Programmable MIDI Instrument controller Design: The No Strings Attached Hammer dulcimer

By Randolph C. Marchany

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Electrical Engineering

Approved:
Dr. Joseph G. Tront, Chair
Dr. James R. Sochinski
Dr. Charles E. Nunnally

September 8, 1992
Blacksburg, Virginia

Programmable MIDI Instrument Controller Design:

The No Strings Attached Hammer Dulcimer

by

Randolph C. Marchany

Dr. Joseph G. Tront, Chairman

Bradley Department of Electrical Engineering

(ABSTRACT)

Real-time digital music system design often involves the translation of formal music notation or human gestures by some input device to Musical Instrument Digital Interface (MIDI) commands which are then transmitted to an electronic music synthesizer. This thesis describes the design and implementation of a microprocessor controlled input device that maps analog signals to MIDI commands and transmits them to a digital synthesizer in real-time. The controller emulates a traditional acoustic folk instrument known as a hammer dulcimer.

The hammer dulcimer is the forerunner of the keyboard family of instruments and incorporates features found in percussive and keyboard instruments. As with any acoustic instrument, its tone is a composite of several partial tones. The controller, in emulation mode, generates the fundamental tone and optionally outputs the partial tones with the just noticeable difference (JND) tolerances described in psychoacoustic research. This feature allows the designer to experiment with the timbre of the fundamental tone. The controller interface succeeds in capturing the gestural movements and translating these events to MIDI commands. It also provides features such as on-demand retuning which allows the musician to play in any tonal center without changing hand positions. Selected MIDI features such as pitch bend, program change and sustain are implemented by the controller. The prototype instrument yields a two octave range from an eight by eight inch sensor grid. Additional grids can be added to increase the range of the instrument.

Acknowledgements

There are not enough words to thank Dr. Joseph Trent for his patience, support and help in this endeavor. I am grateful to him and to Drs. Nunnally and Sochinski for their friendship and help. This work is dedicated to the memory of Dr. George W. Gorsline.

Table of Contents

List of Illustrations

List of Tables

1.0 Introduction

1.1 History of Computer Driven Instrument Controllers

Modern synthesizer design is based on two components: the human gestural interface known as a controller and the digital and analog components that generate sound wave patterns. A typical synthesizer may consist of both cache memory and ROM, a circular queue, a calculator, an input device or controller such as a keyboard and a driver that produces the synthesized output [Parks, 45]. The sound producing component can be either digital or analog and consists of items such as oscillators, filters and modulators. These modules can be controlled by a common parameter voltage and these signals are usually input via the synthesizer's input device. Alles describes in his paper the general process of digital sound synthesis [Alles, 42].

The introduction of the Musical Instrument Digital Interface (MIDI) protocol in 1983 greatly facilitated the use of computers in musical synthesizer production and development. The control units that scan the input interface and produce the actual musical notes can be separated from the actual sound generating portion of a synthesizer. This separation of function allows the instrument

designer to create a wide variety of gestural interfaces. MIDI software is divided into three major categories:

1. sequencer software,

2. librarian software,

3. editor software.

Sequencer software functions as a "word" processor where the "word" is not a traditional language entity but, rather, a time-stamped musical event. Once entered, the sequence of musical events can be altered by a set of commands similar in function to traditional word processing programs. A "recorded" piece can be transposed to any key, have its tempo changed, be repeated often, or have a musical score printed out on a printer. All of these functions can be performed by sequencer commands. This ability to manipulate a musical piece greatly increases the productivity of the composer.

Librarian software emulates the musician selecting, assigning and mapping particular synthesizer sounds. This type of software sends voice control information to the synthesizer on demand by the musician. Most voice control information that can be input to a synthesizer via its control panel can also be input via MIDI commands.

Editor software manages synthesizer components and functions in the same manner that the operating system for a computer manages the computer's resources. The control processor of the synthesizer continuously scans its input switches and "keys" and assigns hardware oscillators to the keys pressed or changes control parameters to these oscillators. Examples of these functions are 1) changing the waveform parameters for a particular voicing, 2) modifying general control parameters per user input, 3) mapping piano keys to specific notes, 4) playing a musical note.

The newest area of research being done is the design of the human gestural interface. Most traditional synthesizers use a piano-like keyboard as the main playing interface. Only in recent years have manufacturers been able to contruct keyboards that "feel" like a real piano. The ability to control this "feel" greatly enhances the expressiveness of a musician and is the subject of a variety of papers [McKay, et al., 20][Moog, 21][Steele, 22][Swift, 23][Federokow, et al., 24]. Factors that influence musical expressiveness and therefore, controller design, include touch, attack, delay and aftertouch sensitivities. In the mid 1980s, Roland Corporation introduced **keyboard controllers** that did not contain the analog component of a synthesizer but did provide wooden, weighted keys that closely approximated the "feel" of a real piano. A microcontroller converted the analog information to MIDI musical event information which was then transmitted to a synthesizer. The design of "touch-sensitive" keyboards has attracted the attention of musicians and engineers over the years [MacKay, et al.,20][Moog, 21][Steele,22].

Digital technology advances have allowed instrument designers to construct musical interfaces emulating other families of musical instruments such as the string, wind, and percussion families. These advances in microcomputer technology have enabled designers to take advantage of faster scan times to better capture the expressive gestures of the musician. Numerous papers have investigated the construction of instrument controllers based on wind instruments [Yunik, et.al, 18][Curry, 8], and percussion instruments [Matthews, 7][Snow,4][Smith,12].

Other researchers have built non-traditional instrument controllers such as the Video Harp [Rubine, et.al, 6] and the HANDS [Waisvecz, 9], the Rolky [Johnstone, 11], the Bio-Muse[Knapp, et. al, 19], the Buchla Thunder [Rich, 39]. They used more modern computer technology to build instrument controllers outside the mainstream of "traditional" electronic instrument design. These units are well suited to capture, simulate and translate a musician's playing gestures into meaningful input to the sound generating modules of an electronic musical instrument. Some controllers did not, however, allow the musician to play notes on a real-time basis, rather, they allowed the musician to play a predefined set of notes.

1.2 Project Justification

The separation of the gestural interface from the sound generating module creates the potential for designing instruments based on families other than the keyboard family. The design of keyboard style controllers has been well documented in literature for the past fifteen years. The design of non-keyboard controller interfaces is a relatively new area of research. Microprocessor evolution allows unique gestural controller interfaces to be designed and tested. This flexibility gives the instrument designer the ability to build an instrument that is limited only "by those imposed by hardware and the dexterity of the human hand" [Aikin, 40].

This project emulates an instrument that requires the "touch" of percussion and string instruments. The No Strings Attached Dulcimer emulates a traditional acoustic instrument, the hammer dulcimer. The hammer dulcimer is the forerunner of modern keyboard instruments such as the piano, harpsichord, clavecin and clavichord. The instrument is played by striking the strings with sticks or plucking the strings with the fingers. By using electronic sensors to emulate the strings of the instrument and a microprocessor to interpret the hand gestures, the author has designed a unique and powerful percussive MIDI controller can be built economically using off-the-shelf components.

The No Strings Attached Hammer Dulcimer is a percussive controller interface that emulates a hammer dulcimer. This controller senses the following information:

1. Note information based on an x-y grid,

2. Touch sensitivity information on the note (how hard it was hit),

3. Sustain information on the note,

4. Control information such as voice switching and pitch bending.

The controller converts this analog input information to valid MIDI control messages and transmits them to a digital synthesizer in real-time. The response time of the controller ranges from 1 to 20 milliseconds. Just noticeable difference (JNDs) parameters for humans distinguishing separate events are on the order of 40 to 60 milliseconds [Licklider, 52; Stewart, 13; Wurthrich, 49]. A musical tempo of 240 BPM (beats per minute) generates one beat every 250 milliseconds. Musical thirtysecond notes (8 notes per beat) would generate an event every 30 milliseconds ((1 second / 4 beats)/ 8 notes per beat)). This tempo is at the upper end of musical playing, thus, the controller is fast enough to simulate simultaneous playing of notes.

This project has improved on the basic design of the percussion style controllers and presents a highly ecomomical method of constructing a fairly powerful MIDI instrument controller. The No Strings Dulcimer uses piezo sensors known as Kynar [Cantrell, 38] arranged in a 4 by 4 grid. This single grid of the prototype instrument provides enough information to generate musical notes in a two octave range. Additional grids can be added to increase the range of the instrument. Synthesizer control information can be sent by activating additional sensor strips on the controller.

1.3 Summary of Chapters

Chapter 2 describes the Musical Instrument Digital Interface (MIDI) specifications in detail. It describes both the hardware and software characteristics and shows some of the weaknesses in the protocol.

Chapter 3 discusses MIDI instrument controller design parameters such as ergonomics, cybernetic issues, psychoacoutic limitations and bounds in controller responsiveness. It briefly describes some of the non-traditional controllers that have been developed in the past decade.

Chapter 4 provides a description of the acoustic hammer dulcimer. It gives a detailed explanation of the timbre structure generated by the hammer dulcimer. It provides a table listing the reverberation characteristics used by hammer dulcimer luthiers to determine the tone of an instrument.

Chapter 5 detailed the hardware used to build the controller. A description of the Intel 8751 microprocessor, Kynar sensor strip circuitry, A/D subsystem circuitry and timing diagrams of the various components is discussed in this chapter.

Chapter 6 describes how the controller software works. Flow diagrams of the individual routines are provided. The design goals and tradeoff issues are discussed in this chapter.

Chapter 7 summarizes the major points of this thesis. It provides some suggestions for future enhancements to the controller as well as possible areas of further research in the controller design area.

2.0 Musical Instrument Digital Interface (MIDI) Protocol Description

2.1 Overview

The Musical Instrument Digital Interface (MIDI) is a computer communications protocol for digital music devices. It evolved from a proposal called the Universal Synthesizer Interface in 1981. In 1982, representatives from Roland, Yamaha, Kawai and Korg started developing the specifications for this new protocol. In 1983, the first successful demonstration of the MIDI standard was made at a national trade show. [Anderton, 57]. The protocol sought to allow hardware interconnectivity and interfacing to computer systems. Software for these computer systems can provide for multi-track recording, playback, musical sequence editing, score display and printing. The MIDI protocol is an **event** based protocol where human gestures dictate a sequence of events to be executed.

The main concept of MIDI is similar to one that describes a player piano. What is encoded is not the actual sound of an instrument but what was played on the instrument. MIDI provides the

added capability of recording and playing back simultaneously. Some of the events that MIDI can handle are:

1. pressing a key,

2. releasing a key,

3. measuring key velocity (how hard it was hit)

4. measuring aftertouch pressure

5. pitch bending, modulation and sustain,

6. synthesizer program (voice) switching,

7. real-time synchronization,

8. sample and system executive dumping.

Key numbers are used to represent each key on a keyboard. MIDI assumes a virtual keyboard that has 128 keys. Standard piano keyboards are 88 keys in length. The MIDI key number to piano key mapping is shown in Figure 1.

The MIDI 1.0 specification was adapted from serial transmission technology that was developed for computer terminals [Loy, 27; International MIDI Assoc., 31]. It describes a hardware interconnection as well as a software protocol description.

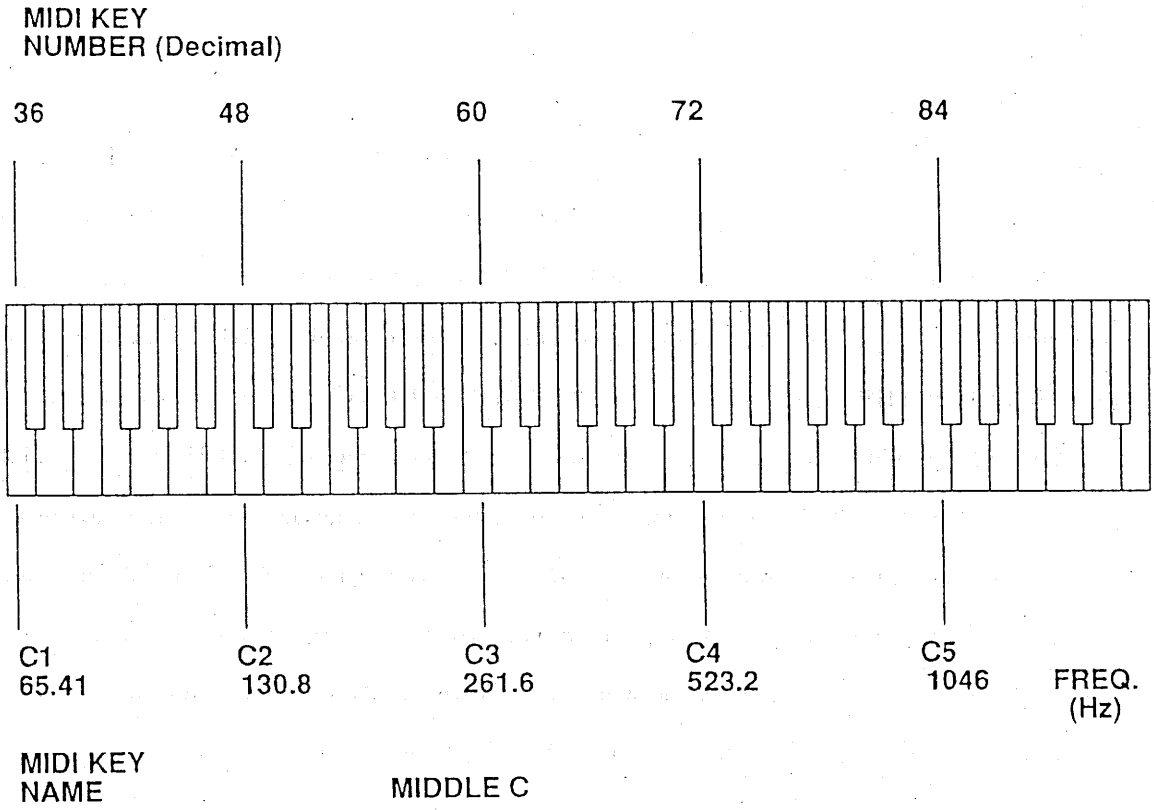


Figure 1. MIDI note number to musical note mapping

2.2 MIDI Hardware Specification

The hardware interconnection shown in Figure 2 is a point-to-point opto-isolated 5 ma current loop that uses a 180 degree 5-pin DIN connector instead of the standard RS-232 connector. Three of the pins are used with the others not connected. The cable consists of a shielded twisted pair that is grounded at the source end. Individual MIDI devices must contain a MIDI input and output jack with a MIDI THRU jack as an option. The MIDI THRU jack passes a buffered electrical copy of the input signal. The transmission baud rate is set at an aggregate rate of 31.25K (31,250 bits per second). Serial MIDI data is transmitted as 10-bit frames containing 1 start bit, 8 data bits and 1 stop bit. The aggregate byte transmission time is 320 μsec [Gualteri, 26]. Most MIDI commands are 3 bytes in length giving a 1 msec per command worst-case transmission time. This gives MIDI the ability to play 500 notes a second worst case allowing a MIDI device to transmit multiple note chord patterns in sequence faster than the JND for temporal sound detection.

Figure 3 show some of the various MIDI interconnection schemes. These are:

1. Unidirectional - master talks to slave.

2. Bidirectional - masters drive each other as slaves.

3. Ring - bidirectional connection to three or more devices.

4. Daisy Chain - one master driving several slaves using the MIDI THRU ports.

5. Star - one master with several uni or bidirectional links.

These connection schemes do not allow for two-way communication as there are no provisions in the protocol for a handshake style of communication.

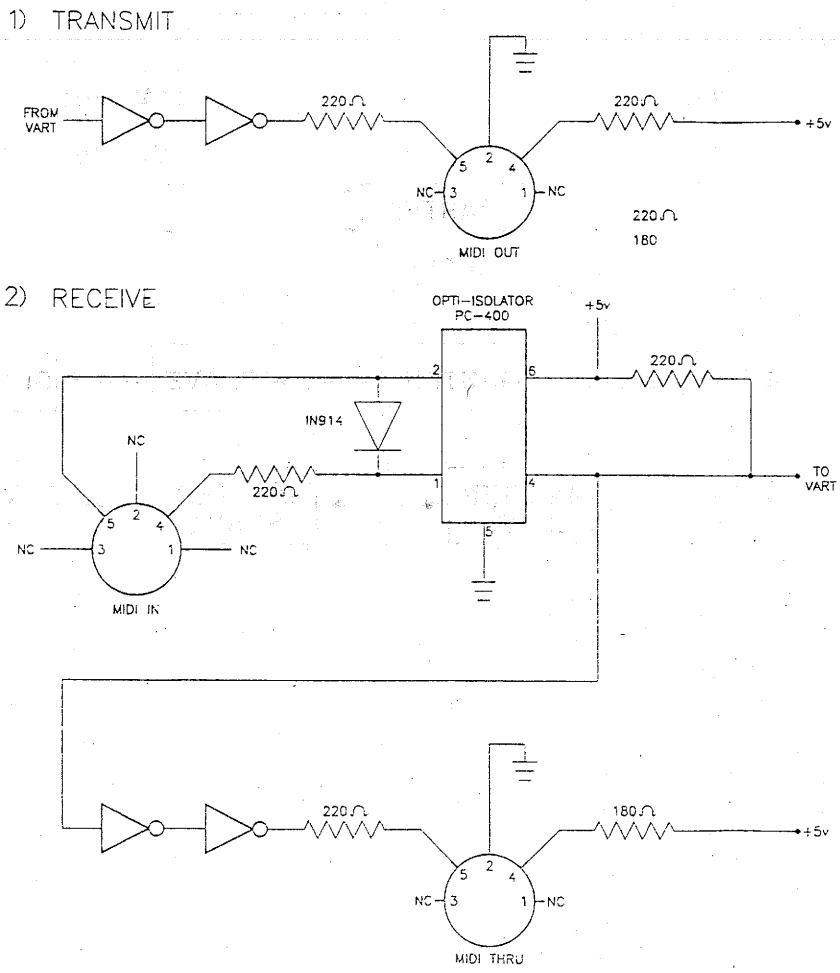


Figure 2. Circuit Diagram for MIDI transmit and receive

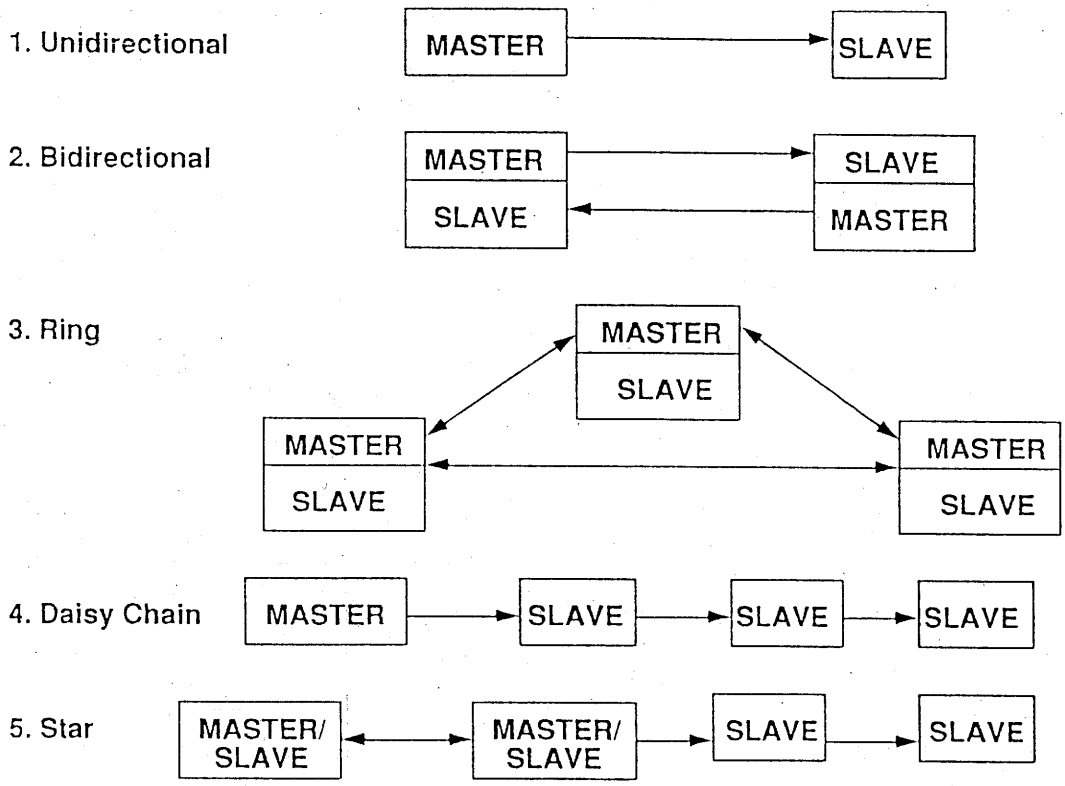


Figure 3. MIDI Interconnection Schemes

2.3 MIDI Software Specification

The MIDI protocol defines two data types. **Status** data types are 8-bit bytes with the most significant bit (MSB) set to 1. They identify the purpose of the data bytes that follow. Status bytes instruct the receiving unit to adopt the status defined in the message immediately. **Data** bytes contain such information as note numbers, key velocity, and channel mode. The data byte has its MSB set to 0. MIDI supports the **running status** mode of data tranmission. This mode shortens tranmission time by sending one status byte for a series of data bytes instead of 1 status byte per command. Once a status byte is sent, its set of operating parameters (channel number and command id) stay in effect until another status byte is sent. **Channels** provide for control of multiple synthesizers or voices in a MIDI network. Channel number recognition is done in a manner similar to the ethernet communications protocol description where all MIDI devices see every message but only respond to the ones addressed to them. Unfortunately, the protocol designers used the term **channel** in two different contexts. The term, channel, used in the first context, refers to a synthesizer being configured to receive data on a particular channel. Once configured in this mode, it will play only those notes that are assigned to that channel. It will pass all other notes to the next device in line. Another name for the term used in this context is **synthesizer channel**. The only provision for dynamically changing channel assignments is by sending the MIDI System Exclusive message. Of course, the musician may physically change the synthesizer setting. MIDI allows for 16 separate channels for music data transmission. Every musical event must be transmitted on one of these channels.

The same term, used in the second context, describes one of the two types of MIDI messages: **channel** and **system**. The two channel message types are **voice** and **mode**. Voice channel messages control the instrument voice. Mode channel messages define the instrument's response to Voice Channel messages[Int'l MIDI Association, 31].

The most common MIDI command is the **Voice Channel Message**. This command contains the data needed to turn an individual musical note on or off. It also provides velocity information which

is used to control the note volume. This command is three bytes in length. The first byte is the status byte which contains the command id and channel number. The second byte contains the note number and the third byte contains the key velocity data. A **note off** command or **note on** command with a velocity of 0 is sent to turn a note off. There is also an **all notes off** command.

Synthesizers map control keys to internal oscillators in order to generate sounds. **Mode Channel** messages define the way these mapping are made to a synthesizer. There are four modes: omni, poly, multi and mono. **Omni** mode causes a MIDI device to accept commands on all synthesizer channels. The device can transmit on only one channel. **Poly** mode allows a synthesizer to play notes transmitted on a single synthesizer channel polyphonically. **Mono** mode instructs the synthesizer to do the opposite, i.e., play one note per synthesizer channel at a time. This feature is used when emulating wind or brass instruments. **Multi** mode assigns individual synthesizer channels to separate voices. Multi-timbral synthesizers use this feature exclusively.

A MIDI **system** command is a sequence of one or more data bytes and is divided into three categories. They are:

1. System Common - these commands are used primarily by sequencer software to describe where in a musical sequence the synthesizer should be.

2. System Realtime - these commands are used to synchronize a network of MIDI devices.

3. System Exclusive - these commands are used to transmit device specific data. Only the header and trailer bytes are defined. The remainder of the data message is assumed to be vendor specific.

The terms **program** and **voice** are used synonomously to describe a particular sound generation setting for a synthesizer. Some examples of programs are trumpet sounds, organ sounds, orchestra sounds and piano sounds. A musician selects a program by pressing a program select key on the synthesizer. This action can be done in software using the MIDI **Program Change** command.

MIDI doesn't define what these programs may be. It simply instructs a synthesizer to switch to program XX where XX is a code ranging from 0 to 127 and is recognized by particular synthesizer.

A pitch bend is an effect produced when a string is stretched slightly beyond its normal length. This produces a "blues" type sound. The **Pitch Bend** command is a two byte channel message that instructs the synthesizer to perform a pitch bend on a particular channel. The musical range of the pitch bend (the range of musical steps for the bend) is not defined by MIDI.

2.4 MIDI Discussion

Loy points out in his paper some of the misconceptions about MIDI [Loy,27]. They are:

1. **MIDI is a bus.** It is not. It is a unidirectional network.

2. **MIDI does not have sufficient bandwidth to capture human performance.** MIDI command transmission sequences take about 1 msec and when compared to the typical 10 msec attack time for percussive instruments, we can see that there is little chance of smearing.

3. **Existing commercial synthesizers run at full MIDI bandwidth.** Some tests have shown that synthesizers take on the average of 2.6 times longer to turn on a voice than process a MIDI command.

4. **The data rate requirement for performance gesture capture is the same requirement as that for synthesizer control.** The 31.25 K baud transmission rate was certainly thought to be adequate when the protocol was written in the early 1980s. However, technological changes have increased both the speed of microprocessors and data transmission rates to levels unheard of in 1983. Current synthesizer technology uses microprocessors that are about 5 years behind the

technology curve. It is only a matter of time before the processor speed of the commercial units will rapidly overtake the current MIDI baud rate.

2.4.1 Some Problems with MIDI

Moore describes certain dysfunctions with the MIDI protocol from a musical standpoint. He defines three basic issues that need to be addressed when learning to play a musical instrument. They are:

1. operation of the instrument itself,

2. customary and contemporary performance practice pertinent to the instrument,

3. literature available for the instrument [Moore, 28].

Psychoacoustic studies have shown the inability of humans to distinguish which of two events occurred first if the time between them is less than 60 milliseconds [Kuivila, 10; Cadoz, 14; Vorberg, 16; Anderson, 41 Woodrow, 48; Wuthrich, 49]. In certain musical situations where the attack time of the sound in question is longer than this window, temporal smearing may "muddy" the sound. Another example of this temporal smearing is the delay time from the musical gesture, e.g., striking a key, to the time the sound is generated. Hidden processing times include the time to interpret the analog sensor signals, the time to convert this data to MIDI data, the transmit time between the controller and the synthesizer and the sound generating time in the target synthesizer [Moore, 28]. All of these times need to fall under the threshold of human distinction in order to prevent a sluggish response model. Failure to satisfy this constraint can cause a temporal smearing of sound attack times.

MIDI reports events to a synthesizer or software controller in a synchronous manner where most musical control parameters are of the continuous feedback variety. MIDI handles this situation by

sampling the data stream for these parameters and adjusting accordingly. This does not appear to be a problem as the processing speed of the microcontrollers increases allowing the sampling time for discrete events to be shorter.

The inability of some MIDI systems to accurately capture certain musical gestures such as glissando and trills is another area of concern. The average time for a full keyboard glissando using 1 hand is approximately 1/2 sec which translates to 204 MIDI notes per second. A trill can generate a note event every 1.6 msec. Again, faster processing speeds of the controllers seems to have relegated this concern to a lesser degree.

The biggest limitation appears to be in the 31.25 KBaud transmission speed. This baud rate is an **aggregate** rate and isn't particularly suited to handle transmission bursts. Moore states that there are three ways to deal with this problem. The first is to clip the information coming in, the second is to precompute the more complex data ahead of time and release it when triggered and the third is allow data at a particular time t to come out at t + 1, t + 2, etc.

MIDI control information tries to capture what the musician is doing real-time, control a synthesizer realtime and control the transmission of information from the musician to the synthesizer. In an acoustic instrument such as the dulcimer, the strings, sound board and all associated components work together to produce the sound. The hammer hits the string which vibrates and the bridges transmit the vibration throughout the box and these vibrations are amplified within the sound box and the listener hears the tone. In an electronic instrument, the input device (the equivalent of the strings) need not be physically connected to the modules that generate the sound. However, it is vital that the device give the performer the same "feel" as the acoustic instrument. Playing an instrument is a combination of factors such as tactile and audio feedback. The performer uses these feedback signals to shape the tone played. Early keyboard synthesizers provided no tactile feedback and keyboardists had to give up a certain amount of control and expressiveness. This remained a problem until Roland Corporation came out with the first keyboard controllers. These controllers were strictly gestural devices only. They were basically MIDI transmit units. The major

advantage of these units was that they provided the "correct" feel of an acoustic piano (weighted, wooden keys, better touch sensitivity and responsiveness). They could be connected to any synthesizer module and represented the first attempt to divorce the input mechanism from the rest of the synthesizer unit.

