A Low-Cost Development System for Digital Audio Signal Processing

Stephen F. Hugo
Anthony Agnello
Eventide, Inc.
1 Alain Way
Little Ferry, NJ 07643

ABSTRACT

This paper describes the Eventide SP016, a commercially available, low-cost digital audio signal processor, and SPUDS, a software environment designed for the development of SP016 programs. The architecture of the SP016 is presented, along with a discussion of the assembly and high-level languages used to write its programs. An overview of the program development cycle is given, along with a description of the structure and features of SPUDS. The complete source code of an SP016 program implementing a simple Schroeder reverberator is listed in the Appendix.

1. INTRODUCTION

The SP016 is a general-purpose audio processor optimized for time-domain applications. Featuring 2 independent A/D and D/A channels, 55K word-delay memory, and a full signal bandwidth of 1kHz, it is a versatile unit designed for professional recording studios, sound-reinforcement, and electronic music applications. All functions of the SP016 can be controlled either from its front panel, a hand-held remote control, or an external computer. The processor programs stored in SPUDS are provided with the unit to perform many types of reverberation as well as delay, flanging, chorusing, filtering, key-slicing, and other special effects. A non-volatile memory allows the storage of user-created programs.

A unique feature of the SP016 is that's it's aimed at the core of the commercial audio processors: its ability to be a complete software development environment for the SP016. SPUDS consists of a compiler and an assembler for proprietary digital signal processing languages developed by Eventide, as well as a linker, an editor, and a fast, high-speed communications subsystem. SPUDS will be portable and will run on the current generation of Macintosh computers and workstations.

2. SP016 SYSTEM ARCHITECTURE

A block diagram of the SP016 is shown in Figure 1. The SP016 is a microprocessor-based system with interconnections of its processors, memory, and I/O.

![Diagram](image_url)

**Figure 1.** Block diagram of the SP016, showing the interconnection of its processors, memory and I/O.
2. THE ARRAY PROCESSOR

A block diagram of the array processor is shown in Figure 2. The array processor can perform signal processing and basic signal processing arithmetic (addition, subtraction, multiplication) or a continuous stream of 16-bit samples derived from the SP2016's two V/3 conversions (shown as ADCS, analog to digital converters) or 16 registers (ALUREG), a 4x48 multiplier (MULT), and up to 1K steps of distributed program memory. The output of the processor may be directed to any of 4 data converters (DACS) or to a digital test port (TPORT) accessible by the SP200.

The ALU executes data movements, arithmetic, and logical instructions using the INSTRAM. Table 1 lists the mnemonic codes for each instruction and its meaning. Most ALU instructions accept either 1 or 2 registers, one of which will always be the destination of the result. Most instructions also come in versions that use the multiplier product as a source operand (those encoded whose mnemonics begin with a H). Although the main data path connecting the various elements of the processor is 16 bits wide, the ALU always operates the most significant 24 bits of the result operand. An important feature of the ALU is its ability to automatically detect and correct overflow of an arithmetic result within a single instruction cycle. This ability to perform integer arithmetic is essential for signal processors that operate on audio data.

The processor's audio data memory consists of 1K 16-bit words of fast static RAM (SRAM) and 65K words of slower dynamic RAM (DRAM). Both memory arrays are automatically configured as signal delay by the hardware, making it easy to set up large numbers of delay lines in memory simultaneously. Having this large memory space dedicated to signal delay is essential for effectively implementing time-domain applications such as synthesis and sound editing. Unfortunately, it precludes the use of data memory for techniques like table lookup in many methods of digital audio processing (Walrath 1979) describes a true synthesizer whose basic architecture is related to the SP2016.)

The SP2016's sample rate can be varied in octaves from 4kHz shown to 5kHz, equalizing in audio bandwidth of 16kHz to 2kHz. The array processor's cycle time is defined in hardware at 200 nanoseconds, yielding a frame length of 1024 steps at 16kHz, 256 steps at 4kHz, etc. The lower sample rates can be used for computationally intensive applications like banks of high-order filters (e.g., equalizers) or when very long delays are required by a program. For example, using the processor's entire dynamic RAM array of 65K words for a single delay line results in a delay of 3.27 seconds at the 2kHz bandwidth. The SP2016 allows the sample rate to be switched while a program is running, which is a special feature or to let users make their own hardware delays. This feature is available for the SP200.

