MIMIC, A Custom VLSI Parallel Processor for Musical Sound Synthesis

John Wawrzynek*
Thorsten von Eicken*

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

In this paper, we present the architecture of a new generation musical sound synthesis machine. The machine is the first of its kind, combining state of the art ideas from multiprocessing and VLSI design in a system that produces sound in realtime by simulating the physical behavior of musical instruments. The machine will be compact, inexpensive, and easily scalable offering much higher performance per unit board area than would be possible using commercially available processors.

Previous generation machines, such as the Caltech UPEs[1] mainly addressed the arithmetic computation needs of instrument simulation. The insights gained with these systems demonstrate the feasibility of the approach but also clearly indicate that single chip systems cannot offer sufficient performance to simulate ensembles of instruments. To overcome this problem the MIMIC system includes a special purpose network managed by an on-chip unit.

Our machine relies heavily on the use of memory, both within our custom processing chips and in commercial memory chips giving rise to the name Memory Intensive Music Integrated Circuit (MIMIC).

1. Introduction

The goal of the MIMIC project is to develop a parallel computer architecture specifically for the task of generating musical sounds based on physical simulation. The machine will be compact, inexpensive, and easily scaled up or down in size and performance to meet a variety of applications.

A primary goal of the project is to design a system that achieves a much higher performance per board area than would be possible using commercially available processors.

This project continues work on custom architectures for music synthesis previously completed at Caltech[1,2]. Custom VLSI chips containing many simple bit-serial processing elements were built and used to implement instrument models based on lumped approximations to acoustic elements. That work provided promising results and validated the basic approach but was limited in its performance because no provisions for parallelism at the system (multi-chip) level were made. The MIMIC project addresses the issue of inter-chip communication in multi-chip systems and, at the same time, the basic processing function of each chip is extended to allow more general modeling techniques.

The architecture we have designed is a multiple-instruction-stream multiple-data-stream (MIMD) multiprocessor comprising identical pipelined processing chips connected via a network. Each chip contains a 32bit ALU, 64Kbit of RAM, a network interface, and a special external DRAM (dynamic random access memory) interface supporting operations using digital delay lines and table lookup.

*This work was sponsored by DARPA contract number N00039-87-C-0182.

The machine is designed as a multi-processor in such a way that it may be scaled in order to fulfill a range of performance goals and applications by adjusting the total number of processing nodes in the system. A standard workstation will be used as a host to our machine and serve the needs of application development and realtime user interaction. Ease of extensibility is enhanced by each node having signal routing primitives that are independent of the size or the topology of the network.

Because of the modest communication needs of the sound synthesis algorithms, bit-serial connections among processing chips and external components suffice. Bit-serial connections reduce the number of pins on each chip and consequently allow the use of smaller packages, achieving higher board density than would otherwise be possible.

A low-end configuration (see Figure1) will include 27 processing nodes (in a 3 by 3 by 3 node cube), each computing about 20 million operations per second (MOPS) where each operation is a multiply-add (a*b+c) where the entire system is computing about 540 MOPS. Not counted are the other simultaneous activities on the chip which manage signal flow through the network, delay lines, and table lookup operations. In our applications the performance of the system will come very close to these maximum bounds. A system of this size is sufficient for performing a realistic simulation of a concert grand piano or a small musical ensemble. A larger system of about 100 nodes on a board could implement an "orchestra in a box".