3.0 Electronic Instrument Controller Design

In recent years, the human/instrument interface has been a popular subject of cybernetic research. Pressing defines the playing of an instrument to be "the transfer of spatial and temporal information from the central nervous system to the system that physically produces the sound" [Pressing, 3]. The stimulus-response model fits this description extremely well because of the nearly one-to-one response between the musician's gestures and the sounds that result from those gestures. A percussive controller further uses these gestural links for power and accurate rhythmic precision.

Digital instrument controllers give the instrument designer a great deal of flexibility in constructing the controller interface. The components of the controller are easily modularized. This flexibility allows for musical information to be input by non-traditional means. The HANDS [Waisvicz, 9] and the Video Harp [Rubine, et al, 6,17] are examples of non-traditional musical interfaces.

The ability to design new gestural interfaces for musical instruments can greatly increase the amount of expressive control a musician has in playing. A wide variety of musical interfaces have been designed in the past but only a few of them attempt to emulate traditional acoustic instruments. String instrument controllers emulate the acoustic versions but have encountered some difficulties duplicating the "feel" of the instruments. The design strategy should create a flexible instrument controller that can combine the better elements of percussive and string instruments. The more

real-time control a musician has over the instrument, the more expressive tools are available for use.

Pennycook describes a hierarchy of timing requirements for a real-time music system [Pennycook, 15]. The hierarchy has the following requirements:

1. Formal music structures - several seconds to several minutes
2. Translation of formal structures
3. Performance execution rates - from .05 seconds to several seconds
4. Input device interaction
5. Streaming of performance data - 1 msec to 100 msec
6. Control updates to the synthesizer
7. Sound synthesis - hardware instruction rates
8. Sound sample output
9. Sound sample conversion - roughly 44,000 samples/second/channel

MIDI instrument controllers must meet the specifications described in items 3, 4, 5 and 6. The higher speeds of today's technology allows the digital instrument designer to capture the subtleties of the playing process more accurately. The key factor in this design model is that of the performance execution of the input devices. The musical tones must be transmitted faster than the JND for hearing separate events. This integration of minimum response times proves to be the most difficult area of controller design. The controller must have the capacity to adequately handle

MIDI transmission bursts, commonly generated whenever multiple notes are played simultaneously.

3.1. Cybernetic Issues in Controller Design

Rubine states that there are gestural parameters which are data such as finger position and force on the instrument and sound control parameters which are the perceptible parameters of sound such as pitch, timbre, amplitude and timing [Rubine, et al.; 6,11]. They also define a controller as a "device that detects gestural parameters and maps them to sound control parameters".

These sound control parameters are divided into two categories called instantaneous and envelope classes. Instantaneous parameters are determined at an instant of time. The hammer strike on the dulcimer is an example of this class of parameter. The amplitude change and decay rate of the note are dependent on the initial strike on the string. Envelope parameters are values that are a function of time over an interval. Bending notes on stringed or wind instruments such as guitars or saxophones are examples of this class of parameter. This class is not easily applicable to an instrument like the hammer dulcimer. Note bending can be done but the range of the bend is limited due to the high string tension of the instrument.

Controlling some of these parameters can be difficult for the hammer dulcimer player. A skilled piano player can control the parameters of ten separate notes using each of his ten fingers. The dulcimer player can control up to four separate notes because the dulcimer player uses only two, three or four hammers to play the instrument. The major parameter that can be controlled by the dulcimer player is amplitude. The player strikes the strings harder to get a louder sound. The force sensor used in this thesis translates the force measurement to an amplitude. Another parameter used by the hammer dulcimer player is rhythmic control. The player can utilize the instrument's percussive capabilities.

The No Strings Attached Dulcimer controller must respond to hammer strikes within a predefined period of time in order to avoid sounding "sluggish". The design parameters are subject to psychoacoustic as well as standard electronic limitations. The performance of the controller must be within the temporal bounds defined by the minimal just noticeable differences (JNDs) in sound control parameters that can be heard by the player or listener [Rubine, 6]. A key JND is that of the minimum amount of time between two events. Licklider reported in his paper [Licklider, 52] that very short clicks up to 2 msec apart were perceived as a single click by the human ear. Wuthrich states that if two or more musical notes are played 20 to 40 msec apart, they are perceived to have been played simultaneously [Wuthrich, 49]. A key factor in determining the minimum time is the type of sound generated by the instrument. Sounds like Licklider's click have very short attack and decay times. Other sounds with longer attack or delay times become less distinguishable because their start and stop times are harder to perceive. Hammer dulcimer tones have very long decay times due to the resonance of the instrument which can result in the smearing of tones over time. The hammer dulcimer player can take advantage of this smearing in order to simulate playing a three note chord. The audience will hear the a vertical sonority ("chord") even though the musician hit the notes at different time intervals. The time intervals, however, are less than the tonal decay time of the instrument.

Rasch's experiments on perception of simultaneous notes show that different onsets are difficult to distinguish in this range even with musically distinct notes. The suggestion of a note being present when it actually isn't can take effect if the time interval is in the 100 to 200 msec range. This continuity effect can be used to create an "audio illusion" whereby the notes don't have to be played at the same time although the listener will perceive them to be played simultaneously [Rasch, 50].

3.2 Ergonomics

The physical layout of the No Strings Dulcimer instrument controller assumes a spacing based on actual measurements taken from the author's hammer dulcimer. There is no standard measurement for string spacings, length of bridges, or bridge width. These measurements are dependent on individual hammer dulcimer builders. One of the advantages of the controller is that these measurements can be changed to accomodate the player.

Tactile feedback from the new instrument should approximate that found on a dulcimer in order to enable the player to utilize the feedback to control the force of the strike. Fitt's law of positioning accuracy states that for a given time per movement, the positioning error is proportional to the length of the motion [Singer, 53]. This basic law defines the constraints on designing the string spacing for the dulcimer. Large string spacing makes the instrument more difficult to play accurately. The musician moves his hands away from his body as he plays higher notes. Typically, the compromise is to place the dulcimer on an inclined stand so that the musician doesn't have to reach very far to play the high notes of the instrument.

The No Strings Attached Dulcimer layout requires no such compromise. Since string lengths are not a factor in the size of the controller, the only dimension to be considered is the distance between the horizontal sensor strips. This distance is typically between 1.25 and 2 inches and can be adjusted to match the player's instrument.

The dulcimer player gets tactile feedback from the string tension in the acoustic version of the instrument. This feedback allows to player to control the "bounce" or rhythm effect of the instrument. A more responsive bounce feedback allows the musician to put a livelier beat to the music. The playing surface can be modified to accomodate a particular player. In the MIDI version of the instrument, an appropriate surface had to be found that would closely approximate the real bounce feeling.

3.3 Some Examples of Unique Instrument Controller Design

A number of approached to instrument controller design merit closer examination. These controllers handle the various features of responsiveness, ergonomics and psychoacoustic elements in different manners. The controllers presented here are the Buchla Thunder [Rich, 39], the Digital Flute [Yanik, et al., 18], the HANDS [Waiscvisz, 9] , Video Harp [McAvinney, 6], Sequential Drum [Mathews, 7] and the Radio Drum [Aikin, 40]. These instrument controllers incorporate techniques that are in common with traditional acoustic playing techniques as well as some radically different playing techniques. Addresses for these vendors, when available, are provided in Appendix F.

3.3.1 Buchla Thunder

The Buchla Thunder, designed by Don Buchla, consists of 25 velocity and pressure sensitive touch pads laid out to fall underneath the fingers and heels of the hands. An LCD screen, a row of smaller touch pads used for programming the device and a long horizontal pad comprise the rest of the device. The device has a 42 note musical range, can store eight complete sets of program changes for all 16 MIDI channels, a short sequencer and a note filter. It is unlike a traditional keyboard controller physically and cannot be treated as a drum machines because of the programming flexibility of the pads.

The user can program each of the pads to perform any of the system functions. Some of these functions include programming the sequencer, transpose notes, do simple MIDI note-on/off functions and filtering notes. The range of flexibility of the device requires some learning time and this is a result of its non-traditional design. Since it is a new "instrument" musicians cannot easily translate previously known methods of playing it. The Thunder pads can be configured to do

standard note on or off when pressed and released or operate as a toggle where a note stays on until the pad is touched again. Each pad can initiate a special effect instead of playing a note and these effects can be triggered in the standard or toggle manners.

3.3.2 The Digital Flute

The Digital Flute is a microprocessor based digital wind controller that emulates a traditional flute. It is not a MIDI controller since it was developed before the standard was developed. It has a novel key arrangement that allows alternate keying patterns to be programmed by the microprocessor. The primary design goals were to provide an interface for musicians who are not familiar with the standard piano-style keyboard and to create a teaching instrument for flute players.

The instrument consists of ten switches, a control potentiometer and a microphone mounted on a plastic tube shaped like a flute. The flutist can press up to three switches to generate three distinct notes. Modifying features such as the attack, decay, "breathiness" and sustain of the note is controlled by blowing into a microphone mounted on the mouthpiece. Each switch has a corresponding id number that is read by a Motorola 6802 processor that converts the switch id to a square wave of the correct frequency. This square wave's amplitude is modulated by the microphone output. The "breathy" tone created by the musician blowing into the mouthpiece microphone is then mixed with the output signal of the square wave generator. The entire signal is then routed through an external amplifier.

The system software runs continously in a tight loop waiting for switches to be pressed. When a switch is pressed, the processor performs a table lookup to determine the correct output frequency. The analog circuit converts the microphone output voltage to a DC control voltage and uses this voltage to modulate the amplitude of the square wave tone generated by the system software. The modulated signal is subsequently amplified for loudspeaker output. This circuit gives the musician a measure of loudness control.

The primary advantages of this instrument controller are: 1) ease of use 2) flexible key manipulation 3) a realistic teaching tool. This controller was built before the arrival of the MIDI standard and so it doesn't incorporate any synthesizer control.

3.3.3 The HANDS

The HANDS MIDI controller was designed by Michael Wasvisz in 1985. The HANDS contain distance/speed, and tilt sensing sensors. The HANDS interface consists of a set of two aluminum plates with sensors, potentiometers and switches strapped under the performer's hands. Finger and hand motions generate analog signals that are decoded by a microprocessor that then transmits MIDI output. There are four mercury switches mounted under the plate that control the "octave-transpose". Moving the hand in a position similar to that of palming a basketball activates these switches. The left hand contains a sonar transmitter aimed at a sonar receiver that is mounted on the right hand. The distance between the sensors is translated to key velocity information. There are three press-buttons on the left hand that control MIDI channel selection. The right hand press-buttons control the synthesizer program choice. In addition to these switches, there is a thumb-wheel potentiometer that controls pitch bends and a toggle switch controlling sustain.

The analog circuitry resides on a single board. The microcontroller is a Rockwell 6511 processor. The software design is simple and similar to that of the Digital Flute. The software scans the input switches for changes and maps input data to MIDI commands.

The HANDS controller was one of the first non-traditional instrument controllers. Its purpose was to demonstrate that the "rich information generated by hand/arm gestures and finger movements/pressures" could be accurately translated into meaningful data suitable for controlling a digital synthesizer. It pointed out the difficulties in learning how to use these new movements in a musical setting. However, the greater flexibility afforded by the controller allows the musician to be more expressive in the performance of a musical piece.

3.3.4 The Radio Drum

The Radio Drum consists of a square playing surface and two mallets that have short range radio transmitters built in them. A radio receiver inside the playing surface tracks the mallet positions in three dimensions. Max Mathews designed this controller to allow a musician to control triggering events and continuous events. By waving the mallets over the playing surface, the player can add features such as vibrato, pitch bends, and tempo controls. Aikin states, "The Radio Drum has a positional accuracy of approximately 1% of the playing surface which translates into 2.10 inch for a 20-inch pad." Pitch layout is reconfigurable and since the controller can sense the mallet position BEFORE it hits the trigger point, it can send the MIDI information ahead of time to the synthesizer.

3.3.5 The Sequential Drum

Mathews developed an earlier instrument called the Sequential Drum. This device is a rectangular device that is hit with drum sticks. The sensors are arranged in an x-y fashion and returns the following sense information: trigger pulse, amplitude, X and Y information. The trigger information starts a next event in a **predefined** musical score. The amplitude of the strike signal controls the loudness of the particular event. The Sequential Drum allows the performer to control all of the control performance parameters such as tempo, dynamics, and timbre except for pitch. The computer has the pitch information stored previously.

The controller measures the signal information by using contact microphones connected to an aluminum back plate and an X-Y grid of grounding wires. The triggering is done by activating a series of one-shots. The unit allows the performer to concentrate solely on the performance factors of the music and not on the actual pitch generation.

3.3.6 The Video Harp

The Video Harp controller optically scans a performer's finger positions on an equilateral triangle playing surface. A neon tube provides the light source and the sensor is a DRAM chip. It can be played with a variety of opaque objects because the sensor detects light or dark sections of the playing surface. The controller can be played using harp, drum or keyboard techniques to generate strumming, percussive, or single note techniques. The main disadvantage of the controller is its very slow 30 Hz scan rate.

The Video Harp hardware consists of a MC68008 processor, RAM, timers and a logic cell array. It is connected to an IBM PC/XT which has a Roland MPU-401 MIDI interface that is used for MIDI output. The 68008 processor determines the finger positions from the triangular sensor and tranmits this information to the IBM PC/XT. The PC then takes this ray list data and converts it to MIDI codes which are output to the synthesizer via the MPU-401 unit. Future versions of the controller will eliminate the PC from the hardware setup.

The sensor sub-regions can be mapped to distinct MIDI functions. For example, one sub-region can be configured as a keyboard region that can be played as a keyboard. Another region can be configured to be played by a bowing motion similar to a violin. A third region can be configured as a conducting region that sends MIDI clock, key velocity, or other real-time information to the synthesizer.

This controller in its present form only plays a predefined sequence of notes. Its scan rate is still too slow to allow true real-time generation of notes. However, the programmable mapping or sub-regions allows for more diverse playing techniques than those found on the traditional acoustic instruments.

3.5 *Discussion*

The controllers mentioned in the previous section all follow the basic design philosophy of creating new methods of **playing** music. All of the interfaces attempt to allow greater degrees of controlling the note modifying parameters, however, only the Digital Flute and Buchla Thunder attempt to deal with the actual real-time note generation. This is mostly due to the lack of processing speed built into the controllers. Their studies have determined that most real-time gestural events occurs in intervals of less than 33 msec but the device scan rates are generally much slower. The newer controllers such as the Video Harp still have this slow scan problem and attempt to alleviate this bottleneck by off-loading some of the computational process to a separate workstation. They also mention using faster hardware to overcome some of these design deficiencies.

In each of these controllers with the exception of the Sequential Drum, the designers used single microprocessors to control the analog portion of their systems. The Video Harp concentrated solely on the controller portion of the real-time music system, leaving the actual sound generation to a digital music sysnthesizer. Using MIDI to communicate to commercial synthesizers simplifies overall controller design and allows the designer to concentrate on capturing the performance gestures more accurately. Controllers such as the Buchla Thunder are very powerful instruments but they require a long learning period to fully take advantage of their capabilities. Controllers such the the Digital Flute and Video Harp use a blend of traditional playing techniques mixed with a small amount of learning.

4.0 Physical Design of the No Strings Attached Dulcimer

4.1 Design Goals

Creating a controller for which a musician doesn't have to learn a new method of playing allows the musician more time to concentrate on producing new music rather than learning new playing techniques. This is one reason why current keyboard synthesizers are so popular among musicians today. It is very easy to translate the mechanics of acoustic piano playing to synthesizer keyboard playing. The only difference between an acoustic piano keyboard and a synthesizer keyboard is in the tactile feedback, sometimes known as **touch** that musicians feel when they press a key. Both types of keyboards look identical providing the keyboardist with a familiar visual reference. The spacing between keys, in general, is identical, again providing the musician with a familiar tactile and spatial reference. These similarities facilitate moving from an acoustic keyboard instrument to the electronic version. However, in order to correctly emulate an instrument such as a cello or flute on a keyboard style synthesizer, the musician must have a good idea of the mechanics of playing the instrument they are emulating. For example, keyboardists generally do not know the subtleties

of playing a percussion or wind instrument. In fact, most keyboardists will tend to play a particular sequence of notes played, often called a **riff**, that would be extremely difficult to play on the instrument they are emulating. An example of this is a keyboardist who has their synthesizer emulating a flute playing chords. This is next to impossible to play on a real flute.

The No Strings Attached Dulcimer combines the percussion and keyboard playing techniques of the traditional hammer dulcimer. Design goals include:

1. The notes should be arranged in a pattern identical to that found on the acoustic hammer dulcimer providing the player with familiar visual references.

2. Spacing between sensor strips should be identical to that found on acoustic hammer dulcimers, thus providing a similar spatial reference for the player.

3. The sensor surface should allow the hammers to rebound or bounce off the sensors in a manner similar to that of the acoustic instrument. This provides tactile feedback to the hammer dulcimer player.

It should also attempt to expand the range and versatility of the acoustic version of the instrument. The acoustic hammer dulcimer is typically tuned in a manner that makes it easy to play in the sharp keys. This feature makes it easy to play with stringed instrument since they are most commonly played in the sharp keys. Playing with horns is more difficult because the horns are tuned to the flat keys and these scales are generally not easy to to play on the hammer dulcimer. Adding features such as dynamic retuning whereby the instrument's pitch level can be changed to play with any instrument family greatly increases the performer's ability to play in any situation.

4.2 Why the Hammer Dulcimer?

The hammer dulcimer is a member of one of the most ancient families of instruments in the world. Almost every culture in the world can claim possession of an instrument in this family. It is known by various names such as the cymbalom (Hungary, Romania), hackbrett (Germany), Qang-chi (China), psalterio (Mexico), santuri (Greece) and santur (Persia). Modern instruments are trapezoidal in shape with 40 to 150 strings. The instrument is played by striking the strings with wooden sticks or plucked by the musician's fingers. The instrument has a wide dynamic range and the sound can be modified by altering the surface of the sticks known as hammers. A loud, sharp tone can be gotten from bare wood hammers. Conversely, a light, soft, harp-like tone can be gotten from felt-tipped hammers. It is this wide degree of flexibility that makes the instrument a prime candidate for emulation.

The hammer dulcimer is a classic example of a "trigger timbre" style of instrument. Trigger timbre is defined to be an "action where a musician performs a gesture and a machine/vibrating body produces the sound in a fixed pre-arranged manner" [Chabot ,8]. Because the musician can vary factors such as impact speed, size and weight of the hammers and rebounds or bounces, no two sounds on the instrument have the exact same timbre.

4.2.1 Physical Layout of the Hammer Dulcimer

Figure 4 described the tuning arrangement of one of the author's hammer dulcimers. This instrument is tuned diatonically and has a 2 1/2 octave range. This particular instrument does not contain all of the chromatic notes within this octave range. The treble bridge is the leftmost bridge and the bass bridge is to the right of it. The player strikes strings on either side of the treble bridge and on the left side of the bass bridge. A string pair played on the right side of the treble bridge is a musical

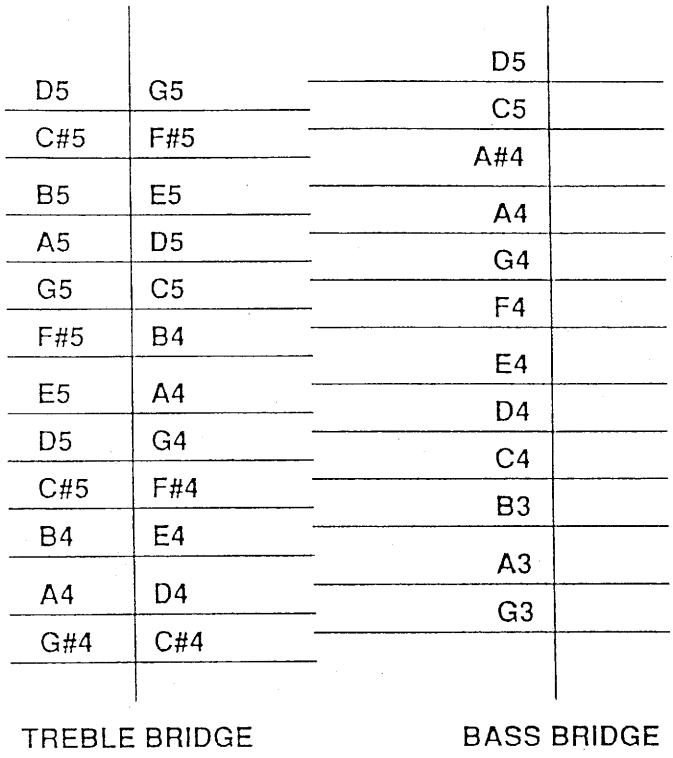
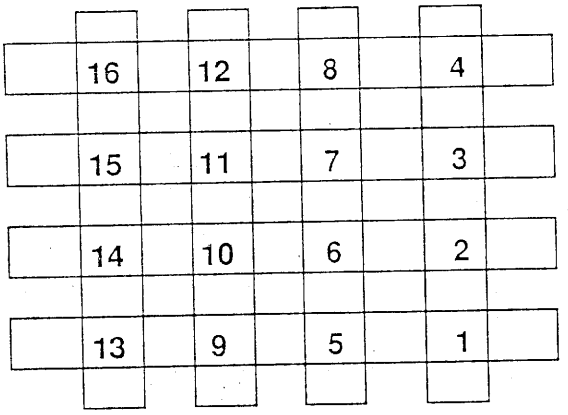


Figure 4. Tuning Chart for a Typical Hammer Dulcimer

fifth interval above the same string pair played on the right side of the treble bridge. A string pair on the right side of the treble bridge is a musical fifth interval above the string pair that is one position below it on the bass bridge. Hammer dulcimers that have this tuning pattern are known as **fifth interval** hammer dulcimers. The instrument can be tuned in different intervals, since the bridges are movable and are held down on the soundboard by the pressure of the strings. Indeed, the international versions of the instrument are tuned to a different interval patterns that are native to a particular culture.

Hammer dulcimers can have more than two bridges. Another of the author's dulcimers has three bridges. The leftmost bridge is a bass bridge that has the lowest notes of the instrument's 3 octave range. The middle bridge (treble bridge) and the rightmost bridge (second bass bridge) are tuned similarly according to the layout shown in Figure 4. This instrument has a 3 octave, fully chromatic range. The hammer dulcimer family is quite versatile in its tuning arrangements. The restriction on the numbers of bridges used is one of keeping the instrument to a reasonable size. Hammer dulcimers with the range shown in Figure 4 tend to be around 40" at the base and weigh around 13-20 pounds. Instruments such as the cymbalom have at least 5 bridges and can be the size of a small piano.

The No Strings Attached MIDI controller uses a sensor grid that provides a tuning arrangement similar to a dulcimer that can be played on either side of **both** "bridges". This arrangements yields a two octave range for the instrument. Figure 5 shows the Kynar sensor grid note mapping and how it emulates the tuning of an acoustic dulcimer. The two leftmost vertical strips correspond to notes that would be hit on either side of a bridge. The rightmost two vertical strips function in a similar manner. The spacing between the strips is identical to the string spacing on the author's hammer dulcimer.



Position	Note
1	Middle C
2	D
3	E
4	F
5	G
6	A
7	B
8	C-5
9	D
10	E
11	F#
12	G
13	A
14	B
15	C-6
16	D

Figure 5. Kynar Grid Note Mapping vs. Acoustic Dulcimer Note Mapping

4.2.2 Tonal Arrangement of the Hammer Dulcimer

As with any acoustic instrument, the hammer dulcimer tone is really a series of partial tones generated when the strings are struck by the playing hammers. These simple tones, typically sinusoidal in form, are combined by a process known as Fourier synthesis [Hofstetter, 55] and form a compound tone. The tonal combination of amplitude, frequency and timbre defines the unique sound produced by the instrument. The amplitude of a note is determined by how hard the player strikes the strings with playing hammers. The frequency of the note depends on which string pair the player hits. The timbre is the sum of the different amplitudes of each partial tone of the final compound tone of a dulcimer note.

Partial tones are denoted by a number which corresponds to what multiple the frequency exceeds that of the prime or original tone. Thus, a second partial tone vibrates at twice the frequency of a prime tone and is said to be an octave higher than the prime tone. The fourth partial which is the octave of the second partial forms the double octave of the prime tone [Fogel, 54]. Fogel states, " the tones of the major scale are also lower octaves of the first five partial tones of the tonic, dominant and subdominant notes". The harmonic points of a string are shown in figure 5.

The amplitude of a hammer dulcimer prime tone and its upper partials depends on how hard the string was hit and the surface texture of the hammer. A very hard hammer surface will rebound almost instantly off the string. This produces a long series of upper partial tones resulting in a sound that is piercing and least pleasing to the ear [Fogel, 54]. A softer hammer surface dampens the string vibration slightly before the hammer rebounds off the string. This dampening effect generates tones of lower intensity for the upper harmonics of the string and a higher intensity for the the prime tone creating an effect that is more pleasing and softer to the ear.

A hammer dulcimer tone will generate sympathetic resonances for other strings whose frequency of their primes or one of their upper partials are in common.

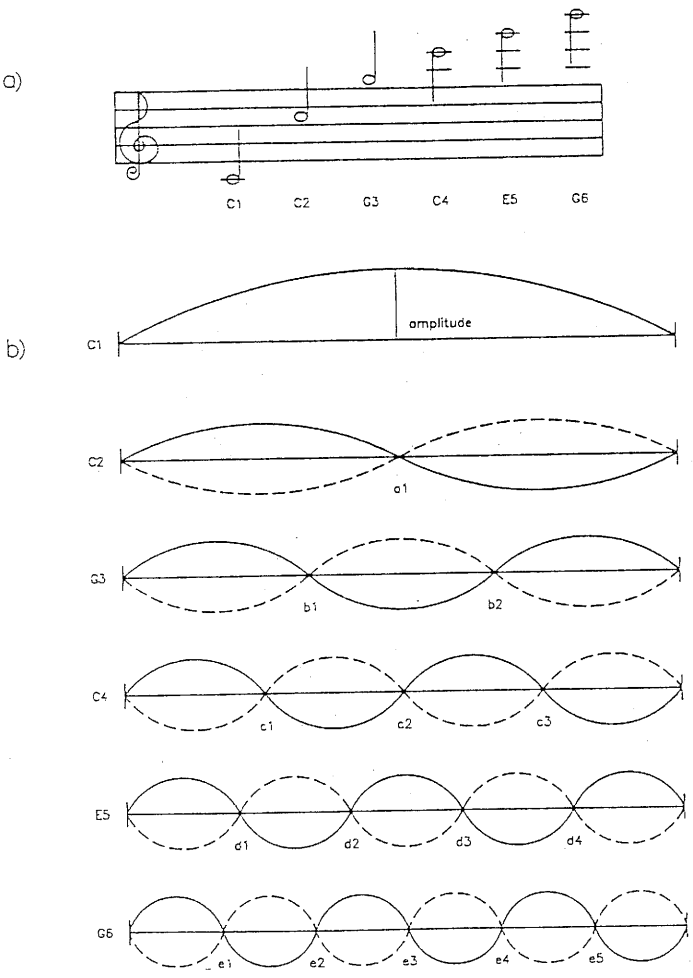


Figure 6. Harmonic points of a compound tone

Recall that a pair of strings can play two separate notes depending on which side of the bridge the player strikes the strings. The bridge is placed at 2/3 the string length so that the note on the left side is a musical fifth interval above the tone on the right side. The entire string vibrates when struck but at different frequencies depending on what side of the bridge we examine. Fogel describes in his paper a simple experiment to demonstrate this sympathetic resonance on a hammer dulcimer. Place a folded piece of paper on the C3 string shown in Figure 4. The paper will move "or even be thrown off" the string whenever a C3 or other C notes are struck. The paper will also move if one of the C3 partials is struck [Fogel, 54]. He concludes that "we have another way in which to verify the existence of the upper partials." The bridge does not completely reflect the waves of the strings as does the fixed end of the string. Some of the energy is transmitted across the bridge to "weakly excite the tone of the other portion of the string" [Fogel, 54]. Therefore, when a note on the right side of a bridge is hit, a corresponding weaker tone from the left side of the bridge is produced. This note, a fifth interval above the primary tone, is one octave below the third partial of the struck note. Fogel also notes that when a note on the left side of the treble bridge is hit, the corresponding right side tone is an octave lower than the fourth interval of the note struck.

