THE ANALYSIS OF A COMPUTER MUSIC NETWORK AND THE IMPLEMENTATION OF ESSENTIAL SUBSYSTEMS

by
ANTONY JOHN WILKS

submitted in fulfilment of the requirements for
the degree of
MASTER OF SCIENCE
in the subject
COMPUTER SCIENCE
at
RHODES UNIVERSITY

SUPERVISOR: MR. R. FOSS
FEBRUARY 1994

ABSTRACT

The inability to share resources in commercial and institutional computer music studios results in non-optimal resource utilisation. The use of computers to process, store and communicate data can be extended within these studios, to provide the capability of sharing resources amongst their users. This thesis describes a computer music network which was designed for this purpose. Certain devices had to be custom built for the implementation of the network. The thesis discusses the design and construction of these devices.

ACKNOWLEDGEMENTS

I would like to thank all those people who had, in one way or another, something to do with this thesis. A special thanks to my supervisor for the effort he made in seeing it to fruition.

CONTENTS

PAGE

CHAPTER 1 - INTRODUCTION

1.1 Computer Music Studio Evolution

Modern music studio practice and equipment have evolved rapidly over the last five decades. The introduction of computer technology into this field is fast improving old and introducing new studio practice and equipment. Computers have been added to the studio environment to generate, process and record sound. The functionality of these computer based devices can be controlled by computer and this has led to the proliferation of computer music studios.

The characteristics of computer music studios allow a single composer to create entire compositions without the need for other musicians to play any of the instruments. The ability for a single user to play multiple instruments is made possible by the powerful control features offered by computers. The great variety of sounds offered by the digital audio generators, and the immense ability to change the sounds using digital processing, make the possibilities for sound creation endless.

Current typical computer music studios are single user in nature. The composer has all the facilities offered by the studio at his disposal even though he may not be using them all. This is not a problem if it is a personal studio, but in institutions and commercial studios there is contention for studio usage. In institutional studios, such as the Rhodes University Computer Music Studio, a user has to book the studio to obtain access. This user then has control over all the resources of the studio for the duration it was booked. This is not an optimal solution, as the user rarely uses all the resources at his disposal. What is needed is a system which allows users to share the resources of the studio.

The full versatility of computers within computer music studios has not yet been fully exploited. Computers can provide the ability to share the resources, in what are currently single user computer music studios. While the inability to share access to resources may be the main problem with typical computer music studios, it is not the only problem. The shared access system also provides solutions to the problems of saving and recalling studio setup, studio complexity, addition of studio resources, audio mixing and MIDI patching. All are problems associated with typical computer music studios.

1.2 Thesis Aim

This thesis discusses how the extended control and communication facilities offered by computers can be used within typical single user computer music studios to share resources. The concept of a computer music network is introduced as the vehicle for providing this shared access. The creation of any computer based system requires that you first address the hardware requirements of the system. The aim of this thesis is to identify the hardware requirements of our computer music network, and to design and construct those devices which are not commercially available.

1.3 Thesis Description

The thesis begins with an analysis of typical computer music studio practice and equipment. This was done to identify the current problems associated with such studios, in particular the Rhodes University Computer Music Studio.

A system, the computer music network, is proposed as a solution to overcoming these limitations, primarily the problem of shared access by multiple users. Current networks that have already been developed are evaluated, and a new network topology is proposed to solve the problems inherent in the Rhodes University Computer Music Studio.

The thesis describes the design, construction and evaluation of devices needed by the computer music network. These devices are not currently commercially available because most of the hardware being produced today addresses the needs of single user studios.

The software of a simple network is described. It was designed and written to evaluate the operation of the devices constructed for the network. The design and evaluation of this test software will supply a starting point for the design and implementation of the software of a complete computer music network.

The conclusion to the thesis describes the adequacy of this computer music network approach in its attempts at solving the problems associated with typical computer music studios, in particular the problem of sharing resources within the Rhodes University Computer Music Studio. The description centres around the adequacy of the devices constructed for the network. Possible future hardware and software developments for the network are discussed.

CHAPTER 2 - CURRENT COMPUTER MUSIC STUDIO ENVIRONMENTS

In this chapter, terminology relating to a single user computer music studio will be introduced. Single user studios will be described in terms of this terminology. This introduction to single user studios forms a necessary basis for examining problems related to the sharing of single user studios.

2.1 Audio devices

Current computer music studios are typically single user studios in which the user has total control over all the audio devices comprising the studio. The user is able to control the audio devices, either by direct physical control, or by the use of controllers. Audio devices fit into one of three categories. They are either audio generators, audio modifiers or audio recorders.

2.1.1 Audio Generators

Audio generators are audio devices responsible for the generation of sound. Audio generators within the realm of computer music are either synthesizers or samplers. Synthesizers, as their name implies, use various synthesis techniques to create audio waveforms electronically. Digital synthesizers use algorithms to control their electronic logic circuitry in the creation of audio waveforms. These audio waveform producing algorithms can create sounds with large numbers of harmonics. Large numbers of harmonics are required to create sounds which are as interesting to listen to as acoustic musical instruments [7].

Samplers, on the other hand, allow the user to repeatedly replay an existing sampled sound of short duration. The sampled sound can be replayed at various speeds by the sampler to alter the pitch of the sound. To sample an existing sound, a digital sampler must first convert the analogue audio signal to a digital representation using an analogue to digital converter. The digital information is then stored sequentially within the sampler's memory. The sampler can then reconvert the stored digital sequence back into an analogue audio signal. The sampler employs a digital to analogue converter to reconvert the digital sequence back to an audio signal. The rate at which the sampler moves through the digital sequence transferring the information to the digital to analogue converter determines the overall pitch of the audio signal. The rate of conversion and the point in time when conversion starts are both controlled by the user [9].

2.1.2 Audio Modifiers

Audio modifiers are responsible for the manipulation of the sound produced by the audio generators. The audio modifiers allow the user to alter, mix and listen to sounds. Altering sounds is done by means of audio effects units. Typical types of effects offered by audio effects units include equalisation, echo, reverb, delay, pitch shifting, compression and expansion. These effects allow the user to create a more complex sound from the sound the audio generators have produced. Audio mixers allow the user to mix sounds produced by the audio generators, and altered by the audio effects units, to produce sounds for recording and listening purposes. Audio amplifiers and speakers allow the user to listen to sounds being produced either directly from an audio mixer and/or being replayed from an audio recorder.

2.1.3 Audio Recorders

Audio recorders allow the user to record his sound compositions. The user can then use the audio recorder to play back the recording for listening, editing and the addition of new material. A recording medium, the medium onto which an audio recorder stores compositions, also allows for the transportation and sale of compositions. Associated with the recording medium are tracks. Tracks are physical areas on the recording medium where an audio signal is recorded.

Audio recorders are either analogue or digital in nature. Analogue audio recorders are slowly being replaced by their digital counterparts. Analogue audio recorders do not have the sound reproduction capabilities of digital audio recorders [33] [25 p1]. Good analogue audio recorders and their associated recording mediums do however abound, and will not be totally replaced by their digital counterparts in the near future due to their price, noncomplex nature, and ruggedness [4] [5] [6] [25 p7].

The classification of an audio device into a single category within these three categories, is not as simple as it sounds, as many devices combine features from more than one category. However, the main features of an audio device allow it to be placed in a category.

2.1.4 Audio Interconnects

Interconnecting the three categories of audio devices are audio interconnects. These interconnects allow for the transferal of audio signals from one audio device to another. These interconnects transfer audio data either in analogue form, or as is becoming more common, digital form. A typical method of interconnecting audio devices is shown in figure 2.1.

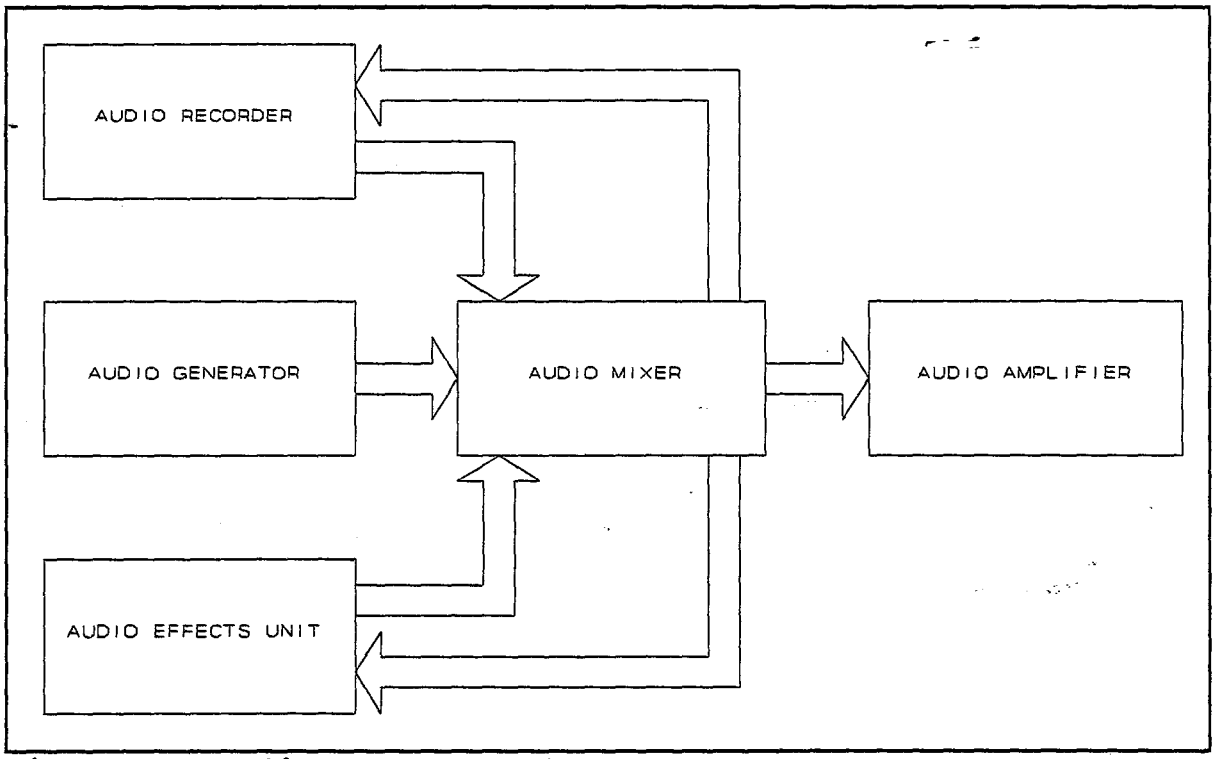

Figure 2.1 Audio Interconnects

The audio signals created by the audio generators are transferred to the audio mixer where the overall gain and tonal attributes of these audio signals is first altered. A different mix of these audio signals is then routed to the audio effects units, audio recorders and audio amplifier. The audio signals returning from the audio effects units are mixed into the audio signals feeding the audio amplifier. The audio signals coming from the audio recorders are modified in the same manner as those from the audio generators. It is necessary to vary gain for audio recorder output signals, as output levels may differ slightly from one recorder to another and from one track to another [17 p478]. The subsequent audio recorder signals are mixed with the audio signal mix produced from the audio generators.

The number of audio devices that can be connected to an audio mixer is dependent on the number of audio inputs and outputs it

has. The numbers of each type of audio device that can be connected to an audio mixer is dependent on the allocation of the audio mixer's audio inputs and outputs to audio generators, audio effects units, audio amplifiers and audio recorders.

2.1.5 Audio Patch Bays

The problem arising with audio interconnects is that one specific interconnection of audio generators, modifiers and recorders does not supply all the requirements that the studio users will have. To overcome this problem, audio patch bays have evolved [8]. Audio patch bays allow the user to control the routing of audio signals between audio generators, modifiers and recorders. Audio patch bays originally required the user to make patches physically. Audio patch bays whose functionality can be accessed remotely by controllers are now available [26]. These allow the user to quickly access and change entire audio patch scenarios. The introduction of an audio patch bay into figure 2.1 is shown in figure 2.2.

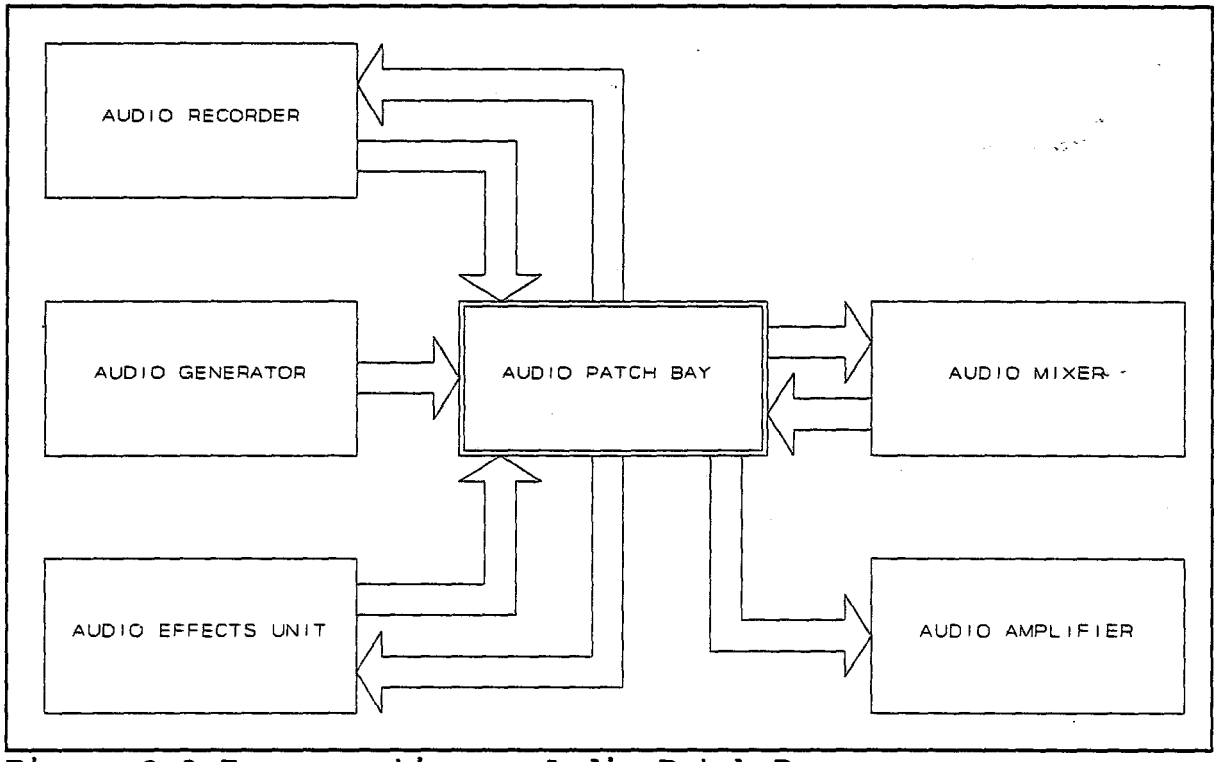

Figure 2.2 Incorporating an Audio Patch Bay

The audio patch bay allows the interconnections between the audio generators and the audio mixer, the audio effects units and the audio mixer, the audio amplifier and the audio mixer, and the audio recorders and the audio mixer, to be modified. Connections can either be reconnected, disconnected or altered. Audio patch

bays also permit direct connectivity between any of the audio devices. This means that audio signals do not have to be routed through the audio mixer in order to travel from one device to another.

2.2 Control of Audio Devices

A major feature of current computer music studio environments is the capability to remotely control and synchronise the devices within the studio. Remote control allows bulky devices such as audio recorders to be placed remotely while keeping control central. It also allows the centralised control of audio generators and audio modifiers. The transfer of control information between devices also gives devices the ability to synchronise to each other. This is important if multiple devices are to work together. Audio device controllers need to transfer control information to the audio devices under their control. In order to make this communication possible and versatile, it was necessary to establish a universal specification for the transfer of control data. All controllers and devices within the computer music studio could then be interconnected, if they conformed to the specification. MIDI (Musical Instrument Digital Interface) is a communication specification which was developed with this purpose in mind [1 p1].

2.2.1 MIDI

MIDI allows controllers fitted with the necessary electronic hardware to communicate with other controllers and devices also fitted with the same hardware. This is made possible by the fact that MIDI is a serial digital communications system running at 31,25 kiloBaud. A MIDI word is composed of ten bits. These ten bits comprise a start bit, 8 data bits and a stop bit [1 p1]. This makes MIDI a fast enough form of communication to control several devices simultaneously in real time. Associated with each audio generator, and in some cases audio modifiers, is a MIDI channel number which allows the user to access any of these audio devices individually.

The transfer of MIDI data is by MIDI messages which each describe a single event . The first portion of the MIDI message is the status word. The status word contains the command portion of the message. The status word has its most significant byte set and that is what differentiates it from a data word. MIDI messages can be grouped into a number of groups and subgroups :

1) MIDI Channel Messages.
   a) MIDI Channel Voice Messages.
   b) MIDI Channel Mode Messages.
2) MIDI System Messages.
   a) MIDI System Common Messages.
   b) MIDI System Real Time Messages.
   c) MIDI Exclusive Messages.

MIDI Channel Messages are the most powerful aspect of the communication standard. In a MIDI system, a MIDI communication link can support up to 16 channels. Devices can be assigned a specific channel number. The status word in all MIDI Channel Messages contains the MIDI channel number. This permits the real time control of different devices on a MIDI communications link. MIDI Channel Voice Messages are the most common form of MIDI messages. These messages are used for real time aspects of control. MIDI Channel Voice Messages are therefore short and consist of a status word and one or two data words. The specific MIDI Channel Voice Message types are:

1) Note On (A key has been pushed).
2) Note Off (A key has been released).
3) Pitch Bender Change (Pitch bender has been moved).
4) Control Change (MIDI controller has been moved).
5) Polyphonic Key Pressure (After touch key pressure sensitivity for each voice).
6) Channel Pressure (One after touch pressure value for all voices).
7) Program Change (Change to the current preset sound).

MIDI Channel Mode Messages allow one to alter the mode of particular devices, usually audio generators, and relate to the control of voices on the device. They determine if voices are played monophonically (Mono On) or polyphonically (Poly On), and if voices are on particular MIDI channels (Omni off) or not (Omni On). The specific MIDI Channel Mode Message types are:

1) Local Control (Connect/disconnect instrument's keyboard from voices).
2) All Notes Off (Turn off all voices that are on).
3) Omni On.
4) Omni Off.
5) Mono On.
6) Poly On.

MIDI System Messages are received by all devices on the MIDI communication link. Response to the particular message depends on the device and how it is programmed. MIDI System Common Messages supply devices with information regarding a composition and how it is to be played. They consist of the following types:

1) Song Position (Position indicator in MIDI sequence).
2) Song Select (MIDI song select message).
3) Tune Request (Perform tuning routine).