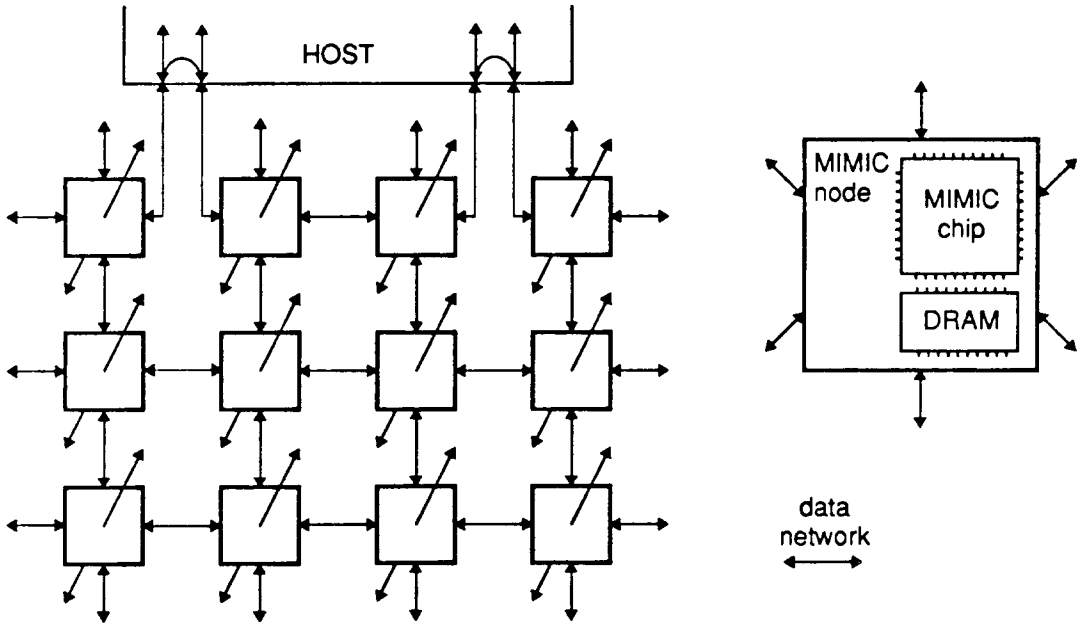

FIGURE 1: MIMIC system level architecture. Processing nodes are typically arranged in a multidimensional mesh. Each processing node comprises a MIMIC chip and a commercial DRAM (dynamic random access memory) chip in approx. 1.5in^2. Nodes connect to their nearest neighbors through bidirectional links for transmitting signal data and control information from the host injected into the network at multiple points. All connections from the host and between nodes are bit-serial.

2. Machine Requirements

Exact system requirements are difficult to pin down. Work on instrument modeling has only begun and is slow because of lack of suitable hardware, we will not know exactly how the machine will be used until it is built. We do know, however, general requirements based on earlier machines and on several experimental musical instrument models.

2.1. Musical Sound Synthesis Background

The processing node is designed to support a few basic primitives that form the computational elements of a particular class of sound synthesis algorithms. Figure 2 shows the set of computational primitives supported by our machine. These primitives form the lowest-level user interface to the machine. Software automatically maps a computation graph represented with these primitives to instructions and routing information for the MIMIC machine.

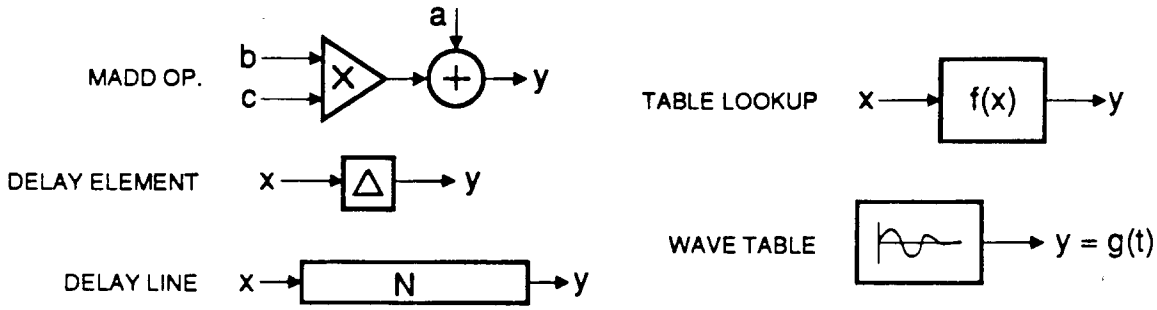

FIGURE 2: MIMIC user level computational primitives. Instrument models are made by building graphs of these primitives. Inputs may come from the output of other primitives or from the host computer.

The physical modeling algorithms have found general application in the synthesis of a wide range of musical instruments including the human voice. They are fairly efficient and produce very realistic sounds[3]. In this technique simple computational analogs to the components of musical instruments are implemented and the interaction of these components is computed in detail. The simple resonator circuit in Figure 3 shows a simple interconnection of primitive elements as may be found as part of a complete instrument model.