4.3 Reverberation Timing

A general technique for calculating reverberation times for notes played on the hammer dulcimer is given. This time is the most important characteristic for determining the tone of an instrument. Reverberation time is defined by Fogel to be "the time which sound takes to die away after the source has ceased to operate." The general formula is: $\mathbf{T = (0.16 * V) / A}$ where:

- **T** is the time of reverberation in seconds
- **V** is the volume of the room the instrument is played in cubic meters

- A is the total absorption in open window units [Fogel,54].

An open window unit is the absorption of one square meter of open window. An open window is considered to absorb 100% of sound since the sound passes out of the room. Fogel's equation for calculating the total room absorption is: $A = c_1a_1 + c_2a_2 + c_3a_3 + ...$ where **c** is the coefficient of absorption shown in Table 1 and **a** is the area of the corresponding material. As can be seen in the table, the coefficients of absorption increase with increasing frequency.

Experiments show that long T values for low-pitched tones and short values for high-pitch tones will cause music to sound more mellow. Shorter T values for low-pitched tones and longer values for high-pitched tones will result in music sounding brighter. In general, longer T values seem more suited for orchestral music and even longer T values are suited for choral music. These T values are used to calculate the duration of a MIDI note being played.

4.4 Discussion

The timbre and reverberation factors are taken into account when the No Strings Dulcimer controller operates in emulation mode. When the controller is in this state, it must transmit at least 5 notes corresponding to the partial tones of varying intensity for each note struck by the player. This is not true Fourier synthesis emulation since the tones output by the synthesizer are compound tones themselves. However, if the synthesizer sound generators can output simple tones then simple timbre emulation can be performed. In addition, the controller should output the corresponding tone produced by sympathetic resonance. A note played on the hammer dulcimer is the loudest only at the moment the string is hit so its volume decreases rapidly afterwards. Fogel defines **beats** to be when " two simple tones of the same or nearly the same frequency excite the same nerve fibers in the ear producing *interference* when caused by two perfectly equal simple tones and *beats* when due to two nearly equal simple tones". The effect produced by **beats** can be heard in a oscillating

Material	Frequencies in cycles/second 250	500	1000-2000
Plaster	0.02	0.03	0.03
Brick	0.02	0.02	0.04
Wood Paneling	.01-.02	.01-.02	.01-.02
Curtains	0.2-1.0	0.2-1.0	0.2-1.0
Audience (per person)	0.43	0.47	0.50
Wood Floor	0.03	0.06	0.10
Carpet	0.05	0.20	0.50
Furniture	0.1-1.0	0.1-1.0	0.1-1.0

Table 1. Absorption Coefficients c (from [Fogel, 54])

fan. An undulating sound is often what listeners say they hear when the fan is running. Pairs of strings on the hammer dulcimer are tuned to the same note but mechanical imperfections cause the two strings to be not tuned to the exact same frequency. Beats caused by the dissonance between the two strings develop as the primary tone dissipates.

Tonal quality is simulated to some degree by generating a MIDI note command for the partial tones that form the primary tone, transmitting these tones one after another and following Fogel's rules on tonal quality to determine the velocity of the partial tones. The tone corresponding to that produced by sympathetic resonance is also sent to the target instrument. The duration of all of the partial tones is set to be equal. See the timbre emulator description in Chapter 6 for additional information.

Fogel's five rules for judging tonal quality are:

1. Simple tones consisting of only the prime tone lack power but are softer to the ear.

2. The most harmonious tones are those where the first six partials are audible.

3. When only the uneven partials are present, the tone is hollow and when a large number of upper partials are heard, the tone is nasal.

4. When the prime tone dominates, the tone is rich sounding. Playing a hammer dulcimer with soft hammers results in a richer tone.

5. The presence of the seventh or higher partial tones results in a brassy, penetrating sound.

The musical tone of the acoustic hammer dulcimer is full and resonant because the strings are under high tension, attached to a wooden frame and pass over fairly rigid bridges. This results in the string vibrations continuing for a longer period of time because they aren't easily dissipated to the sound box. Also, since most dulcimers do not have a damping mechanism, the sounds may tend to overlap in the lower range of the instrument.

The major drawback in this timbre emulation is that the partial tones generated by the instrument do not remain constant in amplitude throughout the duration of the tone. Moorer and Gray verified this effect in their research on the analysis of a violin tone [Moorer, et.al, 56]. This is the reason why modern synthesizers cannot truly emulate an acoustic instrument. In the future as processors become more powerful and smaller, true emulation will be closer to reality. The advantage of this timbre generation is that an instrument designer can experiment with the volumes of the different partial tones to create the "ideal" tonal structure for a particular instrument.

5.0 Hardware System Design

The hardware system is divided into three sections: the 8751 microprocessor, the piezo-electric sensor circuitry and the A/D circuitry. Figure 6 shows a block diagram of the system.

5.1 INTEL 8751 Microprocessor

The No Strings Dulcimer uses an INTEL 8751 microprocessor to drive the MIDI controller circuitry. Musical input is transmitted to the microprocessor via a piezo-electric sensor subsystem that provides sensor coordinate and magnitude information to the 8751. This sensor information is converted to MIDI command messages and sent to the MIDI output port. The Intel 8751 microprocessor was chosen as the driver of the system for a variety of reasons. These include

1. Fast execution time - using a 12 MHz crystal, most 8751 instructions will execute in 1 μsec,

2. Priority interrupt structure - the 8751 has 2 external interrupt control lines that can be prioritized under software control,

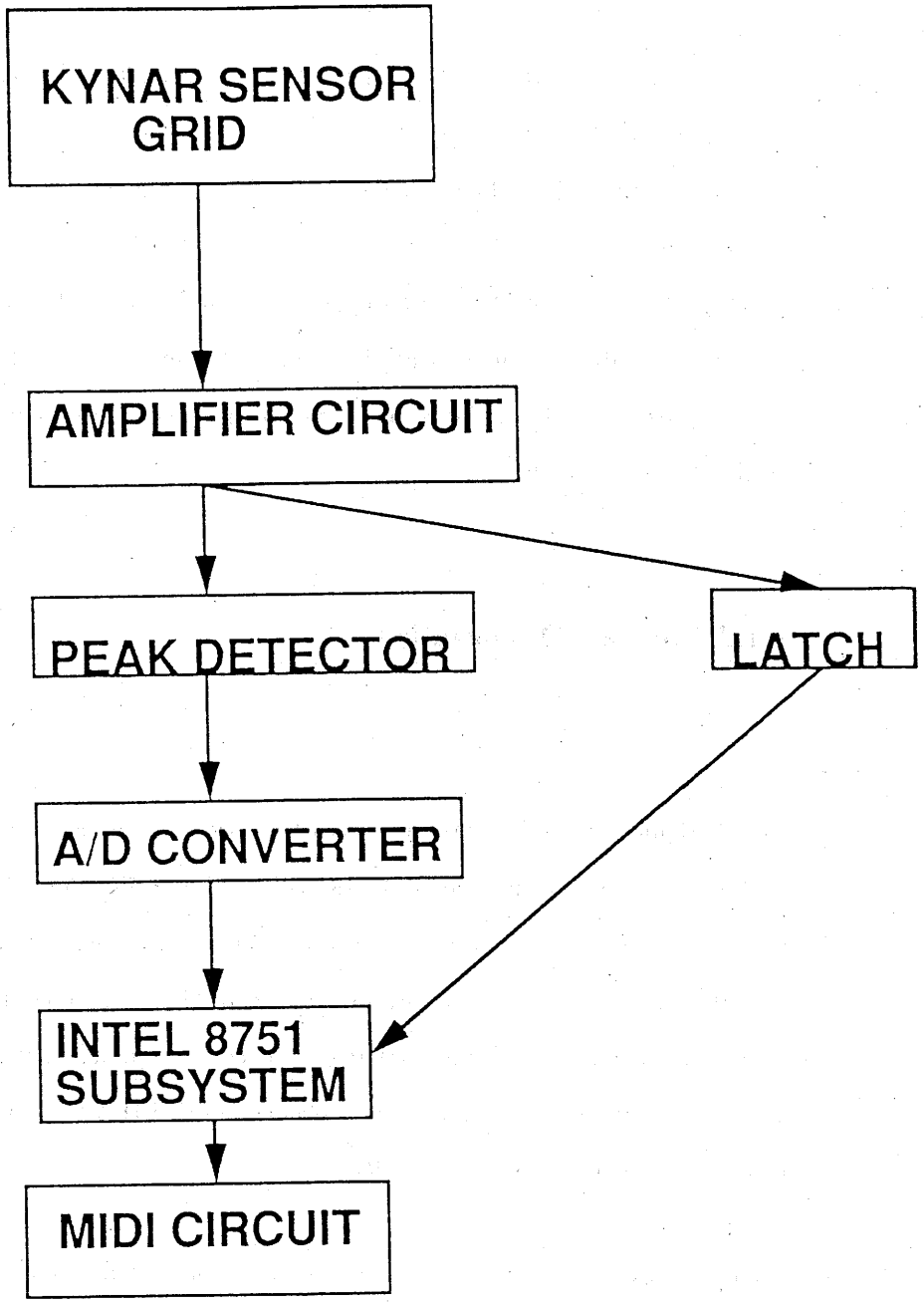


Figure 7. No Strings Dulcimer Controller Hardware Block Diagram.

3. System completeness - the 8751 has 4K of EPROM memory, 4 bidirectional I/O ports, a serial transmit/receive port and 2 internal timers.

The 8751 microprocessor is a member of the INTEL 8051 microcontroller family. It is an 8-bit microprocessor that is optimized for control applications. It can directly address up to 64K of program memory, 64K of data memory with 4K of program memory and 128 bytes of data memory on-chip. It has 32 bidirectional and addressable I/O lines giving a total or four 8-bit I/O ports on-chip. It also contains two 16-bit counters, a full-duplex UART, an on-chip clock oscillator and a 6 source, with 5-vector, two priority level, interrupt structure. The 8751H model used in this project runs at 12 Mhz yielding an average instruction time of 1 microsecond (1 μsec). It uses a single 5V power supply.

5.1.2 Memory Organization

The 8751 memory is divided into two distinct sections, program and data and is shown in figure 7. Program memory can consist of 64K with the lower 4K being on-chip. On CPU power-up or reset, program execution begins at location 0000H. The interrupt vectors are located in the lower part of program memory. They are spaced at 8 byte intervals and in this application contain a jump instruction to the corresponding interrupt processing routine. If the $\overline{EA}$ pin is strapped to VCC, instructions are fetched from addresses 0000H to 0FFFH. Program fetches from addresses 1000H to FFFFH are done from external ROM.

Data memory is also divided into external and internal data memory. External data memory can be up to 64K in length. Internal data memory is divided into 3 blocks totaling 384 bytes in length. Internal data memory addresses are one byte in length. This implies a maximum of 256 bytes but the addressing can accomodate up to 384 bytes by using an indirect memory addressing technique. Direct memory addresses greater than 7FH access a different physical memory space than indirect

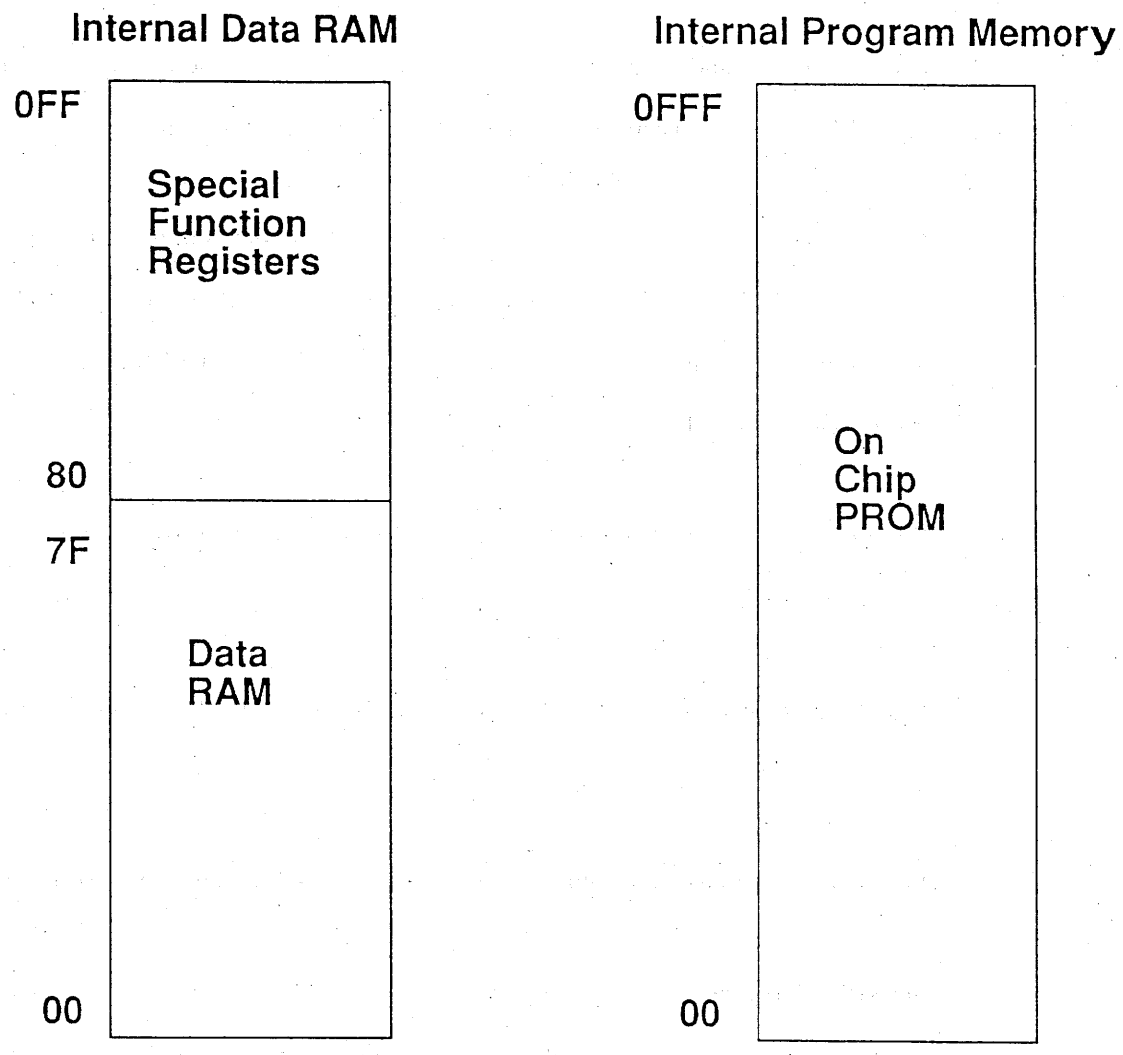


Figure 8. Intel 8751 Memory Organization

addresses greater than 7FH. The lower 128 bytes of Data RAM are grouped into 4 banks of eight general purpose registers. The Stack Pointer (SP) is an 8-bit register and resides in on-chip RAM and grows upwards.

The 8751 supports six addressing modes. They are:

1. **Direct** - the operand is specified by an 8-bit address field in the instruction. Data RAM and Special Function Registers are the only valid addresses in this mode.

2. **Indirect** - a register specified in the instruction contains the address of the operand. Internal and external Data RAM can be accessed in this manner. Register 0 and 1 (R0, R1) or the Stack Pointer (SP) are the address registers for 8-bit address. The Data Pointer register (DPR) is used for 16-bit addresses.

3. **Register** - the register banks can be accessed by instructions that use a 3-bit register designation within the opcode.

4. **Register-specific** - The Accumulator (ACC) or Data Pointer (DPTR) are the target address for certain instructions.

5. **Immediate Constants** - the value of a constant can follow an instruction in Program Memory.

6. **Indexed Addressing** - Program Memory can be accessed in a read-only manner by indexed addressing. A 16-bit register points to the base address of a table and the Accumulator contains the table entry number. The address of the entry is calculated by adding the Accumulator to the base register

5.1.3 Instruction Times

In general, 8751 instructions execute in 1 μsec when running with a 12 MHz clock. The only exceptions are the INC DPTR instruction (2 μsec) and the MUL and DIV instructions (4 μsec). 8751 accumulator specific logical instructions execute in 1 μsec with the others executing in 2 μsec. Data transfer instructions will execute in 1 μsec with the exception of the PUSH, POP, MOV DPTR and MOV DEST,SRC instructions which execute in 2 μsec.

5.1.4 CPU Timing

The 8751 has an on-chip oscillator that can be be used as a clock source. The internal clock generator defines the sequence of states that defines the 8751 machine cycle. An 8751 machine cycle is divided into 6 states and lasts 12 oscillator periods (1 μsec at 12 Mhz).

5.1.5 Interrupt Structure

The 8751 allows interrupts from 2 external sources, 2 timer sources and the serial port. Each of these sources can be individually enabled or disabled or all can be enabled or disabled at once. Each source can be programmed to one of two priority levels. A low priority interrupt can be interrupted by a high priority interrupt but not by another low-priority interrupt.

5.1.6 I/O Port Operation

The 8751 contains four bidirectional I/O ports on-chip. The output drivers of Ports 0 and 2 and the input driver of Port 0 can be used for external memory accesses. Ports 1, 2, 3 have internal pullups while Port 0 has open drain outputs. A port must have a '1' output to it to configure the port for input. All port latches have a '1' written to them on power-up or reset. The output buffers of Ports 1, 2 and 3 can each drive four LS TTL inputs. Each pin on the port can be accessed individually. This allows for great flexibility in configuring and controlling systems.

5.1.7 8751 Timers

There are two 16-bit Timer/Counter registers: Timer 0 and Timer 1. They can be configured to operate as timers or counters. In "timer" mode, the register is incremented every machine cycle. Thus, the count rate is 1/12 of the oscillator frequency producing a count every microsecond when a 12 MHz crystal is used. The 8751 timers are used in this mode for this project. For this application, the timer is configured to operate in MODE 2 which sets the timer up as an 8-bit counter with automatic reload. The timer is preloaded with a value and this value is reloaded on overflow. Timer status is either read from a flag bit or reported through the interrupt facility.

5.2 Kynar Sensor Circuitry

The input device for the MIDI dulcimer controller is a 4 by 4 grid of piezo-electric strips made out of Kynar[Cantrell, 38]. Kynar is a polyvinylidene flouride (PVDF) semicrystalline resin that is manufactured by the Pennwalt Corp. Table 4 describes the electrical characteristics of this material.

Property	Symbols	Values	Units	Conditions
Thickness	t	9,16,28,52	μm	
Piezo Strain Constant	d_{31}	23×10^{-12}	(m/m)/V/m)	lateral clamp
	d_{32}	3×10^{-12}		
	d_{33}	-33×10^{-12}		
Piezo Stress Constant	g_{31}	216×10^{-3}	(V/m)/N/m^2)	
	g_{32}	19×10^{-3}		
	g_{33}	-339×10^{-3}		
Electromechanical Coupling Constant	k	29	%	@100Hz(VF_2 /Vf_3)
	k_{31}	12	%	@100Hz(Vf_2)
Permittivity	ε	106×10^{-12}	F/m	@10Hz
Electrical Impedance	Z	1350	ohms	
Young's Modulus	Y	2×10^{9}		
Capacitance	C	379×10^{-12}	F/cm^2	

Table 2. Kynar Piezo Film Physical Properties (from [Cantrell, 38])

The film sensor is approximately 22 x 165 mm in size and resembles a piece of tape. The sensor strips are used in a variety of applications ranging from optical shutters, ink-jet pumps, flat speakers, blood pressure cuffs and basic impact sensors to vibration sensors.

The Kynar Piezo Film sensor is a thin sheet with a film of metal on each side to act as the electrical connection. When the strip is pushed in one direction, it generates a positive (or negative) DC voltage spike corresponding to the magnitude of down/up movement of the strip. The more the strip moves in one direction, the greater the voltage spike. This property allows a designer to measure how hard a strip was hit. However, the voltage is generated only by the change in position of the sensor and does not remain constant once the sensor reaches its new position. Once the material stops moving, the voltage returns to zero. In Figure 8, a downward movement of the sensor strip to point A generates a corresponding positive voltage level A'. When the strip rebounds to position B, a corresponding negative voltage, B', is generated. This positive/negative cycle of voltages continues until the strip position returns to equilibrium. The input signal generated by the hardware subsystem lasts approximately 3 msec. The output of the sensor strips is filtered by a resistor-capacitor (RC) circuit and amplified by a noninverting operational amplifier circuit. The

sensors are aligned in a 4x4 grid. The sensors aligned in the X direction are called the **horizontal sensors** and the sensors aligned in the Y-direction are called the **vertical sensors**.

5.2.1 Horizontal Strip Circuit

The horizontal sensors determine the magnitude of the voltage produced by the sensor when it is struck by the musician. This voltage is converted to digital form and translated into MIDI key velocity information. The microprocessor strips off the MSB of the data byte and does a table lookup to determine the correct MIDI key velocity. Figure 9 describes the horizontal sensor strip circuitry. The voltage range of the input signal is 0.0V - .3V and is amplified by a standard non-inverting amplifier circuit (labelled AC1) to an output signal range of approximately 0.0 - 4.0 V. The signal first passes through an op-amp (A1) acting as a unity gain follower. This amplifier simply buffers the signal from the Kynar strip to the non-inverting amplifier (A2). The resistors R1 and R2 have the values of 12K and 1K, respectively. Using the standard noniverting amplifier gain equation, **GAIN = (R1 + R2)/R1**, we compute a gain of 13 for the horizontal circuit. The voltage peak detection circuit (labelled PD1) captures the maximum voltage generated by the corresponding maximum deflection of the sensor strip. This circuit will only detect positive peaks because the diode (labelled D1) activates only when the voltage exceeds its forward-bias level. The discharge signal is activated by a microprocessor command to the DM7406 Hex Inverting Buffer chip (labelled HX1). This discharge signal must be held long enough to discharge the capacitor. Experiments have shown a minimum signal duration of 3.8 msec will discharge the capacitor within JND tolerances for minimum time between hammer strikes.

5.2.2 Vertical Strip Circuit

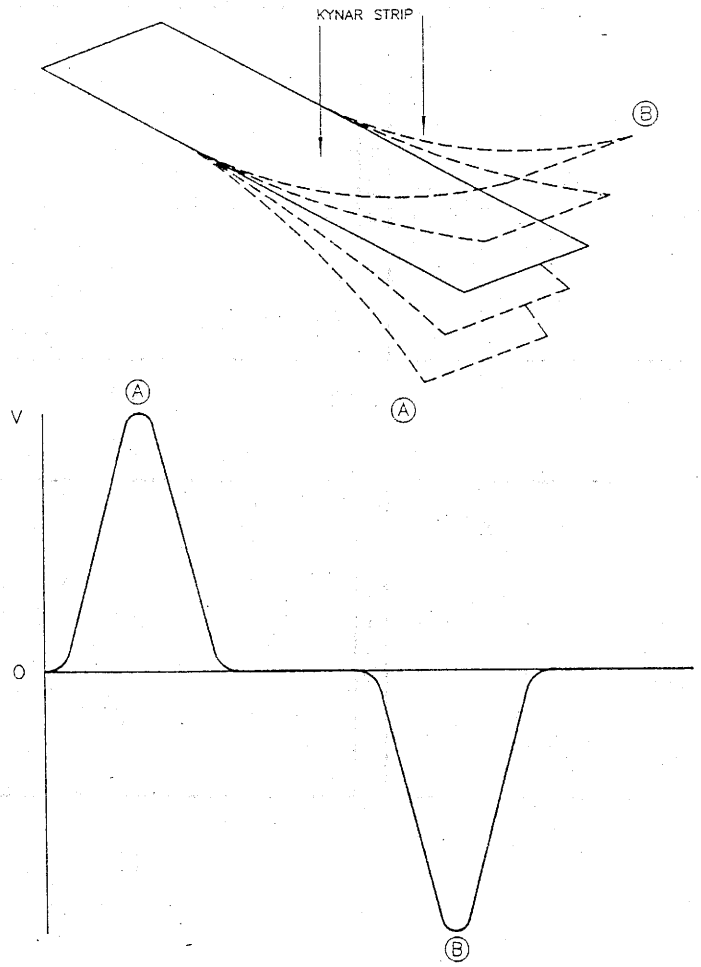


Figure 9. Voltage spike to sensor movement

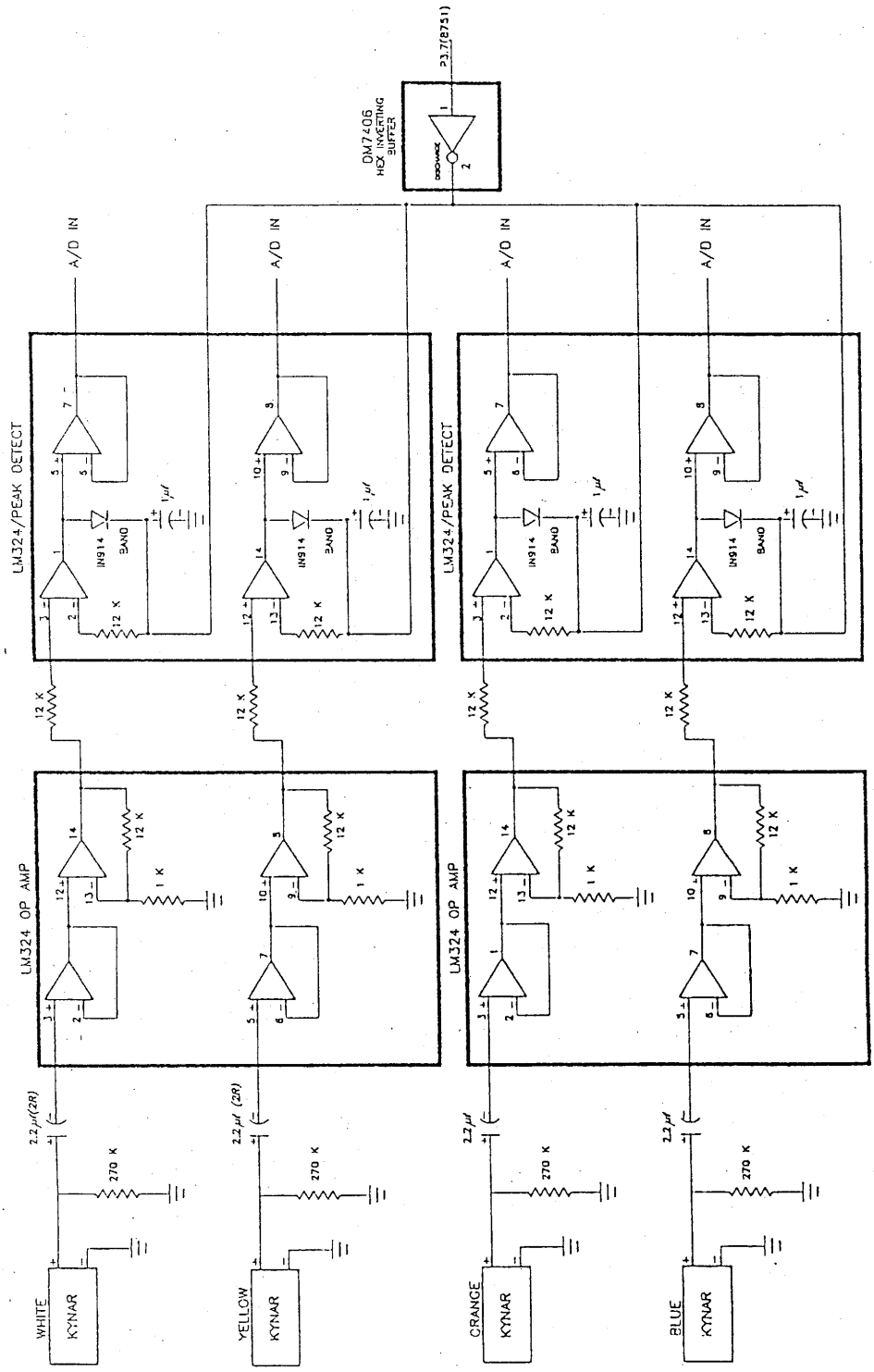


Figure 10. Horizontal Strip Circuit Diagram

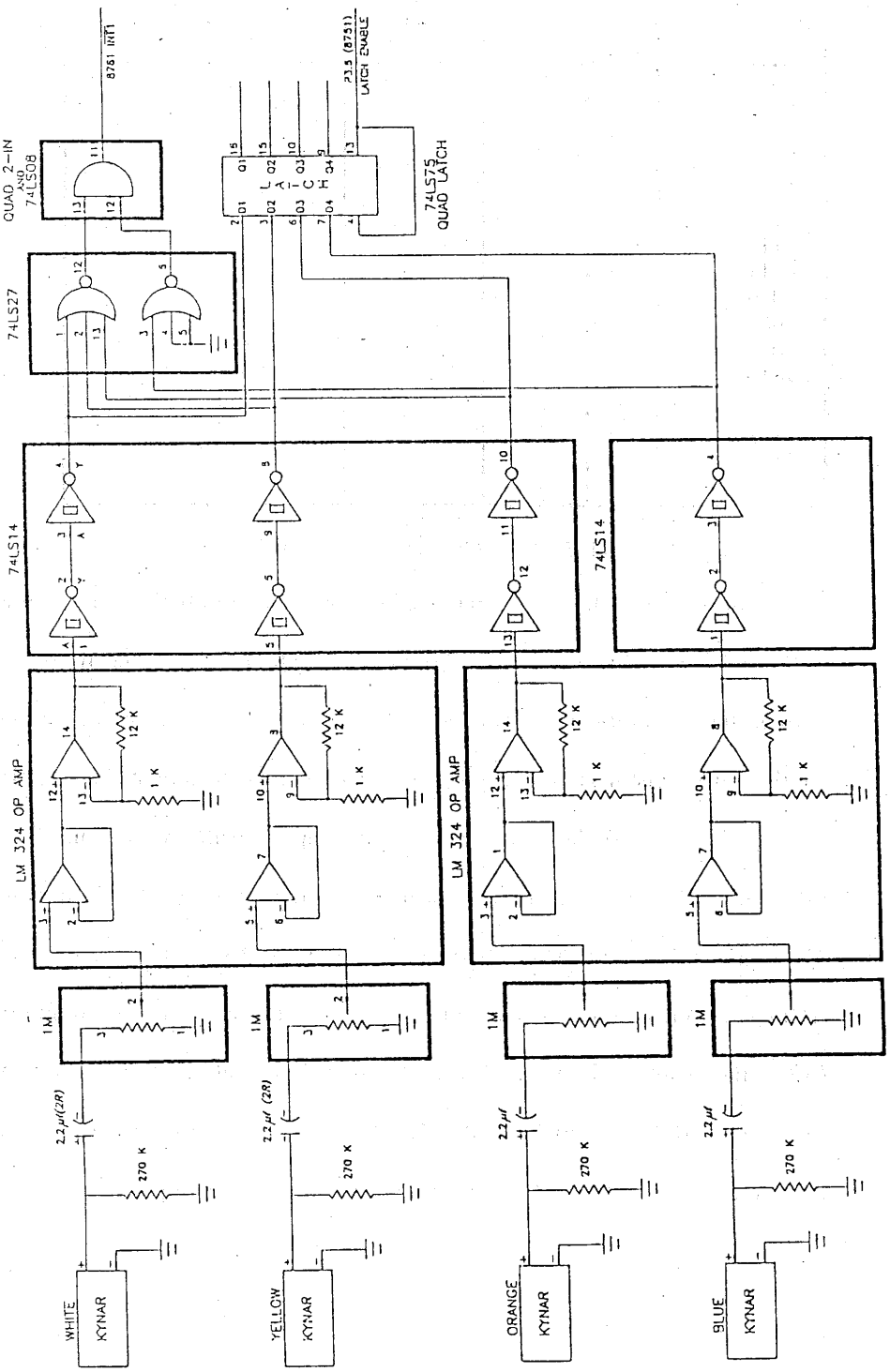


Figure 11. Kynar Vertical Circuit Diagram

Strip	1	2	3($\overline{INT1}$)	
0000	1	1	1	Interrupt
0001	1	0	0	
0010	0	1	0	
0011	0	0	0	
0100	1	0	0	
0101	0	0	0	
0110	0	1	0	
0111	0	0	0	
1000	0	1	0	
1001	0	0	0	
1010	0	1	0	
1011	0	0	0	
1100	0	1	0	
1101	0	0	0	
1110	0	1	0	
1111	0	0	0	

