of the PI which evolved during 1983 to its current state.

The PI architecture was designed to upgrade performance of the IBM PC in terms of computational power and to provide it with a flexible interface to the "real" world, i.e., real-time signals, processes, and events. With the inclusion of the IBM SP, the PC can be viewed as an instrumentation controller with the power of a general-purpose array processor for real-time digital signal processing applications. Without its I/O features, the PI can be used as a general-purpose array processor tightly coupled to the PC I/O channel.

The IBM SP executes each instruction in 100 ns, providing the PI subsystem user with 10 MIPS of 16-bit

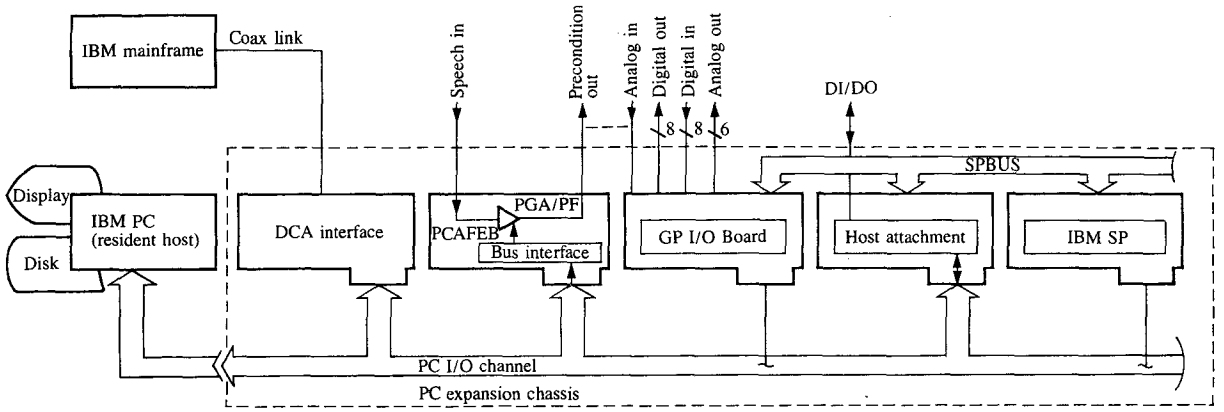

Figure 3

PI—overall functional block diagram.

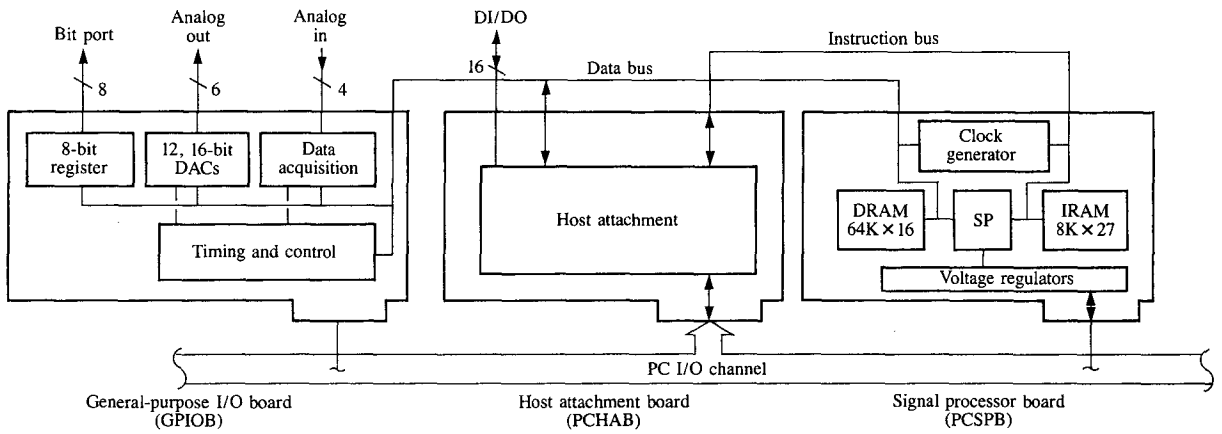

Figure 4

PI-CO Processor functional block diagram.

processing—about 100 times the processing power available from the native PC 8088 microprocessor. In addition to its sheer computational power, the PI provides an extensive array of I/O capabilities with minimal SP and no PC overhead. This is achieved by an interrupt-driven I/O mechanism and pipelining of the "slow" and asynchronous "real-world" events from the "fast" SP execution cycles.

The PI block diagram is given in **Figure 3.** The subsystem consists of three PC boards and two optional cards for signal preconditioning and host communication [10]. The system comprises (1) a signal processor board (SPB-64), (2) a PC host attachment board (PCHAB), and (3) a general-purpose I/O board (GPIOB). These three boards are connected via a signal processor bus (SPBUS) on which all data and program transactions take place **(Figure 4).** PC I/O channel transactions, on the other hand, take place only during PI IPLing and data memory transfers (e.g., storing SP processing results, SP cache swapping, etc.).

