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

Digital Signal Processor:

Software Simulator

By J. AAGESEN

(Manuscript received June 23, 1980)

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

I. INTRODUCTION

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

* Registered trademark of Bell Laboratories.

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

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

II. ARCHITECTURE

2.1 Overview

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

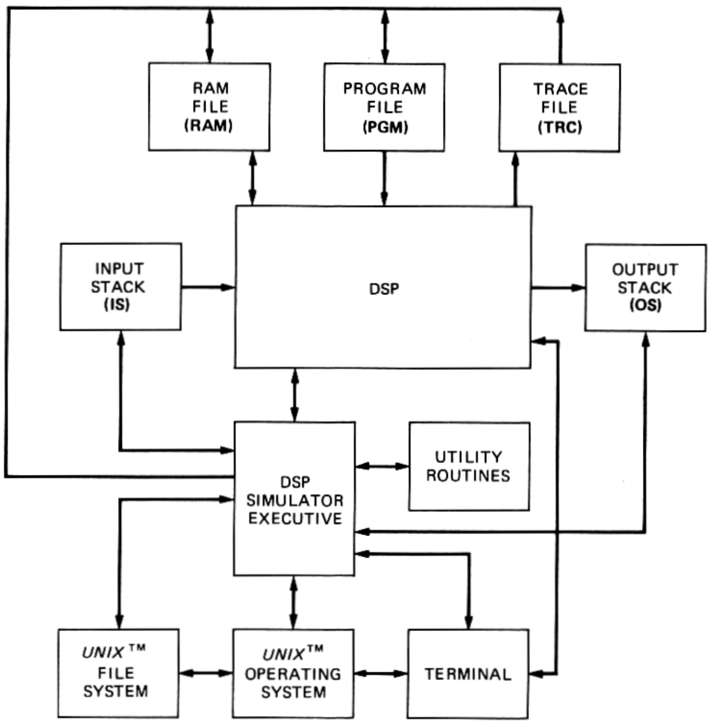

Fig. 1—Block diagram of DSP simulator.

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

2.2 Data formats

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

2.3 File formats

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

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

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

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

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

data[* scale factor][+ offset].

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

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

III. CONDITIONAL OPERATIONS

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

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

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

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

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

IV. COMMANDS

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

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

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

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

The expression has the structure

```
cond op cond op cond ···
```

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

```
sc pc = 10

sc a = 1234.5

when (pc|a){dmp pc; dmp y}
```

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

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

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

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

V. TERMINAL SESSION

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

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

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

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

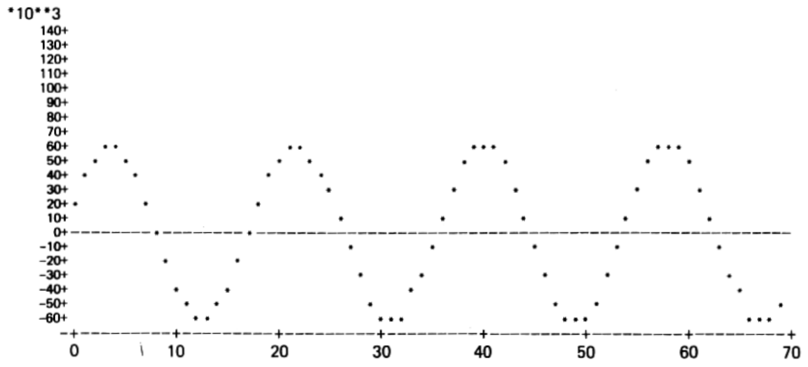

Fig. 2—Terminal session on DSP simulator.

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

VI. ACKNOWLEDGMENT

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

REFERENCES

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