Table 3. Truth Table for Intel 8751 Interrupt signal

The vertical sensors determine that the horizontal sensors have been hit and that A/D conversion of the horizontal sensor voltages should start. The output signal from a vertical strip is routed through the RC-amplifier circuit shown in Figure 10. A 1 Megohm potentiometer was substituted for the 270k resistor (R1). This simple device is used to adjust the sensitivity of each individual sensor strip. Table 5 shows the results of a simple sensitivity test run with different R2, R3, and R4 values. The amplifier output is routed to a Schmitt Trigger to ensure a maximum trigger value. The signal is then routed from a 74LS27 3-input NOR gate (O1, Figure 10) to a 74LS7408 2-input AND gate (A1, Figure 10). The output of the AND gate generates an interrupt signal on the 8751 $\overline{INT1}$ line. Table 7 shows the truth table that is generated by this circuit. An interrupt is signaled when the 8751 $\overline{INT1}$ signal goes LOW. The vertical strips tell the microprocessor that something was hit and to start the A/D conversion of the horizontal strips voltages.

5.3 A/D Circuitry Description

5.3.1 ADC804 A/D Converter

The ADC804 A/D Converter is a CMOS 8-bit successive-approximation analog-to-digital converter that accepts an input signal ranging from 0V to 5V and outputs a digital value ranging from 00H to FFH (full scale). It has a 100 μsec conversion time, 135-ns access time, a conversion span of analog ground (GND) to Vcc and a clock input frequency range of 100 to 1460 KHz. The $\overline{WR}$ signal resets the chip's successive approximation and 8-bit shift registers. The ADC0804 remains in a reset state until either the $\overline{CS}$ or $\overline{WR}$ signals go to a HIGH level. This tranisition starts the conversion process approximately one to eight clock periods later. The $\overline{INTR}$ signal goes low when the converter has transferred an 8-bit data value to the output latch. The converted data is output to pins DB0-7 when the $\overline{RD}$ and $\overline{CS}$ signals are LOW. This transition also resets the the $\overline{INTR}$ signal to HIGH. A HIGH signal on the $\overline{RD}$ or $\overline{CS}$ signals returns the DB0-7 outputs to a high-impedance state.

There are four ADC804 A/D converters and each one is assigned to a different horizontal strip. This subsystem delivers to the Intel 8751 controller a value that represents how hard the strips were hit. This information is then translated to MIDI key velocity. The pin connections are shown in Figure 11 and are as follows: $\overline{WR}$ is connected to the 8751 Port 3.0, $\overline{RD}$ is connected to a 74LS154 4-16 decoder output pin, $\overline{INTR}$ is connected to a NOR-AND circuit that drives the 8751 $\overline{INT1}$ line LOW only when all of the ADC have finished a conversion. The ADC output lines, D0-D7 are connected to a bus that feeds into the 8751 Port 2. Since these data lines are tristate outputs, this ensures that only 1 valid set of data is on the bus at a given point in time.

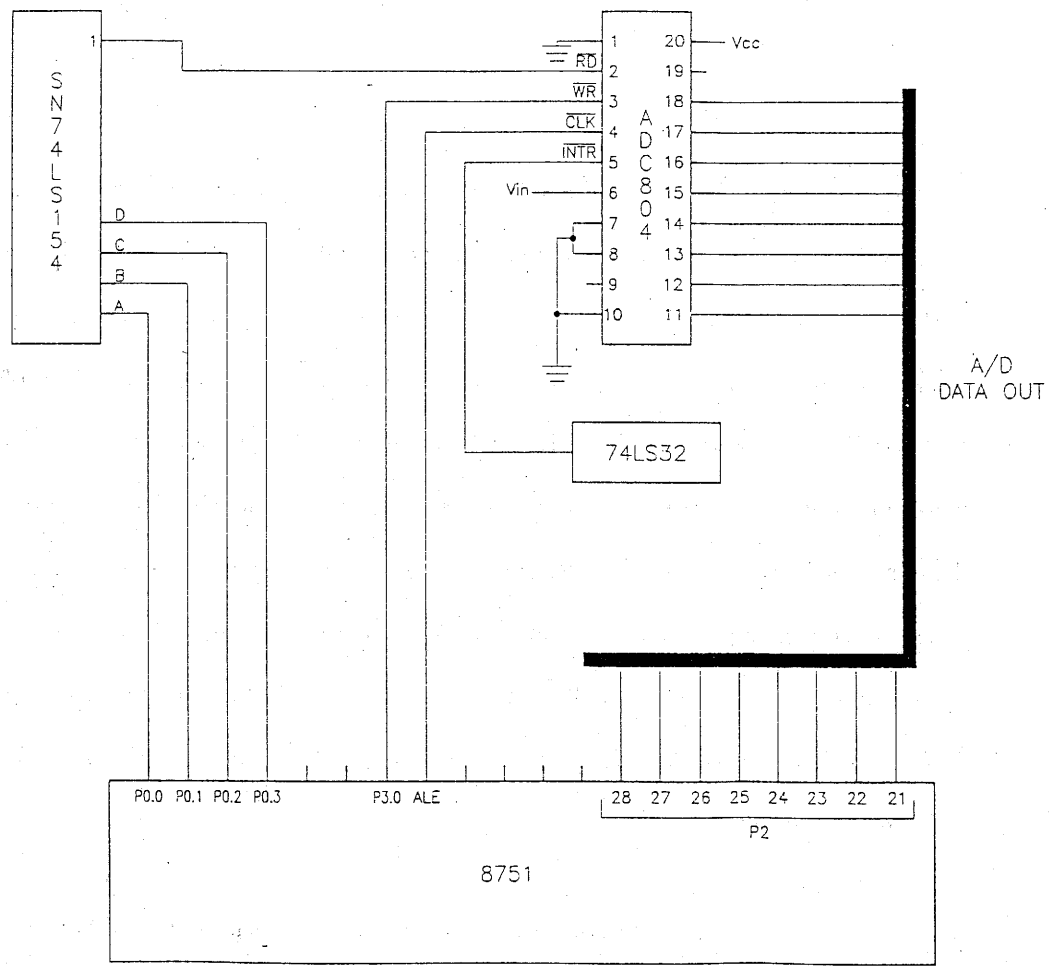


Figure 12. ADC0804 Pinout Diagram

5.4 Discussion

5.4.1 General Background

The hardware subsystem of the No Strings Attached Dulcimer was designed to provide accurate sense information that would tell the microprocessor that 1) a musical note was hit 2) how hard the note should be played. The design criteria were:

1. amplify and filter the Kynar sensor signals to voltage levels acceptable for use in TTL logic,

2. hold the signal levels long enough for A/D conversion,

3. reset all data and control signals in time for the next event.

4. transmit MIDI information from the processor to the target synthesizer.

The Intel 8751 has proved to be adequate in its role as a process controller. The processor's ability to use bit-mapped control signals allowed a wide degree of flexibility in designing control circuits such as the peak detection circuit discharge signal. Another example of this feature was in the ability to split an 8-bit I/O port in half creating essentially an extra control port. The I/O port electrical compatibility greatly reduced the chip count needed to interface to the microprocessor. The 4k program memory did not prove to be a hindrance as was originally feared, however, the small amount of data RAM necessitated the use of software tricks to compensate for this deficiency. The circular buffers used to hold the A/D and vertical sensor id values was smaller than desired.

The controller currently provides only MIDI **transmit** capability only.. The MIDI serial communications circuitry is well documented in current literature [Gaulteri, 26; Kubicky, 29, 30; Int'l MIDI Assoc., 31; Clark, 33, 35; Schmidt, 32; Henry, 34, 36, 37; DeFuria et al. 46; Rona, 47]. The

controller cannot interpret external MIDI information and therefore, has no MIDI **Receive** capability.

5.4.2 A/D Subsystem

After the ADC0804 $\overline{WR}$ signal goes from LOW to HIGH, the ADC0804 $\overline{INTR}$ signal goes from HIGH to LOW when the conversion is completed. However, the signal stays LOW until the corresponding ADC0804 $\overline{RD}$ signal goes from HIGH to LOW to HIGH. In the current setup, the 74LS154 decode chip strobes this signal until the maximum data value is found. If the value is found BEFORE the last A/D chip is read, the remaining ADC0804 $\overline{INTR}$ signals would remain LOW thereby triggering false interrupts. Toggling all of the $\overline{RD}$ signals solved this problem.

The final design of the A/D circuitry calls for a minimum of 4 A/D chips for each two octave cell, i.e., one A/D converter for each horizontal Kynar strip. The vertical strip interrupt routine starts the conversion sequence using a single signal to raise all of the $\overline{WR}$ signals from LOW to HIGH. In the first version of this subsystem, the processor would wait for a A/D converter chip $\overline{INTR}$ to drop from HIGH to LOW to start the READ DATA routine. However, the conversion time period for all of the chips was not constant and this would result in the processor reading the A/D data lines prematurely. The signals had to be passed through a NOR circuit to ensure that the ALL CONVERSION COMPLETE signal (connected to 8751 *INT0*) would be triggered only when **all** of the $\overline{INTR}$ signals were set properly. At this point, valid data was ensured to be present on the outputs of all of the A/D chips.The 8751 drops the $\overline{RD}$ lines for each A/D chip successively by using a 74LS154 4x16 decoder chip. The decoder chip inputs are connected to the 8751 output port and the decoder output lines are connected to the individual $\overline{RD}$ lines. As each output is selected, the output voltage goes from HIGH to LOW causing the A/D chip to present its data on the "bus".

5.4.2.1 Sample Hold vs. Peak Detect

In the original design of the A/D subsystem, LF398 Sample-Hold chips held the voltage produced by the amplified Kynar circuit long enough for the A/D chips to convert the signal to digital form. A simple test configuration, shown in Figure 13a, determined how fast the input signal could ramp up to its maximum value and the optimum capacitor value to use in order to minimize the charging delay time. The ideal shape of the input signal is a square wave but in reality, the signal is bell-shaped. The test would also determine the optimum acquisition based on the how fast the input signal reached its maximum value. The droop rate of the signal measures the fall time of the input signal from its maximum value. This rate determines the upper time bound that the HOLD signal had to be generated. All of these time values could be verified by oscilliscope. Figure 13b shows the output results. The input ramp time and droop rate was insignificant as far as the sample-hold chip was concerned. The original sample-hold circuit used the vertical sensor signal as the HOLD signal for the LF398 Sample-Hold chip. The sensor signals were routed through a flip-flop that was reset by connecting the $\overline{RD}$ signal to the **clear** input of the flip-flop. This generated a relatively long HOLD signal.

The Sample-Hold circuit was replaced by the peak detection circuit shown in Figure 9 (PD1) when experiments showed that there was no guarantee that the HOLD signal would be activated when the input signal reached its maximum value. Software tests showed that the HOLD signal would come slightly earlier or later than the input signal peak, resulting in incorrect values being input to the A/D subsystem. The peak detection circuit follows the input signal waveform and stores this voltage in the capacitor labelled C1 in Figure 9. The capacitor is discharged by a LOW signal from the DM7406 Hex Inverter Chip. The DM7406 acts as the discharge switch in this standard peak detection circuit. The discharge signal has to remain in effect to the time it takes to discharge the capacitor.

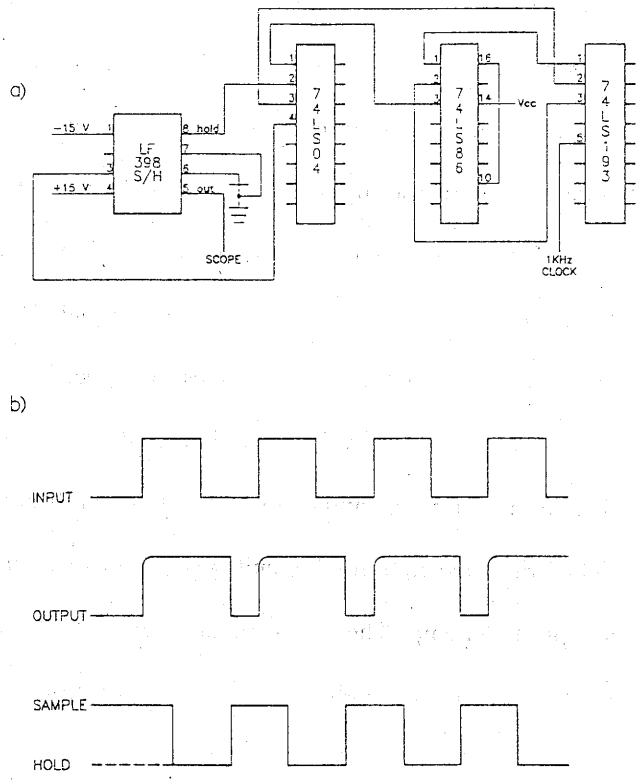


Figure 13. a) Sample-Hold Test Circuit b) Waveform output

5.4.2.2 Why Calibrate the A/D chips?

The calibration of the A/D converter subsystem (ADC0804 A/D converter chip, LM324 operational amplifier circuit, peak detection circuit for the Kynar strips and the DM7406 Hex Inverter Chip). was done in two steps. First, the A/D control signals were manipulated by hand. Secondly, the A/D control signals were manipulated under software control from the 8751 processor. Since force of a hammer strike is converted by the controller software to MIDI key velocity information, the calibration procedures were developed to determine the mapping between force and A/D output values.

This test verified that software control of the A/D converter chips by the 8751 was feasible and would meet the timing constraints of the the controller design. A schematic of the hardware used in the test in shown in figure 12. A kynar strip circuit was connected to the $\overline{INT1}$ line on the 8751. This interrupt tells the processor to start A/D conversions. A known voltage was input to the A/D input line using a 0-10V power supply. The test sequence steps were:

1. Apply a known voltage to the input pin of the A/D chip. This voltage was incremented in whole number units for each successive test.

2. The kynar strip was struck by a dulcimer hammer to simulate a playing strip being hit. This invoked the $\overline{INT1}$ interrupt handler and this routine drops $\overline{WR}$ from LOW to HIGH to start the A/D conversion process.

3. When conversion is complete, the $\overline{INTR}$ signal goes from HIGH to LOW. This event triggers the 8751 $\overline{INT0}$ interrupt routine.

4. $\overline{RD}$ is brought from HIGH to LOW by the interrupt routine and valid data is read into one of the 8751 I/O ports. The data value is output to a hex display connected to another 8751 port.

Input Voltage	Displayed A/D Data Range (Hex)
0V	00 - 0F
1V	31 - 3D
2V	63 - 78
3V	A8 - AD
4V	DF - E6
5V	FF

Table 4. A/D Calibration Values - 50 Observations at each Voltage

5. The test is repeated 50 times for each voltage setting to get an average reading. The power source for the experiment could not be to less than whole number units. This is why the A/D data values are expressed as a range rather than a single number.

The A/D calibration values are shown in Table 6. The table shows the range of data values in hexadecimal for the unit voltages ranging from 0-5V. Fifty observations were made at each unit voltage level and the data values were recorded. The test was repeated on five separate occasions. The data values were consistent regardless of varying room temperature, amount of time powered on or order of voltage change. The A/D calibration values fell within the range specified in the manual test. These tests verified the operation of the A/D to 8751 circuitry and provided the nucleus of the full blown software system.

5.4.2.3 8751 Interrupt Signal Sources

The 8751 $\overline{INT0}$ signal is generated by passing the four A/D $\overline{INTR}$ (A/D conversion complete) signals through three OR gates. Since this signal goes LOW and remains LOW until its corresponding $\overline{RD}$ signal goes LOW, the OR gate network allows the *INT*0 signal to go LOW only when all of the A/D conversions are completed. The 8751 is guaranteed to read valid A/D data at this point.

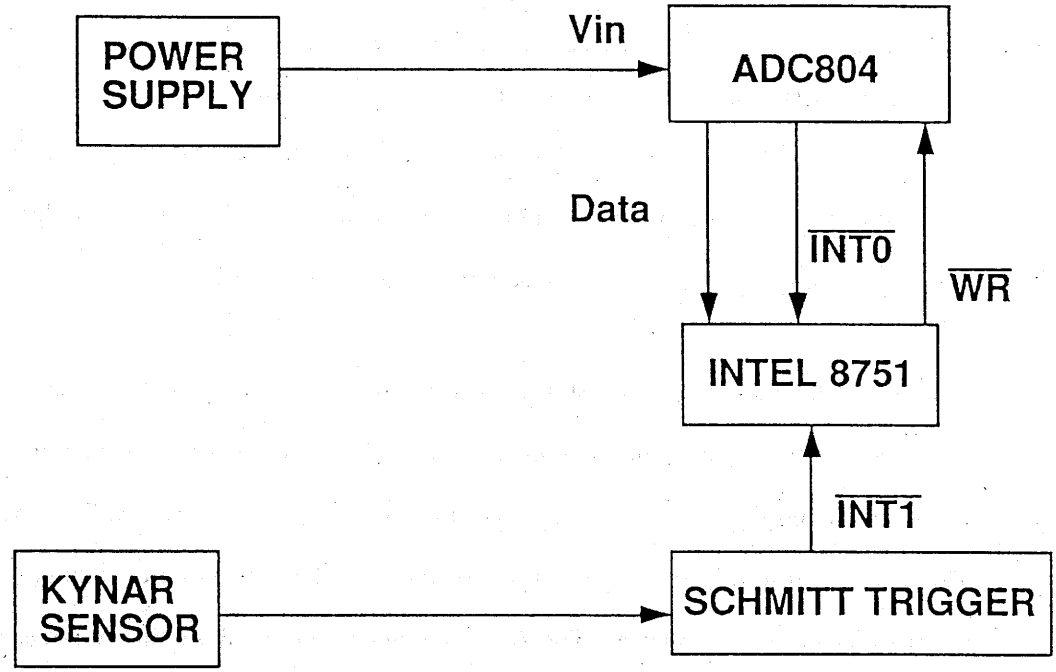


Figure 14. Schematic Diagram for 8751 Controlled Test

5.4.3 Vertical Sensor Design Considerations

The sensitivity of the kynar sensors poses a special problem in isolating the a valid signal from external vibrations. The feedback resistors in the op-amp circuitry yield a gain of 24 when amplifying the sensor signal. This was done in order to capture as many "hammer bounces" as possible. This high gain, however, also increases the susceptability of the sensor grid to falsely trigger an event.

5.4.3.1 Latch Timing Problem

The 74LS75 Latch chip identifies which vertical sensor was struck by the musician. It is enabled by the 8751 vertical sensor interrupt service routine (ISR) and is disabled when the ISR exits. A minimum of 3 machine cycles, 3 μsec, elapses between the $\overline{INT1}$ and execution of the first instruction of the ISR. This interrupt signal is the higher priority interrupt so approximately 4 μsec pass between the interrupt signal and the execution of the first instruction in the ISR. The input signals to the latch come directly from the Schmitt triggers (S1, Figure 10). The output of the Schmitt triggers is not filtered in any way so there is the potential of more than one strip generating enough voltage to register a "1" in the latch. This problem was hopefully solved by mechanical means discussed in section 5.4.4. The struck sensor will always generate the maximum value and will always register a "1" in the latch. Crosstalk from external vibrations may be great enough to register a "1" and this can deceive the vertical ISR.

5.4.3.2 False Triggering - the Problem

The Kynar sensors are sensitive to external vibrations and must be shielded from these sources as much as possible. The major tradeoff in adjusting the sensitivity of the sensors is between the ability to adequately capture the nuances of the musician's playing gestures and the the ability to filter out unwanted interference. The 1M potentiometers provide a first line "defense" for tuning the sensors'

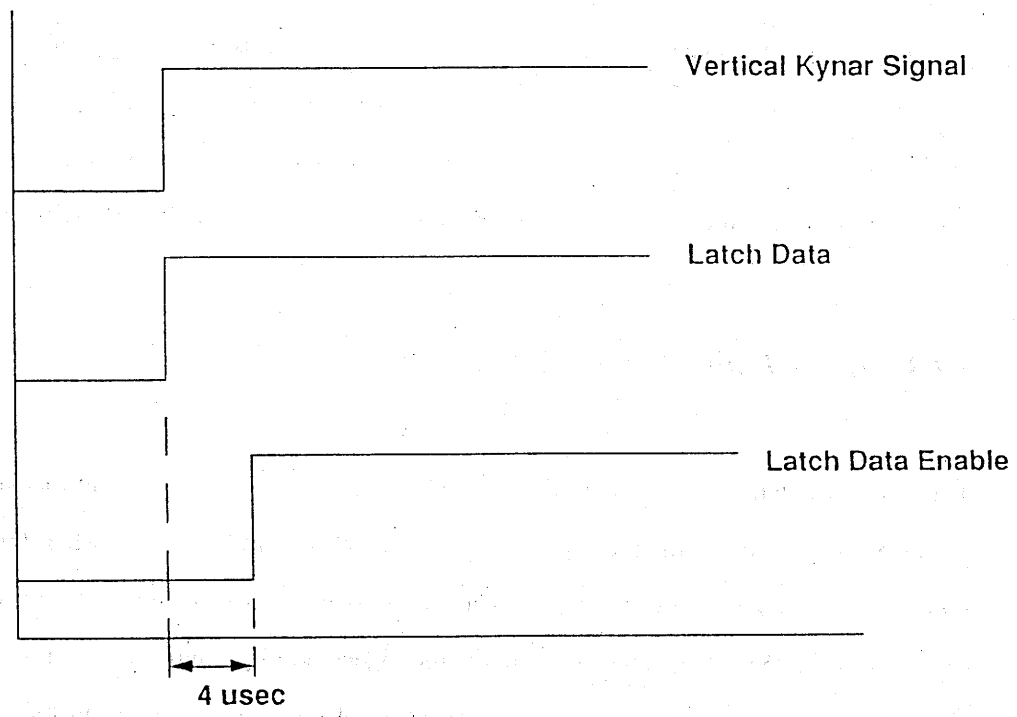


Figure 15. Timing Signals for Vertical Kynar, Latch Data and Latch Disable

sensitivity. The second method of adjustment involves changing the amplifying resistor (R4, Figure 9) values. A combination of both adjustments provides sufficient means of isolating false triggers electrically.

5.4.4 Mechanical Considerations

The sensor grid had to be mounted on a surface that would provide tactile feedback similar to that found on a dulcimer yet dampen any indirect vibrations in order to prevent an incorrect reading. The grid was first mounted on a hard plexiglass surface. This didn't work as well because the surface was a poor source of tactile feedback. The next material chosen was a spongy rubber surface that provided good tactile feedback but allowed false triggers to be registered. Two sensor layouts provided an acceptable compromise between the need to capture as many hammer strikes as possible and the need to dampen sensor sensitivity in order to prevent false triggering.

Figure 16 shows the first three dimensional sensor layout. The vertical kynar sensors are mounted on a 1/4″ foamboard that is glued to 3/4″ sponge rubber. This sponge rubber layer is glued to 3/4″ plywood. The horizontal kynar sensors are mounted on a similarly layered material but they are suspended 1/4″ above the vertical sensors. This was done in order to prevent any vibrations from the vertical sensor from triggering the horizontal sensor. The individual sensors are isolated from each other and are glued to the heavier plywood base. This layered approach to isolating the sensors, in conjunction with the 1M potentiometers appear to provide the best arrangement for isolating the sensors from external stimuli.

Figure 17 depicts the second three dimensional sensor layout. The vertical sensor strips are mounted on a 1/4″ plywood board with cutouts. The horizontal sensors are mounted on a separate 1/4″ plywood surface and are aligned so that the "intersection" of the horizontal and vertical sensors occurs at the cutouts of the top board. Sponge rubber plugs are the only things that touche either sensor. The plugs provide the tactile feedback to the musician. Both layers are connected to each

other by machine mount screws. This arrangement provides good isolation from external vibrations, however, it does not provide a wide a range of MIDI key veolicty information. This is due to the fact that the cutout size and horizontal sensor tension limits how far the sensor can move during a hit.

5.4.4.1 Sensor Grid Arrangement

The sensors are arranged in a classical X-Y orientation in order to provide an way of simulating the acoustic hammer dulcimer tuning scheme. Each pair of vertical strips correspond to the notes on either side of a bridge. The bass bridge on the acoustic instrument is played only on the right side, however, the MIDI controller allows notes on either side of this "virtual" bridge. The sensors could have been arranged in a linear fashion where a sensor corresponds to a single note. This scheme has an musical range of 1 octave. The grid arrangement effectively doubles the range of the controller by increasing the range to 2 octaves. Additional software is needed to correctly decode the grid information, however, the 8751 processor is fast enough to handle this extra load.