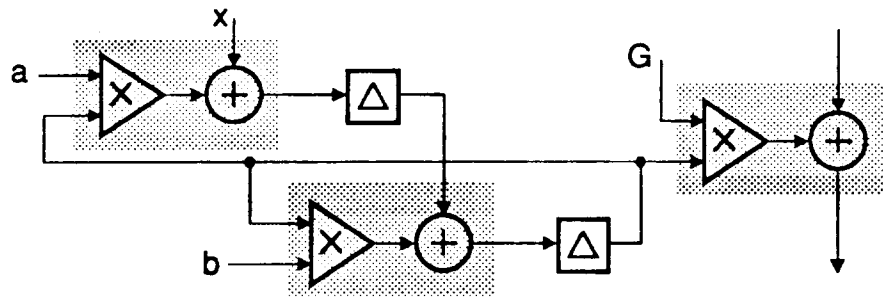

FIGURE 3: Digital resonator circuit. This digital resonator configuration is typical of the circuits used in musical instrument models. The coefficients a, b and G are controlled by the host computer and adjust center frequency, damping, and amplitude respectively. The input x comes from the output of circuits modeling other instrument parts.

This technique is successful because the complex and rich behavior of musical instruments, and thus the complexity of their sounds, is the result of the interaction among many simple elements. Each element, for instance, may be one string on a guitar, the reed of a clarinet, or the bridge of a violin. These elements when taken individually are fairly simple mechanical systems. In fact, in some instances components can be approximated as lossless waveguides and implemented as simple bidirectional delay lines[4]. Other components such as nonlinear elements can be implemented using polynomial evaluation or, for more complex functions, table lookup techniques. Still other physical components may be implemented as standard digital filters.

An experimental piano model has been devised and is shown schematically in Figure 4. It provides a good target task for a small MIMIC machine and is representative of a wide range of musical instrument modeling tasks. It works by modeling the strings, bridge, and sound board of a piano and their interactions. The behavior of all pieces is computed simultaneously to capture the complex coupling of the elements of real pianos.

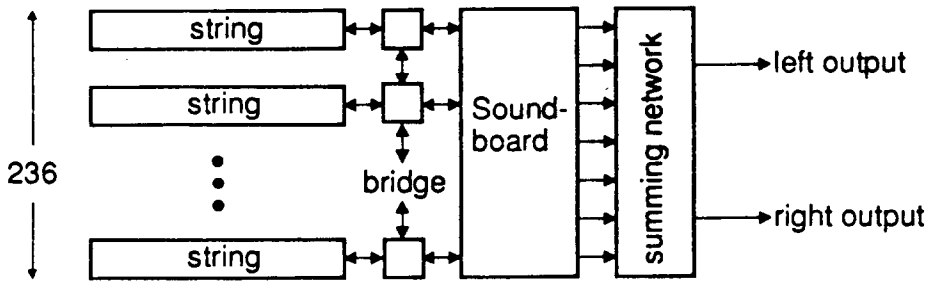

FIGURE 4: Piano Model. This model has been devised to help guide our machine design decisions. It works by modeling the pieces of a piano (strings, bridge, sound-board) and their interactions. The summing network is used to generate stereo sound image.

2.2. Model of Operation

Because the machine will simultaneously generate the sound of many different musical voices in realtime, we provide each processing node with an independent control program so that each node works on different sets of instruments or different parts of a large complex instrument. The program for each node is loaded by the host computer prior to execution and is stored in each node. During execution this program is repeated once every sample time. The current implementation is designed to run at a typical sample rate of 50K samples per second. Our sound synthesis algorithms contain no conditional execution; therefore, the processing nodes contain no hardware for conditionals. The same set of instructions is repeated until changed by the host computer. Program changes, however, are intended to occur on a very slow time scale (e.g., when a new instrument model is desired). Instantaneous changes controlling the "playing of the instrument" are implemented by the host by sending packets that change coefficient values within nodes.

2.3. Design Flow

The computation power per node, memory size per node, delay/table operations, the network size, and inter-node communication needs were estimated by examining the requirements for simulating a small ensemble or a concert grand piano as shown in Figure 4.

We examine the case of a small ensemble: On the average, each instrument requires 4 voices (for example 4 strings). To compute the state of a voice, around 40 MADDs (multiply-adds) have to be executed per sample period. Therefore an ensemble of 30 instruments will require on the order of 4800 MADDs per sample period.

The estimation for a grand-piano is very similar: Each one of the 236 strings is modeled as an individual voice for a total, with the inclusion of the sound-board simulation, of almost 10000 MADDs per sample period.

Each of the voices usually requires one delay line or table lookup operation per sample period.