Program memory for the array processor is split among three 16K 18-bit words:

- INSTRAM contains 18-bit instructions (ADC, DAC, and port and RAM enable bits) and ALU instructions (register operands and opcodes for arithmetic, logical, and data movement operations).
- COEFROM contains 16K 18-bit multiplier/multiplexer coefficients for each step, allowing a multiplier operation to be initiated every 5000 clock cycles.
- DSPROM contains miscellaneous instructions for read and write operations. Access to internal dynamic RAM array is permissible, but static data is not accessible by this program step.

The SP200 has a free read and write access to the three fields of the array processor's program memory. Over 16-bit single-field access can occur once each sample period, scheduled by the hardware to occur at step 1 of the program. The limit of one cycle per sample period is a modest timing constraint, given a sample period of 25.6 usec (0.016kHz bandwidth) and the SP200's clock rate of 1.2 MHz. In contrast to the SP200, the array processor is restricted to reading values from its program memory as the program advances, incrementing them. Unfortunately, this process requires using the array processor to calculate its own instructions or address offsets. In the case of coefficients, however, the entire 18-bit word (XPROD) allows any valid value to be used. Again, the read from the XPROD or D latch to the delay link to be used as a multiplier, this requires processing the ring modulation or the computation of amplitude envelopes.

The SP200 may also read the output of the digital test port (TPORT), which is a 16-bit latch whose inputs are connected to the array processor's data bus. The array processor can be programmed to write to the XPROD exactly as to a D/A converter, enabling the creation of software applications that alter their signal processing parameters dynamically in response to the signal itself. (Walrath 1979). The test port could also easily be added to implement a sampling-to-disk system using the SP2016's as a front end.

2.2 ARRAY PROCESSOR PROGRAMMING

Because of the so-called horizontal micromanipulation of the three memory fields and array processor's pipelined architecture, each step in a signal processing program might consist of several operations. In general, on each step the array processor may perform:

ICMC '85 Proceedings
**TABLE 1**

Memoramic operators for ALU instructions and their axioms. The first row represents the 16 standard instructions, the second is a set of specially enabled alternative instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>A + B = C</td>
</tr>
<tr>
<td>SUB</td>
<td>A - B = C</td>
</tr>
<tr>
<td>MUL</td>
<td>A x B = C</td>
</tr>
<tr>
<td>DIV</td>
<td>A / B = C</td>
</tr>
<tr>
<td>AND</td>
<td>A and B = C</td>
</tr>
<tr>
<td>OR</td>
<td>A or B = C</td>
</tr>
<tr>
<td>XOR</td>
<td>A xor B = C</td>
</tr>
<tr>
<td>NOT</td>
<td>A not C = C</td>
</tr>
</tbody>
</table>

Array processor programs are created by assembling the source code of the program written in a low-level language (commonly machine language) as an ALU language. The form of the comments in an ALU program correspond closely to the architecture of the array processor. In parallelism, and program memory field access is reflected in the ability to specify multiple operations on each array value of the program. As a simple example, the ALU program fragment 1 illustrates a comb filter, (i.e., a repeating delay line) with a loop per-pixel. Larger DSP programs in the SP2016 can be constructed from basic blocks like this comb filter, assembles a concept that the programmer can take full advantage of the array processor's parallelism to exploit a performance-environment that isn't divided at first glance. By way of a benchmark, its latency, at a 16-bit bandwidth the array processor can implement a least 30 second operand in 256 points. (Hammer Filter) It's a Schonberg realization of the one in the Appendix links (with plenty of steps left over to generate some easy reflections on memory and even some extremely impressive improvements on this basic algorithm).