5.4.5 Suggestions for Future Designs

This section presents some suggestions for modifying the hardware subsystem of the controller. The ADC0808/9 CMOS A/D converter with 8-channel multiplexers is one chip that can be used to handle controllers that offer greater than a two octave musical range. This chip performs A/D conversion on one of eight analog inputs. The primary advantage of this type of converter is the reduction of the chip count for the entire package. Its electrical characteristics are similar to the ADC0804 chip currently used in the system.

The Kynar sensor arrangement can be altered from the current grid setup to whatever setup the instrument designer chooses. Instrument luthiers can arrange the sensors in various configurations

in order to test different tuning schemes for prototype instruments. This allows a luthier to try out different tuning arrangements for a prototype without having to build the instrument first, thereby saving time and money.

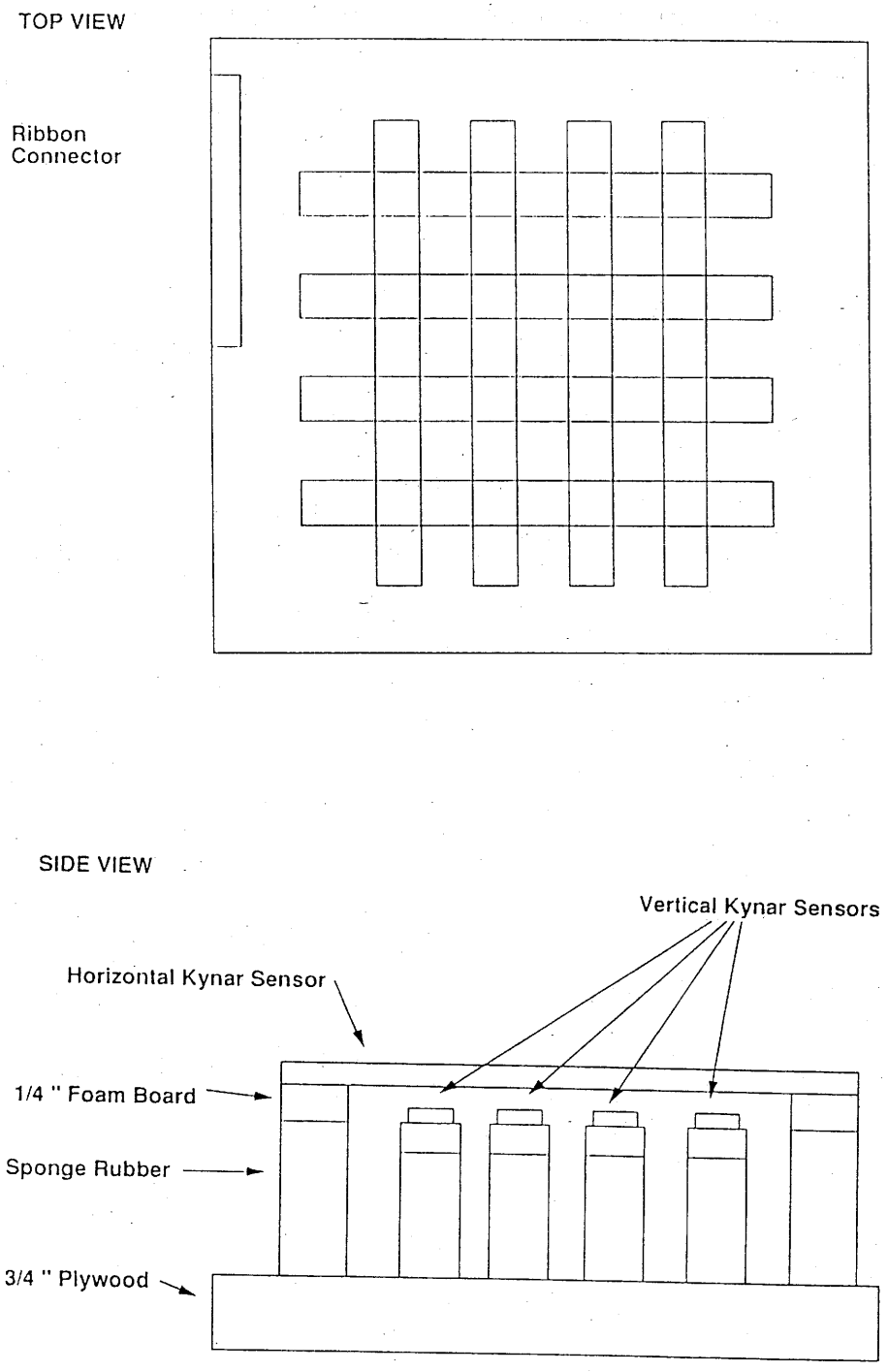


Figure 16. First three-dimensional sensor layout

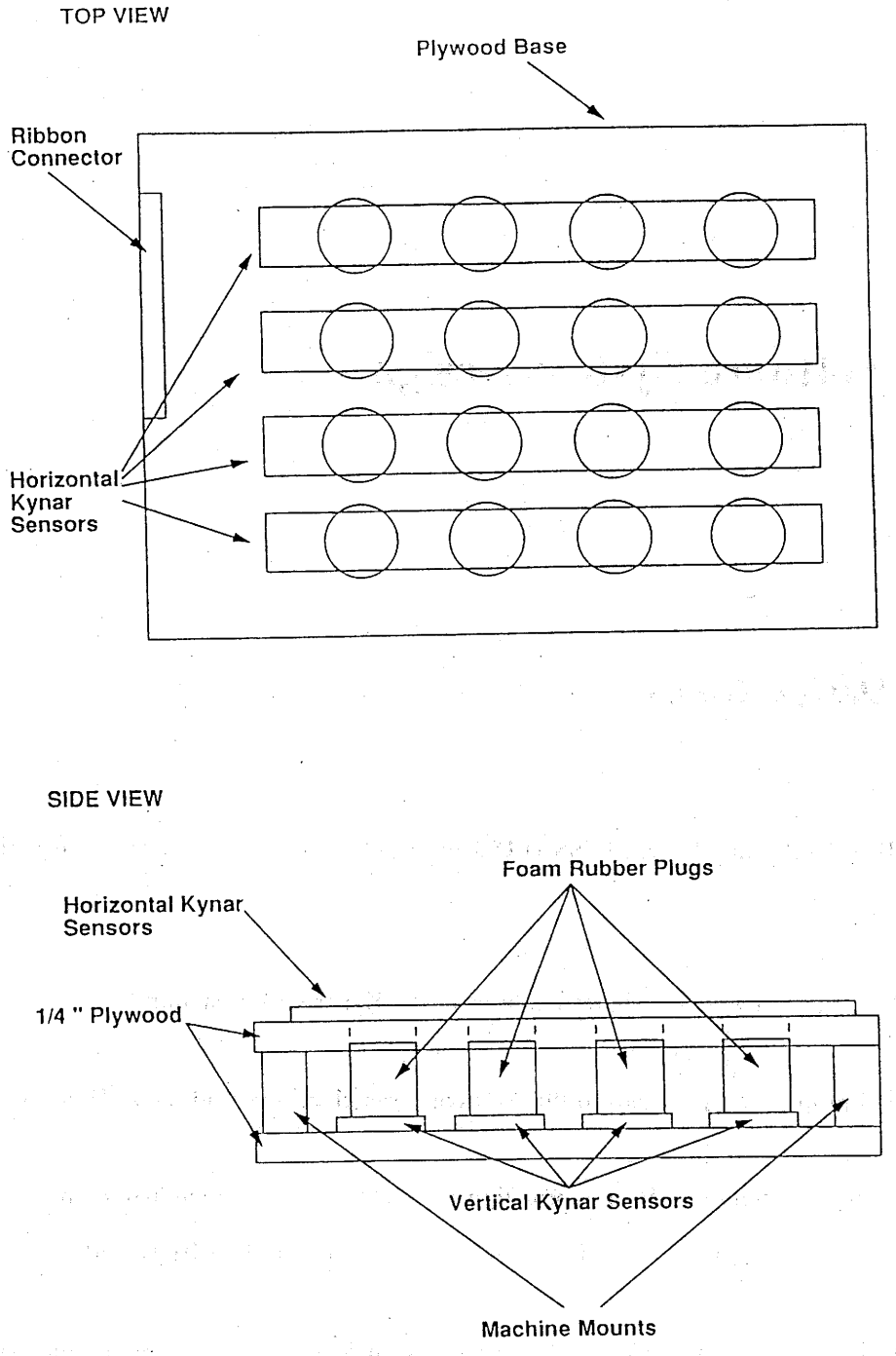


Figure 17. Alternate three-dimensional sensor layout

6.0 Software System Design

6.1 Design Goals

The MIDI No Strings Attached (NSA) Dulcimer software was designed to accomplish the following goals:

1. Derive note and velocity information from the Kynar sensor circuitry.

2. Send proper control signals to the A/D converter chips and read the A/D information.

3. Be able to interpret control information such as pitch bends, modulation and sustain. Translate this information to MIDI data and output it through the MIDI interface.

4. Operate in "normal" mode which translates a single gesture to a corresponding MIDI event.

5. Operate in "emulation" mode where the processor translates the single gesture to a series of MIDI events. This feature is used to emulate the timbre of a tone generated by a hammer strike. It also generates the tones produced by sympathetic resonance.

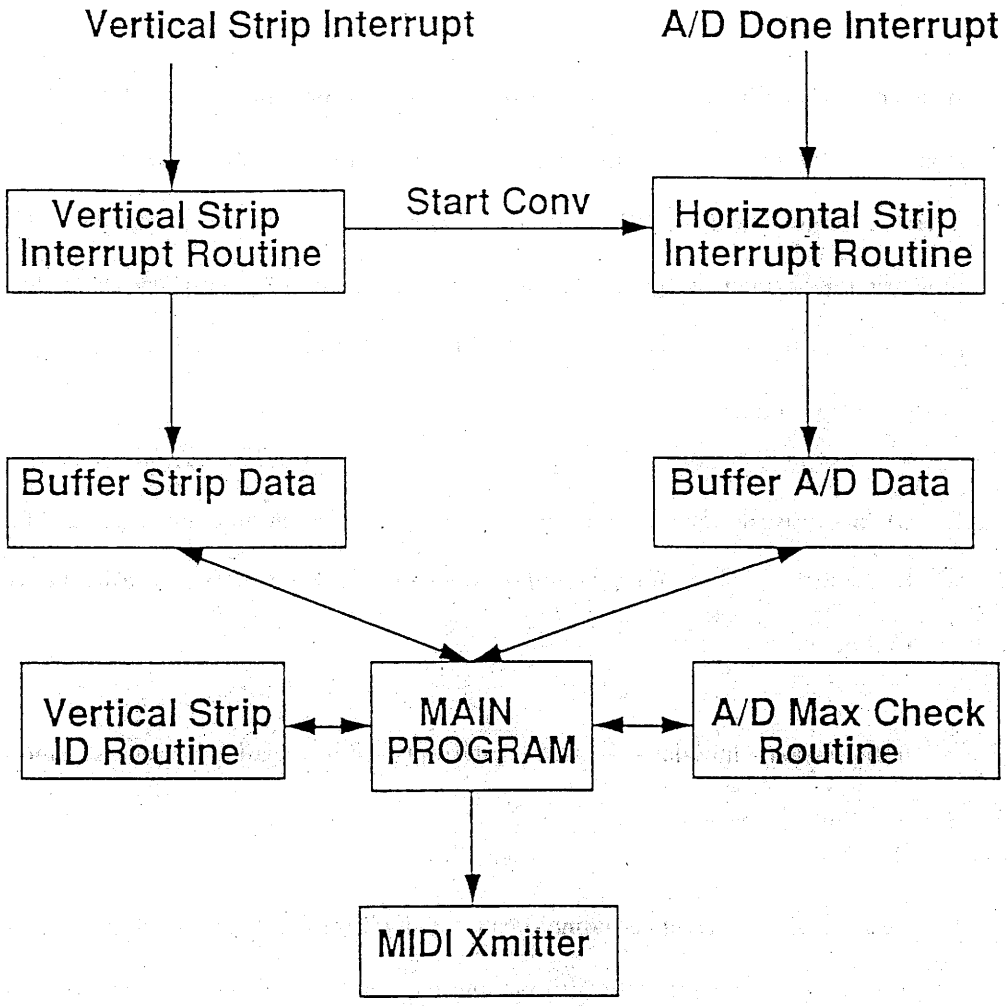


Figure 18. Controller Software Block Diagram

6.2 General Software Description

A diagram of the interaction between the various components of the software controller program is given in Figure 18. Interrupt signals from the hardware subsystem ($\overline{INT0}$, $\overline{INT1}$) instruct the interrupt service routines (ISR) to start A/D conversion of the event data and buffer them for later processing. The main program determines which sensor strips were hit, calculates partial tones if necessary and transmits MIDI messages to the target tone generator. The software is divided into the following modules:

1. **Main** - calls the initialization routine then stays in an infinite loop checking for non-zero data buffer values, determining the indices to the musical note array and calls the MIDI transmitter routine when needed.

2. **Initialization** - initializes the timer, interrupt vector assignments, registers and data buffers for the controller software.

3. **Vertical Strip Interrupt Routine** - reads a value from the latch to determine which vertical strip was hit and stores this value in a circular data buffer. It also pulses the AD804 $\overline{RD}$ line to start A/D conversions of the horizontal strips.

4. **Horizontal Strip Interrupt Routine** - handles the A/D conversion complete interrupt. It reads all of the A/D values and stores them in another circular data buffer. It also delays 4 msec to allow time for the peak detect capacitors to discharge.

5. **Vertical Strip Identifier** - determines which of the vertical sensors was struck and saves the information as a "Y" coordinate into the musical event storage array.

6. **Horizontal Strip Identifier** - searches for the maximum value read in from the horizontal sensors and converts the information to MIDI key velocity data. It also stores the id number of the sensor as an "X" coordinate value into the previously mentioned storage array.

7. **Overtone generator** - calculates the partial tones that would be generated from the primary tone. It transmits these data values after the primary tone data is sent to the tone generator module.

8. **MIDI Transmitter** - This section uses the data value from the Vertical Strip Interrrupt Routine as an index into a musical note table. It uses the data value from the Horizontal Strip Interrupt Handler as the MIDI key velocity information. The routine inserts this data value into the MIDI note message that is to be transmitted. The routine pauses for 3.8 msec and then sends a MIDI **Note Off** command.

6.2.1 Finite State Machine Description

The controller software performs all of the necessary functions such as A/D conversions, table look-ups, MIDI note transmission and reset functions within 20 msec to 50 msecs. This is within the JND time range that two notes played at different onset times are still perceived to occur simultaneously yet the listener can distinguish the individual tones [Rasch, 50, 51]. This requirement is most critical when the software is running in emulator mode because the primary tone **and** its partial tones must sound simultaneously to the listener. Otherwise, each tone will appear as a distinct event leading to a listening experience not envisioned by the musician. Another timing requirement is to perform the previously mentioned functions as soon after the actual event as possible. The faster the response time to an event, the more feedback is returned to the musician.

A finite state diagram of the controller software interrupt processing is shown in Figure 19. The software remains in an "IDLE" state until a hit is detected on a vertical Kynar strip. The software then moves to the "START CONVERSION" state. The four A/D converters are activated to read the values of the horizontal Kynar strips. When the conversions are complete, the driver moves to the "PROCESS DATA" state. Mechanical crosstalk will cause all four strips to generate positive values, but only the sensor that was actually struck will generate the **maximum** value. This value is used as an index into a lookup table for the correct MIDI note. Once the flags are set for MIDI note transmission, the driver moves to the "DISCHARGE" state. The driver sends a DISCHARGE signal to the peak detector circuits in order to prepare to read the next data values. Once this is done, the driver returns to the "IDLE" state.

Figure 21 shows the finite state machine that describes the MIDI transmission process. The software remains in the "IDLE" state while the **X** and **Y** indices into the MIDI note array are being determined. The state machine moves from the "IDLE" to "CHECK Y" state while the interrupt state machine is in the "PROCESS DATA" state. Once the indices have been determined, the machine moves into the "SEND NOTE ON" state, delays long enough to allow the primary tone to develop and then moves into the "SEND NOTE OFF" state. After all pending notes or events are transmitted to the tone generator, the machine moves back to the "IDLE" state.

6.2.2 Detailed Software Description

This section describes in detail the functions performed by the individual controller software modules. The controller software listings are found in Appendix D.

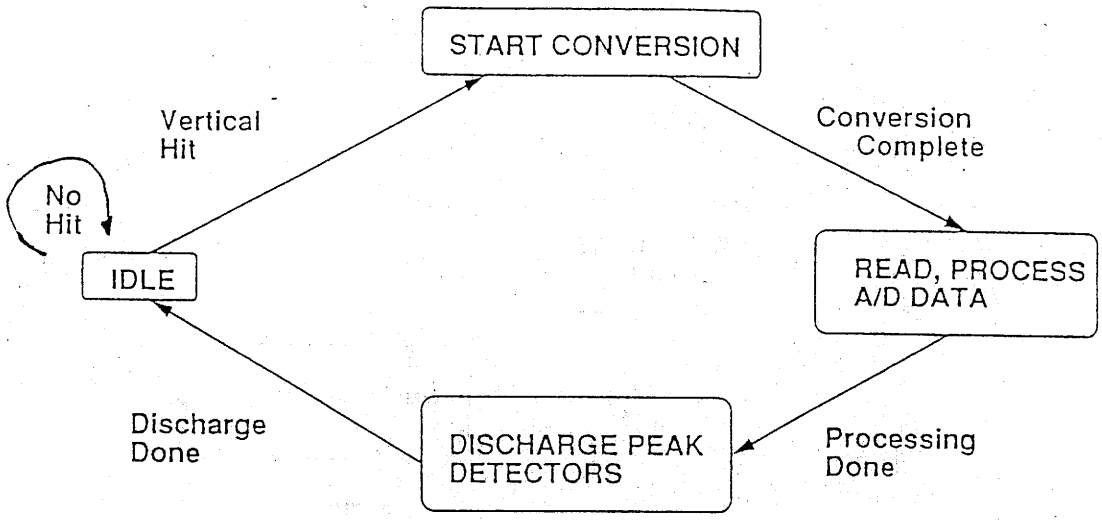


Figure 19. Finite State Definition - Interrupt Processing

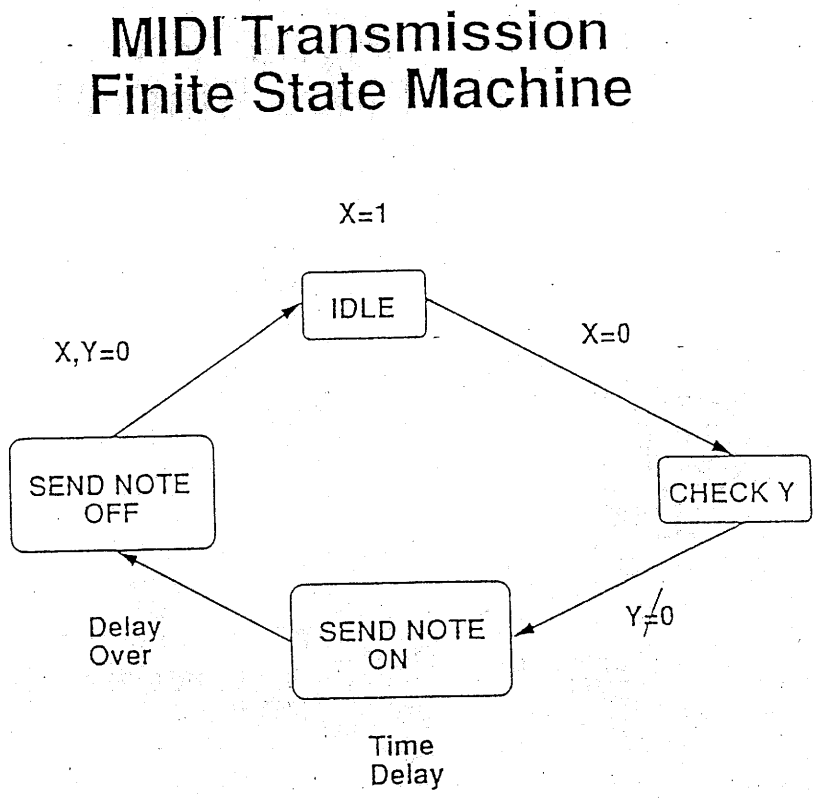


Figure 20. Finite State Definition - MIDI Note Transmission

6.2.2.1 Initialization

The routine INIT initializes the 8751 processor interrupt vectors, timers, serial port, registers, RAM variables, and interrupt priority masks. It pulses the $\overline{RD}$ line on the A/D chips to ensure that the devices are initialized correctly. Timer 1 is turned off by setting the timer control word, TR1, to 0. Timer 1 is thus configured as an 8-bit auto-reload counter by setting TMOD = 20H. The counter will reload the value 0FFH (set by TH1 = 0FFH) into TL1 each time it overflows. Timer 1 will increment its TL1 value by 1 at the end of a machine cycle which is 1 microsecond for an 8751 processor operating at 12Mhz. The 12 Mhz crystal speed was chosen to allow the processor serial port to operate at the MIDI specification of 31.2K baud.

The Serial Port Control Register is set to 42H. This configures the serial port to operate under the following parameters:

1. Mode 1 - 8-bit UART

2. Transmit only

3. 10-bit transmission frame consisting of 1 start bit, 8 data bits and 1 stop bit.

The serial baud rate is determined by the Timer/Counter 1 overflow rate. This baud rate is calculated as follows:

1. Bit SMOD of register PCON = 0 so n = 32.

2. Timer/counter overflow rate = count rate /(256 -TH1). The count rate is equal to 1/12 the oscillator frequency so that the count rate = 12MHz/12 = 1Mhz. 256 - TH1 = 256 - 255 = 1 so the T/C overflow rate = 1MHz/1 = 1MHz. Now, the baud rate = T/C overflow rate/n = 1MHz /n = 31250 baud where n was caculated in step 1. This is the baud rate specified for use in MIDI tranmission.

External 8751 interrupt signals, $\overline{INT0}$ and $\overline{INT1}$ are enabled by setting the Interrupt Enable register, IE = 05H. All other interrupts are disabled. $\overline{INT1}$ is given higher priority since it is connected to the vertical kynar strip circuitry and this event signals the start of the A/D conversion process. The interrupt signals are set to be level activated by setting TCON = 0. This means the interrupt source has to hold the signal level until the interrupt is generated in the microprocessor. Subsequently, the source must deactivate the signal before the interrupt service routine completes in order to prevent false interrupt handling.

6.2.2.2 Vertical Strip Circuit Interrupt Handler

The Kynar vertical strip circuitry generates an interrupt signal on the 8751 $\overline{INT1}$ interrupt line whenever a hammer strike occurs. A minimum of four machine cycles elapse between the $\overline{INT1}$ LOW signal and the execution of the first instruction of this ISR. The Interrupt Service Routine (ISR) enables the 74LS75 latch to save which vertical strip was hit. It does this by clearing port P3.6 (74LS75 **latch enable**). The A/D $\overline{WR}$ signal is set LOW to prepare for an A/D conversion by clearing port P3.0. The ISR delays a three milliseconds to debounce the Kynar signal, decode and store the latch value in a circular buffer in the 8751 internal RAM. The A/D $\overline{WR}$ and latch enable signals are then brought HIGH by setting ports P3.6 and P3.0 to start A/D conversion and reset the latch, respectively. The detailed sequence of steps taken by the ISR are:

1. Save current registers.

2. Clear port P3.6 (latch enable).

3. Switch to register bank 1.

4. Increment an event counter (register 2) and set the A/D $\overline{WR}$ signal LOW by clearing port P3.0.

5. Delay 3 msec to debounce the Kynar signal.

6. Read the latch data, strip off the low nibble and swap bytes.

7. Store the data in the circular buffer pointed to by register 0. Check to make sure the buffer pointer isn't out of bounds and reset it to the beginning of the buffer if it is.

8. Set port P3.6 (latch enable) and port P3.0 (A/D $\overline{WR}$). This starts the A/D conversion.

9. Return from Interrupt

Register 0 in Register Bank 1 points to the next available position in the vertical id buffer. The MIDI transmit routine uses this pointer to calculate the MIDI note number to be sent.

6.2.2.3 Horizontal Strip Circuit Interrupt Handler

The Kynar horizontal strip circuitry feeds a voltage to a peak detection circuit that holds the maximum voltage (shown in point A in Figure 8) of the input signal for the ADC804 A/D converter chip. The routine is invoked whenever the 8751 $\overline{INT0}$ line is activated by going LOW. The A/D chips drop their $\overline{INTR}$ signal LOW when the chip is finished with an A/D conversion. A combinatorial AND circuit ensures that the 8751 interrupt signal is not activated until all of the A/D conversion done signals are LOW. The ISR then reads each of the A/D values and stores the values in a circular buffer in the 8751 internal RAM. The ISR discharges the peak detection capacitors by pulsing port P3.7 HIGH, delaying 4 msec and dropping port 3.7 LOW. The $\overline{INT0}$ signal is disabled approximately 5 msec. The detailed steps taken by the ISR are:

1. Save registers.

2. Switch to register bank 2.

3. Initialize the strip counter.

4. Select the A/D to be read.

5. Read the A/D data, store in the buffer position pointed by register 1.

6. Perform a bounds check on register 1 and reset it to the beginning of the buffer if needed.

7. Repeat steps 4-7 until all A/D converters are read.

8. Set port 0 (A/D $\overline{RD}$) to change the A/D $\overline{INTR}$ signal HIGH and port 3.7 to discharge the peak detect circuit capacitors.

9. Delay 4 msec to allow time for the capacitors to discharge.

10. Clear port 3.7 to reenable the peak detect circuits.

11. Set DONE flag for the main routine.

ARRAY ID	NOTE	HEX VALUE
(1,1)	A5	51
(1,2)	B5	53
(1,3)	C#5	55
(1,4)	D6	56
(2,1)	D5	4A
(2,2)	E5	4C
(2,3)	F#5	4E
(2,4)	G5	4F
(3,1)	G4	43
(3,2)	A4	45
(3,3)	B4	47
(3,4)	C5	48
(4,1)	Middle C4	3C
(4,2)	D4	3E
(4,3)	E4	40
(4,4)	F4	41

Figure 21. MIDI note array (version 1) using Standard Music Note Numbers

6.2.3.4 MIDI Transmitter Routine

The MIDI transmitter uses the information collected by the two ISRs to determine which musical note to send to the synthesizer. The vertical ISR provides the x coordinate and the horizontal ISR provides the y coordinate information used in the table lookup. Figure 21 shows this X-Y mapping to the MIDI note lookup table. The table index is found by performing the following calculation, given a matrix M x N, starting at address BASE: **Address of element(x,y) = BASE = (n * x)** + y. The array indices are scaled to account for the address structure starting at 0.

The routine then inserts the MIDI note value, and the key velocity value into a preformatted MIDI voice channel note on message and sends it out the serial port to a synthesizer. The software delays 3.8 msec and sends out a MIDI note off message. After resetting the RAM data values, the routines returns to the main loop. If the controller is in emulator mode, this routine calls the timbre emulator to calculate and transmit the partial tones.