With the current technology available from MOSIS we expect to build a 20Mhz multiply-add unit. Given a 50Khz sampling rate each processing node can perform 400 MADDs per sample period. The above simulations will therefore require the arithmetic power of 15 to 30 processing nodes.

Calculation of memory bank sizes is based on earlier experience with instrument modeling. Each of the 400 MADD operations per sample period requires on average one coefficient from the host, and one state variable. An additional 200 or so words of storage are used to double buffer coefficients and buffer input/output operations resulting in a total on chip store of 1K words.

If we now distribute the computation across multiple processing nodes, we have to estimate the communication needs between the nodes. The 10 voices which can be grouped into a single node generate about 10 inter-node messages.

To control the simulation the host must periodically update the instrument model coefficients. The required communication performance can be split into two categories: the sustained bandwidth and the instantaneous bandwidth. The estimates for the performance were made by examining worst case conditions.

We assume that each MIMIC node can support 10 musical voices and that each voice is controlled by 40 coefficients supplied by the host for each musical note. Now if each voice can play 10 notes per second, the total coefficient update rate is 4000 updates per second per node. With a sample rate of 50000 samples per second and a expected communication clock rate of 20MHz, we arrive at an average update rate of only 1 update every 12 or 13 sample periods.

More constraining than total average update bandwidth is instantaneous update bandwidth. This bandwidth is needed to assure that when an update is made in the system that a tolerable delay is maintained from the time the change is sent until the effect is heard. For interactive systems a 10ms delay is usually not noticeable. Longer delays are acceptable especially if the delay is constant from note to note. For a conservative measure we assume in our system a situation were we wish to change every voice and allow at most a 10ms delay. Changing 400 coefficients per node in 10ms requires the host to send approximately 0.8 coefficients per sample period to each node. In some cases the update coefficients can be precomputed and sent to each node and stored locally until they are needed. Then a single control word can be sent from the host to trigger their activation. Each MIMIC node contains double buffering on coefficient registers to help implement this scheme.

2.4. Summary of Machine Requirements

Table 1 summarizes our expected system requirements and capabilities for a typical small configuration. The arithmetic requirements are relatively high and communication and memory requirements low compared to a more conventional multiprocessor configuration.

TABLE 1: Summary of Machine Requirement Estimates.

Operation	per node per sample	per node per second
MADD (multiply-add)	400	20M
Inter-node 32bit packets	10	500K
Delay-line/table	30	1.5M
Host updates	1	50K

The estimates are based on the simulation of a small instrument ensemble or a grand piano. The processor clock rate is assumed to be 20Mhz with a sample period of 50Khz.

3. The MIMIC Network

A typical configuration of MIMIC chips is shown in figure 1. The unit of communication between MIMIC chips is a single 32-bit word, usually representing a sound sample value, and a packet is simply a word.

Because the control flow of MIMIC programs is linear (i.e., they have no conditional branches) the communication pattern for each program is predetermined. The routing of all packets can be determined at compile time and loaded (in the form of a program) into the MIMIC chips. No routing decisions are necessary at execution time and all problems relating to dynamic detection and correction of deadlock and collisions are avoided.

The fact that the communication pattern is static and routing is precomputed allows a simple solution to keep routing latency to a minimum. Because no packet headers are needed the packet can routed through each node with only a one bit-time delay.

Communication cost is the price paid for parallel computation. Meshes and other topologies for directly connecting processing nodes, though simple to build, never take full advantage of the raw communication bandwidth provided by the pins on the nodes. Messages sent into the network, while on route to their destination consume resources and block other messages. It is plain to see that network efficiency is inversely proportional to the average path length in the network and that the average path length is a function of the network topology.

During the design of our machine we were interested in knowing what fraction of raw communication bandwidth provided by the hardware could be effectively used in our algorithms. We studied primarily three-dimensional toroidal meshes. The performance of several typical system configurations are displayed in table 2. For these simulations we generated random pairs of nodes as source and destinations for messages and assigned random times within the sample period. The goal of the simulation was to find the maximum number of messages that can be routed within one sample period.

