Music Synthesis by Simulation using a General-Purpose Signal Processing System

by
John Kitamura, William Buxton, Martin Snelgrove and Kenneth C. Smith

Department of Electrical Engineering
University of Toronto
Toronto, Ontario
M5S 1A4

Abstract

A flexible system has been built with programmable digital signal processors (DSPs) and which utilizes a recently designed architecture that is extendible for multiprocessor applications.

This device is modular in design, so that an inexpensive minimum-parts configuration can be expanded into a large and powerful machine. The current prototype is both RS-232 and MIDI compatible.

Several examples of real-time music-related programs have been written and tested. These include a simple 5-voice FM synthesizer, a harmonizer and a simulation of a drum head by first principles.

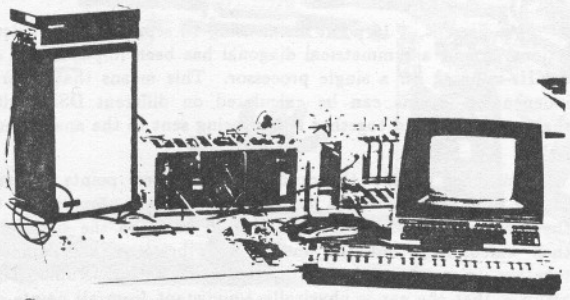
1. Introduction

A programmable general-purpose signal processor has been built which can perform a variety of different algorithms in real time. It is powerful yet inexpensive in its minimal configuration, and can easily be expanded by adding modules to meet the needs of larger processing tasks.

Modules consist of printed circuit (PC) boards, each housing two digital signal processing elements (see figure 1). Processing elements can be linked with 'software patchcords' in a variety of configurations to suit different algorithms. This encourages experimentation with new methods of computation-intensive music synthesis which previously could only be realized with expensive, special-purpose high-speed hardware.

In addition, a cross-assembler and a fully equipped monitor program has been written to aid in the writing and debugging of microcode. A full description of the hardware and software support will follow.

Several synthesis and processing functions have been implemented to test the flexibility of the system and are detailed later in the paper. A particularly interesting example of this is the simulation of a drum from first principles. Simulating an instrument from first principles can be done by modeling the instrument as a series of moving points governed by equations of motion. This can produce realistic effects which are overlooked by more common types of synthesis. The realistic nature of this algorithm coupled with a well chosen input device will make it easier for non-technical musicians to understand and use, since all variables are based on actual physical parameters.



Current Prototype of the General Purpose Signal Processor

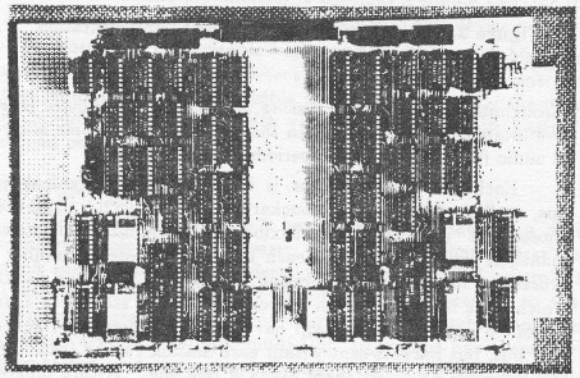


Figure 1. PC board 'module' containing two DSP elements

2. Hardware Description

Processing elements were designed around commercially available Digital Signal Processors (DSPs) to reduce the size, complexity and cost of the hardware. This special-purpose processor, a Texas Instruments TMS32010, is able to perform the high speed computation normally handled by large blocks of specialized hardware. It has been used by other Computer Music oriented groups in the design of their signal processors, notably the S.I.M. 'Soft Machine' [Del Duca1985] and the Compusonics DSP-2000 [Stautner1985].

Since the processor is programmable, it allows great flexibility, as different programs can simply be loaded into program memory and then executed. However, this can limit the efficiency of the processing element due to the fixed internal structure of the DSP, and also to its pared-down and specialized instruction set. For example, it requires four 200ns cycles to evaluate each term of the function $x = a \times b \times c \times \cdots$ even though one multiply can be done in a single cycle. It is also awkward to access internal data ram with more than one level of indirection.

Furthermore, any external data transaction with its internal memory requires a minimum of two cycles, provided the transaction is synchronous. Since full flexibility implies an asynchronous communication system between processors, each TMS32010 will have to spend extra time in polling loops to facilitate its I/O. Because of this delay it is important to find a very fast bus so as not to further decrease the efficiency of interprocessor communication. The bus also has to have a very large

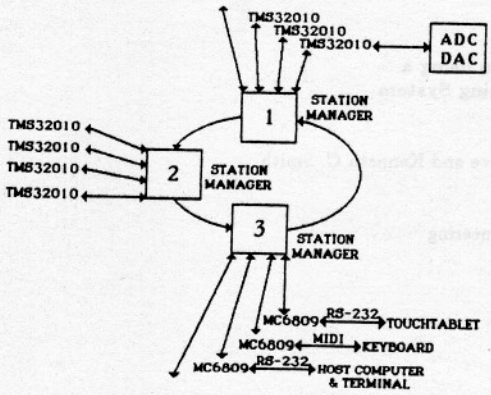


Figure 2. P-Bus Structure and Current Implementation

throughput, since some processing algorithms may require extensive intercommunication within the short periods of time allowed in audio frequency sampling periods.