6.2.3.5 Vertical Strip Identifier Routine

This subroutine determines which vertical sensor was struck. The vertical strip ISR reads the latch value and saves it in the circular data buffer. This routine gets the latch data from the circular buffer in the 8751 internal RAM. The detailed sequence of steps is:

1. Read the data byte from the internal RAM circular buffer.

2. Initialize a bit position counter, register B.

3. Rotate the data byte through the Carry bit.

4. If the Carry bit is set, save the index position stored in Register B in RAM variable, I. If not, go to step 3.

5. Return to the main program.

6.2.3.6 Horizontal Strip Identifier Routine

This subroutine finds the maximum of the values read in from the A/D converter chips and stores it in an internal RAM location for use by the MIDI transmit subroutine. The routine performs the following steps:

1. Set the loop counter equal to the number of A/D converters.

2. Get a data value from the internal RAM circular buffer.

3. If it's the maximum value so far, store it in a temporary internal RAM location. If not, go to step 2.

4. If all the A/D converter values have been checked then return to the main routine. Otherwise, go to step 2.

6.2.3.7 Timbre Emulator

The timbre emulator determines the partial tones generated by the primary tone. It transmits these partial tones instead of the fundamental tone. These five notes are output to the synthesizer with a 320 μsec delay between notes. This delay is due to the MIDI baud rate and the actual amount of time it takes to transmit a note to the tone generator. The partial tones are a minimum of five tones which are the octave, fifth above octave, double octave, third and fifth above octave tones above the fundamental tone. For example, if the primary tone is a middle C (C4), the corre-

sponding partial tones are the C that is one octave higher (C5), a fifth interval above this note, a G (G5), the C two octaves above the primary tone (C6) and the third and fifth intervals above this note, E (E6) and G (G6). The loudness or key velocity of these partial tones is assigned arbitrarily since the actual notes are not struck. By experimenting with these velocity values, the designer could attempt to influence the tonal quality of the resultant sound that is created by the tone generator. Adjusting the loudness of the partial tones relative to the primary tone, following Fogel's rules on tonal quality (mentioned in Chapter 4) results in dramatic differences being heard. A musical octave consists of twelve tones so the routine adds twelve to the indices in order to calculate the first harmonic (octave tone). It adds 7 to these numbers to find the second harmonic (fifth above octave), 24 to get the third harmonic (double octave), 3 to find the fourth harmonic (third above double octave) and 2 to get the fifth harmonic (fifth above double octave). It then transmits these five notes delaying 320 μsec between transmissions. The notes are then turned off at the end of the delay period specified by the MIDI transmitter routine.

6.2.3.8 Controller Main Routine

The main routine sets up the interrupt structure, determines the vertical and horizontal strips ids and calls the MIDI transmitter. In emulation mode, the routine will call the timbre emulator to calculate the MIDI note numbers of the overtones and transmit them to the synthesizer. The detailed sequence of steps is:

1. Initialize all system registers.

2. Initialize the timers, serial port baud rate and interrupt priority structure.

3. Switch to register bank 1 and see if a vertical hit has happened. If so, save registers 2 and 6 from register bank 1 and switch to register bank 2.

4. Check the A/D DONE flag (set by the A/D ISR).

5. Call the vertical, horizontal identifier routines to get the X-Y coordinates. Do this for all the data in the circular buffer.

6. Go to step 3.

6.3 Discussion

The timing and responsiveness of the controller software must fall within acceptable JND tolerances in order to correctly translate the "feel" of the performer to MIDI events. It must initiate a MIDI event as soon as possible after the performance gesture such as a hammer hit or control change. Stewart's study on the effect of responsiveness on the "feel" of a musical piece was one of the first attempts to quantify subjective musical terms such as **snap**, **groove** and **in the pocket** to the actual amount of time the hit occurred around a musical beat [Stewart, 13]. He demonstrated that a **snap** occurs about 5-10 msec before the actual beat and that **in the pocket** occurs approximately 10 msec after the actual beat. He determined the boundary limits on either side of the beat that define the acceptable range of responsiveness. In general, an event that occurs greater than 25 msec after the beat will cause the music to sound as if it were "dragging". This suggests a maximum time limit for the responsiveness of the controller. The controller software must generate a MIDI **Note On** command no later than 25 msec after a hit in order to keep the music from "dragging". Stewart's results are shown in Table 5.

Interrupts are disabled as long as the ISR routines are active, therefore, this "down" time must be kept to a minimum. The ISRs simply grab the data and place them in two circular buffers that can be accessed during the idle state of the software for processing. Figure 22 shows the relationship between the circular buffers and the X,Y coordinate method of identifying the horizontal and vertical sensors that were hit. The sensor strips are arranged in a grid as shown in Figure 5. The **X** and **Y** values map to the horizontal and vertical strips respectively. Each **X** is matched with four possible

Time (msec)	Description
-40	"Get some sleep"
-20	Dragging
-10	In the pocket
-5	Groove
0	The precise beat point
+ 10	Snap
+ 20	Drive
+ 35	Nervous
+ 40	Unacceptable

Table 5. Musical Feel at 130 BPM (from [Stewart, 13])

Y values. The maximum of these four values identifies the horizontal sensor that was struck. The identification process is carried out by the main routine. Since the interrupt disabled time is kept to a minimum, the controller software avoids the pitfalls described in the previous paragraphs because. The 8751 is quite capable of keeping up with the fastest musical playing gestures. The only timing constraints are in the delay loops for debouncing the Kynar strips and discharging the peak detect circuit capacitors. These delay times can be modified to determine the optimum value.

The 8751 maintains state information for each of the interrupt events by saving this information in different register banks. The processor switches context by simply switching register banks. This method avoids having to save pointer information on the stack or in RAM. This context switching allows for post-event processing by routines called by the main driver code. For example, when the controller operates in emulation mode, it outputs the partial that was struck followed by 4 other notes corresponding to the partial tones that form the compound tone of the primary note. Since it is not known in advance which note was hit, this event information has to be saved long enough to calculate and transmit the partial tones. Storing the buffer pointers in different register banks allows processing to be done asynchronously. Earlier versions of the software processed each event as it occurred and thus did not allow for any type of post-processing of the event. Table 8 describes the functions of the registers in each register bank.

Register Bank 0	Description
0	Ptr to next available Vertical Strip buffer slot
1	Ptr to next available A/D buffer slot
2	Vertical Strip Event Counter
3-4	Unused
5	Set equal to R0
6	Set equal to R1
7	Save Area for max A/D value
Register Bank 1	**Description**
0	Ptr to next available Vertical Strip buffer slot
1	Unused
2	# times a vertical strip was hit
3-4	Used by timing loop
5	Save area for latch data
6	Current ptr to Vertical strip buffer
7	Unused
Register Bank 2	**Description**
0	Unused
1	Ptr to next available A/D buffer slot
2	# of A/D's to read
3-4	Used by timing loop
5	Current ptr to buffer
6	Unused
7	A/D Done flag for Main Program

Table 6. NSA Software Register Bank Definitions

6.3.1 The Synthesizer Management Model

Anderson proposed a model for tone generator control that is based on a synthesizer manager (SM) that provides device independent mechanisms for controlling synthesizers. The SM provides support for features such as priority based voice allocation, note starts and ends and logical sustain pedals [Anderson, 2]. He defines **Event Routines** as the routines that "are called at music performance time to play and manipulate sustain pedals." He also states that the SM functions in a manner similar to a computer operating system in that it translates a user request/gesture to a device specific function. The high level requests can be the playing gestures and can be kept the same regardless

Vertical & Horizontal Strip Circular Buffers

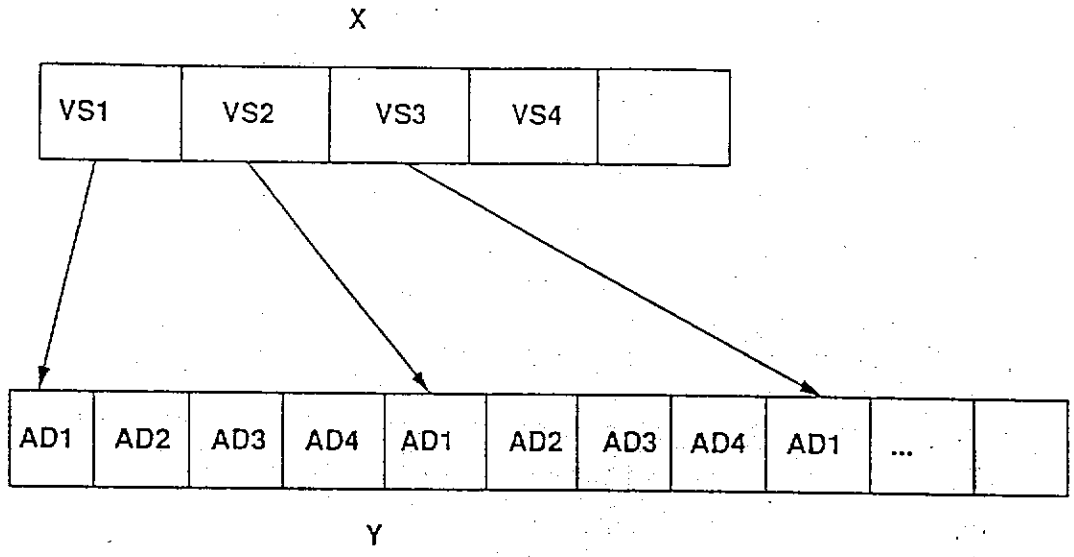


Figure 22. Circular Buffer: X,Y mapping

of the synthesizer type so that the musician or programmer. This virtual interface layer is usually called the **beautification principle** [Peterson, et al., 35].

The NSA MIDI transmitter routine takes note information derived from the sensor grid and outputs it to a tone generator located in a synthesizer. Therefore, the routine can be describes as an Event Routine. The main program performs functions such as determining which sensors were struck, generating overtones, determining MIDI control information and starting a MIDI transmission. The main program can be defined as an SM, albeit a primitive one. The more advanced features in Anderson's model such as priority voice allocation, note preemption scheme and logical sustain pedals are not implemented at this time. However, future releases of the NSA software will incorporate these features into its design.

Modern synthesizers have a finite number of tone generators in order to give them polyphonic capability, i.e., the ability to play more than one note at a time. Typically, the synthesizer editor software will assign a tone generator to a note. The problem with this arrangement occurs when there are more notes to be played than there are tone generators. The editor software then decides which notes should be turned off in order to free a tone generator for the next note. Anderson points out that the note preemption policy considers "the oldest note, ...the softest note...or preference to outer voices." This problem can occur with the NSA controller if the voice selected has a longer decay time than the time between notes struck. The synthesizer will arbitrarily turn off a note in order to free a resource to play the next note. As Anderson points out, this may not be the musically correct decision. The NSA MIDI main program could be modified to adhere to Anderson's cost function, C(X), calculations. He defines the rules to be:

- "If X has the correct patch parameters and a free voice, then C(X) is negative infinity.

- If X has the correct parameters and no free voices, then one note must be preempted in order to play the new note on X. Hence C(X) is the minimum of the priorities of the notes currently sounding on X.

- If X has the wrong parameters and has no notes currently sounding, C(X) is zero.

- If X has the wrong parameters and has notes sounding, then all these notes must be preempted in order to play the new note on X. Hence C(X) is the maximum of the priorities of these notes."

We can see that C(X) determines where and when a note is played.

At present, the NSA MIDI software operates in a discrete event mode where a note is turned on then off before the next note is turned on. Note duration times were calculated using the reverberation formula in Chapter 4. The buffering capability of the software allows asynchronous MIDI **Note On/Off** sequences, however, this feature is not implemented at this time. The controller software was designed using Anderson's design guidelines. The software functions at the lowest level but it is written in a modular fashion allowing relatively easy layering of higher level functions.

7.0 Future Enhancements

The flexability of the controller allows for a variety of areas in MIDI controller design to be explored. The controller was designed to function as an instrument in a live concert setting. To function in this capacity, it had to be responsive to different musicians' touch. As an instrument, its capabilities are limited by the equipment owned by the musician. The ability to add more modules to increase the range of the instrument requires some modification of the controller software. The Kynar strips are one method of designing a sensor subsystem. Infrared sensors can be used to eliminate some of the false triggers that bedevil the Kynar sensors. Magnetic pickups installed on an acoustic hammer dulcimer give the musician the capability of playing both acoustically and through a synthesizer. The controller allows an instrument designer to concentrate on different sensor subsystems.

Another interesting application of the controller involves designing instrument tuning layouts. Luthiers can arrange the sensor subsystem in whatever manner they choose. The sensor were arranged in a 4x4 grid to emulate the tuning of a standard fifth-interval hammer dulcimer. However, the ability to "retune" the sensor under software control allows the musician to play in whatever tonal structure they want. Luthiers can test out note placement by rearranging the sensors in whatever fashion they want. This allows them to test these placements without having to build an instrument first.

8.0 Summary

This thesis discusses the design considerations used to construct an electronic MIDI instrument controller emulating an instrument called a hammer dulcimer. The flexibility of the controller allows for a variety of areas in MIDI controller design. The ability to modify instrument design parameters such as reverberation time calculations to determine optimal MIDI "note on" times, adjusting software performance to fall within JND tolerances, optimizing controller responsiveness to a musical event, incorporating a dynamic retuning feature and sensor design allows the builder to easily adjust the "feel" of the controller [Marchany, et al., 1].

The controller meets the original design criteria described in Chapter 1. It interprets the X-Y grid information derived from the sensor grid. Velocity information can be derived, however, the sensors can only move about 1/4" in order to contact the vertical strip due to mechanical limitations. Since the Kynar sensor generates a signal proportional to the distance moved, the digitized signal varies only slightly. This number when translated to MIDI velocity information produced no audibly noticeable difference in the loudness of the tone. Sustain information is determined by values calculated using the reverberation formula described in Chapter 4. Voice switching commands are presently done by setting a DIP switch. The prototype is designed to accept pitch bend information but this feature is not implemented at this time.

The controller's ergonomic features can be adjusted to fit a particular musician's playing style. The sensor pads are adjustable within limits and can be spaced to match a particular acoustic instrument's string spacing. The sandwich-style layer backing provides adequate feedback to the player. One of the limitations of the foam backing used in the prototype is the noise generated when the hammer stikes the surface. This noise could be distracting when performing in a small setting. Adding a thin rubber layer to mute the contact noise should be sufficient to eliminate this problem. The controller operates well within the time parameters defined in Pennycook's timing hierarchy. It does a very good job capturing and translating into MIDI information, the instantaneous sound control parameters and provides good response time to fast hammer strikes. It does not capture hits that occur within 5 msec of each other because of signal debouncing delays introduced in the software. These delays can be modified with further experimentation in order to increase the responsiveness of the unit.

The controller does not resemble the acoustic version of the hammer dulcimer in appearance. The contact points, however, are where a hammer dulcimer player would expect them to be. A mylar cover sheet with contact points marked in a manner similar to that found on the acoustic hammer dulcimer would provide some visual similarity. The interval tuning arrangements of the various forms of the hammer dulcimer such as the cymbalom, santur and hackbrett are not restricted to the fifth interval tuning found on the North American version of the intrument. They can be based on a variety of standard or microtonal tunings. The controller can be programmed to any tuning that can be translated into MIDI information, thus providing increasing the adaptability of the unit to many forms of music. The ability to retune dynamically enables the musician to play in a variety of ensemble settings without having to change hand positions.

The prototype unit uses the INTEL 8751 microprocessor as the control unit and Kynar piezoelectric sensor strips as the triggering devices. The ADC0804 A/D converter chips were used to digitize the input signals. The prototype meets the MIDI protocol requirements defined in Chapter 2. Controller performance was not affected by the limitations in the MIDI protocol discussed by other researchers. The controller is able to generate musical events within the JND tolerances for

distinguishing separate events. The actual playing motion of the instrument can be described as a synchronous series of events, i.e., distinct hammer strikes at a specific period of time. Because these events are not usually modified afterwards (such as pitch bending or modulation), the controller does not have any difficulty in reporting events to the synthesizer faithfully.

8.1 Suggested Improvements

This section details some of the improvements that can be done to increase the versatility of the controller. The debounce timing wait states can be adjusted to increase the responsiveness of the controller software to very fast hammer hits. This can be done by modifying the the register 3 and 4 values in the vertical and horizontal ISRs. The MIDI "note on" time value can be modified by altering the register 3 and 7 contents in the MIDI transmitter routine, XMIT. MIDI Pitch Bend hardware such as a pitch wheel needs to be connected to the controller in order to implement the pitch bend facility. Currently, the controller can select up to eight different tuning schemes, this restriction is due to the three position DIP switch used to select the tuning. The controller software can be modified to cycle through available tunings. This would require 1 bit of control information rather than the present three bits of data. The unused bits could be used to interpret other control information.

The A/D subsystem uses the ADC0804 8-bit A/D converter chip to digitize the input signal from the sensor subsystem. The prototype currently reads data from four horizontal sensors and each sensor requires a separate A/D chip to interpret the data. The ADC0808/9 8-channel multiplexing A/D chip and the ADC0817 16-channel multiplexing chip can be substituted to reduce the wiring complexity and chip count of the unit. The next version of the controller will incorporate these chips in the sensor subsystem. The incorporation of these chips will greatly simplify the hardware design and requires a rewrite of the horizontal sensor ISR to accomodate the different chip control

signals. Reducing the chip count and wiring complexity greatly increases the reliability of the controller unit.

A wide variety of sensor technologies can be substituted in the controller hardware design. Infrared and fiber optic sensor technologies appear to offer the greatest advantages in creating an instrument that closely resembles the acoustic hammer dulcimer. The author is currently investigating a silicon rubber fiber optic sensor that can be stretched in a manner similar to an actual string. If this sensor technology proves feasible then the controller can be built to look exactly like the acoustic version of the instrument. The player would then strike the "strings" and these sensors would then transmit the appropriate information to the processing unit.

Another interesting application of the controller involves designing instrument tuning layouts. After consulting with some hammer dulcimer luthiers, the author discovered that they viewed the device as a modelling as well as a performance instrument. The luthiers can arrange the sensors in whatever manner they choose thereby allowing them to experiment with various tuning layouts **before** building the actual instrument. This results in saving time and materials. In addition, they mentioned the use of the controller as a teaching device for introductory hammer dulcimer students. The small size of the controller and its only requirement it be connected to a MIDI compatible device make it ideal for teaching in a workshop environment.

In summary, the No Strings Attached Hammer Dulcimer is a unique application of electronic instrument design principles. It has proved to be an easily configurable instrument and is beginning to see use as a performance, modelling and teaching device. The use of standard digital electronics makes the device affordable to the general public. It also can serve as a front end to a VLSI sound synthesis system that functions as a compute engine for sound generation in a more complex real-time music system.

9.0 References

1. Marchany, R., Tront, J. G., Sochinski, J. R., "A Programmable MIDI Instrument Controller Emulating a Hammer Dulcimer", *Proceedings of the 1992 International Computer Music Conference*

2. Anderson D., "Synthesizer Management Based on Note Priorities" , *Proceedings of the 1987 International Computer Music Conference*, Computer Music Association, pp.230-236

3. Pressing J., "Cybernetic Issues in Interactive Performance Systems", *Computer Music Journal*, 14(1), 1990, pp. 12-25

4. Snow D., "BEAT-IT: A Drum Sensor Interface for the Atari ST", *Electronic Musician*, 12/88, pp. 87-93

5. Fischer C. "Fun Under Pressure", *Electronic Musician*, 9/91, pp. 80-83

6. Rubine D., McAvinney P., "Programmable Finger Tracking Instrument Controllers", *Computer Music Journal* 14(1), 1990, pp. 26-41

7. Mathews M., Abbott C., "The Sequential Drum", *Computer Music Journal* 4(4), 1980, pp. 45-59

8. Curry M., "ONE-CHIP PROJECT: Breath-Controlled "Expressor"", *Electronic Musician*, 1/89, pp. 82-85

9. Waisvisz M., "THE HANDS, a set of remote MIDI-controllers", *Proceedings of the 1985 International Computer Music Conference*, Computer Music Association, pp.313-318

10. Kuivila R., "Timing Accuracy and Response Time in Interactive Systems", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.327-329

11. Johnstone E., "The Rolky: A Poly-touch Controller for Electronic Music", *Proceedings of the 1985 International Computer Music Conference*, Computer Music Association, pp.291-295

12. Smith M., "Percussive Control of MIDI Synthesizers", *Electronic Musician*, 7/86, pp. 42-45

13. Stewart M., "The Feel Factor: Music with Soul", *Electronic Musician*, 10/87, pp. 57-65

14. Cadoz C., Luciani A., Florens J., "Responsive Input Devices and Sound Synthesis by Simulation of Instrumental Mechanisms: The Cordis System", *Computer Music Journal*, 8(3), 1984, pp. xx-xx

15. Pennycook B., "Computer-Music Interfaces: A Survey", *ACM Computing Surveys*, 1985, pp. 267-289

16. Vorberg D., Hambuch, R., "On the Temporal Control of Rhythmic Performance", *International Symposium on Attention and Performance VII*, 1978,pp.535-555

17. Rubine D., McAvinney, P., "The Video Harp", *Proceedings of the 1989 International Computer Music Conference*, Computer Music Association, pp.xx-xx

18. Yunik M., Borys M., Swift, G.W., "A Digital Flute", *Computer Music Journal*, 9(2), 1985, pp. 49-52

19. Knapp B., Lusted, H., "A Bioelectric Controller for Computer Music Applications", *Computer Music Journal* 14(1), 1990 pp. 42-47

20. McKay B., Wills B., Carr D., "Polyphonic Velocity-Sensitive Keyboard Interface", *Proceedings of the 1980 International Computer Music Conference*, Computer Music Association, pp.583-594

21. Moog R., "A Multiply-Touch-Sensitive Clavier for Computer Music Systems", *Proceedings of the 1982 International Computer Music Conference*, Computer Music Association, pp.601-605

22. Steele D., Wills B., "A Microcomputer-based Keyboard System", *Proceedings of the 1980 International Computer Music Conference*, Computer Music Association, pp.595-606

23. Swift G., Yunik M., "A Microprocessor based Keyboard Instrument for Microtonal Music ", *Proceedings of the 1982 International Computer Music Conference*, Computer Music Association, pp.588-600

24. Fedorkow G., Buxton W., Patel S., Smith K., "A Microprocessor Controlled Clavier", *Proceedings of the 1980 International Computer Music Conference*, Computer Music Association, pp.96-99

25. Peterson J. L., Silberschatz A., "Operating System Concepts", 2nd Edition, Addison Weslet, 1985

26. Gualteri D.M., "MIDI Output Interface to a Parallel Printer Port", *Computer Music Journal* 10(3), 1986, pp. 79-82

27. Loy G., "Musicians Make a Standard: The MIDI Phenomenon", *Computer Music Journal* 9(4), 1985, pp. 8-26

28. Moore F.R., "The Dysfunctions of MIDI", *Proceedings of the 1987 International Computer Music Conference*, Computer Music Association, pp.256-263, *Computer Music Journal* 12(1) 1988, pp. 19-28

29. Kubicky J., "Build a MIDI Interface for Your PC", *Micro Cornucopia*, June-July 1987, pp. 14-19

30. Kubicky, J., "A MIDI Project", *Byte Magazine*, 6/86, pp. 199-208

31. International MIDI Association, "MIDI Musical Instrument Digital Interface Specification 1.0", International MIDI Association

32. Schmidt P., "The Pocket MIDI Controller", *Electronic Musician*, 9/88, pp. 50-59.

33. Clark K., "The EM MIDI Fader", *Electronic Musician*, 2/91, pp.90-101

34. Henry T., "The Connection: a 4-Voice Midi-to-Analog Synthesizer Interface", *Electronic Musician*, 7/89, pp.84-101

35. Clark K., "Merging with the Integrated MIDI Processor(IMP)", *Electronic Musician*, 11/90, pp.72-79

36. Henry T., "The MIDI-to-Trigger Converter", *Electronic Musician*, 6/89, pp. 58-62

37. Henry T., "The One Chip "MIDI Computer": Meet the 68705", *Electronic Musician*, 4/89, pp. 72-75

38. Cantrell T., "Kynar to the Rescue", *The Computer Applications Journal*, 8/91, pp.88-94

39. Rich R., "Buchla Thunder", *Electronic Musician*, 8/90, pp. 94-101

40. Aikin J., "The Light Touch", *Keyboard Magazine*, 6/90, pp.40-46

41. Anderson D., Kuivila R., "Accurately Timed Generation of Discrete Musical Events", *Computer Music Journal*, 10(3), 1986, pp. 48-56

42. Alles H., "Music Synthesis Using Real Time Digital Techniques", *Proceedings of the IEEE*, 68(4), 1980, pp. 436-449

43. Wawrzynek J., Mead C., Lin T., Liu H., Dyer L., "A VLSI Approach to Sound Synthesis", *Proceedings of the 1984 International Computer Music Conference*, Computer Music Association, pp. 53-64

44. Kahrs M., "VLSI and the Design of Real-Time Digital Sound Processors", *Proceedings of the 1980 International Computer Music Conference*, Computer Music Association, pp. 362-373

45. Parks D., "Hardware Design of a Digital Synthesizer", *Computer Music Journal*, 7(1), pp. 44-65

46. DeFuria S., Scacciaferro J., The MIDI Resource Book, Hal Leonard Books, 1987

47. Rona, J., MIDI: The Ins, Outs and Thrus, Hal Leonard Books, 1987

48. Woodrow H., "Time Perception", *Handbook of Experimental Psychology*, ed. S.S. Stevens, 1951, pp. 1224-1236

49. Wuthrich C., Tunks, T., "The Influence of Presentation Time Asynchrony on Music Interval Perception", *Psychomusicology*, Vol. 8 Number 1, 1989, pp. 31-46

50. Rasch R., "The Perception of Simultaneous Notes such as in Polyphonic Music", *Acustica*, Vol.40, 1978

51. Rasch R., "Synchronization in Performed Music", *Acustica*, Vol.43, 1979

52. Licklider J., "Basic Correlates of the Auditory Stimulus", *Handbook of Experimental Psychology*, New York, 1951

53. Singer R., "Motor Learning and Human Performance: An Application to Motor Skills and Movement Behaviors", MacMillan, New York, 1980

54. Fogel R., "Physics, Music Theory and the Hammered Dulcimer", Whamdiddle Music, 1979

55. Hofstetter, F.T., Computer Literacy for Musicians, Prentice Hall, 1988, pp.51-66

56. Moorer J., Grey J., "Lexicon of Analyzed Tones", *Computer Music Journal*, 1(2) 1977, pp. 39-45

57. Anderton, Craig, "20 Great Achievements in Twenty Years of Musical Electronics", *Electronic Musician* 4(7), pp.28+

58. Francois J., Chabot X., Sibler J., "MIDI Synthesizers in Performance: Realtime Dynamic Timbre Production ", *Proceedings of the 1987 International Computer Music Conference*, Computer Music Association, pp.238-240

Appendix A. Summary of MIDI Status and Data Bytes

STATUS BYTE			DATA BYTE	
Hex	Dec	Function	Byte 2	Byte 3
80	128	Note off - Chan 1	Note No.	Note Vel.
81	129	" - Chan 2	"	"
82	130	" - Chan 3	"	"
83	131	" - Chan 4	"	"
84	132	" - Chan 5	"	
85	133	" - Chan 6	"	"
86	134	" - Chan 7	"	
87	135	" - Chan 8	"	"
88	136	" - Chan 9	"	
89	137	" - Chan 10	"	"
8A	138	" - Chan 11	"	
8B	139	" - Chan 12	"	"
8C	140	" - Chan 13	"	
8D	141	" - Chan 14	"	"
8E	142	" - Chan 15	"	
8F	143	" - Chan 16	"	"
90	144	Note on - Chan 1	"	0 = Note off
91	145	" - Chan 2	"	"
92	146	" - Chan 3	"	
93	147	" - Chan 4	"	"
94	148	" - Chan 5	"	
95	149	" - Chan 6	"	"
96	150	" - Chan 7	"	
97	151	" - Chan 8	"	"
99	152	" - Chan 9	"	
99	153	" - Chan 10	"	"
9A	154	" - Chan 11	"	
9B	155	" - Chan 12	"	"
9C	156	" - Chan 13	"	
9D	157	" - Chan 14	"	"
9E	158	" - Chan 15	"	
9F	159	" - Chan 16	"	"
A0	160	Polyphonic Aftertouch - Chan 1	Note No.	Value
A1	161	" - Chan 2	"	"
A2	162	" - Chan 3	"	"
A3	163	" - Chan 4	"	"
A4	164	" - Chan 5	"	"
A5	165	" - Chan 6	"	"
A6	166	" - Chan 7	"	"
A7	167	" - Chan 8	"	"
A8	168	" - Chan 9	"	"
A9	169	" - Chan 10	"	"
AA	170	" - Chan 11	"	"
AB	171	" - Chan 12	"	"
AC	172	" - Chan 13	"	"
AD	173	" - Chan 14	"	"
AE	174	" - Chan 15	"	"
AF	175	" - Chan 16	"	"

STATUS BYTE			DATA BYTE	
Hex	Dec	Function	Byte 2	Byte 3
B0	176	Control/Mode Change - Chan 1		
B1	177	" - Chan 2		
B2	178	" - Chan 3		
B3	179	" - Chan 4	"	" "
B4	180	" - Chan 5	"	" "
B5	181	" - Chan 6	"	" "
B6	182	" - Chan 7	"	" "
B7	183	" - Chan 8	"	" "
B8	184	" - Chan 9	"	" "
B9	185	" - Chan 10	"	" "
BB	186	" - Chan 11	"	" "
BB	187	" - Chan 12	"	" "
BC	188	" - Chan 13	"	" "
BD	189	" - Chan 14	"	" "
BE	190	" - Chan 15	"	
BF	191	" - Chan 16	"	
C0	192	Program No. - Chan 1	Program No.	
C1	193	" - Chan 2	"	
C2	194	" - Chan 3	"	
C3	195	" - Chan 4	"	
C4	196	" - Chan 5	"	
C5	197	" - Chan 6	"	
C6	198	" - Chan 7	"	
C7	199	" - Chan 8	"	
C8	200	" - Chan 9	"	
C9	201	" - Chan 10	"	
CA	202	" - Chan 11	"	
CB	203	" - Chan 12	"	
CC	204	" - Chan 13	"	
CD	205	" - Chan 14	"	
CE	206	" - Chan 15	"	
CF	207	" - Chan 16	"	
D0	192	Channel - Chan 1	Value	
D1	208	Aftertouch - Chan 2	"	
D2	209	" - Chan 3	"	
D2	210	" - Chan 4	"	
D4	211	" - Chan 5	"	
D5	212	" - Chan 6	"	
D6	213	" - Chan 7	"	
D7	214	" - Chan 8	"	
D8	215	" - Chan 9	"	
D9	216	" - Chan 10	"	
DA	217	" - Chan 11	"	
DB	218	" - Chan 12	"	
DC	219	" - Chan 13	"	
DD	220	" - Chan 14	"	
DE	221	" - Chan 15	"	
DF	222	" - Chan 16	"	

STATUS BYTE			DATA BYTE	
Hex	Dec	Function	Byte 2	Byte 3
E0	224	Pitch Wheel Range - Chan 1	Pitch	Pitch
E1	225	" - Chan 2	Wheel	Wheel
E2	226	" - Chan 3	LSB	MSB
E2	227	" - Chan 4	(0-127)	(0-127)
E4	228	" - Chan 5	"	"
E5	229	" - Chan 6	"	"
E6	230	" - Chan 7	"	"
E7	231	" - Chan 8	"	"
E8	232	" - Chan 9	"	"
EC	236	" - Chan 10	"	"
E9	233	" - Chan 11	"	"
EA	234	" - Chan 12	"	"
EB	235	" - Chan 13	"	"
ED	237	" - Chan 14	"	"
EE	238	" - Chan 15	"	"
EF	239	" - Chan 16	"	"
F0	240	System Exclusive		
F1	241	System Common - undefined		
F2	242	Sys Com Song Pos Ptr		
F2	243	Sys Com Song Select (Song #)		
F4	244	System Common - undefined		
F5	245	System Common - undefined		
F6	246	Sys Com Tune request		
F7	247	Sys Com - end of SYSex		
F8	248	System realtime timing clock		
F9	249	System realtime undefined		
FA	250	System realtime start		
FB	251	System realtime continue		
FC	252	System realtime stop		
FD	253	System realtime undefined		
FE	254	System realtime active sensing		
FF	255	System realtime sys reset		

Table 7. Summary of MIDI command bytes

Appendix B: Supplemental References

Lucht C., "Quad Piezo-Electric Drum Trigger", *Electronic Musician*, 7/86, p.71-73

Serra X., "A Computer Model for Bar Percussion Instruments", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.257-262

Yavelow C., "The Impact of MIDI upon Compostional Methodology", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.21-27

van Manen F., "Ringo - A Percussive Installation", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.193-195

Starkier M., "Real-Time Gestural Control", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.423-426

Rossum D., "Some Aspects of Sample Rate Conversion", *Proceedings of the 1985 International Computer Music Conference*, Computer Music Association, pp.119-124

Adrien J.M., Causse R., Rodet X. "Sound Synthesis by Physical Models, Application to Strings", *Proceedings of the 1985 International Computer Music Conference*, Computer Music Association, pp.264-269

Puckette M., "Interprocess Communication and Timing in Real-time Computer Music Performance", *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, pp.43-46

McKay B., Wills B., "Microprocessor-Supervised Digital Synthesizers", *Proceedings of the 1980 International Computer Music Conference*, Computer Music Association, pp.324-336

Chabot X., "Gesture Interfaces and a Software Toolkit for Performance with Electronics", *Computer Music Journal* 14(2), 1990, pp. 15-27.