The PC and the PI can interrupt each other for attention calling and direct memory access (DMA) initialization; however, all I/O is handled by the GPIOB interrupt mechanism, which is totally transparent to that of the PC.

The PI data memory can be accessed via the PC DMA channel, thus significantly increasing the data transfer throughput between the two processors. In fact, the DMA mechanism enables the PI to share large amounts of data memory with the "slow" PC memory and thus alleviates the need for a large and expensive on-board SP "fast" memory.

• *SP board (SPB-64)*

The SPB-64 contains the IBM SP module, finite-state machine clock generator, SP data memory (64K-word DRAM), SP instruction memory (8K-word IRAM), and power regulation circuitry **(Figure 5).** Since the SP native data memory addressability is 4K words, a page register (DPREG) is augmented to the SP data address bus. Thus, accessing any DRAM page (4K words each) beyond the base page (e.g., page 0) involves a single processor cycle overhead (i.e., loading DPREG with the appropriate page information). This overhead has been proved to be negligible if careful data address handling is exercised.

• *PC host attachment board (PCHAB)*

The PC host attachment board (PCHAB) provides, in essence, a direct interface between the PC I/O channel (PCBUS) and the SP bus (SPBUS). Since the latter has a high bandwidth (160 megabits/s) compared with the PC bus bandwidth of 8 megabits/s, communication is carried out in a byte-serial fashion in which the SP subsystem is seen by the PC CPU as 16 contiguous I/O locations. Thus, all interactive PC/SP commands are I/O read/write operations.

As a resident host, the PC can set the SP mode of operation (RESET, RUN, STOP, etc.) and at the same time monitor its status. This control is extremely useful in a runtime environment where application programs are being developed and debugged [11].

In order to minimize the overhead involved in data transfers between the two machines, the PCHAB contains the host attachment circuitry—a set of registers and drivers **(Figure 6)** which are loaded sequentially by the PC and activated simultaneously by a cycle steal sequencer (CSS). The CSS accepts commands from the PC and initiates the appropriate sequence of control signals to stop the SP, release its buses, and provide direct access to the DRAM, IRAM, and all its I/O devices. At most, the CSS "steals" five SP cycles in order to complete an access to a DRAM location or to an I/O device.

In addition to the normal PC-SP transfers (i.e., the PC CPU is transferring data under program control), a PC DMA channel can be activated by the SP to initiate an SP DRAM cache swapping. Once a DMA transfer is initiated, the PC CPU is disabled and data are moved between the PC memory and the SP DRAM (there are no provisions for PC-to-IRAM DMA transfers). The resulting DMA transfer throughput is 360 kilobytes/s, about twice that of normal PC program-controlled transfers. **Figure 7** depicts the SP DRAM as a fast SP cache connected to the PC memory via a combination of the PC DMA and the SP CSS control mechanism. A typical "cache update" time in the PI is 8 ms per 2 kilobytes, at which time the SP is running at an effective rate of 9 MIPS, a negligible performance degradation in most applications.

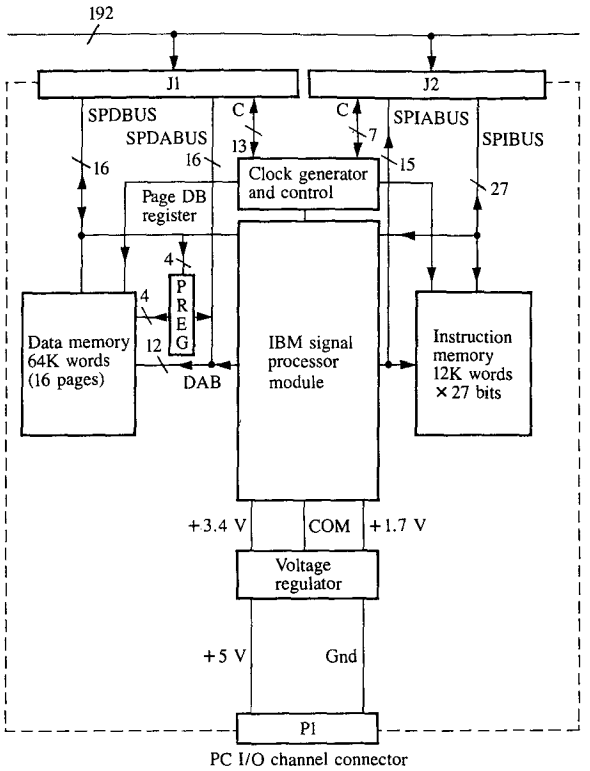

Figure 5

Signal processor board (SPB-64).

• *General-purpose I/O board (GPIOB)*

The IBM SP is particularly suitable for efficient I/O handling. Since its I/O devices are memory mapped, there is no distinction between memory and I/O accesses. However, the I/O interface is not allowed to degrade the SP throughput—hence the need for a front-end I/O handler tightly coupled to the SP data bus. Our solution to the strict I/O handling requirements is the subsystem on a general-purpose I/O board (GPIOB), which frees the SP from polling I/O devices yet permits it a fast context switch time for real-time I/O response.