**2.3 ACCESS CONTROL PROCESSOR**

The 6809 microprocessor is the master controller for the SP2016, responsible for controlling the execution of the array processor as well as handling interrupt functions like sending the alphanumeric display, servicing keystrokes, and keypad movements, and responding to the hand-held keypad supervisory and the GPIO interface. If the SP2016 changes to a non-directional communication over an IEEE-488 bus between the SP2016 and any computer equipped with a GPIB compatible interface. A large set of IEEE-standard microcomputer interfaces are available.

- Emulate the SP2016 from front panel and see the display
- Lockdown control from panel control
- Read/write anywhere on the 6809 or array processor's memory
- Reset the 6809 or SP2016, boot up the array processor
- Switch sample rates, program pages, hexadecimal formats, etc.

The 6809's communication and control routines are included in a ROM-based operating system of approximately 1250 bytes. This embedded is also operating system is an inscrutable for programs written in a high-level language (formally C48, the Signal Processor Users' Laisné-Programming language). SP2016 serves as a flexible, programmable interface between the SP2016's front panel controls (for remote equivalent) and the array processor. The 6809's SP2016 programs execute in parallel with the array processor's DSP programs, interacting with its sequence in keyword and data movements made by the user. This interaction usually takes the form of reading and writing new instructions, coefficients or offsets into the three fields of the array processor's parameter memory, though it could also mean reading the tap positions or writing to locations that control the sample rate, analysis context, or other hardware.
2.4. SPUD PROGRAMMING LANGUAGE

SPUD programs are organized around the idea of parameter-clarified, defensibly apportioned control which the user can examine over a program. SPUD is a topical, parameter-apportioned, stream-based language, for example. The structure of SPUD provides for routines that serve and display the values of a program's parameters, with the sequencing between parameters taken care of by the operating systems. A parameter's value is read into memory (into) a display routine and then written to the display buffer (DBUF) with a display routine. Special routines may also be written to serve and display the states of the display buffer, as needed. To perform one such program function, special utility macros must be written to do so. In conclusion, the utility macros must be to do the functions to display the buffer from the display buffer.

A powerful feature of SPUD is its ability to perform continuous quality control using special software modules in the background while programs and software are being processed at the foreground. Quality control is an effective application for this computer processing. A background routine monitoring a testable, parameterizable test result (for example, the length of time between delay lines, changing Display-Buffer-pitch shift and delay changes in the audio output. In the foreground, meanwhile, parameter-setting and setting services are carried out. The parameter-setting feature can be simultaneously responding to a page cursor in the foreground, executing a routine in the background, and supporting communication with the GPIB interface (or, to quote, the display device is a computer, perhaps). In other words, the foreground and the background routine operate in a high priority line connection, the quality of which processing will continue unimpaired.

The output appearance of SPUD falls somewhat between BASIC and a block structured language like Pascal. Structured flow-of-control constructs are available as well as any structured control constructs. SPUD includes the GOTO and GOSUB targets that are symbolic labels. A complete compilation facility allows the creation of useful SPUD subroutines in calculator file format. For example, it is possible to store in SPUD and called from different programs. A parameter value is determined through the environment of the program, plus some extended arguments and line operations design into the routine, and line computational routines are contained. In pseudocode, the extended binary value, CURRENT DATA values may be used to control the READ statement for look-up evaluation. A large number of special keywords are included that are not used in an ordinary computer language and are used to control values aspects of the SPUD's hardware. Line error checking is done on the specific code-time line and displayed on the fly by the GPIB commands.

As an programming example, the SPUD program targets to list a program's service and display routines for a FGA2 parameter which could interact with the ALU program for a pulse filter that is used in program 1. When the pulsing of the FGA2 parameter, the routine for the display follows by the value of the parameter, formatted a percentage from 0% to 99%. Each time the filter is moved (or insufficient in what it is called) AGCU mode, the display routine display routines are re-executed, thus updating both the feedback control in the array processor program memory and the displayed value on the front panel DLS.
Figure 5. SPUD program development cycle. After compiling and assembling the source text of the SPUD and ALU programs the code files are linked and may be programmed into EPROM or downloaded directly to an SPUD16 via the GPIB.