Swearingen D., "MIDI Programming", *Byte Magazine*, 6/86, pp.211-224

Conger J., "MIDI Programming in C, Part One: MIDI Input and Output", *Electronic Musician*, 9/89, pp. 30-34

Conger J., "MIDI Programming in C, Part Two: MIDI Data Debugger", *Electronic Musician*, 10/89, pp.72-74

Conger J., "MIDI Programming in C, Part Three: Patch Librarian Basics", *Electronic Musician*, 11/89, pp.24-27

Baran N., "Data Acquisition: PCs on the Bench", *Byte Magazine*, 5/91, pp. 145-149

Millman, Halkias, "Integrated Electronics", pp. 570-572

Short K., "Microprocessors and Programmed Logic", pp. 414-425

Mims F., "Getting Started in Electronics", Archer/Radio Shack, 1983

Wentworth K., "But It Worked with My Emulator", The Computer Applications Journal, 8/91, pp. 42-48.

Figueiredo T., "Building High Performance Power Supplies", *Electronic Musician*, 9/88, pp.67-69

Artwick B., "Microcomputer Interfacing", Prentice Hall, 1980, pp. 182-227

Young G., "The Theory and Design of a Multi-intonational Metallophone", Percussive Notes, 26(1), 1987, pp. 44-50

Hurtig B., "The Engineer's Notebook: Twelve Ways to Use Dynamics Processors", *Electronic Musician*, 3/91, pp.66-68

Clynes M., "Secrets of Life in Music: Musicality Realised by Computer", *Proceedings of the 1984 International Computer Music Conference*, Computer Music Association, pp. 225-232

Dannenberg R., "An On-Line Algorithm for Real-Time Accompaniment", *Proceedings of the 1984 International Computer Music Conference*, Computer Music Association, pp. 193-198

Harris C., "A Composer's Computer Music System: Practical Considerations", *Computer Music Journal*, 11(3), 1987, pp. 36-43

Blackwood E., "Easley Blackwood Discovering the Microtonal Resources of the Synthesizer", *Keyboard Magazine*, 5/82, pp.26-38

Keislar D., "History and Principles of Microtonal Keyboards", *Computer Music Journal*, 11(1), 1987, pp. 18-28

Winsor P., DeLisa G., "Computer Music in C", TAB Books, 1991

Fitts P., "Engineering Psychology and Equipment Design", *Handbook of Experimental Psychology*, ed. S.S. Stevens, 1951, pp. 1287-1340

Gregory D., "Using Computers to Measure Continous Music Responses", *Psychomusicology*, Vol. 8 Number 2, 1989, pp. 127-134

McArthur V., "The Use of Computers to Analyze Performance Motions of Musicians", *Psychomusicology*, Vol. 8 Number 2, 1989, pp. 135-141

Gibson D., "An Effective Computer-Assisted Protocol for Music Perception Experiments", *Psychomusicology*, Vol. 8 Number 2, 1989, pp. 191-196

Higgins W., "Resource Guide for Using MIDI in Psychomusicology Research", *Psychomusicology*, Vol. 8 Number 2, 1989

Hoenig, S., "How to Build and Use Electronic Devices Without Frustration, Panic, Mountains of Money, or and Engineering Degree", Little, Brown and Co., Boston, 1980

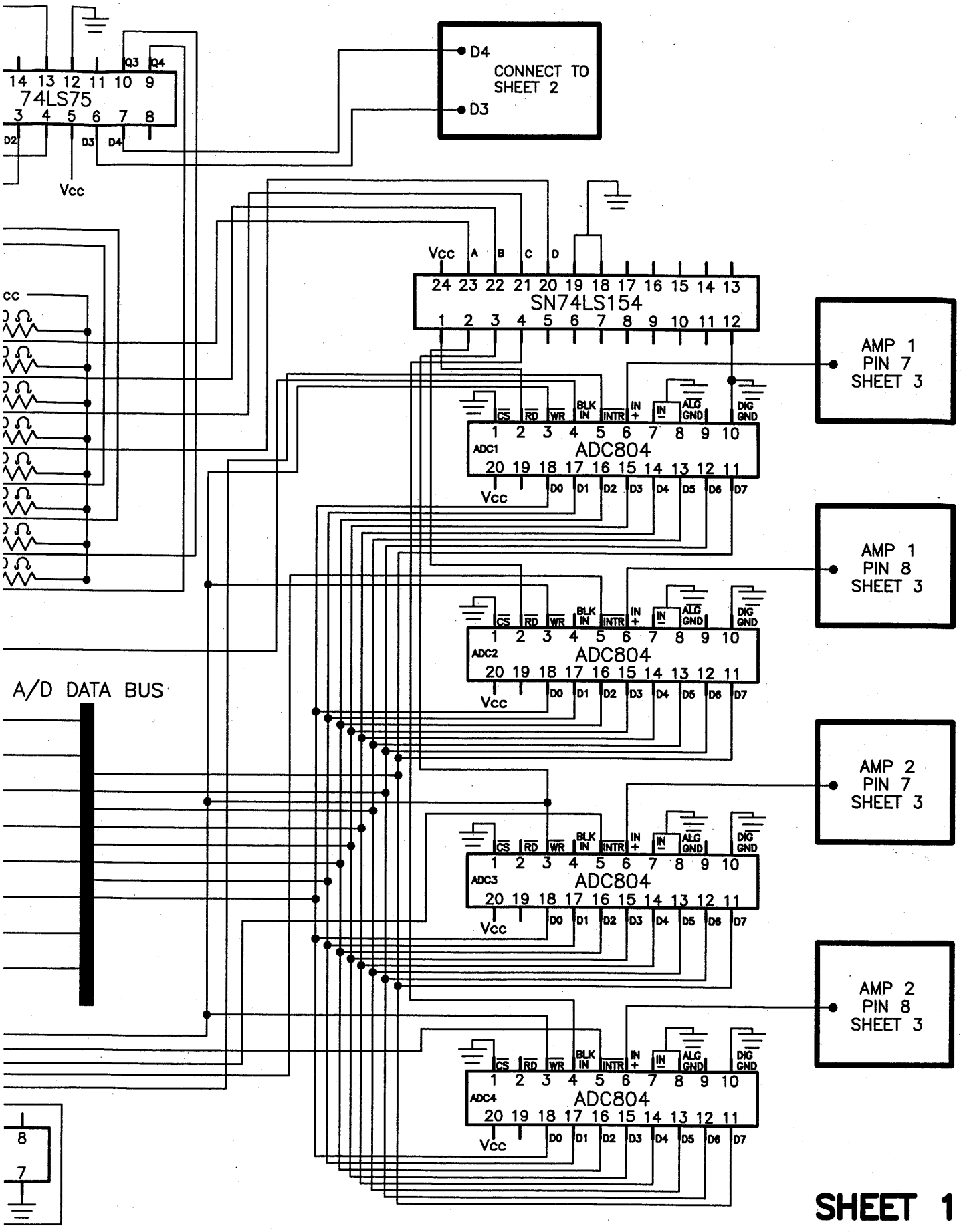
Mims F., "Engineer's Mini-Notebook: Basic Semiconductor Circuits", Siliconcepts, 1986

Mims F., "Engineer's Mini-Notebook: Digital Logic Circuits", Siliconcepts, 1986

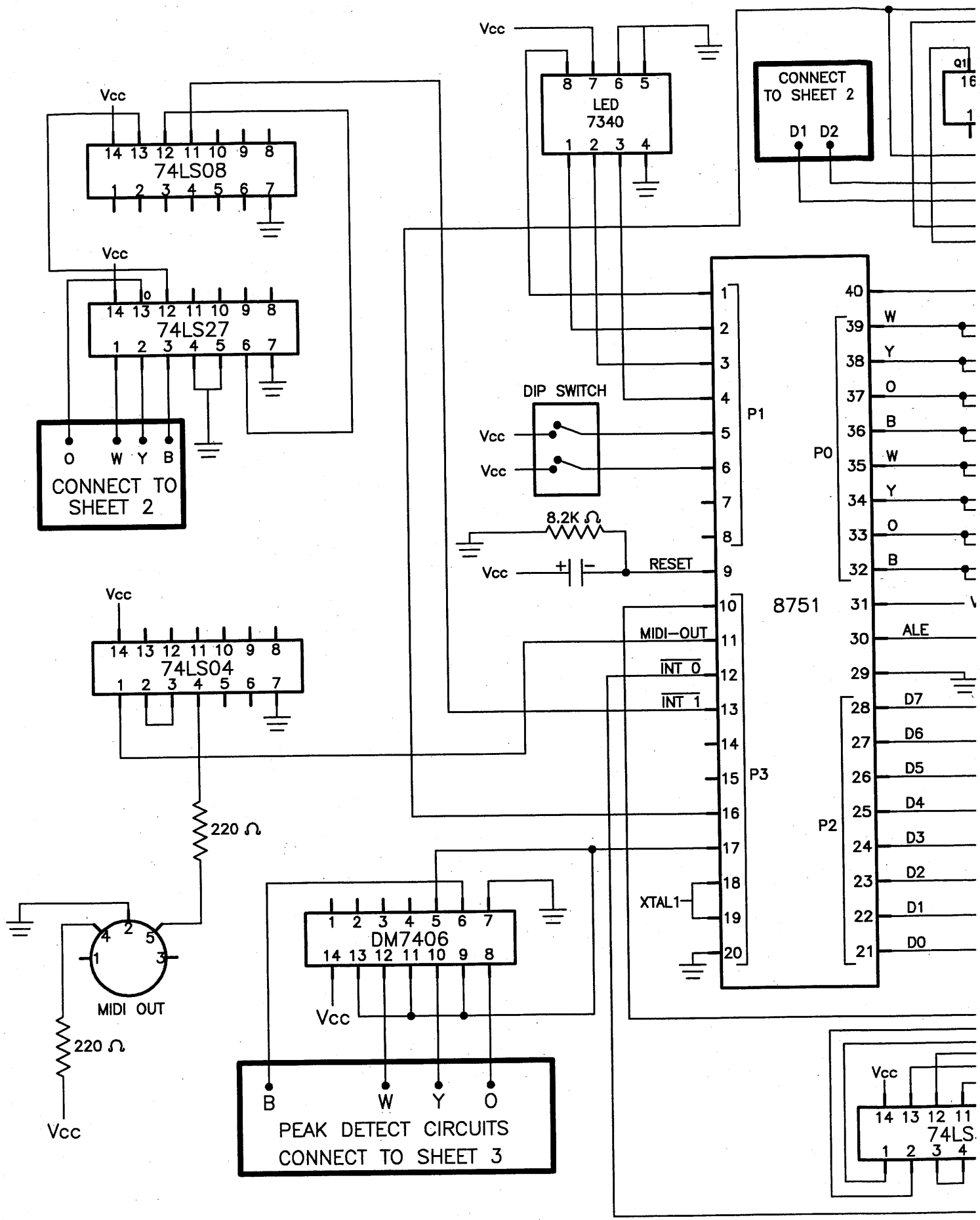
Mims F., "Engineer's Mini-Notebook: Op Amp IC Circuits", Siliconcepts, 1986

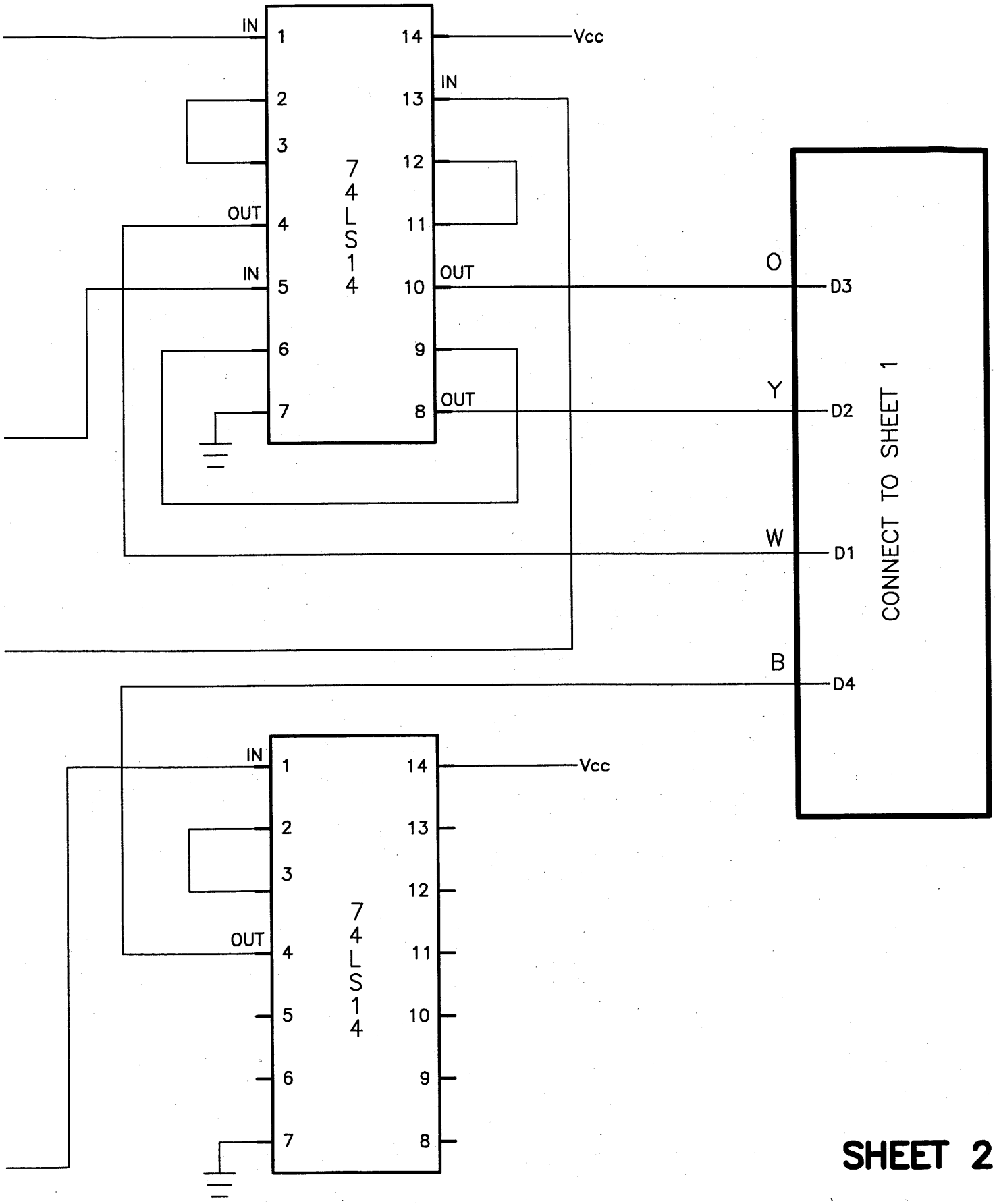
Waverly J., "Digital Design Principles and Practices", Prentice Hall, 1990

Appendix C: Controller Hardware Circuit Diagram

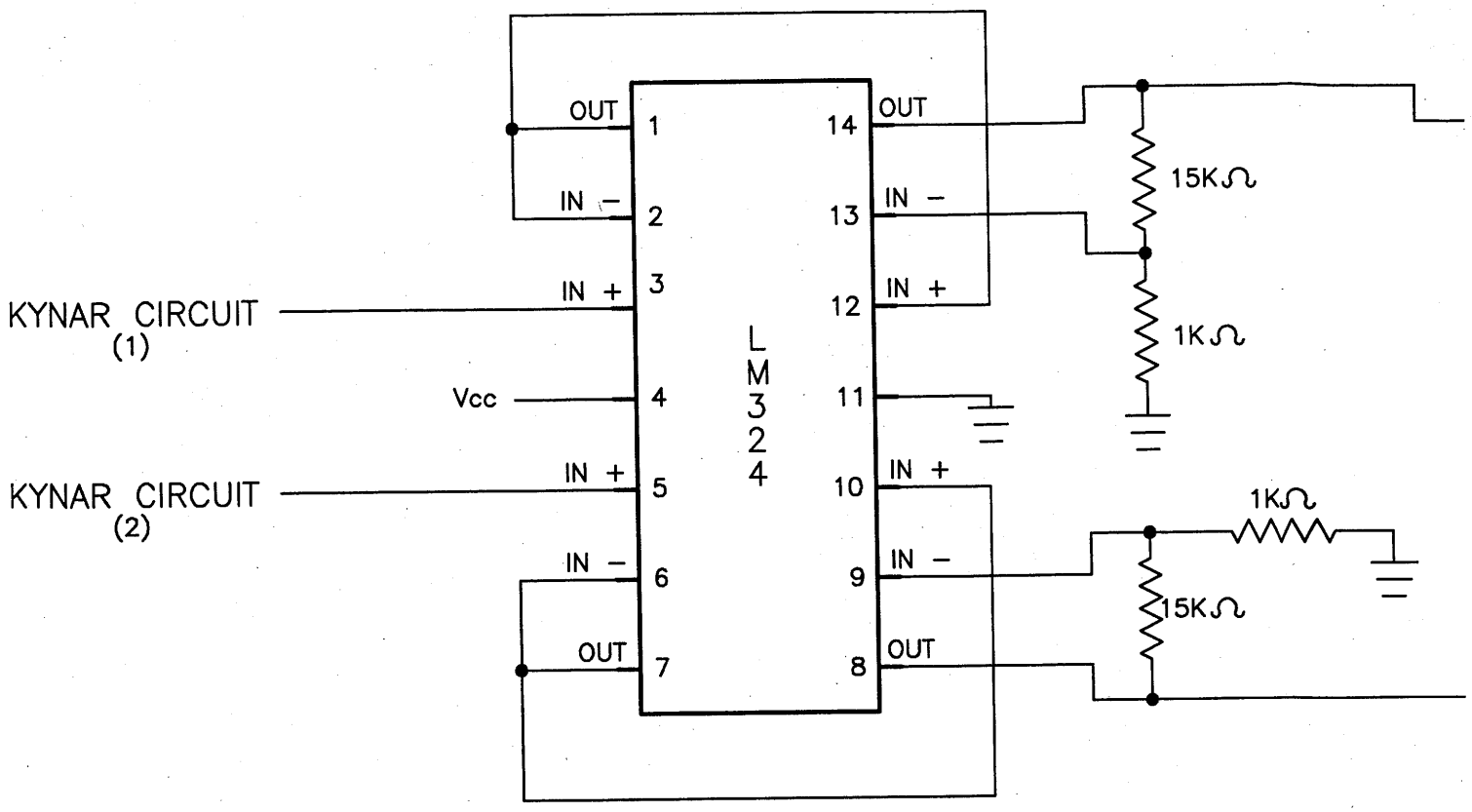

74LS75
14 13 12 11 10 9
3 4 5 6 7 8
Q3 Q4
D2 D3 D4
Vcc
D4
CONNECT TO
SHEET 2
D3
SN74LS154
24 23 22 21 20 19 18 17 16 15 14 13
1 2 3 4 5 6 7 8 9 10 11 12
Vcc A B C D
ADC804
ADC1
ADC2
ADC3
ADC4
CS RD WR BLK IN INTR IN + IN − ALG GND DIG GND
20 19 18 17 16 15 14 13 12 11
D0 D1 D2 D3 D4 D5 D6 D7
AMP 1
PIN 7
SHEET 3
AMP 1
PIN 8
SHEET 3
AMP 2
PIN 7
SHEET 3
AMP 2
PIN 8
SHEET 3
A/D DATA BUS
8
7

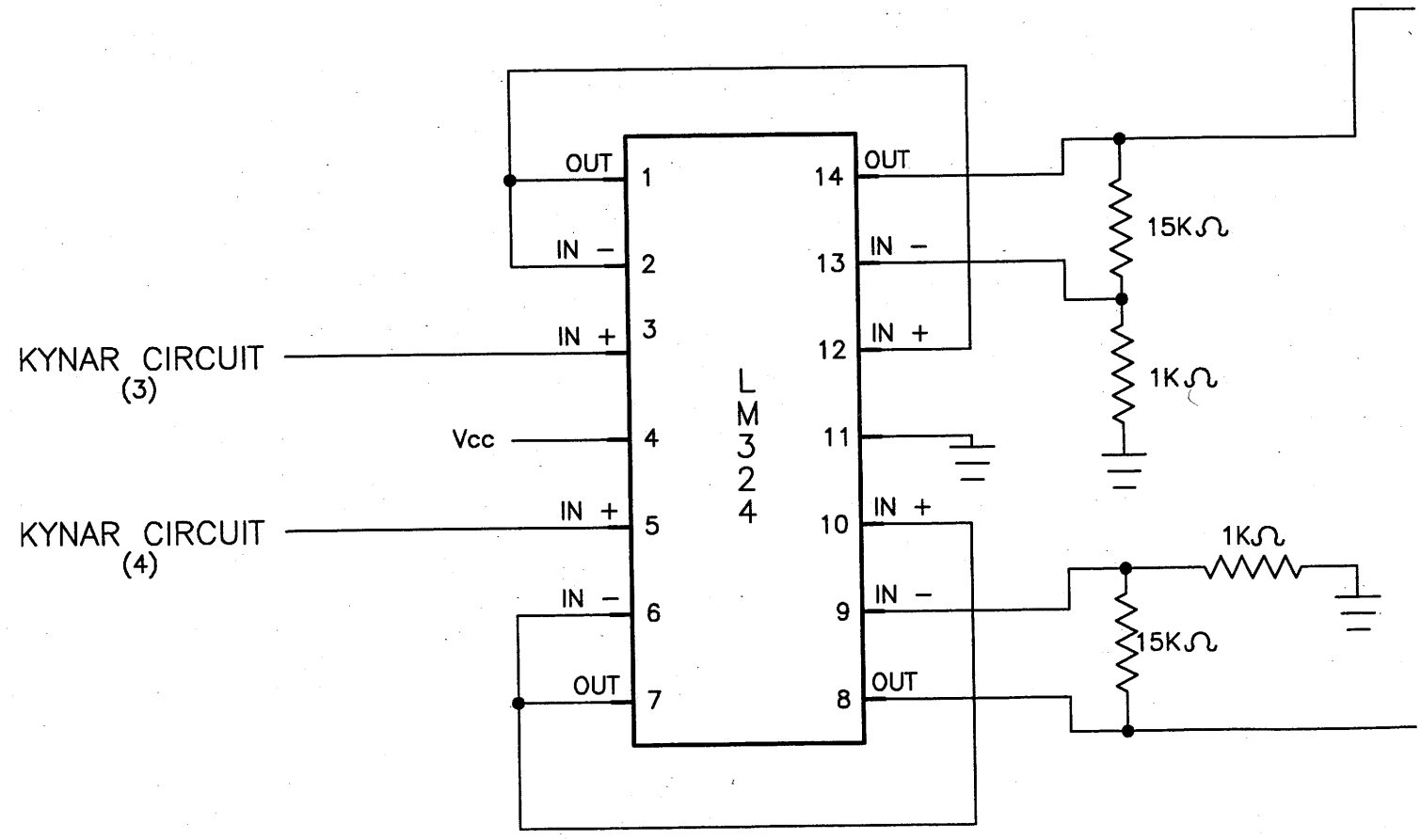
SHEET 1

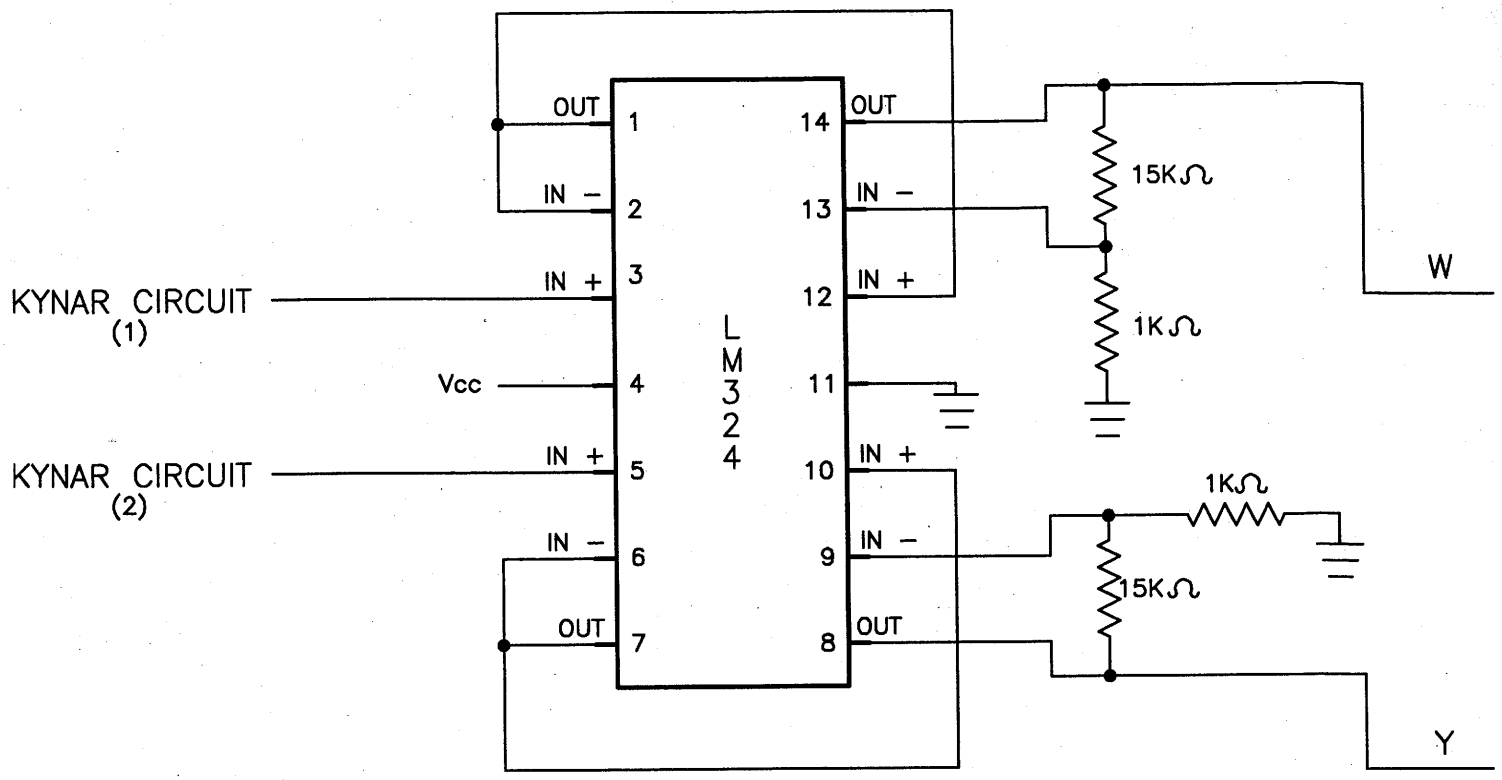

Vcc
8 7 6 5
LED
7340
1 2 3 4
CONNECT
TO SHEET 2
D1 D2
Vcc
14 13 12 11 10 9 8
74LS08
1 2 3 4 5 6 7
Vcc
14 13 12 11 10 9 8
74LS27
1 2 3 4 5 6 7
O W Y B
CONNECT TO
SHEET 2
DIP SWITCH
Vcc
Vcc
8.2K Ω
Vcc
RESET
P1
P0
8751
MIDI-OUT
INT 0
INT 1
P3
P2
XTAL1
ALE
D7
D6
D5
D4
D3
D2
D1
D0
W
Y
O
B
Vcc
14 13 12 11 10 9 8
74LS04
1 2 3 4 5 6 7
220 Ω
MIDI OUT
220 Ω
Vcc
1 2 3 4 5 6 7
DM7406
14 13 12 11 10 9 8
Vcc
B W Y O
PEAK DETECT CIRCUITS
CONNECT TO SHEET 3
Vcc
14 13 12 11
74LS
1 2 3 4