Approximately 70% of the raw channel bandwidth is lost to network topology. We experimented with several routing algorithms all taking advantage of the fact that all routing patterns are known at compile time. The simplest routing algorithm uses buffering, performs well, and is easy to support in hardware. With this algorithm about 60% of the channels are kept busy. This means that at any time an average of 40% of the channels are idle. Better channel utilization is possible with more elaborate routing algorithms. This algorithm was chosen for its simplicity and fairly good performance. The product of network efficiency and channel utilization gives us the effective pin-efficiency. This efficiency factor reveals how much extra communication bandwidth we need to build into the pins of our nodes to achieve a given level of actual network throughput. For systems in our simulations this factor is between 3 and 6. One important result to note in table 2 is that even the least efficient network (125 nodes) is able to transmit 12 messages on average between pairs of nodes.

For our simulations we used a random spatial distribution of messages. In practice our algorithms tend to exhibit locality in the communication patterns and the average path length is actually less than our simulations indicate. The simulation results are therefore pessimistic.

TABLE 2: Routing Simulation Results.

# nodes	# messages routed	network efficiency	channel util.	effective chan util.	#message per node
27	595	.48	.62	.30	22
64	948	.33	.61	.20	15
125	1570	.27	.61	.16	12

3.1. Host Communication

Some of the communication bandwidth lost to idle channels in our system is captured for use in sending update messages from the host processor. Therefore, two separate types of communication are supported by the network: 1) inter-node communications implementing data links in the computation graph, and 2) host-node communication implementing update links. Data links are

characterized at compile time, but the update links are only partially known. The maximum bandwidth needed from the host to each node is known but the actual time when the information will be sent is known only at run time; for it is the result of realtime user interaction with the machine. At compile-time, after the data links are established through the network, daisy-chains are traced out through the remaining unused channels in the network. Then at runtime the host can reach any node along a daisy-chain.

This network arrangement provides a graceful way to trade inter-node data communication bandwidth for host control bandwidth. This property matches well our observation that in our applications models requiring high communication between processing nodes have modest control bandwidth needs and, conversely, losely coupled computations require high control bandwidth from the host.

4. MIMIC Node

4.1. Machine Structure

The requirements of the algorithms and the communications within the machine result in the following node structure. Each node has three major functional units: an arithmetic unit, a memory controller, and a communication switch. The arithmetic unit is similar to those typically found in standard signal processing engines. Its main function includes multiplication, addition, and storing of results in large on-chip register banks. The memory controller performs the address calculations involved in implementing delay lines and performing table lookups. The delay line and table data are stored in standard memory chips. The memory controller allows the relatively expensive arithmetic unit to dedicate its computation to signals. The third major unit, the communication switch, handles the flow of information in, out, and through the node. This switch is much simpler than ones used in general purpose multi-computer architectures. In our case the communication patterns within the network are known before the computation proceeds. Therefore it is possible to "compile" the communication instructions and store them within each node. Each node does not need to adjust its routing function dynamically. We call this fixed routing approach "compiled routing". In addition to routing acoustic signals within the network the node must manage the flow of information from the controlling host computer.

Each processing node comprises a custom VLSI MIMIC chip and a commercial DRAM chip. The internal structure of a MIMIC chip is shown in figure 2. Internal units are connected via two internal 32-bit busses, the IBUS for receiving input from off-chip and the OBUS for sending samples off-chip. The structure and functions of the individual units are described in subsequent sections of this report. The ALU/register unit preforms arithmetic operations on signals. Operands and results are stored in memory arrays labeled MEM1 through MEM4. The network unit provides a synchronous interface to six other MIMIC nodes. Signals are sent and received bit-serially. The network unit allows each MIMIC chip to be a source of data, a data sink, or a intermediate node in passing data within the network. The table unit is a special unit for implementing delay lines and table lookup operations. This unit provides all the signals necessary for controlling the commercial DRAM chip. The host-interface unit contains the control store and the mechanism for receiving setup and control information from the host computer.

TABLE 3: Chip Pins

DRAM control	16
Network ports	12
Clocks/reset	4
Power	4
Total	36

A design goal for our system was to keep the number of pins on our custom VLSI chip to a minimum. This way we can employ a small package, thus maximizing the density of nodes on the circuit board. The communication bandwidth requirements for the music synthesis algorithms indicate that bit-serial communication among chips and with the host computer is sufficient. The external pins requirements of the MIMIC chip are shown in Table 3.