The GPIOB functional block diagram is given in **Figure 8.** Control of the different I/O devices on board is accomplished via a data address bus strobe decoder. This control mechanism enables individual access to each I/O device via a single SP store/load operation. Process

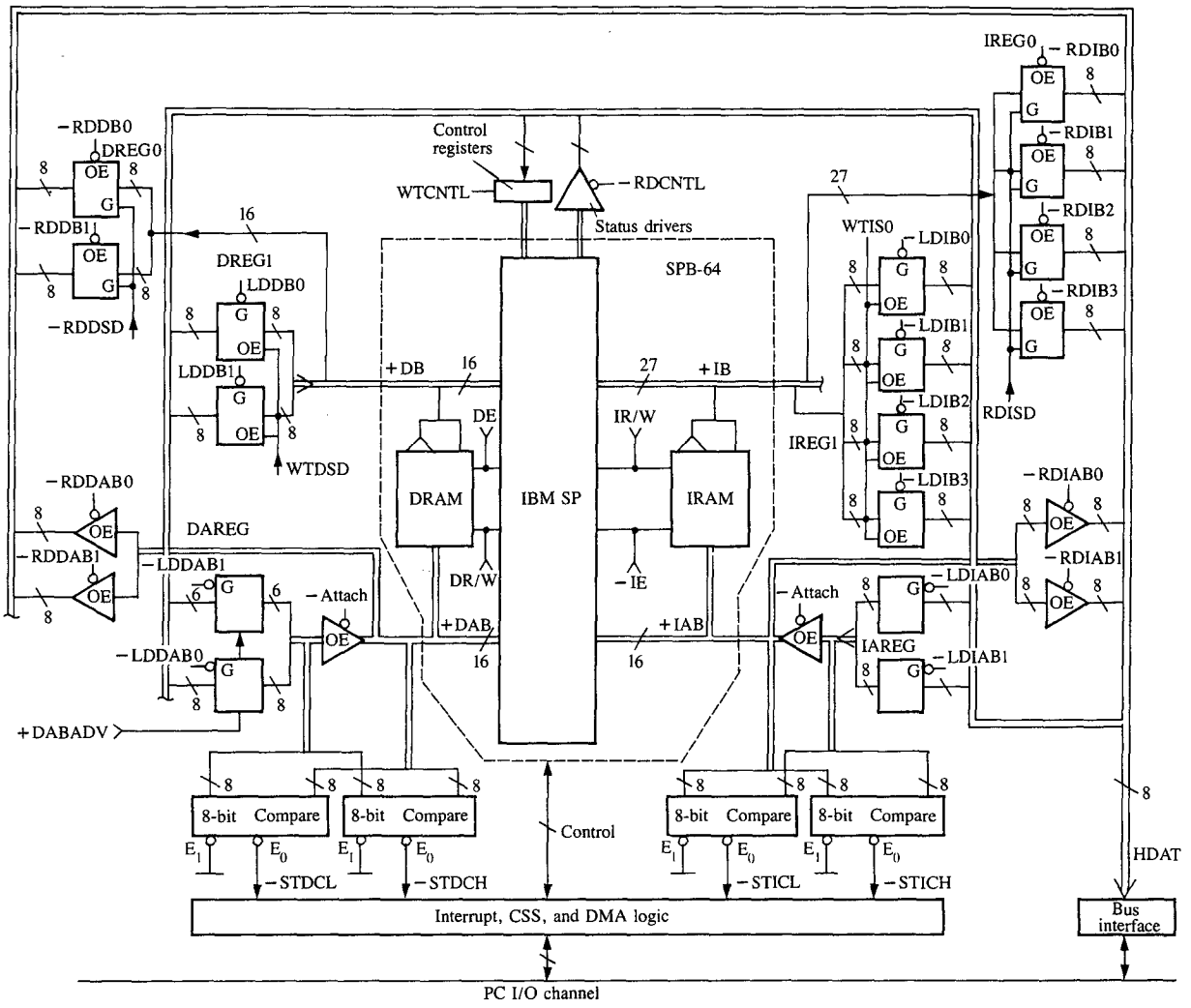

Figure 6

PI-PC host attachment board block diagram.

synchronization and real time clock (RTC) generation is accomplished by a set of six SP-programmable timing elements packaged in two timer modules [12]. With the RTC timing mechanism, sampling can be done in both time and frequency domains; each sampling event can be scheduled by the SP at will or preprogrammed for repetitive operation (time-domain sampling is crucial for the implementation of high-speed modems, whereas frequency-domain sampling is common in speech processing and control applications). Two undedicated timer signals are available to the user for process synchronization (e.g., 12-bit DAC sampling rate generation, single-bit I/O port interrupts, etc.); the remaining four timing signals are hardwired as sampling rate generator and as timing elements for the analog to digital converter (ADC) and the 16-bit digital to analog converter (DAC) subsystem.