MIDI System Real Time Messages form the timing reference aspect of MIDI. They ensure MIDI devices start and stop together at the correct points in a composition, and that they remain in step in between. There are six types of MIDI System Real Time Messages:

1) MIDI Clock (A timing reference).
2) Start (Begin the sequence).
3) Stop (Stop the sequence).
4) Continue (Continue the sequence from the stop position).
5) Active Sensing (Verify the connections between the MIDI transmitter and receivers).
6) System Reset (Instrument to return to default setting).

MIDI System Exclusive Messages are device specific messages. These control parameter settings within specific devices. Each System Exclusive Message begins and ends with a status word. The first status word informs the connected devices of the beginning of a System Exclusive Message and the other informs them of the end. MIDI System Exclusive Messages are used to control the devices described in chapter 4. These devices provide the necessary facilities for creating a computer music studio in which the resources can be shared amongst multiple users.

The MIDI specification has been expanded and now includes MTC (MIDI Time Code). MTC is used to synchronise a MIDI sequencer to a recorder [54]. This enables new sounds to be generated which are synchronised to those already recorded. MMC (MIDI Machine Control) has just been added to the specification to permit a MIDI sequencer to control recorders [56]. This provides the MIDI sequencer with the ability to automatically control recorder functions, such as start, stop and relocate, to name a few. Any MIDI based studio must take full cognisance of these two additions to the MIDI specification. It must provide for their inclusion and communication.

2.2.2 MIDI Controllers

A MIDI controller, within a MIDI based computer music studio, is a device capable of generating MIDI events for the control of devices fitted with the MIDI interface. Keyboards that generate MIDI events when played by a musician, are the most common form of MIDI controllers. These keyboards are usually built into synthesizers and samplers but are not dedicated to the synthesizer or sampler. The MIDI controller can be separated from the synthesizer or sampler by sending it a local control MIDI message and setting it to the off position. This effect can be also be achieved directly through the MIDI controller's function keys.

2.2.3 MIDI Sequencers

The logical step after creating a MIDI controller is to create a device capable of recording and replaying sequences of MIDI control information. MIDI Sequencers were developed for this purpose. Other features were then added to MIDI sequencers which allowed the user to insert, edit and delete MIDI control information. This control information is stored digitally within the MIDI sequencer. The stored information is compact due to the nature of MIDI and is therefore easy to transport and store [10] [11].

A MIDI sequencer has tracks, analogous to an audio recorder, onto which MIDI information is recorded. The MIDI sequencer permits editing of these tracks. The user can perform rhythmic correction, known as quantisation, on tracks. He can perform transpositions to modify pitch. He can change the sound assigned to a track. Tracks can be shifted in time. The pitch, velocity and duration of notes within a track can be altered. Continuous controllers such as pitch bend and modulation wheel can be edited. A MIDI sequencer not only permits the editing of performance data but it also allows the manipulation of entire compositions [18].

MIDI sequencers release the musician from the real time constraints normally associated with live performance. They permit a musician to control multiple devices simultaneously, because MIDI can support multiple channels. They give the musician the ability to replay his composition at any point during its creation.

2.2.4 MIDI Control of Audio Devices

Current audio generators found within computer music studios allow all their functionality to be accessed remotely by MIDI. Audio effects units also allow their functionality to be accessed by MIDI. Audio mixers have had varying degrees of their functionality made available to controllers, but in general most mixers require direct physical control. Amplifiers for listening purposes do not require remote or synchronised control. Thus Audio modifiers have varying amounts of MIDI control over their functionality.

Complete control of audio recorders through the use of remote controllers is not possible. This problem arises due to the nature of the recording medium of audio recorders. The recording medium is a finite resource and must be physically replaced when there is insufficient remaining. The recording medium will also usually require changing with each user and with each song. Remote controllers do not supply audio recorders with the ability to change the recording medium.

A lot of controllers for audio recorders are dedicated and therefore inflexible. Multitrack audio recorders can be quite large devices, so in order to prevent them from occupying excessive room in the working area, they are moved to a remote location and access to their functions is supplied by a small remote controller placed in the working area. The sole purpose of this remote controller is to control the audio recorder for which it was created [17 p400]. The introduction of the MIDI machine control specification is however changing this situation, as it specifies how to implement remote control for audio recorders [12]. Some recorder manufacturers have included MMC into their recorders, and many will follow [12].

2.2.5 MIDI Interconnects

Interconnecting all devices fitted with the MIDI Interface are MIDI interconnects. These interconnects allow for the transferal of MIDI control data between the devices. MIDI Interconnects have a maximum length of 15 meters [1 p2].

One MIDI output can drive one and only one MIDI input [1 p1]. In order for a MIDI signal to be sent to multiple devices, multiple replicas of that MIDI signal must be made in order that the specification be met. This is done in one of two ways. Devices

which have a MIDI In usually have a MIDI Thru. This MIDI Thru creates a replica of the MIDI signal at the MIDI In. This allows devices to be chained to permit them to receive the MIDI signal. This method has drawbacks in that a device must be switched on in order for its MIDI Thru to operate, and timing problems associated with the replication of the MIDI signal arise when chains become too long [1 p1]. The second and preferred method of replication is to use a MIDI Thru box. The MIDI Thru box has multiple MIDI Thrus, all of which replicate the MIDI signal appearing at its single MIDI In. This method overcomes the necessity to power up all devices and also overcomes the timing problems associated with MIDI chaining. Figure 2.3 illustrates the incorporation of a MIDI Thru box into a computer music studio.

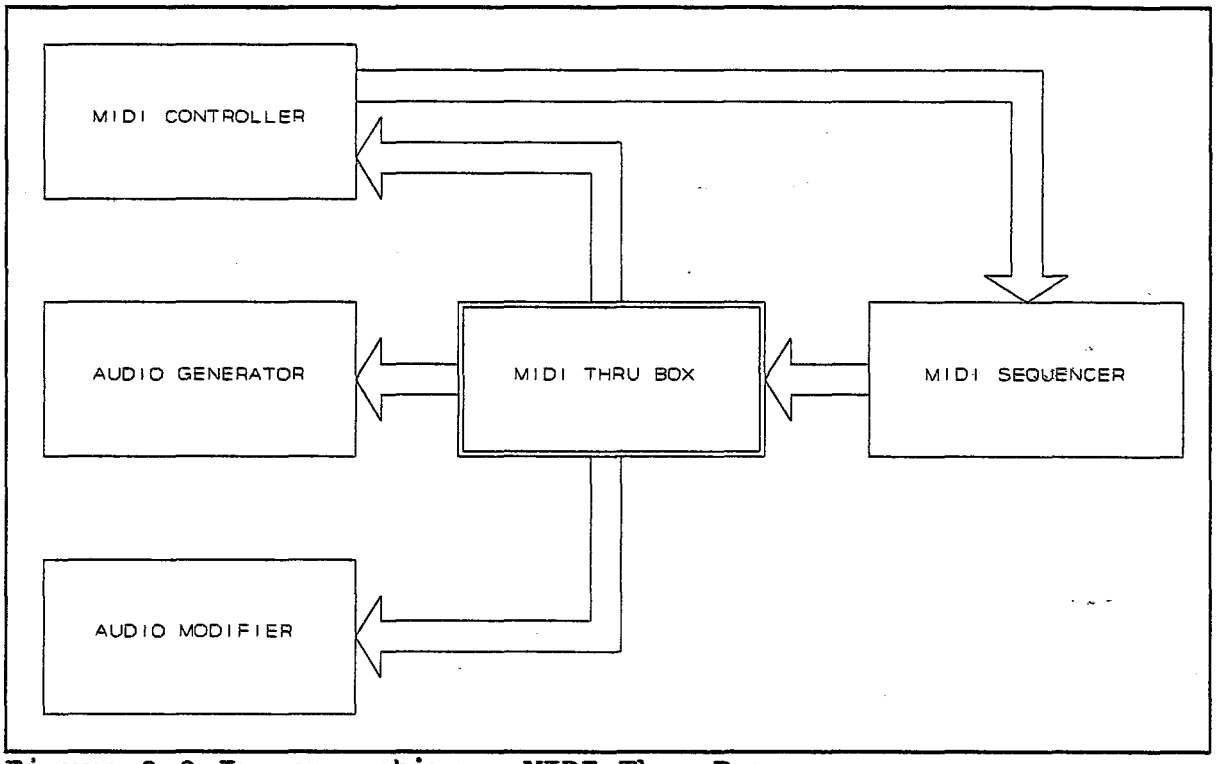

Figure 2.3 Incorporating a MIDI Thru Box

2.2.6 MIDI Patch Bays

The problem arising with MIDI interconnects is that one specific interconnection of devices does not supply all the requirements that the studio users will have. To overcome this problem MIDI patch bays have evolved. MIDI patch bays allow the user to control the routing of MIDI control data between devices. MIDI patch bays originally required the user to make the patch physically. MIDI patch bays whose functionality can be accessed by using MIDI are now available [26].

These MIDI patch bays offer other features such as MIDI merging, filtering, transformations and transposing, they also can be used as MIDI Thru boxes. In MIDI merging, two or more MIDI links can be merged into one. MIDI filtering allows for the removal of certain MIDI events from a MIDI link. MIDI transformation allows MIDI messages passing through the MIDI patch bay to be altered in a manner specified by the user. MIDI transposing, a form of MIDI transformation, permits the pitch of MIDI note information to be altered. The MIDI patch bay must have some form of intelligence to perform some of this MIDI processing. For example, when merging two MIDI data streams into a single MIDI data stream, MIDI messages can arrive concurrently on the two MIDI inputs. MIDI messages must be output on a first-in first-out basis. The second MIDI message must be stored until transmission of the first has been completed. The intelligence in the process of merging is required to store MIDI messages and identify when a complete MIDI message has been received or transmitted. Figure 2.4 illustrates the incorporation of a MIDI patch bay into a computer music studio.

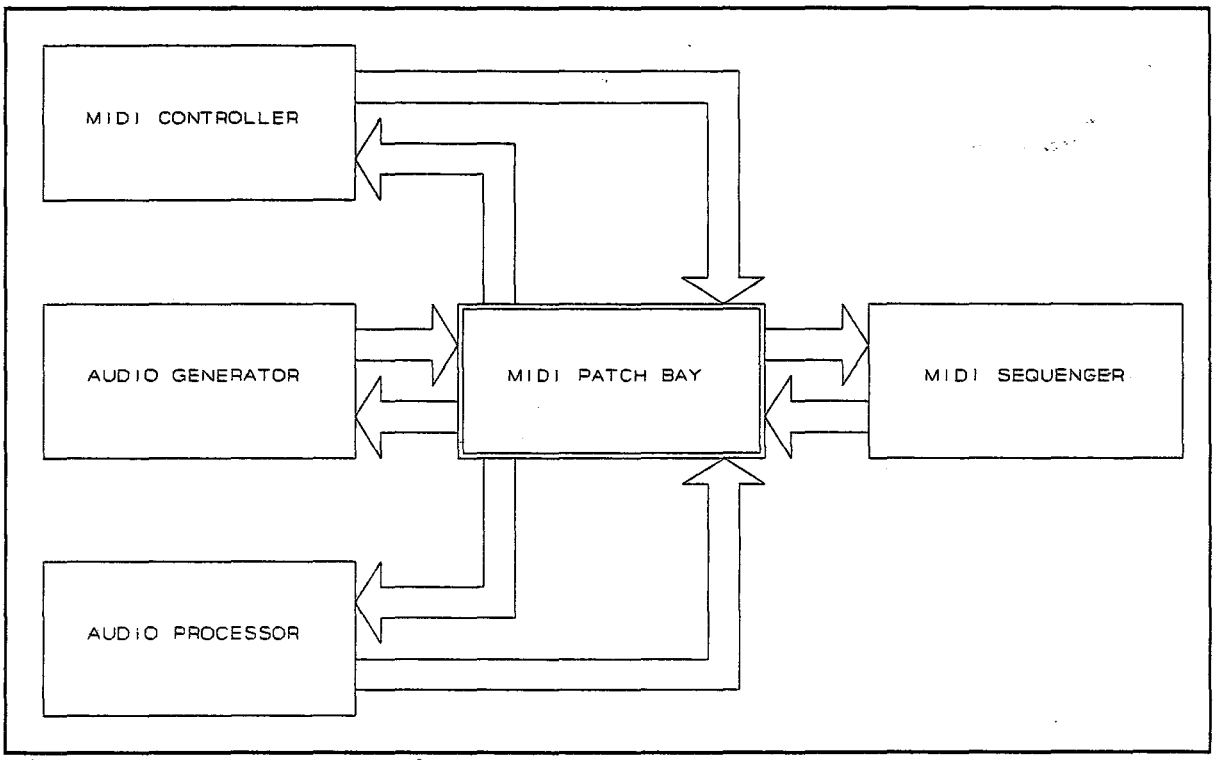

Figure 2.4 Incorporating a MIDI Patch Bay

The MIDI patch bay permits the MIDI controller and other MIDI devices to communicate MIDI information to the MIDI sequencer. The MIDI patch bay also permits the flow of MIDI information directly from one device, to one or more other devices connected to the patch bay. MIDI information does not therefore need to

travel via the MIDI sequencer.

2.3 Synchronisation

It is important that all the devices in a computer music studio are synchronised when playing a composition. This will ensure that all devices start at the same time and then stay in step. This is not as easy as it sounds, due to the diversity of devices now appearing in a computer music studio. Typically, the requirement is to synchronise MIDI sequencers and audio recorders.

The synchronisation of drum machines to MIDI sequencers is done by using MIDI system real time messages. Both devices have a MIDI interface and both are programmed in terms of beats and measures. The MIDI sequencer can start, stop and resume rhythms programmed into the drum machine using the relevant MIDI system real time messages. MIDI clocks received by the drum machine from the MIDI sequencer keeps the rhythm being played by the drum machine in beat with the composition being played by the MIDI sequencer.

The synchronisation of MIDI sequencers and audio recorders is not as trivial. Audio recorders do not operate in terms of beats and measures, but have tracks onto which sound can be recorded. The recording of synchronisation information onto one of these tracks is therefore required. The synchronisation information will have to be an audio signal. When replayed, the audio signal can be translated to MIDI synchronisation information. This MIDI synchronisation information can be used to alter the speed of the MIDI sequencers or audio recorders to keep them synchronised.

SMPTE (Society of Motion Picture and Television Engineers) [15] is a time code specification which evolved to meet the need to synchronise video and its audio component. SMPTE works in terms of hours, minutes, seconds and frames. LTC SMPTE (Longitudinal Time Code) is the audio form of SMPTE and permits SMPTE to be recorded onto an audio track of a recorder for synchronisation purposes. This requires that the user have a LTC SMPTE generator. The SMPTE track marks frame locations on a track, thereby time tagging the other tracks. The need for computer music studios to produce music and sound effects which are synchronised to video has made the use of SMPTE in device synchronisation widespread. The SMPTE track on an audio recorder is however an audio signal, and a MIDI sequencer receives and transmits MIDI. Clearly the SMPTE frame locations must be translated to MIDI timing

information. MIDI time code was developed for this purpose [54]. There are two types of MIDI time code messages : The first message type is used to communicate SMPTE times using MIDI. The second message type is used to communicate SMPTE setup information. This information contains event start times, stop times and cue positions. The conversion of LTC SMPTE to MTC is performed by a LTC SMPTE to MTC converter.

The biggest problem associated with a MIDI sequencer that synchronises itself to MTC, is that the MIDI sequencer can only synchronise itself to a single audio recorder at any one time. A problem arises when there is more than one audio recorder. The MIDI sequencer must synchronise itself to one of the recorders and the synchronisation of recorders to each other must be done by means of a dedicated controller. This limits the type and number of recorders that can be integrated into a studio. It also means that control is not centralised, as the sequencer cannot control the recorders.

With the ratification of MMC [12], this is starting to change. MMC allows the synchronisation of multiple recorders via MIDI [56 p1]. This will make the integration of recorders into a studio a straightforward task. MMC also centralises control in that it gives control of the recorders to the MIDI sequencer.

2.4 Video

Computer music studios that produce music and sound effects for visual media are quite common nowadays, and these studios incorporate a wide array of video equipment. The ability to synchronise MIDI sequencers, audio generators, audio modifiers, audio recorders and video recorders, make computer music studios ideal for the generation of music and sound effects for visual media.

Video recorders can have LTC recorded onto an audio track for synchronisation purposes. It is important that the LTC be frame aligned. This requires that each time code word corresponds to a single video frame. This ensures that frame-accurate tracking of source and destination video recorders occurs. It also ensures that sound generated for a particular video frame occurs within that frame and not at some point inbetween the previous or next frame. The user must have a LTC SMPTE generator which generates frame aligned SMPTE. The SMPTE track can be converted to MTC for MIDI sequencer synchronisation. VITC SMPTE (Vertical Interval

Time Code) [15] is another form of SMPTE intended for exclusive use in video. VITC is inserted into the video signal for recording on video recorders. VITC overcomes the limitations of reading LTC on video recorders, allowing the exact frame location to be established when used in slow or freeze-frame modes. It does not use an audio track on the video recorder, and this audio track can, therefore, be put to other use [16]. VITC will require one to have a VITC SMPTE generator and a VITC SMPTE to MTC converter for synchronisation purposes.

The number of frames per second is dependent on the format of the video material. This varies from one video recording medium to another and from one country to another. 24 frame SMPTE is used in the film industry. 25 frame SMPTE or EBU time code (European Broadcast Union) is used by European television. 30 drop frame SMPTE is used by American television. 30 nondrop frame SMPTE is also used by American television but only for black and white television [16]. The type of SMPTE one uses is therefore dependent on the frame rate of the video material being used.

2.5 The Rhodes University Computer Music Studio

2.5.1 Studio Equipment and Layout

The Rhodes University Computer Music Studio comprises seven MIDI controllable synthesizers and a sampler as audio generators. Five of the synthesizers have keyboards. There is a MIDI guitar which generates MIDI events when it is played. To perform audio processing there are two audio effects units, an eight channel audio mixer, an Iota Systems Midi Fader and a stereo audio amplifier. The Iota Systems Midi Fader [2] is a MIDI controllable eight channel audio fader which enables the user to employ a MIDI sequencer to control mix levels. There is an eight track audio tape recorder and a standard stereo audio tape recorder. Both audio recorders are analogue. There are no digital recording facilities. A computer based MIDI sequencer allows the musician control over the synthesizers, sampler, effects units and the audio fader.

All audio signal transmissions occur in the analogue domain. There are two manual audio patch bays which allow the user to manually reconfigure the audio connections. They provide, however, only a subset of the interconnectivity possible, as not all the audio sources and audio sinks can fit on the audio patch bays.

A manual MIDI patch bay allows the user to alter the MIDI interconnects. The MIDI patch bay contains a MIDI Thru box with sixteen outputs providing the necessary replication of the MIDI signal produced by the sequencer. There is a "Lone Wolf" "MidiTap" [3] which allows the user to perform MIDI merging, filtering and transposing. It can also be used as a MIDI Thru box. The unit has an interface which permits computer control. It has optic fibre interconnects which allows it to communicate to other "Miditap" units. The studio is a typical small computer music studio, based around a computerised MIDI sequencer.