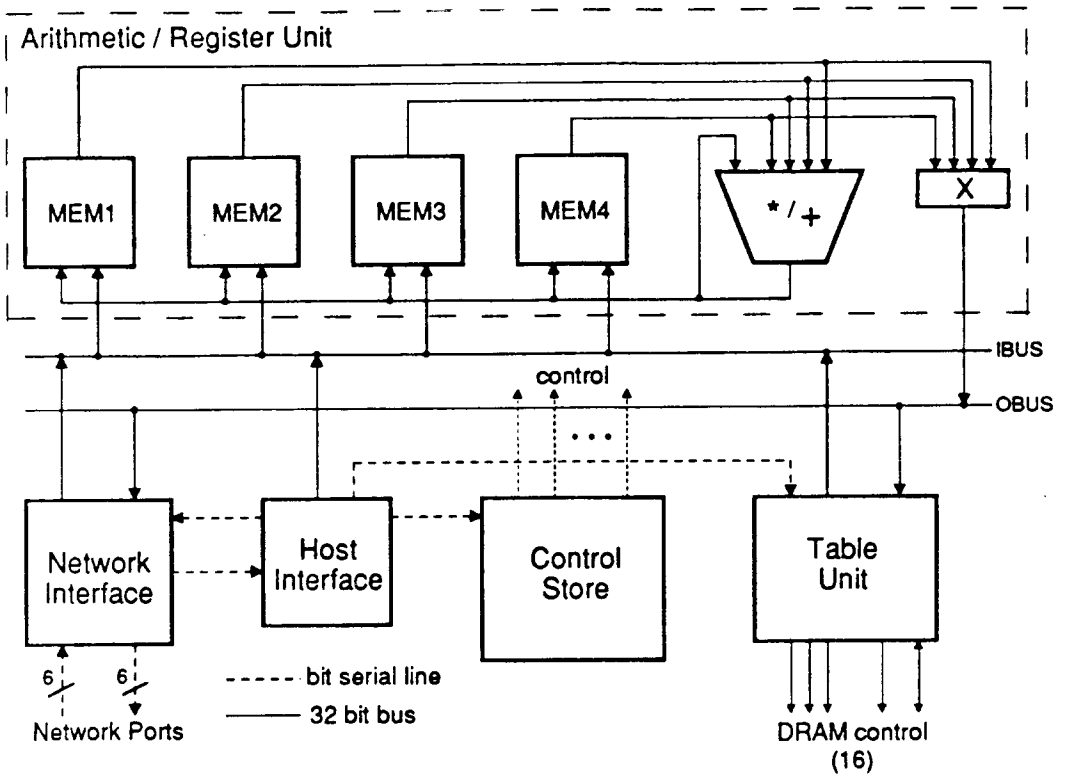

FIGURE 5: MIMIC Node Structure.

4.2. ALU/Register Unit

The ALU/Register (AR) unit performs arithmetic operations on signals as needed by sound synthesis algorithms. The AR unit consists primarily of a 32-bit by 32-bit fixed point parallel multiply/add unit (ALU) and four memory banks as shown in Figure 5. The memory banks are used to store intermediate results, state variables, and coefficients supplied by the host computer. Because all arithmetic operations are memory-to-memory operations, the memory banks can be viewed as large register files.

The AR unit is designed to efficiently implement the multiply/add operation. This operation is primitive to sound synthesis algorithms as well as many other signal processing tasks. Each operation fetches three operands and produces one result. To provide the necessary data bandwidth to the multiply/add unit, four memory accesses are required on each processing cycle.

A common way to achieve concurrent memory accesses is with multi-ported memory arrays. Multi-porting has the disadvantage that the chip area consumed by a the memory is significantly more that the single-ported version (almost 2 times for dual-ported, more than 3 times for quad-ported). We have chosen to implement four separate memory banks to achieve four memory accesses per cycle. At compile time the memory access patterns are analyzed and program variables and coefficients are assigned to memory banks in such a way as to provide conflict free access to four memory locations per processing cycle. Of course, it is possible to write programs with variable use patterns where memory accesses conflict. Memory access conflicts result in sequential memory access resulting in lower system performance. We have found that with our sound synthesis algorithms very few conflicts occur (a few percent).

The number of words per memory bank is primarily governed by the available chip area. The current implementation of the MIMIC chip has four 256 word memory banks.

The current MIMIC node implementation uses a 256K by 4-bit DRAM yielding a total sample storage of 32K samples per MIMIC node. Because each sample is 32 bits long, each delay line or table operation requires eleven clock cycles allowing up to 36 delay line or table operations per sample period.