The ADC and the 16-bit DAC subsystems are pipelined through 512-word pipeline FIFOs, whereby the 12-bit DAC subsystem is buffered by a single pipeline register. The "deep" pipelining is particularly useful in segmented data acquisition tasks where signals are being digitized into the ADC FIFO without any SP intervention. Once a "frame" or a segment has been acquired, the SP is interrupted once, the FIFO data are moved to the appropriate area in the SP cache space, and the required processing of the data segment is started.

The following is a list of I/O devices supported by the GPIOB:

1. A four-channel, 12-bit ADC buffered by a 512-word-deep pipeline FIFO (synchronous port).
2. A four-channel, 12-bit DAC buffered by a single pipeline register (all four channels are independently controlled and synchronous).
3. Two 16-bit DACs buffered by a 512-word-deep pipeline FIFO. The two converters are independent, individually controlled, and synchronous.
4. Eight single-bit input ports (asynchronous).
5. Eight single-bit output ports (asynchronous).
6. Six programmable timers for real time clock generation and I/O control.
7. A seven-way interrupt controller.

The analog input data acquisition subsystem has the following characteristics:

- 12-bit resolution with 5-ns aperture uncertainty across all channels.
- 100 000 samples per second for single-channel acquisition.
- 512-word-deep pipeline FIFO.

The main DAC subsystem has the following characteristics:

- Two channels, 16-bit resolution.
- 125 000 conversions per second.
- 512-word-deep pipeline FIFO.

The auxiliary DAC subsystem has the following characteristics:

- Four channels, 12-bit resolution.
- Single-level pipeline buffering (i.e., all channels can be loaded individually and sampled simultaneously via RTC control).
- 125 000 conversions per second.
- A Z-axis control signal provided during data conversion to facilitate X-Y display trace control.

As depicted in the block diagram of Fig. 8, there is minimal SP overhead involved in any I/O operation. Once the RTCs are programmed and the interrupt system enabled, the GPIOB acts as a separate I/O processor tightly linked to the SP via its interrupt mechanism. A typical interrupt latency time in the PI is 10 SP cycles, or one microsecond. For a process sampled at the highest sampling rate (i.e., 100K samples per second), this latency is merely 10% of that sampling interval.

The PI is currently used as the front-end processor for the Yorktown Automatic Speech Recognition System, and as such needs only a preconditioning unit for microphone signal interface prior to its acquisition by the GPIOB. Other signals can be easily handled by device-dependent preconditioning units and sensors.

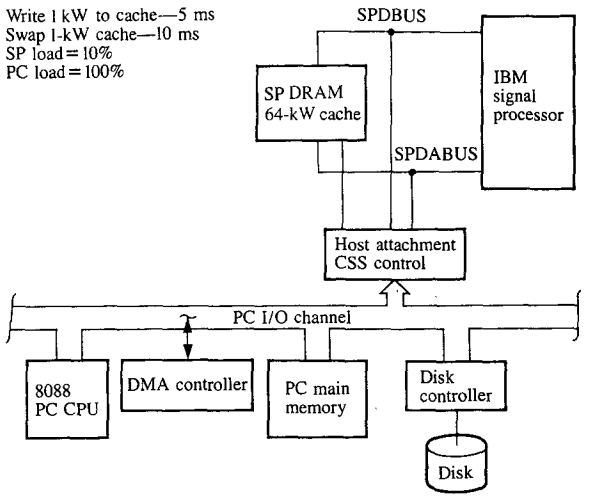

Figure 7

PC—SP data block swapping (cache update).

- *Main SP memory—cache enhancement*

In several PI applications, a larger data store is a critical requirement for sustaining the throughput of the signal processor. Typical examples are speech and pattern recognition, where a large statistical (or template) data base must be resident in the SP data store [13] and must be accessed by the SP during its normal execution cycles. Unfortunately, running the SP at a 100-ns cycle time requires the use of fast memory devices having at least 50 ns access time. Static memories, satisfying this speed requirement, are not as dense as their dynamic counterparts due to the excessive heat dissipated by their storage cells. As a consequence, the packing ratio of static memories does not allow the incorporation of large amounts of storage on the SP board (the 64K words currently implemented on the PI SPB-64 is about the maximum memory size attainable in today's technology).

Since SP application programs typically operate on vectors (or arrays of data), there is no need for the SP to randomly access data in all the available memory (note that the IBM SP instruction store is separated from the data store and that the argument above is not applicable to its implementation). Rather, large vectors (arrays) of data residing in a main SP memory (MSPM) can be accessed by the SP sequentially via a memory control unit (MCU) which facilitates automatic prefetching of the next location of data when the SP is latching the data from the currently accessed location. The SP must merely set an initial vector address to be accessed and set the MCU control register for the required transfer [14].

