DSP-Sound: A Software Synthesis Package for Real-Time DSP-based Systems

Russell F. Pinkston
Music and Audio-Visual Research Center
Department of Music
The University of Texas at Austin
Austin, Texas 78712
music@emx.utexas.edu

ABSTRACT: This paper describes a software synthesis package called DSP-Sound, which is currently being developed at the University of Texas at Austin. The package is designed for use in a host system containing one or more digital signal processors, AD/DA conversion hardware, and a MIDI interface. The software is divided into three basic components: a MUSIC250-style synthesis language which provides the standard unit-generators coded as DSP assembler macros; a host-control program which supports both real-time and non-real-time use of the DSP(s); device drivers and application programming interfaces for the DSP(s) and MIDI.

Background: This project is based on some recent work done at the MAVRC involving real-time software synthesis using the Motorola 56001 and TMS320C25 Digital Signal Processors. The principal goal of that project was to find a means of combining the benefits of real-time performance control via MIDI with the flexibility and power of software synthesis, and to do so in a system which was generic, relatively inexpensive, and easy for musicians to use. To satisfy these criteria, an IBM PS/II Model 60 and an Apple Mac II were chosen as the primary host platforms, a symbolic compiler called Patchwork was written which allowed users to design synthesis algorithms graphically, and a real-time synthesis language was developed - first for the Accelerando Box (a stand-alone DSP module containing a single Motorola 56001) and subsequently for the IBM Audio Capture and Playback Adapter (an inexpensive audio card with 16-bit stereo A-to-D and D-to-A converters and a TMS320C25 DSP chip). It proved to be easy to design and test instruments using this system, but it soon became apparent that none of the DSP chips currently available was capable of synthesizing more than a few voices of musically interesting sounds in real-time. Hence, to be of any practical use as a performance instrument, it would necessary to employ multiple DSPs. However, a single-DSP system might make a suitable computer music "workstation," if there were a mechanism for using it in a non-real-time, or "batch" mode. Work on such a mechanism was begun, but it soon became clear that with our current software, it was difficult to develop instruments which could operate gracefully and efficiently in both real-time and non-real-time modes. In addition, the experience of porting our DSP software from the Motorola 56001 to the TMS320C25 suggested that we needed to devise a more generic synthesis language in order to maintain source compatibility over a potentially wide variety of DSP chip architectures.
A Generic Synthesis Language for DSP Chips

DSP-Sound is a software synthesis language designed for use with Digital Signal Processors. It is based on Keith Lines's original MUSIC5600C but its unit generators and other functions are designed to be as non-system-specific as possible and to be usable in either real-time (RT) or non-real-time (NRT) mode. Its syntax should be familiar to anyone who has used one of the MUSIC64 style languages (MUSIC64, MUSIC64, etc.) or its descendants (CSOUND, etc.). Of all these languages, DSP-Sound is closest in concept and syntax to Barry Vercoe's MUSIC360, in that the "orchestra" consists of assembly language macros, which generate highly efficient in-line code for the target system. Unlike MUSIC64, however, DSP-Sound is designed for real-time performance control and its control mechanism is a MIDI data stream. Moreover, it can't make any assumptions about the target system's word size, data representation, operating system interface, etc. Consequently, all functions must follow strict conventions with respect to the ranges and types of input and output arguments; arithmetic expressions must be limited to very basic operations, and constants entered through special macros for variable initialization. These data types are supported: real numbers between +/- , 1, signed integers between +/- >2 4 WORDSIZE-1 and fixed point numbers with 8 integer bits and WORDSIZE-8 fractional bits. Support for double precision is also planned.