2.5.2 Problems Associated with the Use of the Studio

There are several inherent problems associated with using the studio and its facilities. These problems are shared by current educational and commercial studios to varying extents. The main problems are discussed below, together with techniques and technologies which have been used in an attempt to overcome them.

2.5.2.1 Resource Utilisation

A user wishing to use the studio has to book it in advance. The user then obtains total access to all its resources during the booked time slots. Access to unused resources during these times by another person is not possible, and the resource must remain unused. This is the main problem encountered in all computer music studios which have more than a single user. Unplugging and relocating resources to permit multiple users is a solution, but is not always practical and is a tedious task. There are no commercial products that provide full shared access to the resources of a computer music studio.

2.5.2.2 Studio Setup

Each user has different requirements of the studio, and, depending on those requirements, will configure the studio's resources accordingly. This means that every user will initially have to go through the routine of configuring the studio to meet his requirements. This initialisation time wastes the users creative time, and reduces the effective utilisation time of studio resources. Ideally, the initialisation of the studio should occur automatically when the user identifies himself to the studio, and indicates the composition on which he wishes to work. For this to happen, all the functionality of the studio must be remotely controllable and the control must be

centralised. MIDI-controllable MIDI and audio patch bays have the potential to solve this problem. The introduction of MIDI Machine Control (MMC) into recorders promises to help users to initialise recorders faster [12]; the automatic issuing of machine control commands by the MIDI sequencer will initialise recorders to the necessary operating mode and will also locate the recording medium to the correct position.

2.5.2.3 Studio Complexity

A further problem associated with the studio is that of the technical ability of the users. This has a limiting factor on the user, as he cannot achieve that which he does not know to be technically possible. He may also be incapable of operating all the devices available to him within the computer music studio. To ensure that studio facilities are correctly utilised, a technician usually operates certain devices within the studio. The technician also understands the studio layout and knows the facilities that it can provide to the user. The control of the studio's resources through the use of computers would help to solve this problem. The computers would contain additional software helping the users with studio usage. They would also contain the software allowing the users to alter the studio setup. The software could prevent the user from incorrect studio usage. It would also permit the automation of certain studio functions, such as patching. These facilities would avoid the need for a technician to scrutinise every device operation the user made.

2.5.2.4 Addition of Resources

The addition of new resources to the studio can be a problem. Some careful thought must be given before the device can be connected into the existing scheme of things to ensure that its functionality can be correctly accessed. This problem has been largely overcome already by the use of audio, video and MIDI patch bays, whether they be manual or MIDI controllable [26]. The inputs and outputs of a particular device are made available on the patch bay.

2.5.2.5 Audio Mixing

Audio patch bays do not overcome the finite audio mixing resources available within a studio, and at some point in the growth of a studio an audio mixer with increased mixing

facilities has to be purchased. An audio mixer which incorporated both patching and mixing facilities, and was expandable without the need for replacement, would overcome this problem.

2.5.2.6 MIDI Patching

Most MIDI patch bays consist of a fixed size matrix which allows a fixed number of MIDI sources to be connected to a fixed number of MIDI sinks [26]. Some MIDI patch bays are expandable allowing for an increasing number of MIDI sources and sinks [3]. Expandable MIDI patch bays are however expensive due to the amounts of processing power required for the real time routing and processing of large amounts of MIDI data. What is required for MIDI patching is a cheap expandable MIDI patch bay.

Techniques and technologies are available to either totally, or partially solve some of these studio usage problems. For others there is as yet no solution. What is needed is a new approach to the configuration of shared single user computer music studios which addresses these problems fully. The main motivation behind this new approach is to improve studio resource utilisation.

CHAPTER 3 - THE CONCEPT OF A COMPUTER MUSIC NETWORK

As discussed in the conclusion to the last chapter on current computer music studio environments, there are problems associated with the sharing of computer music studio resources. In this chapter we introduce the concept of a computer music network and explain how such a system can overcome these problems.

3.1 Shared Resources

The aim of a computer music network is to share the resources of a computer music studio among the users, with a view to improving resource utilisation. The users should not be required to physically move resources or patch connections to them. The function of the computer music network is to communicate the media found within the network to and from each of its users. The types of media found within current computer music studios include MIDI, RS232, RS422, SMPTE, audio, digital audio, video and digital video. It is these media that the network will have to communicate. The media relate to both control and performance aspects of composition. The control related media permit the control of devices. The control related media are MIDI, RS232 and RS422. The performance related media are the sounds and visual images which result from these devices. A real time response is required in the communication of most forms of the media.

The concept of sharing the resources of a computer music studio by using a local area network (LAN) is not a new idea. Proposals have been made for the creation of such a LAN. At their core is the problem of the distribution of the multiple forms of media. The communication of MIDI in a LAN was proposed by Buxton [30]. The introduction of MIDI servers into a LAN permits the communication of MIDI data across the network. The problem with Buxton's proposal is that most LANs today have communication protocols that do not support the real time communication of data.

ArcoNet (Artist's Computer Network) [32] is another network proposal for the real time communication of media by using a LAN. The ArcoNet proposal is only concerned with the communication of control media, such as MIDI and RS232, and not performance related media. This has the advantage that it decreases the real time requirements on the LAN. Both Buxton's proposal and the Arconet proposal require you to have other forms of interconnects to carry the performance related media.

There are also implementations that attempt to distribute control and performance related media via LANs. Lone Wolf's medialink protocol was developed for a LAN with real time communication capabilities [31]. The protocol permits the user to prioritise the communication of data within the network, providing for the real time communication requirements of MIDI, audio and video. Lone Wolf has already marketed the Miditap unit [3]. This unit uses optic fibre interconnects for the real time communication of MIDI data to other Miditap units on the LAN. Lone Wolf plan to introduce other tap units which allow the real time communication of control data such as RS232, and performance data such as audio and video over the LAN [31] [65]. Devices incorporating Medialink can be controlled directly over the LAN [65].

Another implementation has been Sonic Solution's SonicNet [65]. SonicNet uses a high speed real time network, Fibre Distributed Data Interface (FDDI), for the distribution of digital audio between digital audio workstations. The digital workstations can access, edit and replay any digital audio data within the network. FDDI is a LAN which supports synchronous operation with transfer rates of 100 megabaud [51 p166].

Currently, there is not a computer music network implementation which provides for the distribution of multiple digital audio channels and MIDI over the same LAN. In creating a MIDI based computer music network, the LAN can be used to communicate some or all of the different forms of media. At the low end of the scale one can use the LAN to only transmit non real-time control information. All real time and performance related media found within the network can be communicated by other interconnects. MIDI, SMPTE, analogue audio, analogue video and digital video could be communicated as they are. Specifications exist for the transmission of single, dual, or stereo digital audio signals (AES/EBU) and multiple digital audio signals (MADI) along single interconnects [39]. These are point to point transmissions rather than network transmissions.

At the high end of the scale the introduction of high speed real time LANs such as FDDI could be used to implement Buxton's MIDI servers allowing for the communication of MIDI on the LAN. This concept could then be expanded to include the communication of digital audio on the LAN, using a method similar to the SonicNet implementation. A digital audio channel requires a much higher bandwidth than MIDI for communication. FDDI is capable of

simultanously supporting the transmission of 80 uncompressed high quality digital audio channels [65]. Another LAN, Asynchronous Transfer Mode (ATM) ATM offers transmission rates in excess of 100 megabaud [67] and could even provide for the transmission of digital video on the LAN. A digital video channel requires a higher bandwith than a digital audio channel for communication. The advantage of communicating all the forms of media using the LAN is that it decreases the physical number of interconnects within the computer music network.

Within our implementation of a computer music network, ethernet is used for the transmission of control information that does not have a strict real time communication requirement. This is done because a real time response cannot be guaranteed with ethernet's CSMA/CD protocol [50]. MIDI, audio and video signals, the real time aspects of control and performance are not, therefore, communicated by the LAN. Our implementation requires media interconnects for the communication of each form of real time media. This does increase the number of interconnects, but these can be kept to a minimum by using other techniques, introduced later in this chapter.

3.2 Centralisation of Control

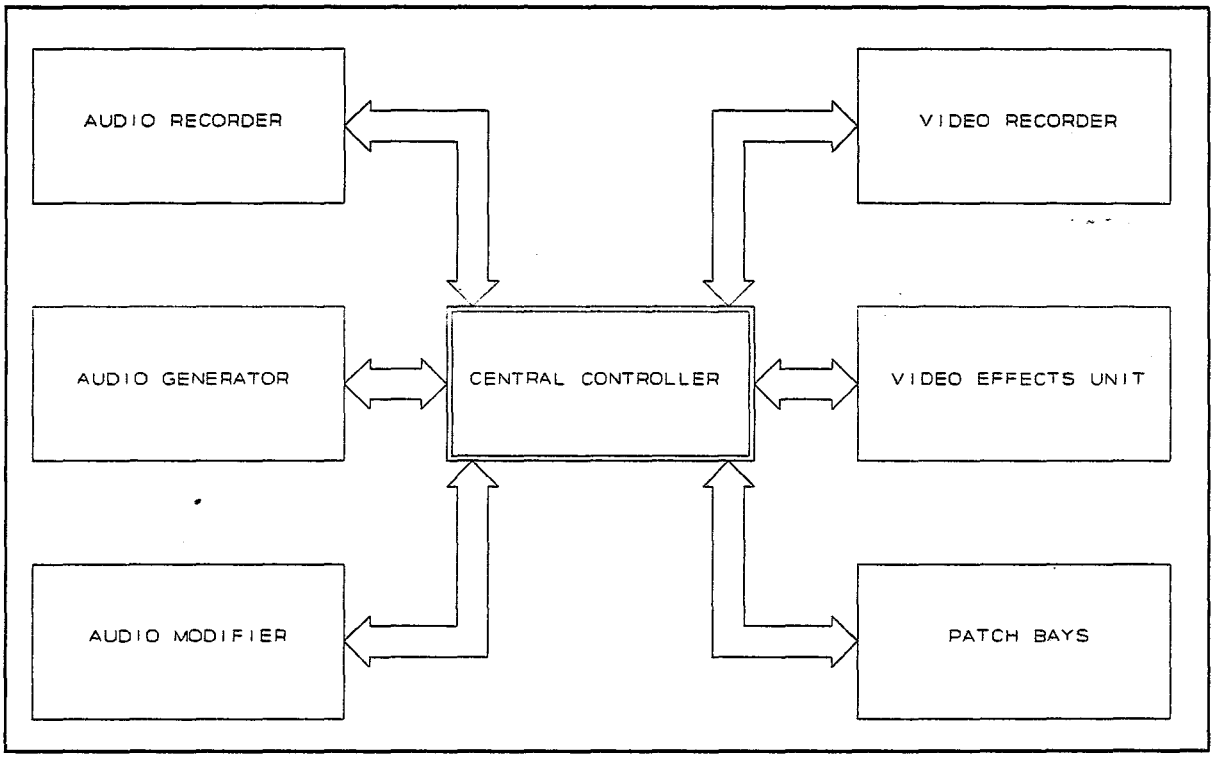

Figure 3.1 Centralised Control

The trend today is towards the centralisation of control within

a computer music studio [12]. This implies that the entire functionality of the studio can be controlled by a single controller within the studio. The control is provided by a single computer running the necessary software. The computer contains hardware providing the necessary interfaces between the user and the computer and the computer and the studio resources. The centralisation of control reduces the demands on the user as he only has to communicate with this single computer. The computer translates his requests to control messages which are relayed to all the devices within the studio. Multiple devices can be synchronised and controlled simultaneously without intervention by the user [12]. Figure 3.1 illustrates the concept of centralised control within a current computer music studio and shows the types of devices that require control.

This trend is important, since control is more easily distributed over a LAN once it has been centralised to a single computer. The computer controlling the studio resources becomes a server [30] and the devices it controlled remain centralised around it. This is illustrated in figure 3.2.

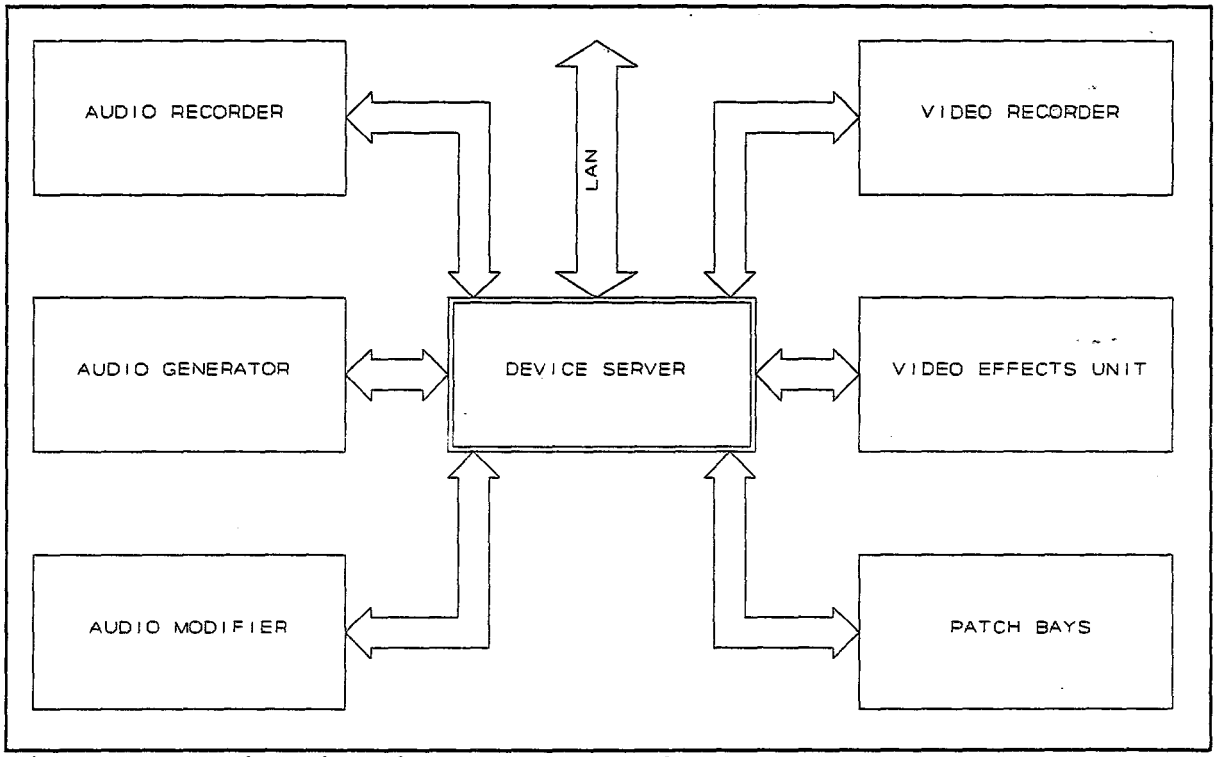

Figure 3.2 Distributing Control using a LAN

The facilities supplied by the central computer within the single, local user computer music studio, are then distributed across the entire LAN. The problems associated with current LANs and their poor real time response severely limits the development

of this type of network [30] [32]. Most of the devices controlled by the central computer must respond in real time if the system is to work. However, the concept of centralised control should not be totally disregarded when creating a computer music network. It can still be useful to centralise control over devices which do not require a real time response and then distribute access to others by using the LAN. Recorders can be controlled over a LAN. So can their support devices such as SMPTE generators and SMPTE to MIDI time code converters. This is possible as many recorders support computer communication standards, such as RS232 [20] and RS422, to access their functionality. An example of this control is illustrated in figure 3.3.

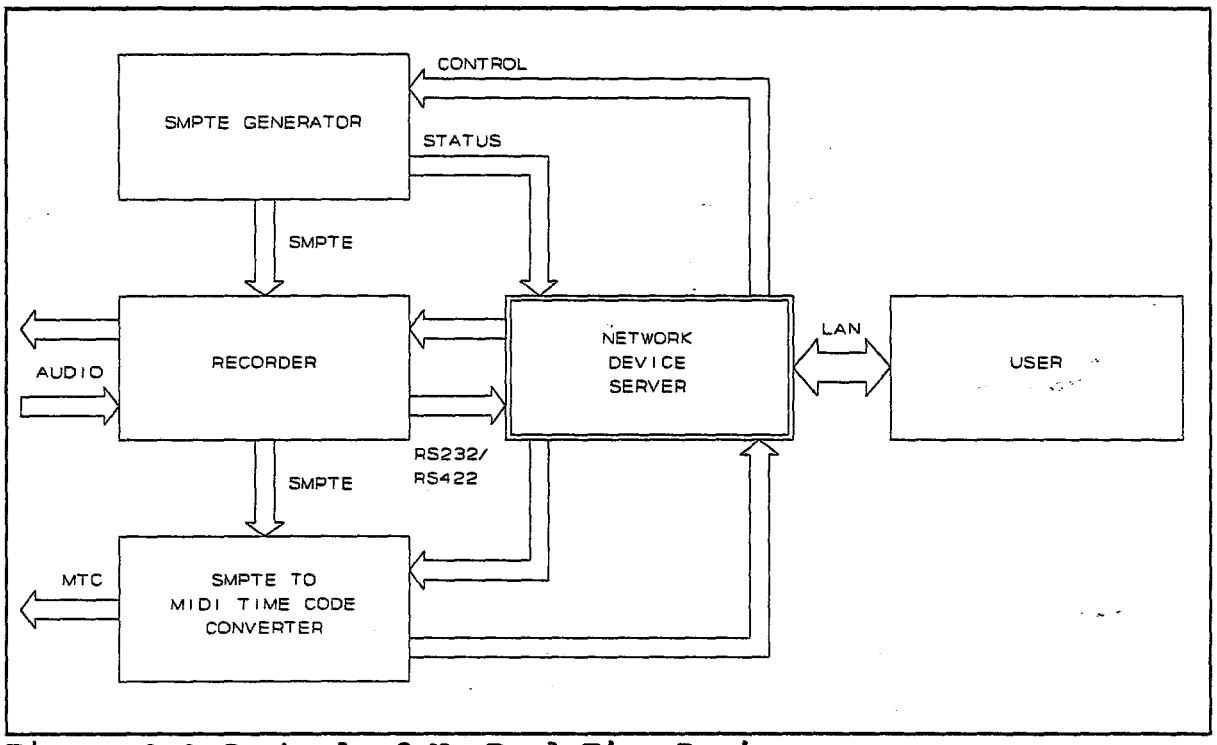

Figure 3.3 Control of NonReal Time Devices

3.3 Centralisation of Resources

There are a large number of media interconnects required in the network to carry both control and performance related media. To reduce the number of media interconnects between each user and the devices within the network, the resources are centralised to a particular place, which will be referred to as the main studio. The distribution of the different forms of media takes place in the main studio. These media can be processed within the main studio and the result communicated to the user. The processing of the media reduces the number of interconnects required between the main studio and the user. The real time forms of media within

the main studio are MIDI, SMPTE, audio, digital audio and video. SMPTE can be disregarded for communication purposes as it can be converted to MIDI time code and communicated along MIDI interconnects. Either digital audio or analogue audio can be used for communication as it is possible to convert between the two forms. The communication of analogue audio is not ideal, as analogue signals are more prone to noise over communication interconnects, than digital audio. This is not a large problem as the audio transmission lengths within the network are short.