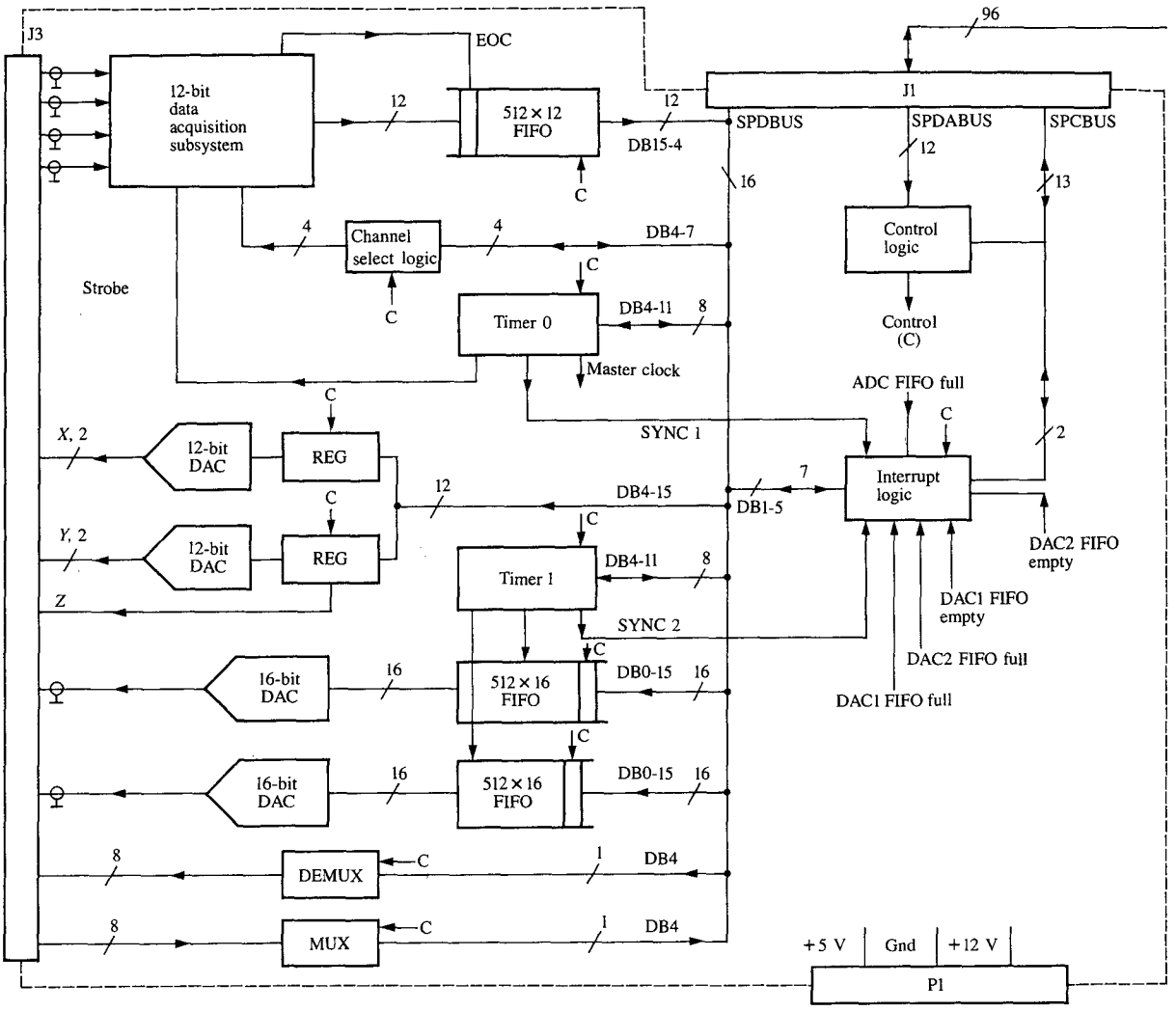

Figure 8

General-purpose I/O board—block diagram.

The MSPM throughput is attributed to the pseudo-static memory technology used in its static column CHMOS dynamic memory devices. The unique feature of these devices is the ability to randomly access each one of its column bits with an access time comparable to that of a fast static memory [15].

The PI MSPM is organized as one megaword for full 16-bit transfers, and two megabytes for single-byte transfers **(Figure 9)**. It provides two modes of operation: random and sequential. In the random mode, the SP page register (DPREG) is set to the higher page (i.e., page 15) and any SP access to memory is routed to the MSPM. There, an 8-bit programmable page register allows selection of 256 4K-word pages for a total of one megaword. The access time in this mode is three SP cycles, where two wait states are transparently inserted into a memory access SP instruction. In the sequential mode, on the other hand, a programmable 20-bit counter is used to sequentially access the memory via an SP memory-mapped I/O address. The prefetch mechanism allows the MSPM to "keep up" with the SP cycle time and no performance degradation is experienced [16].