Each processing element is housed on a hierarchical ring bus, called the P-Bus [Loucks1980], [Rose1982], [Rose1985] chosen for its speed and flexibility. A number of station managers lie on a ring, and each manager accesses a number of processors in a star configuration (see figure 2). Thus packets which have to be sent from a processor on one station to a processor on another station will have the throughput of a general ring bus, but packets sent within each station can be transferred immediately, provided there exists a free slot. Within different stations, these data transactions can occur simultaneously. Thus overall throughput is much greater than that of a ring bus.

The current P-Bus prototype consists of three stations capable of each housing four processors. It is 38 bits wide, which allows 24 bits of data, 12 bits of source/destination address and 2 bits of control to travel in parallel around the ring. Each station manager handles data transactions with its processors via a standard handshaking procedure. The bus is also extendible, in that extra stations can be inserted in the ring (increasing the number of nodes) and extra processors can be added per station.

At present, there are six independent processing elements housed on the P-Bus. These are arranged as pairs of DSP elements on a PC board at a hardware cost of under $200 per board and a count of about 45 TTL chips. Each board is approximately 7×10 inches and slides into a 19 inch card cage, also containing the P-Bus station managers. There is also a DSP element specially configured to communicate with an analog (ADAC) port, a 16-bit bidirectional stereo interface capable of audio quality standards and with a variable sampling rate (see figure 3). All other processors must gain access to the analog interface through this DSP, which can set or disable the sample period counter.

Accompanying the DSPs are several general purpose processing boards employing Motorola MC6809s which are used as peripheral interfaces, including RS-232 and MIDI. The MC6809s receive and process control information into packets, store them, and send them on to the DSPs when required. A multiprocessor monitor featuring single-stepping, disassembly and the ability to read or write the DSP program and instruction memory has been implemented and runs on one of the MC6809s.

Thus DSP programs can be stored and downloaded from one of the MC6809s, or downloaded through a MC6809 via the RS-232 port from an external host computer. Programs can then be debugged with the aid of the disassembler and single stepping monitor. Another useful feature, breakpoints, will soon be added to this monitor.

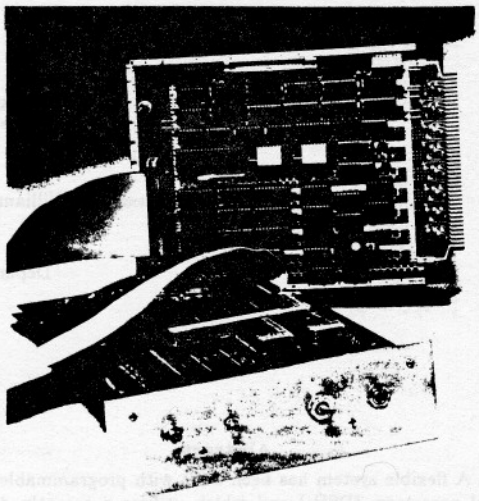


Figure 3. Specialized DSP and Analog Interface

3.1 Drum Synthesis by First Principles

Music synthesis of conventional instruments usually involves attempting to reproduce the harmonic structure of the output of the instrument by manipulating a number of oscillators. Although this can be done quite effectively, the variables by which the output is adjusted do not often correspond with actual physical parameters. Additionally, precise control over the parameters becomes awkward when the harmonic spectrum of the instrument is rapidly shifting.

An alternative to this kind of synthesis is simulation from first principles. When the actual sound of the instrument is created by a simple vibration or motion it is possible to simulate this motion and thus actually recreate the sound of the instrument from first principles. The surface of a drum was chosen as a suitable model because its structure is simple to model and it produces a rich, time-varying and inharmonic spectrum which would be costly to reproduce by more conventional means.

The drum is modeled as a series of point masses on a surface, each capable of moving in only one dimension. These points are arranged on a $n \times m$ grid with a fixed outside perimeter. The following equations of motion are then used to calculate the vibration of the points. For each point, with initial displacement u, and velocity v the new displacement and velocity are given by the following recurrence relations:

$$u_{n+1} = u_n + v_n \Delta t$$

$$v_{n+1} = v_n + \left[k \nabla^2 u + b (\nabla^2 u)^3 - a v_n \right] \Delta t$$

where $\nabla^2 u$ for each point on an x,y grid is determined by:

$$\nabla^2 u_{x,y} = u_{x-1,y} + u_{x+1,y} + u_{x,y-1} + u_{x,y+1} - 4u_{x,y}$$

and where k is the 'tightness' coefficient, b the non-linear coefficient, and a the damping factor of the drum.