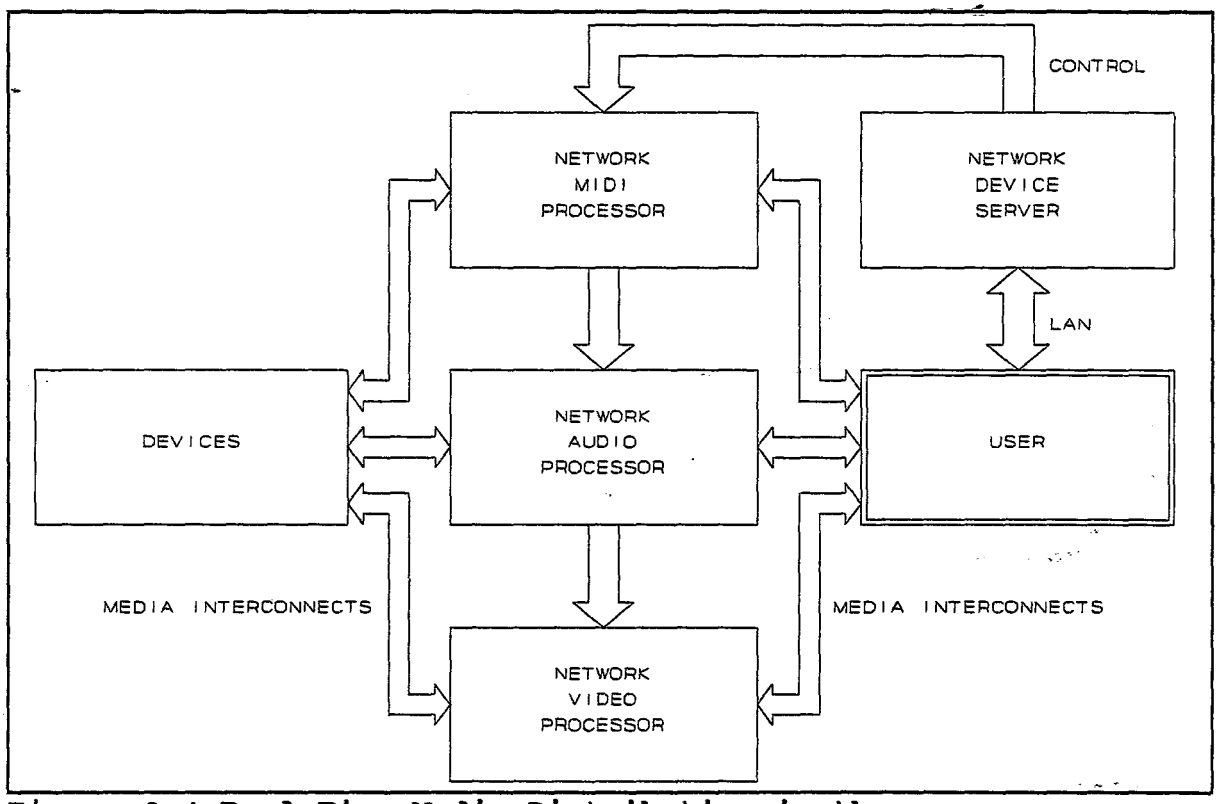

Figure 3.4 Real Time Media Distribution in the Computer Music Network

The processing of MIDI, audio and video within the main studio are the main techniques used to reduce the number of media interconnects between the main studio and the user. The type of processing depends on the type of media. Control over the media processing systems must be by the way of the LAN. This will allow the users of the network to control the processing of the media to suit their requirements. This concept is illustrated in figure 3.4. Ethernet does not exhibit real time communication abilities except with low communication traffic [30]. The media processing systems should exhibit real time processing capabilities. This will permit real time processing to occur on LANs which can provide a fast enough communication response time.

The centralisation of resources to reduce communication interconnects is in strong contrast to Lone Wolf's medialink protocol [31] which allows for resources to be distributed throughout the network. The centralisation of resources does offer other advantages over distributed resources. Access to the physical resources is more easily controlled if they are localised. A technician who may be required to perform studio duties, such as tape changes, will have all the recorders in one place.

3.4 Network MIDI Processing

The distribution of MIDI within the main studio and to the users is performed by a MIDI patcher resident within the main studio. No other form of MIDI processing is necessary within the main studio. The patching of MIDI is required in order that MIDI data transmitted from the user reaches those devices in the main studio to which he has access. It also ensures that MIDI data generated by these devices can be transmitted to the user. The user requires that the MIDI data he produces is received by several devices. In order to minimize the number of interconnects between the user and the main studio, the MIDI patcher can route the users MIDI data to any number of devices within the main studio.

To transmit MIDI data from the main studio to the user, a single MIDI interconnect is used. The user only requires a single MIDI interconnect, as there are only three types of MIDI data he will wish to receive: MIDI system real time messages, MIDI time code and MIDI sysex messages (refer to section 2.2.1 for a description of these MIDI types). The user will only need to receive one of these types at any one time. MIDI system real time messages and MTC supply timing information. Currently, the user can only use timing information from a single device to synchronise his MIDI sequencer(s) to that particular device. The timing information must either be in the form of MIDI system real time messages or MTC, not both. MIDI sysex messages supply the user with data about device setup. Device setup data does not need to be communicated while the MIDI sequencer is actively using MIDI time code or MIDI sysex real time message for synchronisation purposes.

The MIDI patcher residing in the main studio must be expandable to cater for an increasing number of MIDI sources and sinks. One can visualise a MIDI patcher as a matrix in which a MIDI source

can be connected to any group of MIDI sinks. In order to cater for an increasing number of MIDI sources and sinks, this matrix must be expandable. This can be achieved by designing a MIDI patch unit which has an open bus architecture. This allows additional MIDI patch units to connect to other patch units in such a way as to increase the dimensions of the matrix (this is discussed further in section 4.2.1). At the time when the system was designed only Lone Wolf's Miditap could supply this type of functionality [3]. The design and construction of such a MIDI patch bay was necessary because of the absence of a MIDI implementation specification for Lone Wolf's Miditap unit [1 p61] [3]. The lack of the MIDI implementation specification meant that server control over the MIDItap units could not be achieved.

The direct transmission of MIDI between the user and the main studio is not without problems. The MIDI specification states that MIDI cables have a maximum length of fifteen meters [1 p.2]. This length may not be sufficient. This can be overcome by using a different transmission specification, such as RS422 or RS423, for the transmission of MIDI data [34]. Lone Wolf also supplies a fibre optic transmission system for the direct transmission and reception of MIDI data. These systems overcome the transmission length limitations of MIDI. Care must be taken when designing the converters that the ground loop problem that MIDI overcomes is not reintroduced by a poor design [1 p1].

MIDI patching within the computer music network is controlled by way of the LAN. The routing of some MIDI interconnects can be done automatically by network when the user receives control of the devices he has booked. MIDI data eminating from the user can be routed to all of the booked MIDI devices.

3.5 Network Audio Processing

For a user of the computer music network to create a musical composition, it is important that the interconnection of audio signals between audio generators, audio modifiers, audio recorders and video recorders within the main studio be correct. It is not possible for all the audio sources and sinks within the main studio to be made available to each user. This would require a prohibitive number of analogue interconnects between the main studio and the user. It would also be difficult to maintain, as new audio interconnects would have to be added every time a new audio device was brought into the main studio. Digital transmission of multiple audio signals along a single

interconnect could be used as a solution to reducing the number of interconnects. This is provided for in the new Multichannel Audio Digital Interface (MADI) standard [39].

A simpler solution to drastically reduce the number of audio interconnects between the users and the main studio is to mix and patch audio signals within the main studio. The number of analogue audio interconnects between the user and the main studio is then no longer dependent on the number of audio devices in the main studio. The number is dependent on music standards. The most common form of communicating music to an audience is in stereo. The number of audio interconnects required from the main studio to the user is therefore two. Two audio interconnects can also be made to carry audio signals from the user to the main studio. The remote patching and mixing of audio signals does pose a problem especially when there are multiple users. The audio signals of one user must be completely independent of other users, they must not interfere with those of another user.

Remotely controllable audio patch bays could be used for the patching of audio signals. Current commercially available audio patch bays are the 360 System's AM-16 Series Crosspoint Switchers [35] and the Intone MIDI Maestro Audio and MIDI Patch Bay [26]. The AM-16 series allows for expansion of the patch bay through the use of expander modules [36]. This is a useful feature as it will cater for the increasing audio routing requirements in a growing computer music network.

The only way to provide automated mixing facilities is through a remotely controllable audio mixer such as Yamaha's DMP7 [37] or Mark of Unicorn's MIDI Mixer 7s [38]. The performance offered by these mixers is poor in comparison to their manually controlled analogue counterparts, due to their poor signal to noise ratio. Both mixers offer expansion facilities. This is a useful feature as it will cater for the increasing audio mixing requirements in a growing computer music network.

Richmond Sound Design market audio fader and router modules that can be used to create customised remotely controllable audio router/mixers [68]. The audio mixers provide audio reproduction of a professional standard. Creating an audio mixer for the network using this technology is expensive, for reasons explained in section 4.1.5.1. Another combined audio router/mixer system is TRAILS [69]. This system was not considered as its fixed internal structure would not provide the required audio routing

and mixing facilities required by the network.

The fixed structure of audio mixers impose a limitation on the creation of audio patching and mixing scenarios. This is because they only supply a subset of the possible connectivity. The ideal situation is to allow the user to connect any audio source or any group of audio sources to any audio sink, as shown in figure 3.5. The mix level of each audio source fed to the audio sink should be controllable. Current audio mixers supply this functionality to varying degrees. The function and connectivity of current audio mixers is described section 2.1.4. The functionality of audio mixers in this regard can be extended by the addition of patch bays, as described in section 2.1.5. This is a partial solution and still does not allow the user total connectivity. Current audio mixers, by their physical structure, force connectivity rules upon the user. The advantage of this is that it prevents audio loops from being formed within the system.

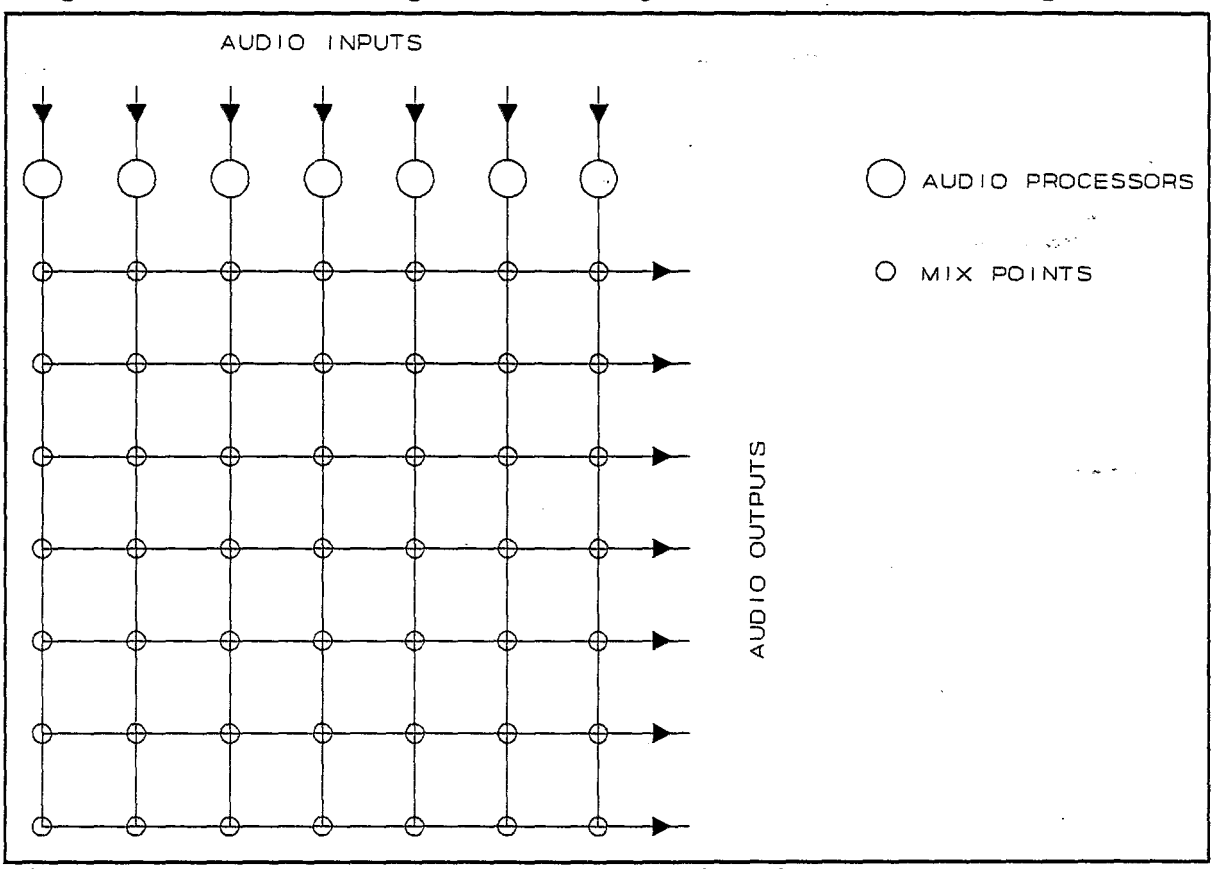

Figure 3.5 Topology of the Ideal Audio Mixer

The routing and mixing of audio signals within the computer music network are controlled by the LAN. Audio connectivity rules, rather than being enforced by the physical structure of devices, can be enforced by software. This modifiable set of rules means

audio mixers with less limiting physical structures can be created, giving to the user a great deal more functionality.

Current audio mixers do not only supply mixing facilities [17 p473]. Incorporated into the units are facilities which allow for the modification of the input level and the modification of the tonal attributes of each audio signal. In figure 3.5 the audio processors perform this function. The input level is modified using a gain control, which ensures that the signal entering the other circuitry of the audio mixer is at the optimal level. This optimal level ensures that all the signals entering the audio mixer are brought to the same level for mixing purposes. It also ensures that an optimal signal to noise ratio is maintained within the unit. Tonal modification permits the user to boost or cut the signal levels of particular bands of the audio spectrum. The tonal modification capabilities of an audio mixer varies from manufacturer to manufacturer and from model to model. Synthesizers which have built in tonal modification capabilities [40] are now available. Tonal modification circuitry is therefore no longer necessary on every input to the audio mixer. MIDI controlled audio effects units typically supply a number of audio processing abilities including tonal modification [22 p9]. Audio inputs requiring sophisticated processing can have this supplied by audio effects units. An audio mixer within the computer music network therefore need only supply rudimentary tonal modification facilities.

The only solution to providing the required routing, mixing, gain control, tonal modification and expansion abilities was to design and develop an audio processor patcher/mixer with these features. The expandability feature caters for the increasing routing, mixing, gain control and tonal modification requirements in a growing computer music network. This number of audio sources and sinks will increase with the addition of more audio devices and workstations to the network. The design of this system is described in section 4.1.

3.6 Network Video Processing

The distribution of video signals between the devices resident in the main studio and the users can be done via a video patch bay that is resident in the main studio. No other form of video processing is required in our network implementation. The video patch bay permits multiple users to patch video sources to video sinks. Control of the video patch bay is by way of a server

resident on the LAN. This is illustrated in figure 3.4. The video patch bay allows any video source to be connected to any video sink [14]. The video patch bay should be expandable to cater for both an increasing number of video sources and video sinks.

3.7 Remote Workstations

Each user requires his own satellite workstation from where he can book resources in the main studio. From the workstation he can remotely control those resources. The control and performance aspects of composition are communicated between the workstation and the main studio using the techniques already described. A workstation should not limit the user in his ability to create compositions. It must provide at minimum all the control that is available when the resources are used in a conventional manner.

The ability to control devices remotely by a LAN permits the user to store data regarding their setup. The stored control information can be used for a fast initialisation of the devices for each composition. This reduces the time spent by the user in initialising, while increasing the time spent on composition, thus improving resource utilisation.

3.7.1 Workstation MIDI Sequencers

The workstation can be configured to have the MIDI sequencer software on the same computer as the workstation network software, or alternatively, the MIDI sequencer can be independent of the workstation network computer. This will permit the use of any current commercial MIDI sequencer, microcomputer based or otherwise, to be used at the workstation. This is a great advantage as it allows musicians to use the MIDI sequencer of their choice. Our initial implementation has the network software operating on the same computer as the MIDI sequencer but they operate entirely independently of each other, as described in chapter 5 [50].

The initial computer music network implementation is illustrated in figure 3.6. This shows the MIDI sequencer software and the network software running on the users workstation computer. It also shows the interconnects between the main studio and the workstation. The SMPTE generator and SMPTE to MIDI time code converter are resident in the server. This requires audio interconnects from the audio recorder to the server to carry the SMPTE. There must also be a MIDI interconnect from the server to

the MIDI patcher, to route the MTC to the correct workstation. Control of the audio recorder is by RS232 [20]. The MIDI patcher and the audio processor patcher/mixer are controlled by MIDI messages generated by the server. The MIDI control of these devices is discussed in chapter 4. The initial computer music network does not incorporate video handling capabilities, these can be added at a later stage.

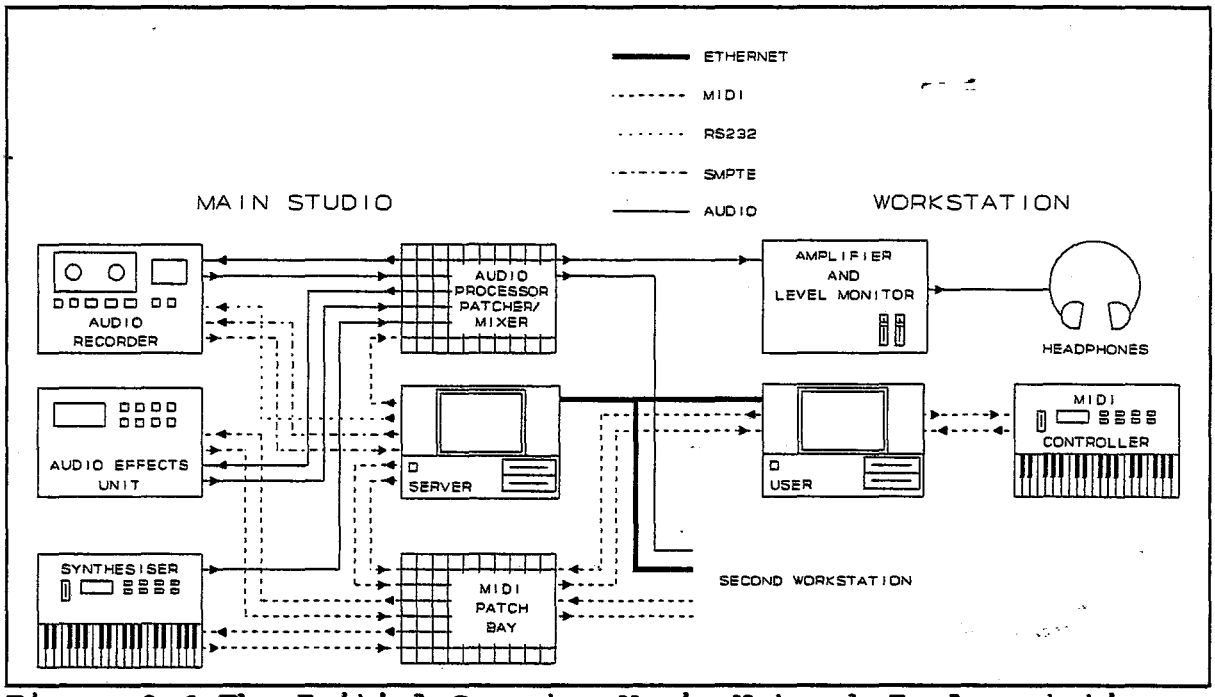

Figure 3.6 The Initial Computer Music Network Implementation

3.7.2 Workstation MIDI Controllers

Each workstation has its own MIDI controller which allows the user to record MIDI data using the MIDI sequencer in the normal manner. It also permits the user to directly control devices in the main studio. The MIDI controller may combine a synthesizer to provide the user easy access to voices. These voices can be used for the initial development of compositions. Audio interconnects for carrying audio signals from the workstation to the main studio can be introduced to enable audio generated at the workstation to be included in the compositions.