Figure 6. The SPUD system is structured as a group of subcircuits which are managed by a Monitor. The Monitor supervises the user from the host DOS by processing a complete development environment.

system. The resulting 512-Kbyte EPROM can be programmed into
EPROMs and permanently installed in an SPUD16 or downloaded over
the GPIB directly into the address space of the 6502 for immediate
execution and testing.

The user provides a fair degree of independence between the
ALU and SPUD programs, since either can be rewritten and assembled
or recompiled by the other even exists. An ALU program may
export memory labels, constants and delay line state information
to the SPUD program (using special symbol declarations) with the
actual values of the exported symbols resolved only at link time.
Careful use of export can make the program re-execute and
re-compilation of one program if necessary even when the other one has
been radically replaced or restructured, a process that may occur many
times during the program development cycle.

3.2 SPUDYSTEM DEVELOPMENT ENVIRONMENT

The complete program development sequence (assemble-
compile/link/download/execute) is maximized and enhanced by
the structure and provided within the SPUDYSTEM development
environment. Although SPUDYSTEM runs as an
application program under the operating system of its host
computer, it actually includes many of the services that
would normally be provided by an operating system. It tightly
integretizes the SPUD compiler, ALU assembler and linker under a common
command interpreter called the Monitor. The Monitor can also call
on a powerful general-purpose editor, a program promoter
subsystem, and sophisticated file management and GPIB
communication services (figure 4). The Monitor is a monitory,
intelligent shell that translates the user from microcomputer
operating system gobbledygook and memory deals which
aren't germane to the
SPUD16 program development process.

SPUDYSTEM'S various subsystems are divided from the Monitor
command interpreter by typographical rules from the keyboard or using a line of
screen commands to ease forwards into the command line.
System directed utility prompts change its context dynamically as
a command is built, allowing any one point in the command
command set of synonymally valid keyboard choices. The result is
a command entry scheme which has the self-prompting features of a menu
but doesn't clutter or dominate the screen. The Monitor also
provides powerful command line editing features like a
command recall stack, a full range of backspace/delete keys and
cursor controls, horizontal scrolling for overlength files, a
4-function calculator with one conversion, etc.:

To provide maximum functionality, SPUDYSTEM uses a dedicated
file system which runs on top of the host file system. This scheme has
multiple advantages: users can easily add files to exist
SPUDYSTEM to perform file maintenance tasks, the system is
optimized to handle those types of files necessary for the SPUD16
development, and files are provided that may not be available on
the host OS, for base: included

- Individual disk directories keyed to user IDs, which can be listed
from any subsystem
- Without the operating system's file protection
- Navigation command files to automate command sequences, with
prompts for user-supplied or default parameters
- Log files to capture sequences of commands or GPIB messages
- Default filename and types to simplify command entry
- Default file recovery and automatic 한번

Through its file, SPUDYSTEM can interface with up to four
SPUD16s and an external SPUD program over the GPIB. Any
of the SPUD16s High-level mnemonic commands can be issued
directly from the keyboard over the GPIB with the response
switched back to the SPUD16 front panel, SPUDYSTEM's CRT screen, or a next
file. This feature can be used to write command files that emulate
the SPUD16 front panel computer or change its other operating
characteristics, either for debugging purposes or to provide
automation for line performance testing.

In most software development systems, the author must
make sure that the build of the unit is consistent with a host
editor. For SPUDSYSTEM, considerable effort was spent creating a
fast, screen-oriented editor for SPUDYSTEM. The resulting editor
features:

- Memory-resident text storage for very fast operation
- Vertical and horizontal scrolling to 128 columns
- Bidirectional find and replace commands using any word and any
mount characters
- External file merging and saving (total or partial file by line
numbers)
- Undo of accidental deletes

177 CMC '85 Proceedings
4. IMPLEMENTATION

SBC Odyssey is implemented as a large modular program of approximately 35,000 lines written in Pascal. From its inception, this program was intended to be portable to other hosts, so care was taken to adhere to the ISO language standard, and apart from writing or separate compilation of program modules, to isolate the inevitable compiler and operating system-dependent features in a single module of C/V Pascal. The original implementation was written on 6809-based Hewlett-Packard Series 200 desktop machines, and an implementation is currently in the works for MS-DOS computers (i.e., the IBM PC and its clones). Implementations for other systems are being considered, and potential users and implementors are invited to contact Odyssey.