Last, in contrast to static memories, dynamic memories require a refresh cycle approximately every 15 microseconds; thus, a three-cycle wait condition is occasionally experienced in both access modes if a refresh cycle coincides with an SP transfer request. Due to the refresh, the PI throughput is expected to degrade by no more than 2% even during

continuous MSPM accesses. During SP cache accesses, however, the PI throughput is independent of the refresh cycles, which are executed concurrently in the MSPM.

• *Software and programming considerations*

The parallel nature of the IBM SP requires special attention to efficient programming and to throughput maximization via pipeline utilization. For this purpose we developed two programming environments for the SP: compiletime and runtime. In compiletime, the IBM SP program and data assemblers (PC-based) are supported along with a host (System/370)-based simulator [17]. In the runtime environment we identify two modes of operation: stand-alone and host-supported. In the former, SP programs and data are assembled by the PC-resident SP assemblers; in the latter, program and data preparation is done on the host mainframe.

The runtime environment enables the user to manipulate the running condition of the SP and employs the PC screen and keyboard as the primary "operator's panel" of the signal processor. Operating within the runtime environment, SP programs or data can be written and altered in the instruction or data memories, respectively. SP instruction and data-memory contents can be displayed on the PC screen and changed via an online assembler/disassembler without affecting normal SP operation (i.e., the SP can run while instructions and/or data are read/written to its memory using the cycle-steal mechanism).

The PI assembler language and programming "style" are best exemplified in the following description of a typical PI application. The assembly language mnemonic of the IBM SP is described elsewhere in this issue [5]; here we use it to demonstrate the unique features of the SP for computation-intensive kernels.

Nearest-neighbor computation kernel—An SP programming example

Suppose we want to compute the nearest neighbor (NN) to a test vector x from a set of NP prototypes in N-dimensional feature space—a common problem in pattern recognition, dynamic programming, and speech analysis. We are asked to use the Euclidean metric requiring at least one multiplication per space dimension. The problem can be formulated as follows:

(1) Initialize minimum distance.
(2) NN = 0.
(3) Do $j = 1$ to NP by 2.

If:

$$\sum_{i=1}^{N} (x_i - y_i^j) < \sum_{i=1}^{N} (x_i - y_i^{j+1})^2,$$

Then NN $\leftarrow j$;
Otherwise NN $\leftarrow j + 1$;
End.

The inequality above can be expressed as

$$\sum_{i=1}^{N} x_i^2 - 2\sum_{i=1}^{N} x_i y_i^j + \sum_{i=1}^{N} (y_i^j)^2 < \sum_{i=1}^{N} x_i^2 - 2\sum_{i=1}^{N} x_i y_i^{j+1} + \sum_{i=1}^{N} (y_i^{j+1})^2.$$

The first terms in both sides of the inequality above are canceled out and the last terms are independent of x, hence can be computed prior to the kernel invocation. We are left with the following relation:

$$\sum_{i=1}^{N} x_i y_i^j - A^j \gtrless \sum_{i=1}^{N} x_i y_i^{j+1} - A^{j+1}.$$

(A^j are the sum-of-squares of prototype j.)

Note that the summations above are simple sum-of-product operations, for which the IBM SP is optimized. As a result, a computation-intensive kernel was translated into a pipelineable (transfer-intensive) kernel that can be efficiently executed on the SP.

The SP assembly language implementation of the NN computational problem is given below as a callable SP routine. The SP assembly notation defines five distinct fields: the label, transfer code, destination/operand, ALU opcode, and comment fields. The multiplier pipeline is two instructions deep, so the multiplier result can be used two cycles after the last operand is loaded into RX or RY registers. From that point on, the SP can provide a product each cycle. In this example, FVEC is the test vector in N-dimensional space, PROT(k, i) is the ith component of the