The MIDI data produced by the workstation MIDI controller has to be merged with MIDI data returning from the main studio. This facility is necessary as the user may wish to record MIDI data from the MIDI controller while the MIDI sequencer is controlling other devices in the main studio. It may happen that the MIDI sequencer is synchronised, by MIDI, to one of the devices within

the network. The MIDI controller data must then be merged with the MIDI synchronisation data. Fortunately, most current MIDI sequencers support hardware which already allows the merging of more than a single MIDI input.

3.8 Device Synchronisation in the Network

The control of non-MMC recorders by the LAN is illustrated in figure 3.3. The overall view of the computer music network is shown in figure 3.6 and illustrates just how non-MMC recorders are incorporated into the network. The remote control of current recorders is either by RS232 or RS422. Figure 3.3 shows more clearly the peripheral devices which are associated with recorders for the purposes of synchronisation. The function of the SMPTE generator and SMPTE to MIDI time code converter are described in section 2.3. The functionality of the recorder, SMPTE generator and SMPTE to MTC converter must be LAN controlled if remote workstation control, by a user, is to be achieved. It is not only necessary to control the functions of these devices over the LAN, the status of these devices must also be communicated over the LAN. This enables the user to know exactly what SMPTE time the SMPTE generator and SMPTE to MTC converter are at. It also enables the workstation user to determine when a recorder has started or completed a function.

MTC produced by the SMPTE to MTC converter is routed by the MIDI patcher to the workstation MIDI sequencer. The MTC is used by the MIDI sequencer to synchronise to the recorder. The MTC being received at the workstation side could be used to perform other synchronisations. This is discussed further in section 6.2.5.2.

MIDI sequencers, such as Steinberg's Cubase, incorporate the control of MIDI machine controlled recorders [66]. For full control of MMC recorders a MIDI interconnect carrying MIDI data from the recorder to the MIDI sequencer is required for each recorder. The MMC specification does not recommend merging the MIDI data produced from each recorder because of the introduction of time delays [56 p3]. These time delays prevent the timeous reception of machine control responses by the MIDI sequencer. The MIDI sequencer may therefore continue supplying machine control commands to an already overburdened recorder.

The described network topology will work, but only if a single recorder is used by a workstation. This is because there is only a single MIDI interconnect carrying MIDI information from the

main studio to a workstation. Increasing the number of MIDI interconnects from the main studio to the workstations is not a viable solution, as it is desirable to keep the number of interconnects as small as possible. Also the number of interconnects between the main studio and the workstation should be device independent. Introducing a new device to the main studio should not require the installation of a new interconnect to each workstation.

The best solution to introducing MMC recorders into the computer music network, as it stands, is to create a multiuser MMC controller within the main studio. This MMC controller will have all the MMC controlled recorders attached to it. Workstation users can then patch their MIDI sequencer to the MMC controller using the MIDI patcher. The MMC controller will relay the machine control commands and responses between the workstation MIDI sequencers and the recorders. The machine control responses are merged by the MMC controller before being sent to the workstation MIDI sequencer. The inherent problem of data delays is overcome by the MMC controller which intelligently buffers the flow of machine control commands and responses. The MMC controller will also perform the necessary synchronisation of multiple recorders. This will be done by relaying the MTC produced by the master recorder to the slave recorders. The MMC controller will route the MTC to the workstation MIDI sequencer, enabling it to synchronise to the recorders.

CHAPTER 4 - ESSENTIAL SUBSYSTEMS

This chapter details the implementation of those subsystems that are required for the construction of the computer music network described in chapter 3. These subsystems are the audio processor patcher/mixer and the MIDI patcher. The audio processor patcher/mixer is required for the processing, patching and mixing of audio signals. The processing features permit the rudimentary modification of the tonal attributes of the audio signals entering the system. The patching and mixing features of the system allow any audio source to be patched to any group of audio sinks and the level at the patch points to be controlled. The MIDI patcher is required for the patching of MIDI signals. It permits the workstation to connect to multiple MIDI devices in the main studio, for the communication of MIDI data between the workstation and the MIDI devices.

4.1 AN AUDIO PROCESSOR PATCHER/MIXER

The aim of this section is to explain the development of the hardware necessary for the creation of a remotely configurable audio processor patcher/mixer (APPM). The APPM is responsible for the processing, routing and mixing of audio signals within the computer music network. It resides in the main studio together with all the other devices shared on the network. The APPM is intended for a real time network of computer music workstations, in which the workstations and their users are in close proximity to the main studio. Analogue audio is passed between the main studio and the workstations, making long transmission lengths undesirable, because of signal loss and distortion. The system must provide the same functionality as a modern professional mixer to each of the workstation users [17 p461]. The remote multi-user audio mixing requirements for the network will ensure that the APPM can also operate in current single user studios.

4.1.1 Description of Functionality

The APPM is capable of the real time processing, routing and mixing of a number of audio outputs from various audio generators, audio modifiers and audio recorders. It creates a set of audio output signals to feed the audio modifiers, audio recorders and workstation users. Each remote workstation user can have real time control over the processing, mixing and routing capabilities of the APPM. These capabilities are supplied to him by the APPM, the network server, and the network software. Figure

4.1 shows the APPM and its connection types within the network.

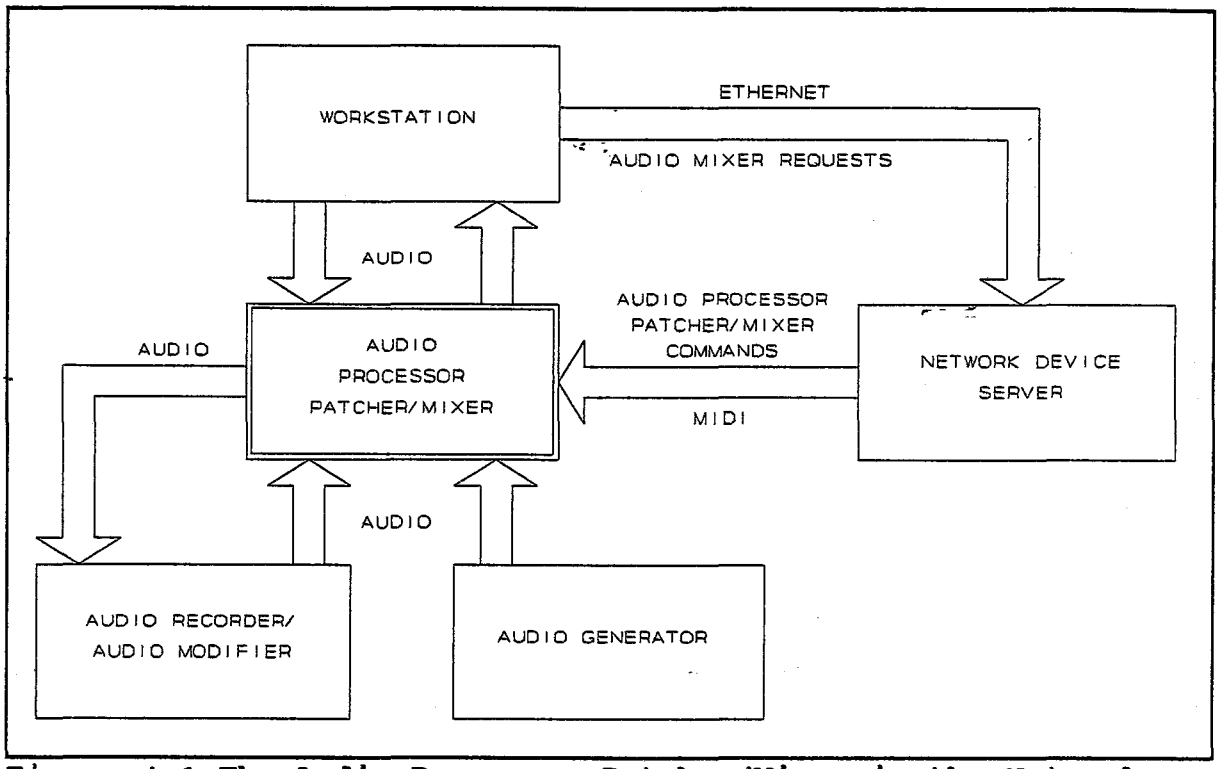

Figure 4.1 The Audio Processor Patcher/Mixer in the Network

The processing features of the APPM give the workstation user the ability to alter at least the gain and tonal attributes of the audio signals feeding a particular audio input of the APPM.

The number of inputs and outputs of each of the units creating the APPM is fixed. The APPM is however expandable, both to accept a larger number of audio inputs and produce a larger number of audio outputs. It is possible to connect a number of the basic audio processor, patcher/mixer units to produce a larger APPM. This will allow for the creation of a larger network with many workstations, audio generators, audio modifiers, audio recorders, and video recorders.

The ability to expand the APPM requires that the phase of the audio signals at its outputs be the same as at the inputs; no phase inversion must occur. This will prevent possible cancellation of the audio signals by the mixing of uninverted and inverted audio signals.

The APPM is composed of two types of units:

1) The Audio Patcher/Mixer Unit (APMU) which is responsible for both the routing and mixing of audio signals.

2) The Audio Processor Unit (APU) which is responsible for gain and equalisation control.

Each audio input to the APPM only needs to pass through a single gain and set of equalisation controls, whereas it may be required to pass through several patcher/mixer controls. Hence, to reduce redundancy of hardware, it is necessary to split the hardware for doing these two functions between two different types of units.

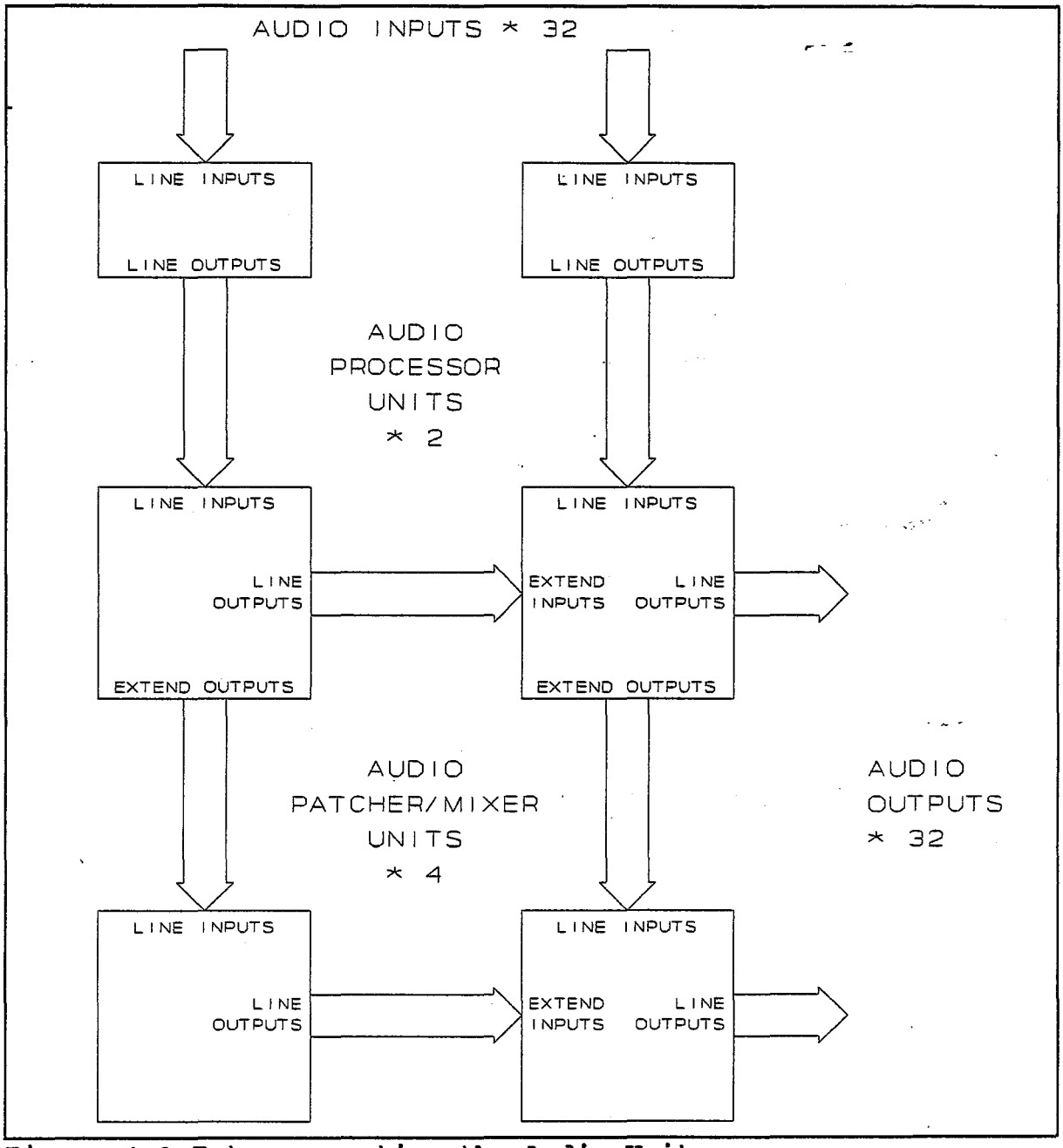

Figure 4.2 Interconnecting the Audio Units

Figure 4.2 illustrates how the two different types of units are interconnected. It shows how an audio signal only flows through one APU, but can flow through many APMU's. It also shows how

several units can be interconnected to create an APPM with an increased audio input and audio output handling capacity.

Commercially available MIDI controlled audio effects units can be incorporated into the structure of the APPM. These units can be placed between the APUs and the APMUs and would be controlled by a network server. This can be used to overcome the limited audio processing abilities supplied by the APU and to supply any required specialised audio effects.

Internal control over the APMUs and APUs composing the APPM will be by way of a dedicated microprocessor control unit residing in each respective unit. Commands are sent to these microprocessor control units via MIDI (see figure 4.1). This microprocessor control unit is discussed in section 4.3, as it is also relevant to the MIDI patcher.

The APPM is required to sink both domestic line level signals (-10 dBu) or studio line level signals (0dBu) [17 p472] and to source both domestic and studio line level signals. This will enable the APPM to be used with both domestic and studio audio devices. The use of domestic audio devices in conjunction with studio audio devices occurs widely in many budget- conscious studios today. Some studio audio devices do not conform to any standard and therefore require that their signal levels be modified. The audio generators, audio modifiers [21 p48] [22 pAdd-39], and audio recorders [20 p7] within the main studio all operate at either domestic or studio line level, or somewhere inbetween the two levels. The audio signals passing between the main studio and the workstations are at studio line level to reduce the effects of noise. It is the responsibility of the gain controls within the APU to alter audio signals to the desired level (see appendix 1.1 and 1.3 for input and output audio signal range specifications).

If other devices such as microphones or acoustic instruments are to be connected to the APPM, preamps or normal manual studio mixers will have to be used to raise the signal levels to line level. Audio differential line drivers and receivers can be used to further reduce the amount of noise introduced into the audio signals passing between the main studio and the workstations.

The APPM does not supply the user at a workstation with information regarding the relative levels of audio signals. This function is supplied by level indicators coupled to the two audio

lines feeding each workstation. The workstation user can patch the desired audio signal to these lines in order to monitor its level. This requires that he set the gain level for each audio signal in turn. First he creates a patch between the workstation and the audio device generating the desired audio signal. The patch is direct and should not be attenuated. All other patches to the workstation should be rerouted or muted. The workstation user then sets the equalisation and alters the gain level to determine settings which will produce enough dynamic range in the audio signal, while not exceeding the signal handling capacity of the APPM.

The specifications for the APPM were decided upon after assessing the requirements of the system within the network being developed, and current commercial studio mixing requirements [43 p32]. The application specifications had to allow the APPM to be used in all current single user and future multiple user computer music studio applications. The technology employed within the system should be assessed with these criteria in mind.

4.1.2 Feasibility and Technology Study

Having assessed the requirements of the APPM, the next step was to locate available technologies for producing the hardware for the APMU and the APU. Hardware was required for the gain and equalisation controls of the APU, and for the routing and mixing of audio signals within the APMU. Methods for remote control over these gain and equalisation controls, and routers and mixers also had to be investigated. Technologies for the remaining electronics, namely the power supplies and audio input and output buffers, are readily available, as are the construction technologies.

The investigation into the audio processing technologies involved looking at the processing of audio both in the analogue and digital domains. Processing and recording audio in the analogue domain has well established techniques and technologies [24 p1]. It is difficult to process signals in the analogue domain under digital control and meet acceptable distortion requirements [41 p159]. This is mainly due to interference between the digital and audio signals.

The processing and recording of audio in the digital domain is still a young technology, relying heavily on current computer technology. As computer technology improves so does the digital

audio processing and recording technology [24 p4]. Standards for the transmission of digital audio are also still in the evolutionary stage [39]. Currently, it is expensive to use digital audio technology. It requires the use of some of the fastest computing technologies available for processing digital signals in real time [24 p137]. A high density recording medium is required for its storage [24 p248]. The communication of digital audio requires interconnects with high bandwidths, especially if multiple digital audio channels flow down the same interconnect [39]. There are a number of advantages to processing and recording audio in its digital form. No distortion of signals occurs in the digital domain [24 p1], remote control over processing is simple to implement, and the degree of control the user has over the audio signals is much greater than in the analogue domain [24 p136].