The system has been in use as an in-house facility for approximately one year. The version of SBC Odyssey distributed to users is essentially identical to the one used to develop the factory programs supplied with the SP206. Thus SBC Odyssey users have at their disposal a complete set of both of the SP206 program development.

5. CONCLUSION

The SBC206, together with the SBC Odyssey development environment constitutes a powerful system for originating and developing digital audio signal processing programs. While the SP206 is a fully functional signal processor as supplied "off-the-shelf" for programming, its programmability permits the user to develop unique applications that weren't envisioned by the original designers. The relatively low cost of the SP206 and the 6809 chip set used in Odyssey's personal computer-system means that DSP research and development is no longer limited to those with access to mainframe computers and large R&D budgets.

ACKNOWLEDGMENTS

The authors wish to express their appreciation to Mark Chayes and Frank Davey for their work on earlier versions of the SP206 hardware and software.
Listing 3:

SPUD

*-------------------------------------- SCHROEDER REVERBERATOR -------------------------------*
* The single parameter in this program allows the user to select from 5 different decay times. *
* by modifying the parameter value.                                                        *
*--------------------------------------------------------------------------------------------*

! ------------- DECLARATIONS --------------

PROGRAM "SCHROEDER REVERB" [28]  
EXTSTEP  
INPUT STEP, Decay_time  
DEFINIT  
DECAYTIME = 0.  
FSCALE = .99999  
DEFVAR  
* +1, +newcoeff = 1  

! ---------------- PARAMETER ROUTINES -------

DEFAULT  
PRM(DECAYTIME) = 3  
PS DECAYTIME :  
SELECT  
LIMIT = 5  
newcoeff = READ(Dec_table)  
PRM(DECAYTIME)  
FOR (i = 0 TO 12)  
STEP 4  
DOE(Decay_time * i) = newcoeff  
NEXT  
DOE(input_step) = (SCALE * newcoeff) / 4  
END  
PD DECAYTIME :  
DISPLAY "DECAY TIME:"  
DEFINE "R90 (N SECONDS)"  
END  

Dec_table  
DATA .707, .856, .915, .94

109  
ICMC ’85 Proceedings
SCHROEDER REVERBERATOR

- This array processor program implements Schroeder's algorithm for reverberation.
- It requires 4 parallel comb filters followed by 2 series all-pass filters. The
- comb filters have long delay to provide long-term reverb decay, while the all-pass filters
- deliver high echo density.

PROGRAM "SCHROEDER REVERBERATOR"

DECLARES

PROSIZE = 128

DELAY

COMB[1] 30MS

COMB[2] 40MS

COMB[3] 45MS

ALPASS[1] 1.7MS

ALPASS[2] 1.7MS

DEFCONS

G = 75

FEEDBACK = 707

SCALE = (1/FEEDBACK) / 4

EXECUTABLE STATEMENTS

NOP

Decay_step:

NOP

PAD m in = 0

PAD m out = 0

temp = 0

comb_sum = 0

comb[1] = 0

comb[2] = 0

comb[3] = 0

Alpass[1] = 0

Alpass[2] = 0

ADC1 X SCALE

(Compound input)

ADC1 [0-MS] X FEEDBACK

(Compound input)

ADC1 [0-MS] X 9999

X = Comb[1]

(Compound input)

ADC1 [0-MS] X FEEDBACK

(Compound input)

ADC1 [0-MS] X 9999

X = Comb[2]

(Compound input)

ADC1 [0-MS] X FEEDBACK

(Compound input)

ADC1 [0-MS] X 9999

X = Comb[3]

(Compound input)

ADC1 [0-MS] X FEEDBACK

(Compound input)

ADC1 [0-MS] X 9999

X = Alpass[1]

(Compound input)

ADC1 [0-MS] X 9999

X = Alpass[2]

(Compound input)

ADC1 (Audio output)

REFERENCES


New Jersey: Essex Inc., Inc.


Massachusetts: Karger Academic Press.


ICMC '85 Proceedings