Like MUSIC360, each statement in a DSP-Sound orchestra is a call to a particular assembly-language macro. Within each macro, code may be included for any of three basic control sections: Setup, Initialization, and Performance. The Setup section is executed once at program load time, initialization typically at the beginning of each new note event, and Performance on every sample. Since DSP-Sound is intended for use in real-time synthesis, a slightly different approach to instrument design is taken. A synthesis algorithm is assumed to be executing all the time, with a limited number of its parameters being modified at more-or-less unpredictable moments by incoming MIDI messages, e.g., note-on/off, velocity, aftertouch, pitchbend, and various continuous controllers. Several different parameters may be affected by the same MIDI event, but there are relatively few types of MIDI events that a typical instrument would need to take into account. Like a Yamaha DX7 voice, most of an instrument's parameters will be constants stored in external tables, which can be filled before execution begins. Consequently, DSP-Sound does not provide a full range of mathematical functions during initialization and performance, on the assumption that changing MIDI values can be mapped onto complex functions using lookup tables, and that complicated calculations are inherently inadvisable to real-time synthesis, anyway. Instead, any special mathematical functions required to calculate filter coefficients, etc., are placed in the Setup section, which can be overlaid after its execution.

Real-Time and Non-Real-Time Libraries

"Efficient use of available resources" must be defined differently for real-time and non-real-time modes. In real-time mode, the overriding concern is to minimize the number of DSP instruction cycles required for a particular operation. Consequently, variables should be sized on-chip RAM whenever possible, indexes addressing modes and frequent branches should be avoided, inline code should be used rather than calls to functions and subroutines, etc. Since the use of subscripted variables and re-entrant code would tend to reduce performance, instrument multiplicity (e.g., polyphony) is handled by simply duplicating all code and variables - one is much more likely to run out of instruction cycles than to run out of program or data space. In non-real-time mode, however, it is not so important to count cpu cycles, although speed is always desirable. Here, the limiting

IBM CO A 369 90 PROC. ENG.
factor is likely to be the rate at which the host can service the needs of the DSP chip, especially the reading and writing of digital audio files to disk. Hence, a different principle must be applied to the DSP coding: take maximum advantage of the available program and data space, so as to perform as much of a task as possible in a single pass. Consequently, the macros in the NRT Library use slower, but more plentiful external RAM for both programs and data. Instruments are coded as functions which can be called as often as necessary to produce a desired order of polyphony, with separate data space allocated for each note’s parameters and variables. All macros which would normally handle real-time I/O functions (Audio IN and OUT, MIDI handlers, etc.), instead direct their inputs and outputs to the host. Timing is no longer controlled by dac interrupts and internal clocks, but is based instead on counting samples. The beauty of the system is that the user only needs to write one synthesis program: the same instrument can be compiled with the RT Library and performed in real-time under MIDI control, then re-compiled with the NRT Library and used in a batch job without requiring any changes to the original code.

The Host Software

As part of the original "Accelerando Project", two different kinds of "Play" programs were written to control the DSP directly from the host (rather than MIDI) - one for real-time and one for non-real-time mode. The real-time version read an ASCII score file which was essentially a direct translation of a delta-time-stamped MIDI data stream and simply sent each new message to the DSP along with the time remaining until the next message; the DSP then signalled the host when the time had expired, the host sent the next message and new delta time, etc. For non-real-time mode, a different Play program was written. Similar in structure to Paul Lansky’s MIX, it could read a command file, read programs into the DSP, and synthesize an entire score once "part" at a time, "mixing" each part into a sound file stored on disk. This strategy allowed the DSP code to be relatively simple, since an "instrument" was never required to synchronize more than one voice at a time. It could be extremely inefficient, however, with respect file I/O, unless the host could allocate very large amounts of memory for disk buffering, and the DSP program space was often underutilized. With the DSP-Sound language, however, only one Play program is required. The host does not need to know if the DSP is running in real-time mode or not, since the mechanism for communicating with it is always the same: the DSP informs the host when it is ready for the next message and the host sends that message along with the number of samples remaining before the next message will arrive. The host software does need to know if input and output audio files must be opened and accessed, but since the DSP is controlling the timing of events, it makes no difference which mode is actually in effect. During a performance, if the DSP is unable to generate as many simultaneous voices as the score requires, it has the responsibility to handle the situation gracefully. The new Play program is thus much closer to the classic MIDI sequencer model than it is to the traditional MUSICn model: a "score" is no longer merely a collection of time-stamped notes for various instruments which must be performed sequentially. Instead, it is an organized collection of "parts" or "tracks," which allows the user to select which voices should be played by which instruments for each run-through of the piece.

References