At present, a 16-point drum using 10 separate point calculations around a symmetrical diagonal has been implemented at 44KHz running on a single processor. This means that several independent drums can be calculated on different DSPs with their outputs mixed together before being sent to the analog output.

In order to 'hit' the drum, one (or more) points are displaced from zero, with the displacement directly proportional to the striking force. The motion of each point on the surface is then calculated, and the individual vibrations are summed together to produce the output of a resonating drum. This assumes that the ear is physically equidistant from all points on the surface.

The sound of this drum simulation can be varied by altering its various parameters. The magnitude of displacement is increased as the drum is struck more forcefully, yielding a greater volume. By displacing different points on the grid simulation, the drum can be struck in different areas, such as in the centre, or closer to the rim. The pitch of the drum can be altered by changing k, the degree with which each point affects its neighbours, which is actually a measure of the tightness of the drum head relative to its mass per unit area.

Also, a damping term a is added to simulate losses due to acoustic radiation and to prevent the drum equations from becoming unstable. This allows the user to specify the duration of the drum sound (but is not like specifying an ADSR amplitude envelope). A non-linear term b can also be added, so that larger displacements of the points on the drum will affect the 'tightness' parameter. Thus, as in a real drum, striking the drum with a greater force will produce a distinctive output, rather than a scaled version of a previously recorded vibration. The effect of a positive b is that the pitch goes slightly 'sharp' during the attack, by an amount varying with the force of the 'hit'.

Typically, drum synthesizers use the technique of reading prerecorded samples of actual drum beats. The pitch of the drum is usually changed by reading the sample at different rates. If two successive beats overlap, the output of the initial beat can be either truncated or summed with the output of the following beat. In either case, they do not directly interact. In the simulation, as in a real drum, hitting the drum while it is still resonating means that two beats will interact, due to the non-linear terms.

The simulation also allows the user to change the tightness of the drum while it is still resonating. This produces an effect not unlike a 'talking drum'.

The drum can be configured into any shape by changing the interaction parameters. As an example, it was found that a interesting spectrum of harmonics was produced when the drum was elongated into an oblong shape.

The drum equations are recalculated during each sample period. At the end of the calculation, the DSP checks to see if the user has entered any new parameters. If so, these are read in and used in calculating the next sample.

It should be noted that all of these parameters can be altered by the user in real time, and that these parameters are modeled on actual physical characteristics of real drums, so that non-technical musicians can easily learn how to create effective sounds.

Because all of the parameters mentioned above can be altered in real time, a suitable interface had to be found to generate these values. Parameters are sent via a serial RS-232 port to one of the 6809s, which then translates the data into packets and send them over the P-Bus to the DSP. In order for the simulation to be successful, the interface had to look and feel like a drum to the user.

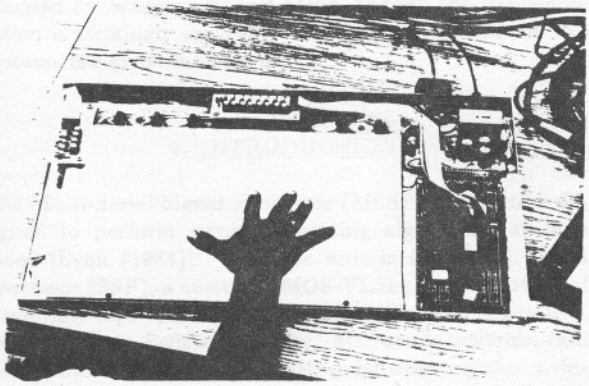


Figure 4. The Touchtablet as a Drum Input

For these reasons, an area-sensitive touch tablet was chosen (see figure 4). This tablet consists of a large (44×32cm.) area which senses the capacitance of a finger or hand touching the surface. The tablet processes this into positional and area information, and sends this data out to an RS-232 port. Therefore, positional data can be mapped directly onto the points of the simulated drum, and area can be mapped onto the magnitude of displacement. Actually, a better interface would have been a pressure sensitive touch tablet, rather than an area sensitive device, but one was not available at the time.

The remaining parameters, resonance, pitch and non-linearity, could have been input through another serial port from sliders or footpedals, but were added as functions of the tablet for simplicity.

Thus the tablet provides an easily adaptable interface for this drum simulation, since striking the tablet surface is much like striking a real drum. Furthermore, since the remaining variables alter realistic parameters, it becomes a simple task for the non-technical user to familiarize himself with this device.