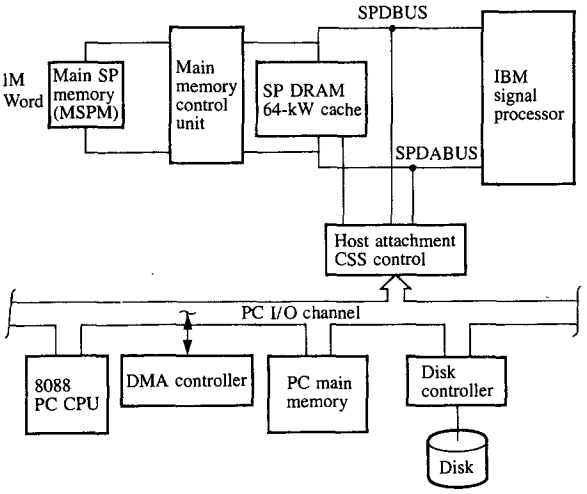

Figure 9

Main SP memory attachment to the IBM SP-based PI.

```
                Labeling by the Nearest Neighbor. Distance calculation is done as follows:

                Compute: SUM(FVEC(I)*PROT(K,I))-0.5*SUM(PROT(K,I)**2 FOR I=0,.,N-1
                         AND K=0,..,NP -1

                Note that the second sum, SMSQR(j), is precomputed prior to the routine invocation.

                Input:   FVEC (0,...N-1), N, SMSQR(NP), PROT(NP*N)

                Output:  LABEL (0....NP-1) IN RB

                Destroys: RB, TEMP1

*
#LABEL    SXM   SAVERX     Y,SB        * SAVE RX
          SBM   SAVERY                 * SAVE RY
          SAM   SAVERA                 * SAVE RA
          LDYI  0+I                    * RY = RI
          LDYJ  0+J          Y,SB      * RY = RJ, RB = RI
          SBM   SAVERI       Y,SB      * SAVE RI, RB = RJ
          SBM   SAVERJ                 * SAVE RJ
*
          LDX   -1024                  * RX = -0.5 = INITIAL MINIMUM DIST
          LDI   0                      * RI = 0
          LDYJ  0                      * RY = RJ = 0 (loop counter)
*
#DISTCAL  SXM   TEMP1        CL,SA     * TEMP1 = INITIAL MINIMUM DIST
          LMX   FVEC                   * RX = FVEC(0)
#LABLOP   LMY   PROT+I       RDL,SA    * RY = PROT(I,0), ROUND RA
          LMX   FVEC+1                 * RX = FVEC(1)
          LMY   PROT+1+I               * RY = PROT(I,1)
          LMX   FVEC+2       M+A,SA    * RX=FVEC(2), RA=RA+PROT(I,0)*FVEC(0)
          LMY   PROT+2+I               * RY = PROT(I,2)
          LMX   FVEC+3       M+A,SA    * RX=FVEC(3), RA=RA+PROT(I,1)*FVEC(1)
          LMY   PROT+3+I               * RY = PROT(I,3)
          LMX   FVEC+4       M+A,SA    * RX=FVEC(4), RA=RA+PROT(I,2)*FVEC(2)
          LMY   PROT+4+I               * RY = PROT(I,4)
          LMX   FVEC+5       M+A,SA    * RX=FVEC(5), RA=RA+PROT(I,3)*FVEC(3)
          LMY   PROT+5+I               * RY = PROT(I,5)
            .       .          .       * .
            .       .          .       * .
            .       .          .       * .
          LMX   FVEC+N-1     M+A,SA    * RX = FVEC(N-1), RA = RA+PROT(I,N-3)
                                       *   *FVEC(N-3)
          LMY   PROT+N-1               * RY = PROT (I,N-1)
          SXM   DUMMY        M+A,SA    * RA = RA + PROT(I,N-2)*FVEC(N-2)
          LDI   N+I                    * RI = RI + N
          SXM   DUMMY        M+A,SA    * RA = RA + PROT(I,N-1)*FVEC(N-1)
          LMX   SMSQR+J                * RX = SMSQR(J)
          LMX   TEMP1        A-X,SA    * RX = DISMIN, RA = RA - SMSQR(J)
          BRND  #NOREP       A-X       * IF DISMIN<RA DO NOT LABEL
          SXM   DUMMY                  * NOP
#REP      LDYJ  0+J                    * RY=RJ= INDEX OF CURRENT PROTOTYPE
          SAM   TEMP1        Y,SB      * DISMIN = RA, RB = PROTOTYPE INDEX
#NOREP    LDYJ  1+J                    * RY = RJ+1 (INCREMENT LOOP COUNTER)
          LMX   NP                     * RX = K = # OF PROTOTYPES
          BRAD  #LABLOP      X-Y       * IF Y =/ 0 GO TO LABLOP
          LMX   FVEC         CL,SA     * RX = FVEC(0), RA = 0
*
#LBLEXIT  LMI   SAVERI                 * RESTORE RI
          LMX   SAVERA                 * RESTORE RA
          LMX   SAVERX       X,SA      * RESTORE RX
          LMY   SAVERY                 * RESTORE RY
          BRM   RETURN                 * RETURN TO CALLING PROGRAM
          LMJ   SAVERJ                 * RESTORE RJ, RB contains the index
*                                      * of the NN to FVEC
```

Figure 10

Example program for the nearest neighbor computation kernel.

kth sample from a set of NP prototypes (each prototype is an N-dimensional vector).

Notice in the kernel shown in **Figure 10** that the SP pipeline is full and the only limiting factors in the execution speed are memory accesses; all other arithmetic operations are performed in parallel with the data fetches. The total number of SP cycles needed for "labeling" by the above kernel is NP * (2 * N + 11). Without the pipeline optimization (i.e., reducing the problem to a multiply/accumulate-type operation), the same kernel would require NP * (6 * N + 11) SP cycles and for $N > 10$ we would likely see computation times three times slower than that demonstrated above. Note that the SP pipeline could not be utilized in its entirety since the elements of the summation

terms in (3) must first be computed and then "stored" in the SP arithmetic registers in order to obtain the product.