5. Implementation

Figure 6 shows the chip floorplan for the MIMIC processor node. The layout is targeted for 1.2μm CMOS technology. Chip dimensions are 9.0mm by 8.4mm. We expect power consumption to be a modest 250mW per chip.

The status of the MIMIC project is as follows: we have written both a high and low level simulator of the MIMIC chip and tested the design using several application programs based on physical simulation of musical instruments. Software has been written to generate program-variable to memory bank assignments and scheduling of arithmetic and input/output instructions. Layout of critical sections has been completed for floorplanning and size and performance estimates. We are currently in the process of fine-tuning the low-level design of the chip and generating the complete chip layout.

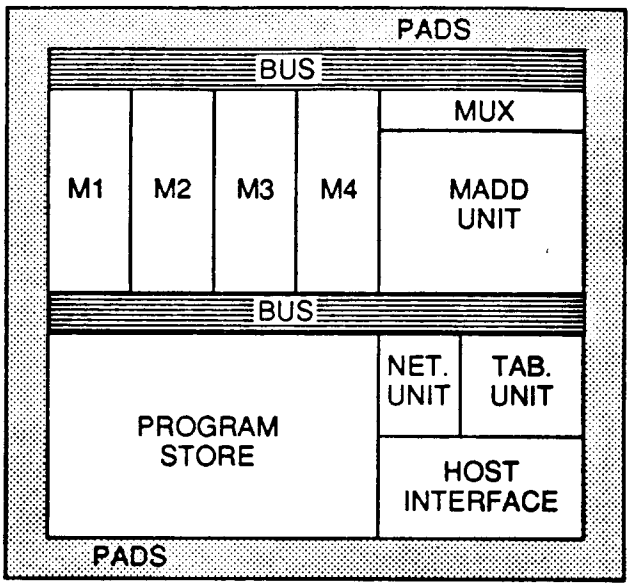

FIGURE 6: MIMIC Processor Node Floorplan.

6. Summary

In this paper we have presented the architecture of a novel machine, called MIMIC, designed specifically for the task of producing sound in realtime by simulating the physical behavior of musical instruments. Although the machine is special purpose many of the ideas have direct application in other realtime signal processing tasks.

The machine achieves high performance per unit board area by using custom VLSI processing chips in conjunction with commercial DRAM chips. The high system performance required to simulate complex instruments or small ensembles of instruments precludes single processor solutions. The MIMIC architecture is designed as a MIMD array of processing nodes interconnected by a special network and controlled by a host computer.

We have shown, that this network can be implemented using low pin-count bit-serial lines. It is scaleable from a few nodes to over 100 nodes. The routing can be performed statically at compile time greatly reducing the complexity of the on-chip network interface unit. Moreover, the static schedule can guarantee the realtime performance.

We have shown that by concentrating on the primitive operations found in our application, musical sound synthesis, we are able to build a flexible processor node which provides performance comparable to latest generation commercial DSP processors.

The performance obtained per unit board area (20MOPS in approx. $1.5in^2$) cannot be matched by commercially available DSP processors. Those processors use large, high pin-count packages,

consume over 1 Watt of power, and are not designed to work in a multi-processor configuration.

Acknowledgments

We would like to thank the members of CS292I (Fall 1988) for their enthusiastic participation in class discussions and their work on the MIMIC project. In alphabetical order they are: Gino Cheng, Paul de Dood, Mark Kwong, Angie Lyons, Hai Nguyen, Chris Perleberg, and Rob Vaterlaus. Thanks to Neil Getz and Fred Obermeier for carefully reviewing drafts of this report.

References

1. John C. Wawrzynek, "VLSI Concurrent Computation for Music Synthesis," No. 5247:TR:87., Doctoral Thesis, Depart. of Computer Science, California Institute of Technology, Pasadena, CA, 1987.
2. John C. Wawrzynek and Carver A. Mead, "A VLSI Architecture for Sound Synthesis," *VLSI Signal Processing*, pp. 277-297, Denyer and Renshaw, Addison Wesley, Reading, MA, 1985.
3. M. E. McIntyre, R. T. Schumacher, and J. Woodhouse, "On the Oscillations of Musical Instruments," *Journal Acoustic Society America*, vol. 74(5), 1983.
4. J. O. Smith, "Efficient Simulation of the Reed-Bore and Bow-string Mechanisms," *Proc. Intl. Computer Music Conf.*, Computer Music Association, The Hague, The Netherlands, 1986.