Note that if a suitable high-speed multi-touch tablet were used (a tablet which can detect simultaneous independent contact points) it would be possible for the user to 'mute' the drum by holding down a point with variable pressure, while hitting a second point.

3.2 Other Functions

Other algorithms which have been tested include a rudimentary 5-voice FM synthesizer, and a 3-voice 'harmonizer' for pitch shifting and chorus effects which can be MIDI controlled. These functions require only one TMS32010 processor, so that different programs on different boards can be 'patched' together via software links. Simple multi-DSP algorithms like these have been tested, and more complex configurations are currently being implemented. One of the MC6809s has been converted into a general purpose MIDI I/O device with record and playback features.

The 5-voice synthesizer produces five FM pairs of oscillators read from wavetables which can be downloaded with a variety of waveforms. The pitch and timbre can be controlled by any compatible device, including the touchtablet or a MIDI keyboard. By altering the software, the voices can be linked together in different configurations as desired.

The harmonizer reads in an an input signal, differentiates it and stores these values in a buffer. It is then read and integrated at a different rate in order to shift the pitch relative to the input. Three such voices can be created on one DSP, and controlled by any RS-232 or MIDI input.

Since all of the above functions can be performed within a processing element, they can simply be linked together by routing the output data from one DSP to another via software patches (see figure 5). Thus the interconnecting P-Bus need only

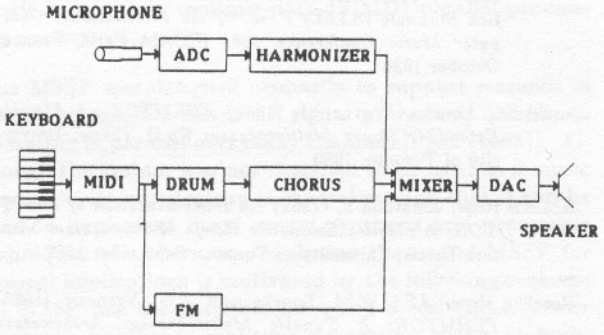


Figure 5. Potential Configuration of Processing Elements

route the final output data of one DSP to the input of another within each sample period.

Note that because of the hierarchical structure of the P-Bus it will be more efficient to group together the processors which require the most intercommunication on one station manager. This will allow two or more independent groups of processors to talk freely amongst themselves without interference from the other groups. However, this means that although any DSP element can be linked with any other by software, some patches will be more efficient than others.

4. Future Work

Future work includes voice synthesis and analysis with LPC techniques, leading up to a high-quality vocoder. Several pitch detection algorithms will also be tested. Because of the computation-intensive nature of these programs, it will be necessary to share these tasks over two or more processing elements. Thus the efficiency of the interconnecting bus will be very crucial for these algorithms, and will be closely investigated.

In conjunction with this, techniques in debugging programs which run on two or more independent processors will be studied. This includes creating a monitor which will allow the user to single step or utilize breakpoints in a program which is shared over a number of processors.

5. Conclusions

This device has proven to be useful as a general-purpose tool. It provides a cost-effective way of experimenting with a wide variety of synthesis techniques, since only new code has to be generated. To aid in this, a fully equipped multi-DSP monitor and cross-assembler have been written to aid in the development of real-time multiprocessor algorithms. Because of the modular nature of the design, the machine can be expanded to meet larger processing demands.

With this prototype, a library of interesting demonstrations have been developed to show the capability of this kind of synthesizer. The modules and P-Bus are undergoing a revision in design which will result in a more compact and portable version of this machine as a potentially commercially available product. Coupled with an easily accessible library of handy functions, and software support for the creation of new algorithms, this machine may find use as a general-purpose synthesis tool and development system.

6. Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grants A4894 and A1753.

7. References

[Del Duca84] Del Duca, L, F. Galante, M. Lupone, G. Nottoli, N. Sani. (1984) Studio Report on the Societa Di Informatica Musicale (S.I.M.) *Proc. of the International Computer Music Conference 1984* IRCAM Paris, France, October 1984

[Loucks80] Loucks, Wayne M. (1980) *FERMTOR: A Flexible Extendible Range Multiprocessor* Ph.D Thesis, University of Toronto, 1980

[Rose82] Rose, Jonathon S. (1982) *An Implementation of FERMTOR: A Flexible Extendible Range Multiprocessor* Masters Thesis, University of Toronto, September 1982

[Rose85] Rose, J.S., W.M. Loucks and Z.G. Vranesic (1985) *FERMTOR: A Tunable Multiprocessor Architecture* IEEE Micro, August 1985

[Stautner84] Stautner, John P. (1984) Musical Recording, Editing, and Production using the Compusonics DSP-2000. *Proc. of the International Computer Music Conference 1984* IRCAM Paris, France, October 1984

[TI83] *TMS-32010 User's Guide*, (1983) Dallas, Texas: Texas Instruments Inc., 1983