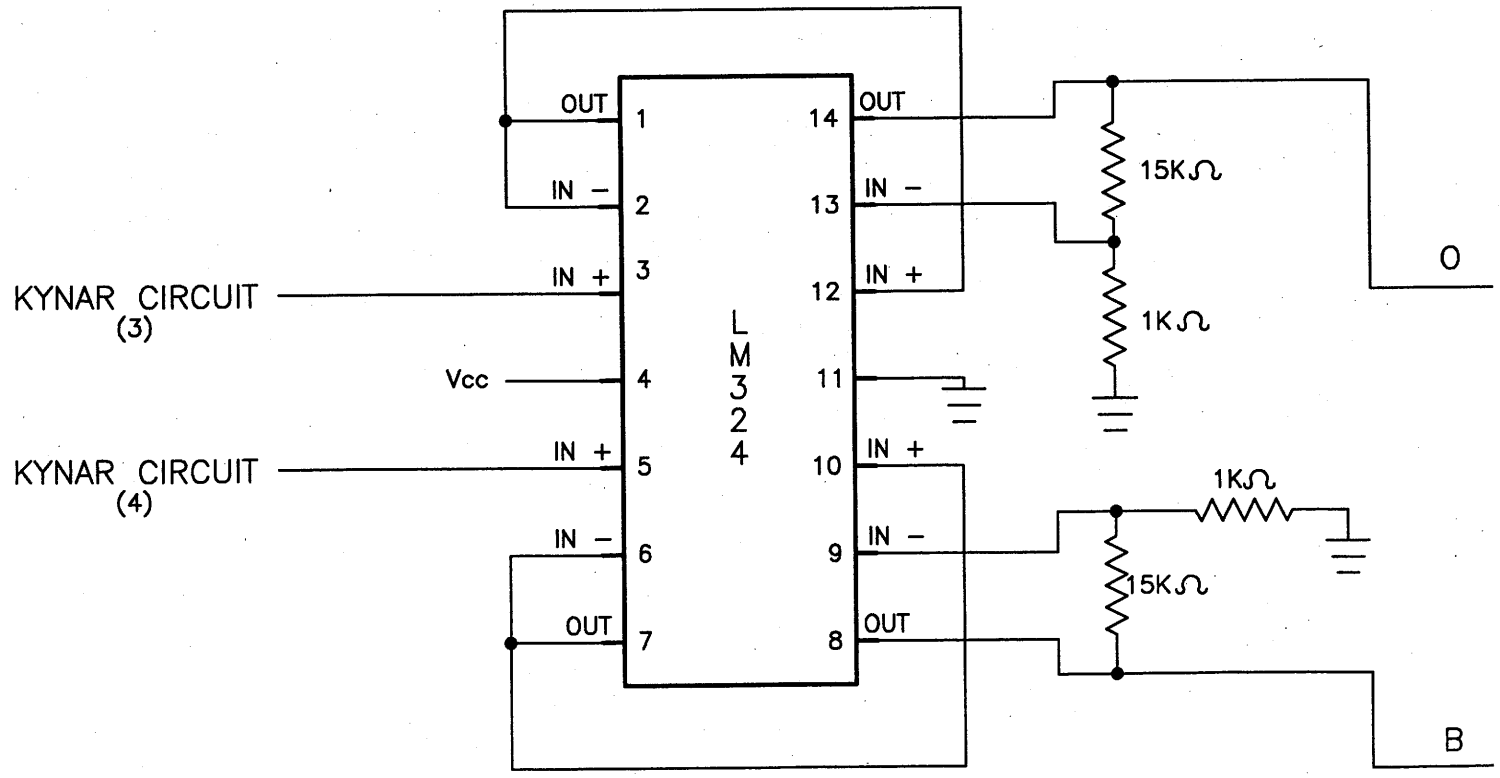

IN
1
14
Vcc
IN
2
13
3
12
7
4
L
S
1
4
OUT
4
11
IN
5
10
OUT
6
9
7
8
OUT
O
D3
Y
D2
W
D1
B
D4
CONNECT TO SHEET 1
IN
1
14
Vcc
2
13
3
12
OUT
4
11
5
10
6
9
7
8

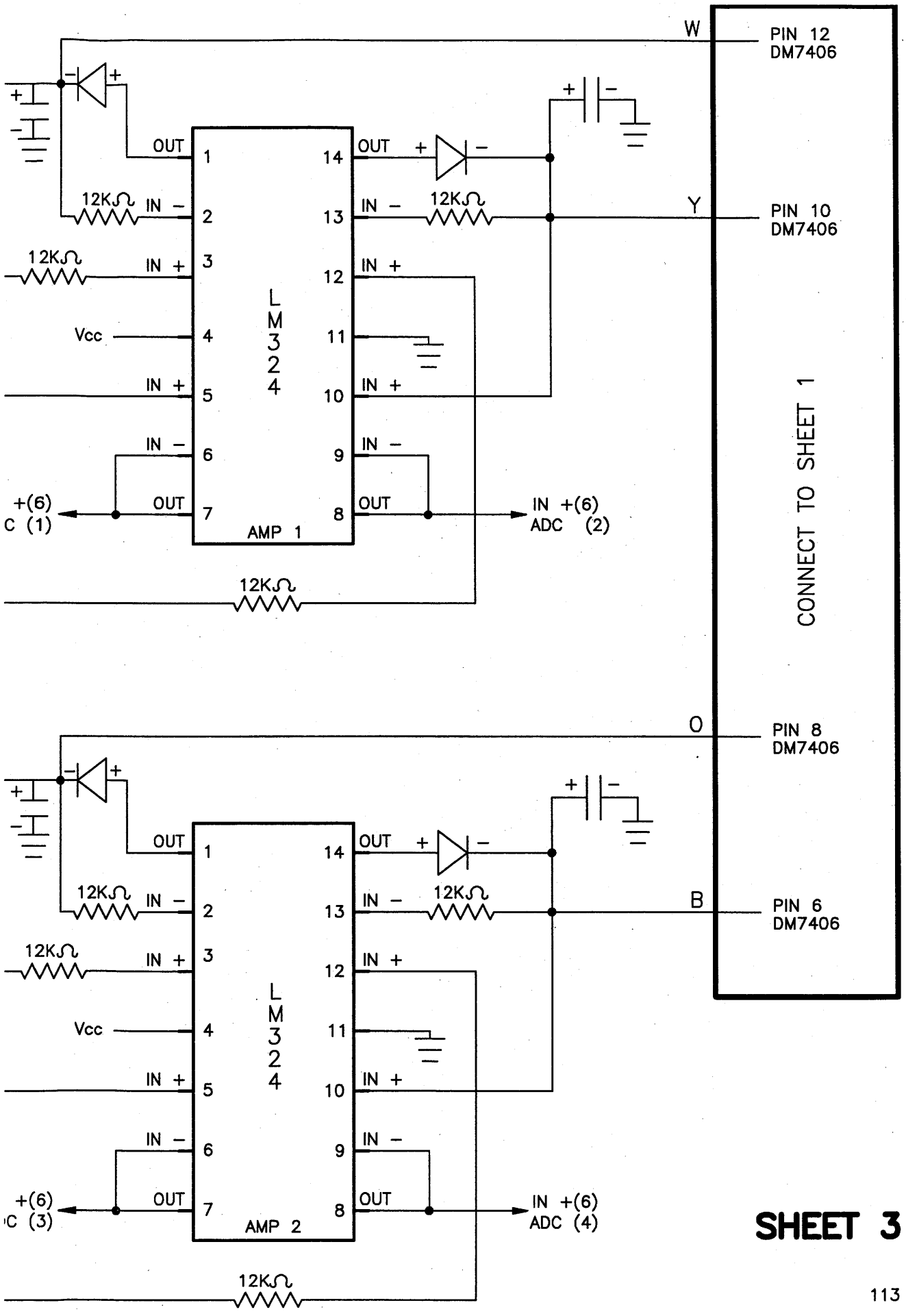
SHEET 2


OUT
1
IN −
2
IN +
3
KYNAR CIRCUIT
(1)
Vcc
4
IN +
5
KYNAR CIRCUIT
(2)
IN −
6
OUT
7
LM324
14
OUT
13
IN −
12
IN +
11
10
IN +
9
IN −
8
OUT
15KΩ
1KΩ
1KΩ
15KΩ


OUT
1
IN −
2
IN +
3
KYNAR CIRCUIT
(3)
Vcc
4
IN +
5
KYNAR CIRCUIT
(4)
IN −
6
OUT
7
LM324
14
OUT
13
IN −
12
IN +
11
10
IN +
9
IN −
8
OUT
15KΩ
1KΩ
1KΩ
15KΩ


OUT
1
IN −
2
IN +
3
KYNAR CIRCUIT
(1)
Vcc
4
IN +
5
KYNAR CIRCUIT
(2)
IN −
6
OUT
7
L
M
3
2
4
14
OUT
13
IN −
12
IN +
11
10
IN +
9
IN −
8
OUT
15KΩ
1KΩ
W
1KΩ
15KΩ
Y


OUT
1
IN −
2
IN +
3
KYNAR CIRCUIT
(3)
Vcc
4
IN +
5
KYNAR CIRCUIT
(4)
IN −
6
OUT
7
L
M
3
2
4
14
OUT
13
IN −
12
IN +
11
10
IN +
9
IN −
8
OUT
15KΩ
1KΩ
O
1KΩ
15KΩ
B


W
PIN 12
DM7406
Y
PIN 10
DM7406
O
PIN 8
DM7406
B
PIN 6
DM7406
CONNECT TO SHEET 1
OUT
IN −
IN +
12KΩ
Vcc
LM324
AMP 1
AMP 2
+(6)
C (1)
IN +(6)
ADC (2)
C (3)
ADC (4)

SHEET 3

Appendix D: Controller Software Listings

```
; this program is the is the driver code for the
; MIDI dulcimer controller. It is written in Intel
; 8051 assembler code.
; AUTHOR: Randy Marchany
; DATE: 8/92
; the vertical strips are id'd in the kynar routine
; The note buffer contains 4 tuning arrangements selected
; by setting a DIP switch that is read into p1.4-7.
; This port is read continously.
; this routine is completely rewritten from the
; previous midint series. The interrupt handlers
; have been streamlined and the max id check and
; vertical strip id check are now done in the
; main program. the interrupt routines use the register
; banks to store pointer into the data ram where
; the values are stored.
;
  name midint8
  org 0
  ljmp  main                    ;main program
  org 03h
  ljmp  ad804                   ;a/d int handler
  org 13h
  ljmp  kynar                   ;int1 handler
  org 100h
main:                           ;start of main program
  mov  sp,#050h                 ;init the stack pointer
  acall  init                   ;init regs
  acall  display                ;and leds
  acall  setup                  ;setup timer,interrupt parms
  setb  ea                      ;enable interrupts
again:
  clr  ea                       ;disable interrupts
  mov  psw,#08h                 ;switch to reg bank 1
  djnz  r2,got1                 ;vertical strip hit?
```

```
  inc  r2                    ;reset r2
  setb  ea                   ;enable interrupts
;
;                              scan p1.4-7 for the tuning selected
;                              it should be a number from 0-8.
;
  mov  psw,#00h              ;switch to reg bank 0
  mov  a,p1                  ;get the port value
  anl  a,#0f0h               ;strip off the low stuff
  swap  a                    ;move to lower nibble
  mov  r7,a                  ;have to figure this out
 acall  display              ;output it
  sjmp  again
got1:
  mov  a,r2                  ;get vert event counter
  mov  r2,#01h               ;reset flag
  push  acc                  ;save it
  mov  a,r6                  ;move to a-reg
  push  acc                  ;save pointer
  mov  psw,#10h              ;switch to reg bank 2
  mov  ie,#01h               ;disable vert ints for now
  setb  ea                   ;renable interrupts to check a/d
adwait:
  cjne  r7,#01,adwait        ;done flag so we know we have valid
  clr  ea                    ;a/d pointers
  dec  r7                    ;reset a/d done flag
  mov  a,r5                  ;move to a-reg
  push  acc                  ;save a/d pointer
  mov  psw,#00h              ;switch to reg bank 0
  pop  acc                   ;move r1 to this bank
  mov  r1,a
  pop  acc                   ;same for r0
  mov  r0,a
  pop  acc                   ;same for r2
  mov  r2,a
  mov  ie,#05h               ;restore vert ints
proc:
  setb  ea                   ;enable interrupts
  acall  vertck              ;get the vertical strip id
  acall  adck                ;get the max a/d value
  acall  xmit                ;got a value
xmbuf:
note1:                         db 90h    ;midi status byte
vel:                           db 0ffh   ;note id
wait1:                         db 45h    ;velocity
note2:                         db 90h    ;midi status byte
  db  00h                      db 0ffh   ;note id
  db  0dh                  ;velocity 0 (off)
  sjmp  again              ;eom
;                          ;loop forever
;
;                              this routine scans the vertical strip latch for the strip that
;                              was hit.
;
vertck:
  push  psw                ;save psw
  mov  a,@r0               ;get the latch data value
```

```
  inc r0                          ;bump pointer
  mov r4,#01h                     ;init the id counter
ky1:
  rrc a                           ;rotate to carry
  jc ky2                          ;got a bit set so jump
  inc r4                          ;nope
  cjne r4,#05h,ky1                ;check next bit position
ky2:
  mov i,r4                        ;save in data ram
  pop psw
  ret
;
;                                   this routine read all 4 a/d data values, searches for the
;                                   max value, saves it and returns
;
adck:
  mov b,#04h                      ;keep track of a/d values
  mov max,#00h                    ;reset max counter
hcheck:
  mov a,@r1                       ;get the a/d value
  inc r1                          ;bump pointer
  cjne a,#0fh,less                ;< 0f? (relative 0)
  sjmp maxck                      ;no, so continue
less:
  jc next1                        ;yes, so get next value
  cjne a,#0f0h,more               ;> F0?
  sjmp maxck
more:
  jnc next1
maxck:
cjne a,max,max1                   ;new max > old max?
  sjmp next1                      ;= so use old one
max1:
  jc next1                        ;< old max so get next value
  mov max,a                       ;save the new max
  mov j,b                         ;save the index in ram
next1:
  djnz b,hcheck                   ;check next value
  ret
;
; this routine determines which note was hit and
; xmits the proper midi note to the synth. the note
; id was set by the a/d interrupt handler.
xmit:
  pop dph
  pop dpl                         ;load data ptr
  acall display
;
;                                   we determine which tuning was selected
;                                   and set the appropriate base register
;                                   address
  mov a,r7                        ;get the parm
  cjne a,#00h,sc2                 ;tuning 1?
  mov dptr,#tuning1               ;point to it
  sjmp sxmit                      ;
sc2:
```

```
  cjne  a,#01h,sc3                ;tuning2?
  mov  dptr,#tuning2              ;point to it
  sjmp  sxmit
sc3:
  cjne  a,#02h,sc4                ;tuning3?
  mov  dptr,#tuning3              ;point to it
  sjmp  sxmit
sc4:
  mov  dptr,#tuning4              ;is the default
;
; get the note from the 2-d array. the address of scale(i,j)
; is (scale + (4Xi) + j)
;                                   i = vertical id, j = horizontal id
;
sxmit:
  mov  a,i                        ;get the vertical id
  dec  a                          ;scale to 0,0
  mov  b,#04                      ;# of cols
  mul  ab
  add  a,j                        ;add offet to row j
  dec  a                          ;scale to 0-n index
  movc  a,@a+dptr                 ;get the note
  mov  r4,a                       ;saveit in buffer
  mov  dptr,#xmbuf                ;point to xmit buffer
  clr  a
  movc  a,@a+dptr                 ;get 1st char to xmit]
rt2:
  jnb  ti,rt2                     ;xmit rdy?
  clr  ti                         ;set as busy
  mov  sbuf,a                     ;output to serial port
  inc  dptr                       ;bump pointer]
  clr  a
  movc  a,@a+dptr                 ;get next char
  cjne  a,#0ffh,rt2a              ;look for note flag in buffer
  mov  a,r4                       ;sub note value
rt2a:
  cjne  a,#90h,rt3a               ;eom? or Delay?
  mov  r3,#0fh                    ;delay
l0:                                 mov  r7,#0ffh
loop:                               djnz  r7,loop
  djnz  r3,l0
rt3a:                               cjne  a,#0dh,rt2        ;eom?
xit:
  mov  a,#1                       ;return to main
  jmp  @a+dptr
;
; interrupt handler for ad804 chip. bring rd down
; to read data value and bring it back up
; we read the 4 ad's looking for a max value
; value. the routine uses r1 of REGISTER BANK 2
;                                   regs  use
;                                   r1  pointer to next avail a/d buffer spot
;                                   r2  # of horizontal strips to read
;                                   r3  used by timing loop
;                                   r4   " " "
;                                   r5  current pointer before reset
;
```

```
;                                   r6  unused
;                                   r7  a/d done flag for main routine
;
ad804:                              push psw
  push acc
  push  b                          ;save regs and
  mov  psw,#10h                    ;switch to reg bank 2
  mov  r2,#05h                     ;strip id
  mov  a,r1                        ;save start of this sequence
  mov  r5,a                        ;for main routine (adck)
next2:
  djnz  r2,cont                    ;select next a/d
  sjmp  out                        ;no more a/d to read, so exit
cont:
  mov  a,r2
  mov  p0,a                        ;select next a/d
  nop
  nop
  nop
  nop
  nop                              ;delay to allow data bus time
  mov  a,p2                        ;read a/d data
  nop
  mov  @r1,a                       ;save in buffer
  inc  r1                          ;bump pointer
  cjne  r1,#050h,next2              ;see if end of circular buffer
  mov  r1,#40h                     ;yes so reset to beginning
  sjmp  next2                      ;get next a/d value
;
;                                   discharge the peak detect caps by pulsing
;                                   the 7406 chip connected to p3.7
;
out:
  mov  p0,#0ffh                    ;this clears a/d rdbar also
  setb  p3.7                       ;discharge caps
  mov  r3,#07h                     ;set up delay
111:                                mov  r4,#0ffh   ; to 1.7msec
self:                               djnz  r4,self
  djnz  r3,111
;                                   mov  p0,#00fh   ;
  clr  p3.7                        ;renable caps
  mov  r7,#01h                     ;set a/d done flag for main prog
  pop  b
  pop  acc                         ;restore regs
  pop  psw
  reti
;
; int1 interrupt handler. starts when a vertical
; kynar strip is hit. it brings wr down on all
; the ad's to start the conversion
; p3.6 is connected to enable on the latch
; for the vertical strip id. Uses R0 of REGISTER BANK 1
;                                   regs  use
;                                   r0  pointer to vertical hit buffer
;                                   r1  unused
;                                   r2  # times vertical strips are hit
```

```
;                                       r3  used for timing loop
;                                       r4  "  "  "
;                                       r5  save area for latch data
;                                       r6  current pointer before reset
;                                       r7  unused
;
kynar:
   push  psw
   push  acc
   push  b
   clr  p3.6                            ;latch enable down
   mov  psw,#08h                        ;switch to reg bank 1
   mov  r2,#02h                         ;temp counter
   clr  p3.0                            ;bring wr down
;
;                                        delay 3.8 msec to debounce int signal
;
                                        mov  r3,#1fh
ky11:                                   mov  r4,#0ffh
ky22:                                   djnz  r4,ky22
   djnz  r3,ky11
;
; get the vertical id from the latch
;
   mov  a,p0                            ;get the data
   anl  a,#0f0h                         ;strip off low nibble
   swap  a                              ;move to lower nibble
   mov  @r0,a                           ;save it in buffer
   mov  a,r0                            ;get the current ptr and
   mov  r6,a                            ;save for main loop
   inc  r0                              ;bump pointer
   cjne  r0,#40h,kexit      ;end of buffer?
   mov  r0,#33h                         ;yes so reset to beginning
   mov  a,r0                            ;save for main loop
   mov  r6,a                            ;
kexit:
   setb  p3.0                           ;bring wr back up
   setb  p3.6                           ;renable latch
   pop  b
   pop  acc
   pop  psw
   reti
;
; this routine inits the timer and interrupt registers
; with the proper values. It sets the serial baud rate
; to 31.2k (for midi) and enables int0,1.

setup:                                    nop     ;setup routine
   clr  tr1                             ;set up timers
   mov  th1,#0ffh                       ;value to reload timer 1 for
        ;31.2 K baud
   mov  tmod,#020h                      ;set up 8 bit auto-reload
   mov  tcon,#00h                       ;clear tcon
   mov  scon,#42h                       ;8 bit uart mode
   setb  tr1
   mov  ie, #05h                        ;enable int0 and 1
```

```
   mov  ip, #00h                    ;set priority to be equal
   ret
;
;this routine initializes the registers
;
init:                                mov  a,#00h
   mov  r0,a
   mov  r1,a
   mov  r2,#00h
   mov  r3,#00h
   mov  r4,#00h                     ;
   mov  r5,#00h
   mov  r6,a
   mov  r7,#00h
   mov  p1,#0ffh
   mov  p2,#0ffh
   mov  p3,#0ffh                    ;keep wr bar down on power up
   mov  p3,#0feh                    ;here
   mov  i,a                         ;
   mov  j,a
   mov  max,a                       ;zero the counters
   mov  b,a                         ;
   mov  psw,#08h                    ;switch to reg bank 1
   mov  a,#33h                      ;put addr of vert strip buffer
   mov  r0,a                        ;put in r0
   mov  r6,a                        ;and r6
   mov  r2,#01h
   mov  psw,#10h                    ;switch to reg bank 2
   mov  a,#40h                      ;put addr of a/d strip
   mov  r0,#00h
   mov  r2,#00h
   mov  r1,a                        ;put in r1
   mov  r6,a                        ;and r6
   mov  r7,#00h                     ;a/d/done flag
   mov  psw,#00h                    ;back to reg bank 0
   ret
;
;                                   this routine displays a hex value on the DISPLAYS
display:
                                    mov  p1,a     ;output the value
                                    ret
;
;                                   this is the tuning storage area. We have 4 different
;                                   tunings for now. 1) straight fifth interval
;                                   2) 2 octave d-major 3) 2 octave e-flat major
;                                   4) triple bridge tuning like my dulcimer
tuning1:  db 51h,53h,55h,56h,4ah,4ch,4eh,4fh
                                    db 43h,45h,47h,48h,3ch,3eh,40h,41h
tuning2:  db 51h,53h,55h,56h,4ah,4ch,4eh,4fh
                                    db 45h,47h,49h,4ah,3eh,40h,42h,43h
tuning3:  db 52h,54h,56h,57h,4bh,4dh,4fh,50h
                                    db 46h,48h,4ah,4bh,3fh,41h,43h,44h
tuning4:  db 32h,34h,36h,35h,45h,47h,49h,4ah
                                    db 3eh,40h,42h,43h,37h,39h,3bh,3ch
;
;                                   this is the data memory area for the program
```

```
;                                   we store stuff in location 70 and 71H
;
max     data  30h     ;max value read
i       data  31h     ;vertical strip id
j       data  32h     ;horizontal strip id
vertbf  data  33h    ;start of vert strip buffer
hbuf    data  40h    ;start of a/d data buffer
        end
```

Appendix E: MIDI Controller Contacts

This appendix contains the vendor addresses of some of the controllers mentioned in this paper.

1. **Buchla Thunder**, Buchla Associates, Box 10205, Berkeley, CA 94709. Phone: 415-528-4446. Suggested retail price: $1990.00

2. **Video Harp**, Sensor Frame Inc., Phone: 412-683-9500

3. **Kynar**, Atochem North America, Piezo Film Sensor Division, P.O.Box 799, Valley Forge, PA 19482, 215-666-3500

4. **Digital Flute**, M. Yunik, M. Borys, G.W. Swift, Dept. of Electrical Engineering, The University of Manitoba, Winnipeg, Manitoba, R3T 2N2 Canada

Vita

Randolph Marchany was born in Fredericksburg, VA on July 13, 1953. He received a B.S in Computer Science from Virginia Polytechnic Institute and State University in 1980. He has been working at the Virginia Tech Computing Center as the senior member of the Academic Computer Systems group for the past 17 years. He is one of the foremost hammer dulcimer players in the United States and teaches master level workshops around the country. A member of **No Strings Attached**, an acoustic band featuring the hammer dulcimer, he has performed in venues ranging from the John F. Kennedy Center for the Performing Arts to CBS-TV's "Morning News" program with artists such as Mary Wilson and the Supremes, Bela Fleck and the Flecktones, Doc Watson, John Hartford and John McCutcheon. The band has released seven albums, four of which have won or were nominated for the National Association of Independent Record Distributors (NAIRD), "INDIE" award for "Best Album - String Music Category" in 1986, 1987, 1988 and 1990. This award is the independent record companies' version of the Grammy awards.

His current professional interests include developing distributed workstation system management tools, Unix system security and developing Unix training seminars. His musical interests include developing methods to combine acoustic and MIDI instruments, MIDI controller design and performing in concert at Carnegie Hall.