An APPM in which the processing of all the audio signals occurred in the digital domain would need a number of digital and digital/analogue devices. The first requirement would be analogue to digital (A/D) converters to convert the analogue audio signals to quantised signals in the digital domain [24 p35]. In order to realise acceptable distortion levels and bandwidth, A/D converters capable of sampling at a rate of at least 40 kHz [24 p27] and with a resolution of at least 16 bits [24 p18] would be required. Multiple digital signal processors (DSPs) would be required to process the audio signals, with several microprocessors being required for the routing of digital information to and from the DSPs as well as for controlling the DSPs. The large number of DSPs and microprocessors are required because of the large amount of digital information being produced by the A/D converters [24 p136]. Finally, digital to analogue (D/A) converters are required to convert digital audio signals back to the analogue domain [24 p29]. The cost of such a system is prohibitively large for studios with limited budgets. Implementing and debugging such a system would also be extremely difficult due to its multiprocessing requirements and real time constraints.

Having considered the requirements of processing audio signals in the digital domain, it became obvious that it would be easier and cheaper to leave the audio signals in the analogue domain and control their processing by way of some form of digital/analogue interface.

There are currently three techniques by which the level of

analogue audio signals can be controlled digitally. The first of these, and by far the most common, is the use of Voltage Controlled Amplifiers (VCAs) [41 p149]. VCAs have been designed to produce mixers meeting the specifications required in todays modern studios, while adding the required mixing automation. Computers generate the required control voltages for the VCAs by using D/A converters.

The second technique involves using D/A converters as digitally controlled resistors in attenuators [41 p150]. This technique is used in a number of commercially available audio faders [2 p35]. It does however tend to inject a lot of noise, called 'zipper noise', into the audio signal due to capacitive coupling between the audio and digital paths, and the large number of solid-state switches that open and close in the audio path when the level is changed. For this reason it is avoided when high quality audio reproduction is required.

The third technique involves using discrete solid-state switches to create a digitally controlled attenuator. Solid-state switches are available as digitally controlled multiplexers. These multiplexers can be used to create switched resistor networks which are the building block of the digitally controlled attenuator. This technology provides a solution for the required routing of audio signals, as well as for attenuation. This is an advantage which neither the VCAs or D/A converters can provide. High quality integrated digitally controlled resistor attenuators have become available subsequent to this implementation [70].

Problems associated with using discrete solid-state switches can be largely overcome using discrete devices. The first and greatest problem associated with solid state switches is capacitive coupling between the audio paths and the digital control paths. This problem permits the harmonics found in the digital control signals to bleed into the audio signals. It can be prevented by keeping digital paths as far away from audio paths as possible and by separating them with a ground plane. The problem cannot be totally cured however, due to capacitive coupling between the switch control signal and the switch itself. Other techniques which can be used to further reduce the problem are the placing of low pass filters in the audio signal path to filter out the unwanted ultrasonic harmonics, and reducing the effective number of switches within the audio path.

The second problem associated with solid-state switches is that

of a nonlinear response, due to the variation of the switch resistance with applied potential. This can be overcome by operating the solid-state switches at a constant potential, thus minimizing the effects of their nonlinear response [19]. The resistance of a closed solid-state switch is constant with a fixed applied potential and therefore no nonlinear distortion will occur to the applied signal. The nonlinearity of the solid-state switches is further reduced by choosing the supply voltage for the solid-state switching devices with care [25 p7-45].

The use of discrete solid-state switches, throughout, in the creation of the APPM results in an elegant design which is not complicated by the use of diverse technologies and their associated interfacing problems.

4.1.3 The Digitally Controlled Attenuator

The Digitally Controlled Attenuator (DCA) is designed to perform the task of a high quality VCA for the purpose of level control in the APPM. This includes controlling gain levels, equalisation levels and mix levels. The level is directly digitally controlled and does not require the generation of an intermediate voltage as in the case of the VCAs. The DCA consists of a series of potential divider ladders each performing a different, but related, level of attenuation. The attenuator with the finest level control is the first in the audio signal path. The attenuators are connected in series, in order, with the attenuator with the coarsest level control coming last. This is to maintain the maximum signal to noise level ratio within the DCA [17 p300].

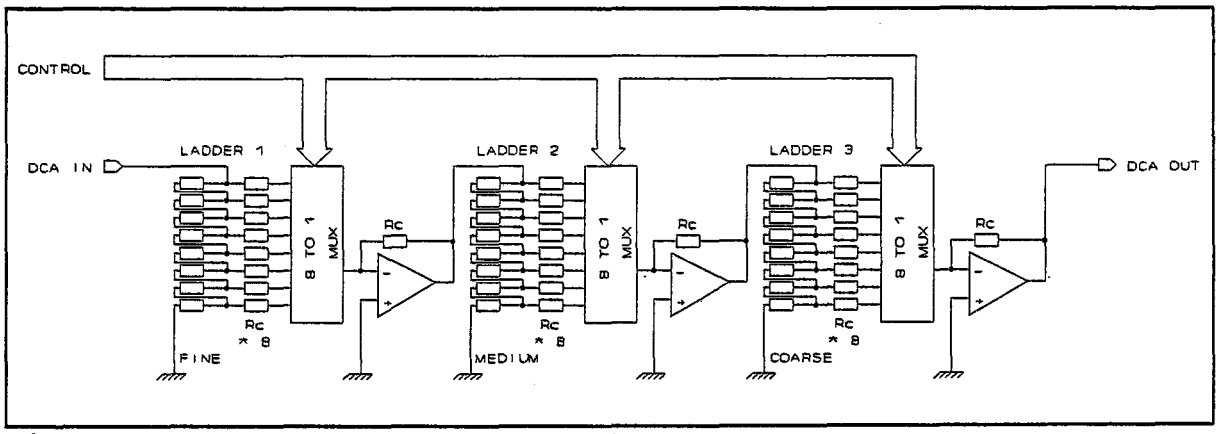

Figure 4.3 **Fundamental set-up of a DCA**

As can be seen from figure 4.3, Rc is connected via the multiplexer to the inverting input of the operational amplifier.

The inverting input of the operational amplifier is a virtual earth, as its noninverting input is coupled to earth. The potential divider ladder can therefore be simplified to the configuration shown in figure 4.4. Each voltage divider in the ladder is formed by resistors Ra and Rb.

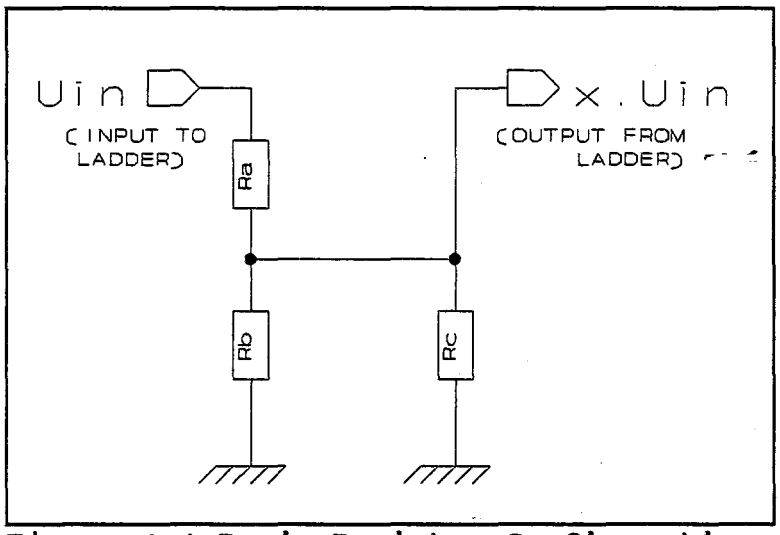

Figure 4.4 Basic Resistor Configuration of the ladders

The calculation of the potential divider resistors required by the DCA is formulated below:

The input potential (Uin)
The attenuation factor (x) $\quad 1 <= x <= 0$

$$x = \frac{\frac{RbRc}{Rb + Rc}}{\frac{RbRc}{Rb + Rc} + Ra} = \frac{RbRc}{RbRc + RaRb + RaRc}$$

then $\quad xRbRc + xRaRb + xRaRc - RbRc = 0$
then $\quad Rb\,(xRc + xRa - Rc) = -xRaRc$

then $\quad Rb = \frac{xRaRc}{Rc - xRc - xRa}$
if $\quad Ra + Rb = Rd$
then $\quad Ra = Rd - Rb$
therefore $\quad xRbRc + xRbRd - xRb^2 + xRcRd - xRbRc - RbRc = 0$
then $\quad xRb^2 + Rb(Rc - xRd) - xRcRd = 0$
then $\quad Rb^2 + Rb(Rc/x - Rd) - RcRd = 0$
then $\quad Rb = \frac{Rd - Rc/x + \mathrm{sqrt}((Rc/x - Rd)(Rc/x - Rd) + 4RcRd)}{2}$

Having now calculated Rb for a specific value of x, the corresponding value of Ra can be calculated. The values of Rc and Rd are constant and are determined from the operational amplifier's specifications. Rc and Rd should not excessively load the output of the operational amplifier, as this will lead to distortion. The values of x depend on the total attenuation required, the number of division ladders in the DCA and the number of divisions within the ladder. A program for the calculation of the resistor networks was written to facilitate the design of the DCA, and can be found in appendix 7.1. DCA resistor values generated by this program and used in the construction of the prototype DCA can be found in appendix 7.2. This DCA fulfils all the requirements for level control within the APPM.

Unfortunately the resistor values in these ladders cannot be realised in practice as resistors are only manufactured with specific values. This problem can, however, be overcome by using two resistors in parallel (Rx and Ry) to approximate the required resistor (Rf) as closely as possible.

$$Rf = \frac{RxRy}{Rx + Ry}$$

then

$$Ry = \frac{RxRf}{Rx - Rf}$$

Now we can choose a resistor (Rx) and calculate the resistor (Ry) which is required in parallel in order to create the correct resistor (Rf).

In the prototype, carbon resistors with a tolerance of five percent were employed. Better results could have been achieved if metal oxide resistors had been employed. Metal oxide resistors have a tolerance of 2 percent or better, and therefore produce more accurate attenuation steps within the DCA. They also produce less noise than their carbon counterparts. The resistor ladders required by the DCA could be integrated, as could the entire DCA. The integration of the resistor ladders would be the next logical step, as it would reduce the size of the DCA, while improving accuracy and decreasing distortion. The component requirements for the prototype DCA can be found in appendix 4.1.

4.1.4 The Audio Processor Unit Hardware

This section describes the hardware required for the gain and

equalisation controls within the APU. The APU supplies the necessary gain controls and most of the equalisation requirements within the APPM. The APPM handles all the audio signals within the main studio of our computer music network. Figure 4.2 shows how the APU fits into the APPM. The gain and equalisation for each audio input signal to the APU is performed by an audio processor. This modular approach aids in prototyping, in constructing, and in trouble shooting the hardware. The gain and equalisation controls within an APU are controlled by a microprocessor control unit resident within the APU. The microprocessor control unit is described in section 4.3, and figure 4.14 within that section is an internal block diagram of the APU.

4.1.4.1 Hardware Decisions

A decision had to be made regarding the number of line inputs and outputs available on an APU. Each audio processor within the APU has a single audio sink and produces a single audio source. This audio source can be split to produce several outputs. The number of audio inputs is therefore dependent on the number of audio processors within the APU. The number of audio outputs is both dependent on the number of audio processors and to how many outputs the audio processor's audio source is split. The number of audio processors that can be included into a single unit is dependent on the volume of hardware required to create the audio processors. The volume of hardware is dependent on the amount of functionality an audio processor supplies.

Each audio processor must supply a gain control. This is necessary to cater for variation in levels produced by different devices. It also allows compensation for variation in the levels of signal produced by the same device. A variation of level can occur from recording to recording on audio recorders and between different voices on audio generators. The audio processors must then also supply audio effects. The normal audio effects associated with manual analogue audio mixers are limited to equalisation. Other effects such as reverberation [17 p230], delay [17 p 222] and echo can more successfully be implemented digitally. Their analogue counterparts are far more bulky and offer far fewer features. Manual analogue audio mixers compensate for this problem by permitting external audio effects units to be coupled to them [17 p233]. Our audio processors also perform their audio processing within the analogue domain. They are therefore subject to the same limitations as manual analogue

audio mixers. The APPM is able to support external audio effects units, as discussed in section 4.1.1.

A complete assessment of the effects offered by manual audio mixers needs to be made before deciding on the effects to be supplied by the APU. The equalisation offered by a manual analogue audio mixer normally consists of a bass, treble and a parametric midrange control for each audio input to the audio mixer. The bass control permits the user to boost or cut the lower frequencies in the audio spectrum. The treble control permits the user to boost or cut the higher frequencies in the audio spectrum. The midrange, being parametric, permits control over the parameters of centre frequency, bandwidth/Q, and the amount of boost or cut [17 p289]. The need for a parametric midrange arises out of the necessity to correctly place a human voice within the audio spectrum of a composition. This is done by cutting the frequencies of audio generators in the area of the spectrum where the vocalist sings. The balancing of audio signals within a composition, provided by the equalisation controls, ensures that each signal occupies an area of the audio spectrum [17 p297]. This prevents one signal from obscuring another. The human ear is also most sensitive in the midrange of the audio spectrum [24 p8], making the extra control provided by a parametric midrange, in this part of the audio spectrum, very useful.

Parametric equalisation is not the only technique by which effective control of the midrange spectrum can be achieved. Graphic equalisation can also be used to provide this control. A graphic equaliser divides the audio spectrum into several, set centre frequency, set bandwidth, boost/cut controls [17 p289]. The position of these controls gives a graphic depiction of the output response of the graphic equaliser. The user can then boost or cut these fixed bands within the audio spectrum. When all the boost/cut controls are set flat, there is no gain or attenuation over any part of the audio spectrum. The output of the graphic equaliser exhibits a flat frequency response.

Equalisation within the APU was implemented using a graphic equaliser. Implementing a parametric midrange using a state-variable (multifeedback) filter [61 p175] [62 p30] required four digitally controlled attenuators (DCAs), two to control the centre frequency and one each for the bandwidth and boost/cut controls. Poor stability, manifesting itself as oscillations, arose due to the large number of operational amplifiers required

to make a digitally controlled state-variable filter using this technique. It is easier to divide the audio spectrum up into several bands using filters. A DCA is used to boost or cut the signal produced by each filter. The outputs of the DCAs are mixed together to create the graphic equaliser output, as illustrated in figure 4.5.

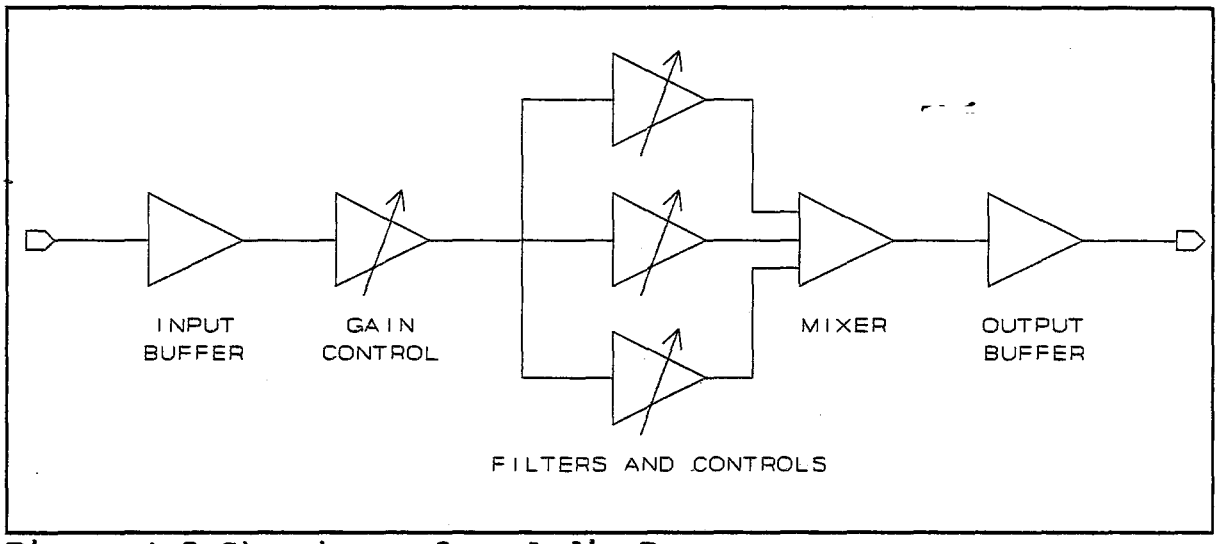

Figure 4.5 Structure of an Audio Processor

The graphic equaliser must exhibit a flat frequency response with boost/cut controls set flat. There must also be no phase shifting of signals. These forms of audible distortion must be avoided in the design of graphic equalisers.

The prototype graphic equaliser has only three filters, one for bass, one for midrange, and one for treble. This allowed for the creation of an APU with sixteen audio processors, enough to feed a single APMU, described in section 4.1.5. The number of bands available within the graphic equaliser, of each audio processor, can be increased to as many as are necessary. The larger the number of bands required however, the fewer the number of audio processors that will fit into an APU. This occurs due to the space required by the additional hardware, and the need to keep the APU to a uniform size. In some cases, such as in the use of audio recorders, equalisation may not be required. The equalisation circuitry can therefore be omitted from the APU on those particular audio channels.

The ability to expand the APPM requires that the phase of the audio signals at its outputs be the same as at the inputs; no phase inversion should occur. This will prevent possible cancellation of the audio signals by the mixing of noninverted

and inverted audio signals. The APU was designed so that no phase inversion occurred within the audio processors.

The specifications for the APU were decided upon after assessing the requirements of the unit within the APPM (see section 4.1.1). The specifications given in appendix 1.1 and 1.2 are the minimum requirements for the APU and give the quantisation of the gain and equalisation controls in the implementation.

4.1.4.2 Hardware Design

The APU Circuitry is comprised of four sections :
1) The Line Input Buffer Circuitry.
2) The Gain Control Circuitry.
3) The Equalisation Circuitry.
4) The Line Output Buffer Circuitry.

The line input buffer circuitry ensures an impedance match between the source of the audio signal and the gain control circuitry. The gain control circuitry permits the alteration of the audio signal level produced by the audio source. This ensures that all audio signals being mixed have approximately the same peak level and optimum signal-to-noise ratios. The equalisation circuitry allows a remote workstation user to set the bass, midrange, and treble boost or cut controls for a particular audio source. This feature will also permit the automated change in tonal settings. The line output buffer circuitry ensures that the outputs of the APU have a sufficiently low impedance to drive several audio sinks, and that the phase of the audio output is correct. The circuit diagram of the audio processor can be found in appendix 2.2 and the part list in appendix 4.2.

4.1.4.3 Hardware Prototyping, Implementation and Testing

Prototyping the audio processor was necessary to evaluate and improve its behaviour. The original prototype was built on a single sided printed circuit board. The density of components on the circuit board was not important here. Control was achieved by a simple hardware interface to an IBM AT's bus. This interface provided two octal latches for storing data for the DCAs, from the AT's data bus. It also provided strobe lines for transferring this data to the gain and equalisation DCAs in the audio processor. Software was written to control all the functions on the audio processor. An audio signal was injected into the audio processor and the output then monitored. The prototype behaved

extremely well. There was no interference on the audio signals by the digital signals. The gain and equalisation level quantisation gave no audible indication that a step type attenuator was used. The complete unit could then be constructed as detailed in appendix 2.2 and 2.5.