The NN programming example, as well as those given elsewhere in this issue [5], demonstrates the performance capabilities of the IBM SP. Unfortunately, there exists no "high-level" optimizing compiler that can generate "compact" and efficient parallel code for the IBM SP. Therefore, the user must "manually" optimize time-critical kernels for the minimum number of SP cycles when the application so demands.

Concluding remarks

The Personal Instrument (PI) system presented in this paper constitutes the first effort at integrating proven Personal Computer and Signal Processing technologies into one cohesive system. The PI uniqueness lies in the way its components are interfaced so as to maximize the throughput of the PC CPU and its SP coprocessor without compromising any limited PC bus bandwidth.

Acknowledgments

Thanks are due Greg Daggett, Edward Epstein, and Mary Garrett for their contribution to the Yorktown PI realization. The constant support and help of Gottfried Ungerboeck and Dietrich Maiwald is appreciated. Finally, Ken Davies' continued support of this project and his help in reviewing the manuscript of this paper are much appreciated.

References

1. R. Childs, J. Crawford, D. House, and R. Noyce, "A Processor Family for Personal Computers," *Proc. IEEE* **72,** No. 3, 363–376 (March 1974).
2. *IBM Personal Computer Disk Operating System Version 2.0*, 1982, Order No. 6024001; available through IBM branch offices.
3. C. Galand, C. Coutourier, G. Platel, and R. Vermot-Gauchy, "Voice-Excited Predictive Coder (VEPC) Implementation on a High-Performance Signal Processor," *IBM J. Res. Develop.* **29,** No. 2, 147–157 (March 1985, this issue).
4. A. V. Oppenheim, *Applications of Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.
5. Jean Paul Beraud, "Signal Processor Chip Implementation," *IBM J. Res. Develop.* **29,** No. 2, 140–146 (March 1985, this issue).
6. G. Ungerboeck, D. Maiwald, H.-P. Kaeser, P. R. Chevillat, and J. P. Beraud, "Architecture of a Digital Signal Processor," *IBM J. Res. Develop.* **29,** No. 2, 132–139 (March 1985, this issue).
7. G. Ungerboeck, D. Maiwald, H. P. Kaeser, and P. R. Chevillat, "SP16 Signal Processor," *Proceedings, IEEE International Conference on Acoustics, Speech, and Signal Processing* (IEEE Catalog No. CH1945-5/84), 1984, pp. 16.2.1–16.2.4.
8. F. Jelinek, L. Bahl, and R. Mercer, "Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech," *IEEE Trans. Info. Theory* **IT-21,** No. 3, 250–256 (May 1975).
9. D. Maiwald and H. P. Kaeser, IBM Research Division, Rüschlikon, Switzerland, private communication, June 1983.
10. *IBM Personal Computer 3278/79 Emulation Adapter Technical Reference* 1984, Order No. 1502336; available through IBM branch offices.
11. G. Daggett, IBM Research Division, Yorktown Heights, NY, private communication, 1984.
12. "INTEL 8254-2 Programmable Interval Timer," Preliminary Specifications, *Intel Components Handbook 70*, INTEL Corp., Santa Clara, CA, 1983.
13. L. Bahl, F. Jelinek, and R. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pattern Analysis & Machine Intelligence* **PAMI-5,** No. 2, 179–190 (March 1983).
14. G. Daggett, IBM Research Division, Yorktown Heights, NY, private communication, June 1984.
15. J. Fallin, "The IRAM—An Innovative Approach to Microprocessor Memory Solutions," *National Computer Conference Proceedings*, AFIPS Press, 1983, pp. 16–19.
16. E. Epstein, IBM Research Division, Yorktown Heights, NY, private communication, August 1984.
17. D. Maiwald, "PIE—Software Simulator," *Internal Report No. 3/1984*, IBM Zurich Research Laboratory, Zurich, Switzerland, January 1984.

Received August 8, 1984; revised October 21, 1984

Gideon Shichman *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Shichman is currently the manager of speech engineering in the speech recognition group at the Thomas J. Watson Research Center in Yorktown Heights. After graduating from Yale University, New Haven, Connecticut, he joined IBM Research in 1982 in the Department of Systems and Information Sciences. Prior to his graduate work at Yale, Dr. Shichman was the research and development director of Digitronics Israel Ltd., Tel-Aviv, Israel. He earned his B.Sc. in electrical engineering from the Technion, Israel Institute of Technology, in 1975, followed by an M.Sc. in electronics from Tel-Aviv University in 1978. He earned an M.Sc. and a Ph.D. from Yale University, both in systems and information sciences, in 1980 and 1982, respectively. Dr. Shichman is a member of the Acoustics, Speech, and Signal Processing, the Engineering in Medicine and Biology, and the Institute of Electrical and Electronics Engineers computer societies.