4.1.4.4 Evaluation of the Unit

The unit was evaluated using a small test network described in chapter 5. This permits workstation access to the gain and equalisation controls provided to modify the audio signal sources of booked devices. The unit performed well in this environment. It did however take longer to set gain and equalisation levels than on a conventional mixer. This was a problem with the user interface though and not the APU. The user interface could be changed in a number of ways to improve control over gain and equalisation levels, as described in section 6.2.5.1 and 6.2.5.2.

4.1.5 The Audio Patcher/Mixer Unit Hardware

This section describes the hardware required for the routing and mixing of audio signals within the APMU. The APMU supplies the necessary routing and mixing requirements within the APPM. The APPM handles all the audio signals within the main studio of the computer music network. Figure 4.2 shows how the APMU unit fits into the APPM. For ease of construction the design had to be modular. The construction of one of each module was all that was required for testing and evaluation. The APMU is controlled by an internal microprocessor control unit. The microprocessor control unit is described in section 4.3, and figure 4.15 within that section is an internal block diagram of the APMU.

4.1.5.1 Hardware Decisions

The internal structure of the APMU can be conceptualised as a two dimensional matrix with inputs on one axis and outputs on the other. This can be seen in figure 4.6. A module had to be created which would perform the routing and mixing of audio. The module is required to route a single audio input to the APMU to a single audio output from the APMU. The module would also be required to control the level of the audio signal appearing at the output. The mixing of audio signals at the outputs from the APMU requires that the output of a module should be able to mix with those of several others. The functions of the module should all be digitally controllable. The audio patcher/mixer node (hereafter

abbreviated to "node") was created to fulfil these requirements. These modular nodes supply all the mixing and routing requirements for the APMU.

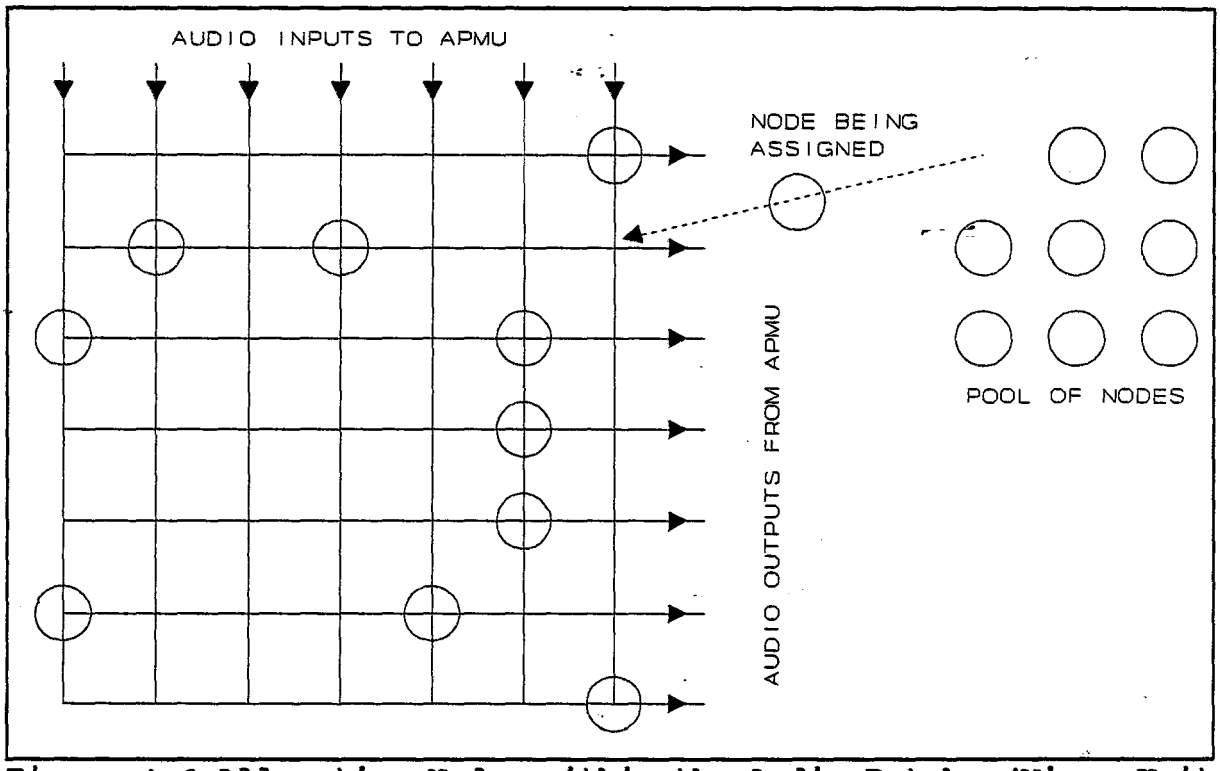

Figure 4.6 Allocating Nodes within the Audio Patcher/Mixer Unit

The nodes do not have a fixed position within the matrix. They can be assigned to any crosspoint within the matrix. Unassigned nodes belong to a 'pool of nodes'. A node is assigned from this pool to a crosspoint in the matrix, making a patch between a particular audio input and output. The level of signal passing from the input to the output is controllable, permitting the desired mix control. Nodes can also be removed from the matrix and assigned back to the pool of nodes. A mix scenario with mixing occuring at each point within the matrix will not occur within the computer music network. The idea behind having a pool of nodes is to overcome the need for having a mix point at every point within the matrix, and therefore reduce hardware redundancy. A system which has mix points at every point within the matrix can be created using Richmond Sound Design audio fader modules [68]. Such a system has more redundant hardware than the node based system.

A decision had to be taken regarding the number of line inputs and outputs available on an APMU. Too few inputs require that one needs a larger number of APMUs in order to supply the desired audio routing and mixing capabilities. This also has the effect

of increasing the amount of MIDI data required to alter mix levels and patch settings within the APPM. This is because MIDI messages are only specific to a single unit within the APPM. A MIDI message contains data identifying which unit the data is intended for. You therefore have the overhead of identification data when transmitting MIDI messages to the units. This results in a decrease in the transmission rate of effective data over that particular MIDI line. The format of these MIDI messages is shown in appendix 5 and is discussed in more detail in section 4.3.1.

On the other hand, an APMU with too many inputs requires an excessively large amount of electronics within the unit to perform the necessary audio routing and mixing. This makes it both clumsy in size and difficult to construct. The number of line inputs is tied to the node implementation technology. The nodes use analogue multiplexers, constructed from solid-state switches, to select the audio input signals. Typically there are only 8 or 16 switches in commercially available analogue multiplexers [25]. A good compromise between all the above constraints was an APMU with 16 inputs.

The factors governing the choice of the number of line inputs to the unit affects the choice of the number of line outputs. The audio signal output destination is also controlled by an analogue multiplexer constructed from solid-state switches. It is important that the number of audio inputs and audio outputs are equal. This ensures that the stacking of units is kept simple, as coupling of APMUs occurs on a one to one basis. It was clear that the APMU should have 16 outputs. The resulting APMU has enough audio inputs and outputs to perform most audio routing and mixing requirements.

An important aspect of the APMU is the number of nodes available within each unit. While too few nodes would result in an inadequate audio patching/mixing capability, too many would result in unnecessary redundancy of nodes and expense in construction. While it may be considered necessary to be able to route and mix every line input to every line output, in reality, at any given time, the routing and mixing facilities required are only a fraction of the functionality supplied by such a unit. The choice of the number of nodes is dependent on the number and types of devices to which the APPM is connected, and the mix scenarios that the users of the system will wish to implement. These requirements are not entirely predictable, but generally

follow accepted rules. It is, for instance, accepted that an audio output channel of a device should not be connected to the input of the same channel.

There are three classes of devices with audio outputs: audio generators, audio modifiers and audio recorders. There are three classes of devices with audio inputs: audio modifiers, audio recorders and workstations. Allowing each audio signal entering the APMU to be connected to one of each three classes of devices with audio inputs, requires 3 by 16 nodes, or 48 nodes. The APMU would then provide routing and mixing facilities similar to a 16 channel conventional audio mixer. Within the network, the APMU can share these facilities amongst multiple users.

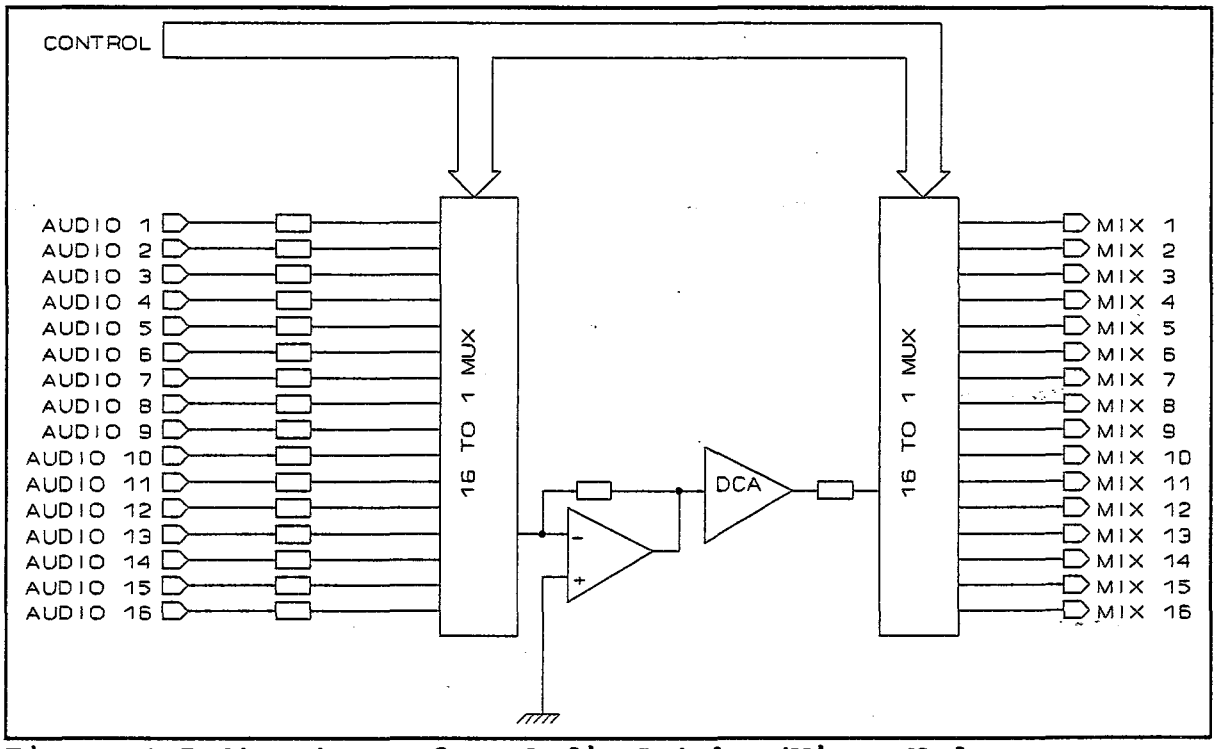

Figure 4.7 Structure of an Audio Patcher/Mixer Node

The maximum number of nodes per unit was fixed at 48. The number of nodes within an APMU can, however, be chosen to suit a particular application. An APMU can be constructed with 1 to 48 nodes present within it. This limit can be exceeded by stacking units in parallel. This means that the number of nodes available in the area of the matrix occupied by an APMU, can be increased by multiples of 48 until there is a node available for each crosspoint in the 16 by 16 matrix. The structure of the node is shown in figure 4.7. It comprises an analogue input multiplexer to select the source of audio signal, a digitally controlled attenuator to attenuate this signal, and an analogue output

multiplexer to mix the audio signal, created by the DCA, onto the correct output line.

On power up, or the resetting of an APMU, the input and output analogue multiplexers within the nodes are all disabled. This means that no audio signals are present on the inputs of the DCAs within the nodes. The audio output signals of the DCAs are not routed to any output. The DCAs are also set to have the maximum attenuation possible. This ensures that the initial and reset state of the system is known. It also minimises any chance of unwanted crosstalk between audio signal paths. Nodes are placed in this state every time they are disconnected.

The user has enough dynamic range to entirely mute an audio signal if so desired. Due to the fact that mix levels will be under digital control, the mix level is quantised into a series of attenuation steps. It is, therefore, very important that the size of the step be sufficiently small so as not to be audible [41 p151]. A step size of 0.2dBs was chosen. This is smaller than commercially available audio attenuators [2 p36]. The total human acoustic range, from the lowest audible sound to the threshold of pain, is of the order of 120dB [41 p38]. Audio mixers that offer a 100dB range of mix control are quite acceptable. It was necessary to further quantise the mix levels, because the number of 0.2dB attenuation steps required to meet the 100dB range was larger than the 128 MIDI steps available. To communicate a number higher than 128 by MIDI requires two bytes. This would affect the rate at which changes could be made to mix levels, as it takes longer to communicate two bytes than one. The further quantising of the mix levels was based on the technique used by Iota's MIDI-Fader [2 p36]. This involves increasing the size of the attenuation step with greater attenuation levels, as detailed in appendix 1.3. It is more difficult to audibly perceive the increased step size with lower volumes.

The specifications for the APMU were decided upon after assessing the requirements of the unit within the APPM (see section 4.1.1). The specifications given in appendix 1.3 and 1.4 are the minimum requirements for the APMU and give the quantisation of the mix levels in the implementation.

4.1.5.2 Hardware Design

The APMU audio routing and mixing circuitry is comprised of three sections:

1) The Line Input Buffer Circuitry.
2) The Node Circuitry.
3) The Line Output Buffer Circuitry.

The function of the line input buffer circuitry is to ensure that the impedances of the devices feeding audio onto the APMU's inputs match the input impedance of the nodes, ensuring that a fanout problem does not occur. This problem occurs when the load impedance is too low. It causes strong attenuation of the source signal and the possibility of distortion. The circuitry also allows for expandability by supplying a buffered thru for each audio input. This audio thru is used to feed other APMUs required to route and mix the same audio signals. Appendix 2.3.1 shows the circuit diagram for the line input buffer circuitry, and appendix 4.3.1 supplies the component part list required by this circuitry.

Specific techniques were employed in the design of the node printed circuit board to obtain the best possible results. Audio and digital paths were kept as short as possible. The audio paths were placed on one side of the PCB and the digital paths on the other. All unused areas of the PCB were grounded. This was done to prevent crosstalk between the digital tracks and the audio tracks [41 p159]. The power supply rails are decoupled at regular intervals, by capacitors, to ground, primarily to stabilise amplifier operation, but also to prevent crosstalk via the power supply rails. A further technique that could have been used to minimise crosstalk would have been the introduction of a ground plane into the board. This was not deemed necessary as the prototype worked well without a ground plane. The circuit diagram of the node can be found in appendix 2.3.2, a printed circuit board layout in appendix 3, and the part list in appendix 4.3.2.

The line output buffer circuitry ensures that the outputs of the APMU have the correct output level for an associated input level, that the output impedance is sufficiently low, and that the phase of the audio output is correct. The inversion of audio signals, corresponding to a 180 degree phase shift, is undesirable, as the side effects cannot be determined for a system with such diverse applications. This is achieved by reinverting the inverted signal produced by the nodes. The buffer circuitry takes the mix bus, containing the outputs from the nodes and the mix extend audio inputs, as its input. The mix extend audio inputs allow external access to the APMU's mix busses, allowing for necessary expansion. The outputs of several APMUs can be mixed together,

with the mix extend inputs permitting an APMU audio output to contain audio signals from more than sixteen audio sources. The circuit diagram of the line output buffer can be found in appendix 2.3.3 and the part list in appendix 4.3.3.

4.1.5.3 Hardware Prototyping, Implementation and Testing

Prototyping the node was necessary to evaluate and improve its behaviour. The original prototype was built on a single sided printed circuit board. The density of components on the circuit board was not important here. Control was achieved by a simple hardware interface to an IBM AT's bus. This interface provided two octal latches for storing data for the node, from the AT's data bus. It also provided strobe lines for transferring this data from the latches to the input and output multiplexers, and the DCA, of the node. Software was written to control all the functions on the node. Audio signals were injected into the node and the outputs then monitored. The prototype behaved extremely well. There was no interference on the audio signals by the digital signals. The mix level quantisation gave no audible indication that it was a step type attenuator. A double sided printed circuit board was then developed for the node. The node was again evaluated using the AT and it performed as well as the prototype. The complete unit could then be constructed as detailed in appendix 2.3 and appendix 2.5.

4.1.5.4 Evaluation of the Unit

The unit was evaluated using a small test network described in chapter 5. This permits workstation users to create patches and to set mix levels for booked devices. The unit performed well in this environment. It did however take longer to set mix levels than on a conventional mixer. This was a problem with the user interface though and not the APMU. The user interface could be changed in a number of ways to improve control over mix levels, as described in section 6.2.5.1 and 6.2.5.2.

A surround sound application was also developed using the APMU. This system permitted a user to specify, with a simple notation, how mix levels change with time. The computer into which these values are entered is synchronised to the sound source using MIDI Time Code, or SMPTE (see section 2.3 for a description of synchronisation techniques). This computer also has direct real time control over the APMUs. Each audio mix produced by the APMUs is fed to a loudspeaker in the sound field. This allows the user

to specify, where in the sound field a particular sound image appears to originate. This system permitted investigations into two and three dimensional surround sound effects, using only sound level control. The system has been used successfully for several shows, including a slide show accompanied by music given at the 1993 National Arts Festival.

There was however one problem with the APMU. There was cross-talk between the digital data controlling the nodes and the audio produced by them. This was more prominent on certain audio outputs. The source of the problem was capacitive coupling between the mix bus, which is fed by the output multiplexers of the nodes, and the digital signals used to control the multiplexers. The coupling occurs on all the nodes, thereby compounding the problem. This is the reason why the problem was missed during prototyping. Only one node was built for prototyping purposes. The problem could be overcome by introducing a ground plane into the node printed circuit board. This would prevent capacitive coupling between the digital and audio signal paths as they are on opposite sides of the PCB, as shown in appendix 3.

4.2 A MIDI PATCHER

The aim of this section is to explain the development of the hardware for a remotely configurable MIDI patcher. The MIDI Patcher is intended for the real time routing of MIDI in a computer music network. The workstations are in close proximity to the main studio due to the limited transmission distance of MIDI. The MIDI patcher resides in the main studio. There it is required to perform the MIDI routing of MIDI messages from the MIDI workstation sequencers to the MIDI controlled devices within the main studio. It must also return MIDI messages, generated by the MIDI devices within the main studio, to the workstation MIDI sequencer.

4.2.1 Description of Functionality

The MIDI patcher is capable of the real time routing of MIDI messages between the MIDI sources and the MIDI sinks in the computer music network. The MIDI patcher only supplies routing facilities, as the computer music network design only requires this capability from the system. It is not required to perform MIDI merging, filtering or transformations. These facilities, if required, must be provided at the workstation. The network was

designed to avoid the need for these forms of MIDI message processing within the MIDI patcher. This was necessary due to the large amounts of MIDI information requiring routing within the network. Any other form of MIDI message processing would require large computer processing capabilities.

Each remote workstation user has the capability to control the MIDI routing abilities of the MIDI patcher. This capability is supplied to him by the MIDI patcher server and the network software. Figure 4.8 shows the MIDI patcher and its connections within the network.

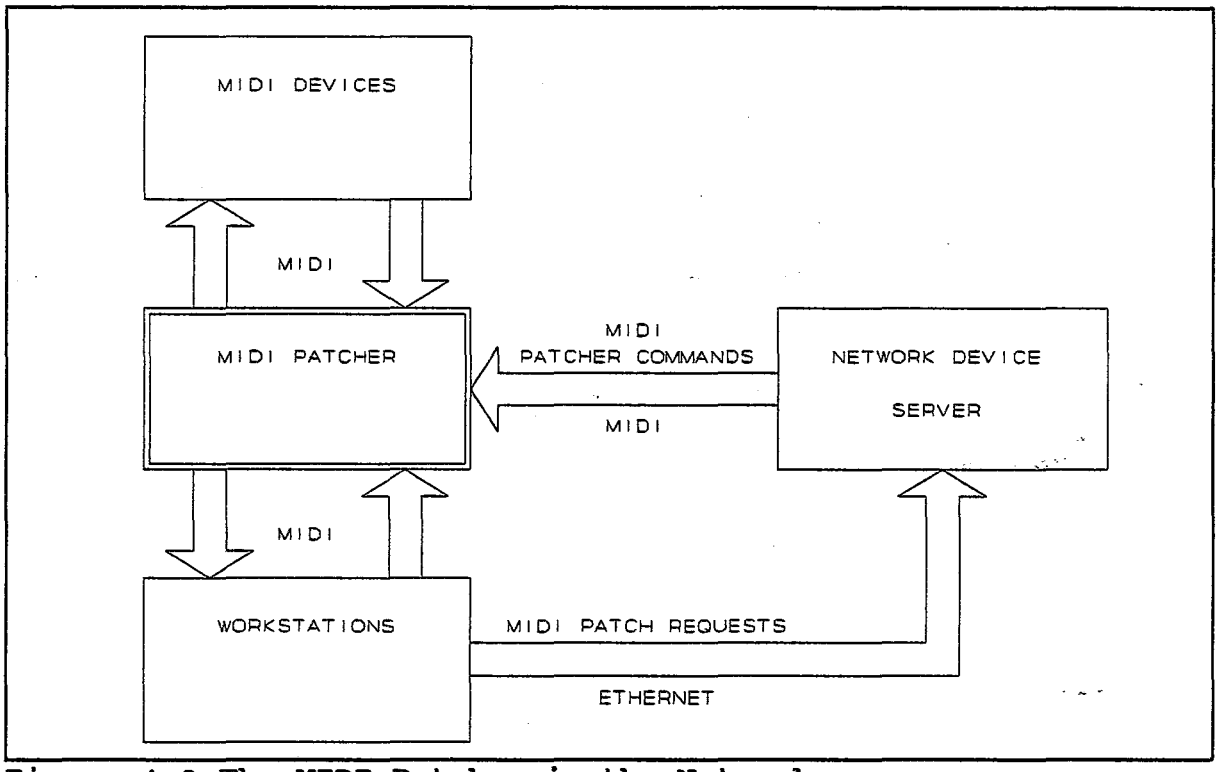

Figure 4.8 The MIDI Patcher in the Network

Each workstation contains a MIDI sequencer needed for the real time control of multiple MIDI controllable devices. Most of the MIDI devices are resident within the main studio. They include such devices as synthesizers, samplers and audio effects units. The necessity for MIDI devices to be patched arises from the need to allow the workstation's MIDI sequencer to control a particular subset of the MIDI devices within the main studio. The sequencer also needs to obtain setup and synchronisation data from MIDI devices within the main studio, as discussed in section 3.4. This requires that two MIDI interconnects exist between each workstation and the main studio. One for carrying MIDI messages from the workstation to the main studio, and the other for carrying MIDI messages from the main studio to the workstation.

This MIDI patcher permits the MIDI messages generated at the workstation's sequencer to be communicated to multiple MIDI devices within the main studio. It also permits a single MIDI device within the main studio to transfer MIDI messages to the workstation's sequencer.

A network usually expands with time as more devices are added and the number of users increases. It is therefore necessary that the MIDI patcher be expandable so as to cater for the growing MIDI routing requirements. To allow for this expandability, the MIDI patcher is a system composed of interconnectable units. Additional workstations and/or MIDI devices can be catered for by connecting more of these units into the MIDI patcher. These MIDI patch units (MPUs) form the building blocks of the MIDI patcher. The expandability of the MIDI patcher is illustrated in figure 4.9.

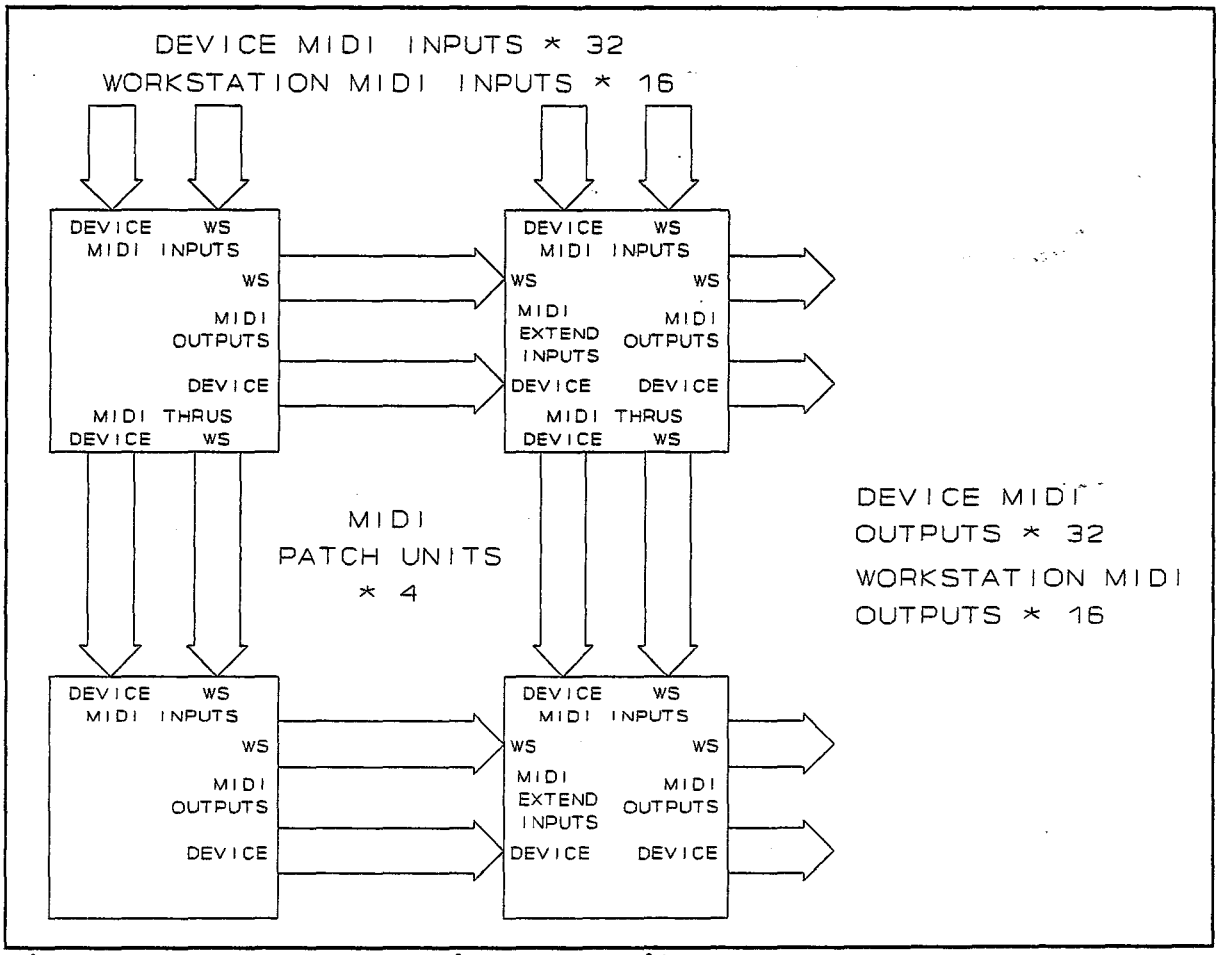

Figure 4.9 Interconnecting the Units

It is necessary to make a decision about the number of MIDI sources and sinks a MPU can support. This will govern the number of MIDI devices and workstations that can be coupled to a single

unit. The workstations are regarded as having a single MIDI source and sink. This is generally true of all MIDI devices. There will, therefore, be a one to one correspondence between the number of MIDI sources and sinks on the MPU. Should a MIDI device or workstation deviate from this rule, the corresponding unused MIDI source or sink on the MPU can be disregarded or used for another purpose.

MIDI messages only flow between the workstations and the MIDI devices. They do not flow from MIDI device to MIDI device or from workstation to workstation. Factors affecting a decision regarding the ratio of workstation MIDI sources and sinks to MIDI device sources and sinks, must be considered. Firstly, the creativity of a user is dependent on the availability of resources. Compositions produced in current computer music studios usually involve more than a single MIDI device. On the other hand, the number of MIDI controlled devices required to create a composition is decreasing with the increased capabilities offered by new devices. The diversity of sounds offered by different audio synthesis techniques and by the use of samplers, does however, increase the demand for the number of devices required for composition. Secondly, MIDI controlled audio modifiers cannot be used in isolation, they need to alter audio produced either by an audio generator or recorder. A one to one correspondence between MIDI devices and workstations from this perspective makes no sense. Having considered these factors a ratio of one workstation to two MIDI devices was decided upon. This permits a workstation user to employ one audio generator or recorder in conjunction with a single audio modifier. This must be seen as a worst case scenario, generally there will be fewer workstation users and the resources will be allocated in varying proportions.

4.2.2 Feasibility and Technology Study

There are several commercially available MIDI patchers, however none of them fitted our requirements as discussed in section 3.4. It was therefore necessary to evaluate current technologies to locate one that would allow the design and construction of a MIDI patcher that did meet our requirements.

There are two techniques by which MIDI can be routed. The first group of routers can be classed as intelligent MIDI routers, and the second group of routers can be classed as dumb MIDI routers.

Intelligent MIDI routers, as their name implies, have a processor between the MIDI sources and sinks. The processor has a dual function. It routes MIDI messages from the correct MIDI source to the correct MIDI sink(s). It also detects the end of MIDI messages to ensure that MIDI rerouting only occurs at the end of a MIDI message. This ensures that MIDI messages are not corrupted by rerouting in the middle of their communication. MIDI devices receiving corrupted MIDI messages may have unpredictable responses. The disadvantage of intelligent routers is that they require a large amount of hardware to provide the necessary MIDI reception, processing and MIDI transmission. Rerouting does not occur instantaneously on request, as MIDI messages must be communicated in entirety before rerouting can occur. The router creates a delay between the reception of a MIDI message and its transmission, as the processor takes a finite amount of time to route the MIDI message.

The dumb MIDI router does not require as much hardware to implement as its intelligent counterpart, as routing does not occur via a processor. Switching circuitry is used by these devices to effect the routing of MIDI messages. The switching circuitry is under control of a processor. This technique avoids the need to have the interface hardware required by the processor, to receive and transmit the multiple MIDI signals. It must still, however, have MIDI reception and transmission hardware, because a MIDI source may be required to feed several MIDI sinks. The MIDI specification does not permit this, as one MIDI output can drive one and only one MIDI input [1 p1]. The MIDI signal must therefore be replicated to provide this facility and this requires the additional hardware. Opto-isolators are used in the reception of MIDI signals [1 p1]. The use of simple switching circuitry does not overcome the pulse width distortion problems created by these opto-isolators. Due to the transfer characteristics of opto-isolators pulse widths are altered. This problem does not occur within an intelligent MIDI router as the MIDI data is reformed by the transmission circuitry. A partial solution to the pulse width distortion problem is to use high quality opto-isolators in the reception circuitry to minimise the effect. This will prevent the problem compounding itself when multiple dumb MIDI routers are interconnected to provide extended MIDI routing facilities.

The dumb MIDI router does not suffer from the processor MIDI routing propagation delay that its intelligent counterpart does, as the routing circuitry of the dumb MIDI router does not

introduce a delay. This is due to the straight-through nature of the routing circuitry.

A dumb MIDI router does not exhibit a delay between a command for rerouting being issued and the rerouting occurring. The router does not wait for MIDI message communication to be completed before rerouting. This technique can cause MIDI message corruptions which may have unpredictable effects on MIDI devices and sequencers. The corruption of MIDI messages by the MIDI patcher can be prevented by ensuring that there are no MIDI messages on the signal paths at the time of switching, and that if MIDI messages are corrupted, they are not received by either MIDI devices or sequencers. Techniques capable of supplying this functionality are discussed in the conclusion, section 6.2.6.

A dumb MIDI router was chosen instead of an intelligent MIDI router for the MIDI routing requirements of our computer music network. The additional MIDI message processing facilities of an intelligent MIDI router were not required.

4.2.3 The MIDI Patch Unit Hardware

The number of MIDI devices and workstations that a single MPU can support is related to the technology used in the implementation of the routers. Multiplexers capable of routing 8 and 16 channels abound and consequently the MPU was designed to support 8 workstations and 16 MIDI devices. This was in keeping with the ratio of one workstation to two MIDI devices.

The MIDI routing circuitry of the MPU was designed to comply with the MIDI hardware specification [1 p1].

The MIDI Patch Unit circuitry comprises four sections :
1) The MIDI input interface circuitry.
2) The MIDI output interface circuitry.
3) The device to workstation routing circuitry.
4) The workstation to device routing circuitry.

The MIDI input interface circuitry converts the logic levels of the MIDI signals to that of the routing circuitry, in this case Transistor Transistor Logic (TTL) [52 p287]. It provides the MIDI inputs for MIDI sources, namely from the workstation MIDI sequencers and the MIDI controlled devices in the main studio. It also provides MIDI extend inputs to provide for the necessary expansion facilities as illustrated in figure 4.9. A high speed

opto-isolator, the 6N139, was used in the construction of the MIDI input interface circuitry [53 p25]. This reduces the problem of cumulative pulse width distortions which occur when multiple MPUs are chained together [1 p1]. The circuit diagram for the MIDI input interface circuitry can be found in appendix 2.4.3 and the part list in appendix 4.4.3.

The MIDI output interface circuitry converts the logic levels of the routing circuitry, TTL, to that of MIDI. The MIDI outputs feed the workstations and MIDI devices. The MIDI thrus are exact replicas of the MIDI inputs to the MPU and provide for the necessary expansion facilities as illustrated in figure 4.9. The circuit diagram for the MIDI output interface circuitry can be found in appendix 2.4.4 and the part list in appendix 4.4.4.

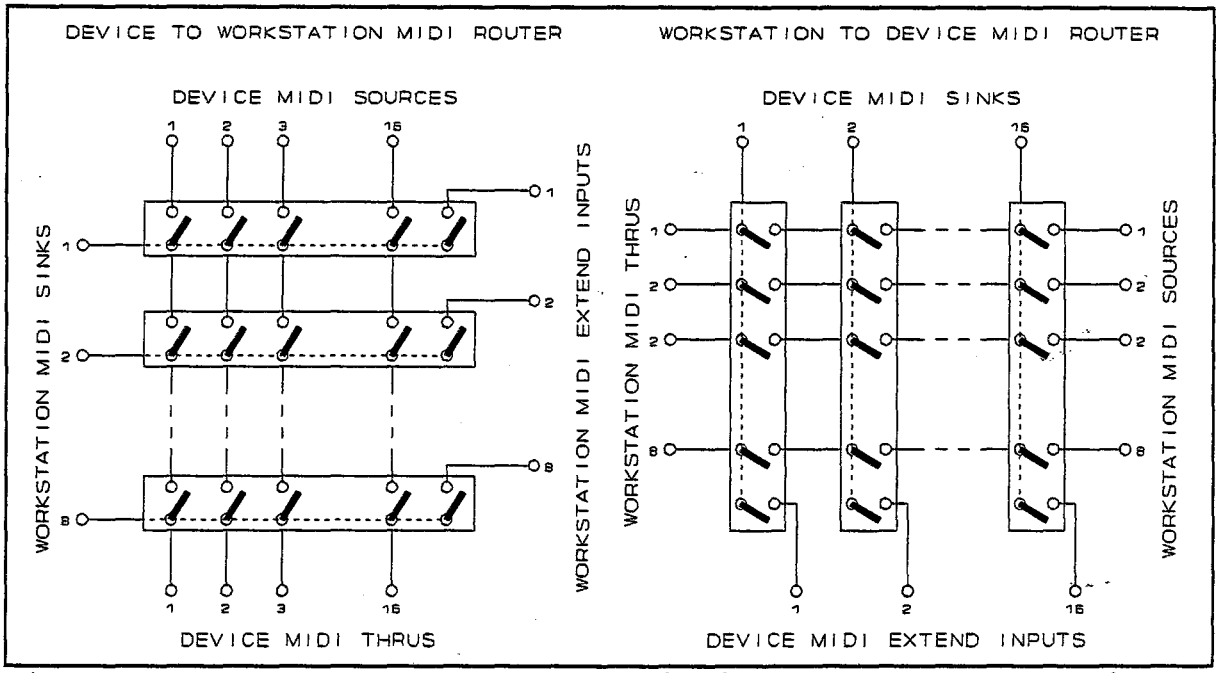

Figure 4.10 MIDI Router Topology within the MIDI Patch Unit

The device to workstation routing circuitry is responsible for the routing of the digital information from a device MIDI source to a workstation MIDI sink. The workstation to device routing circuitry is responsible for the routing of the digital information, representing the MIDI messages, from a workstation MIDI source to a device MIDI sink. The router circuit diagrams can be found in appendix 2.4.1 and 2.4.2, and the parts lists in appendix 4.4.1 and 4.4.2. These routers are placed in parallel, allowing a MIDI message from a single source to be routed to several sinks. This allows the workstation MIDI sequencer to feed MIDI to more than a single device. It also allows the devices to return MIDI to each of the workstations. The topology of the

routers within the MPU is illustrated in figure 4.10.

The MIDI Thru facility allows the stacking of routers to cater for an increasing number of MIDI sinks as illustrated in figure 4.11. The MIDI thrus of the first MPU are coupled to the MIDI sources of the second MPU providing it with exact replicas of the MIDI sources coupled to the first MPU. The number of MIDI sinks that can therefore be connected to those MIDI sources is therefore increased.

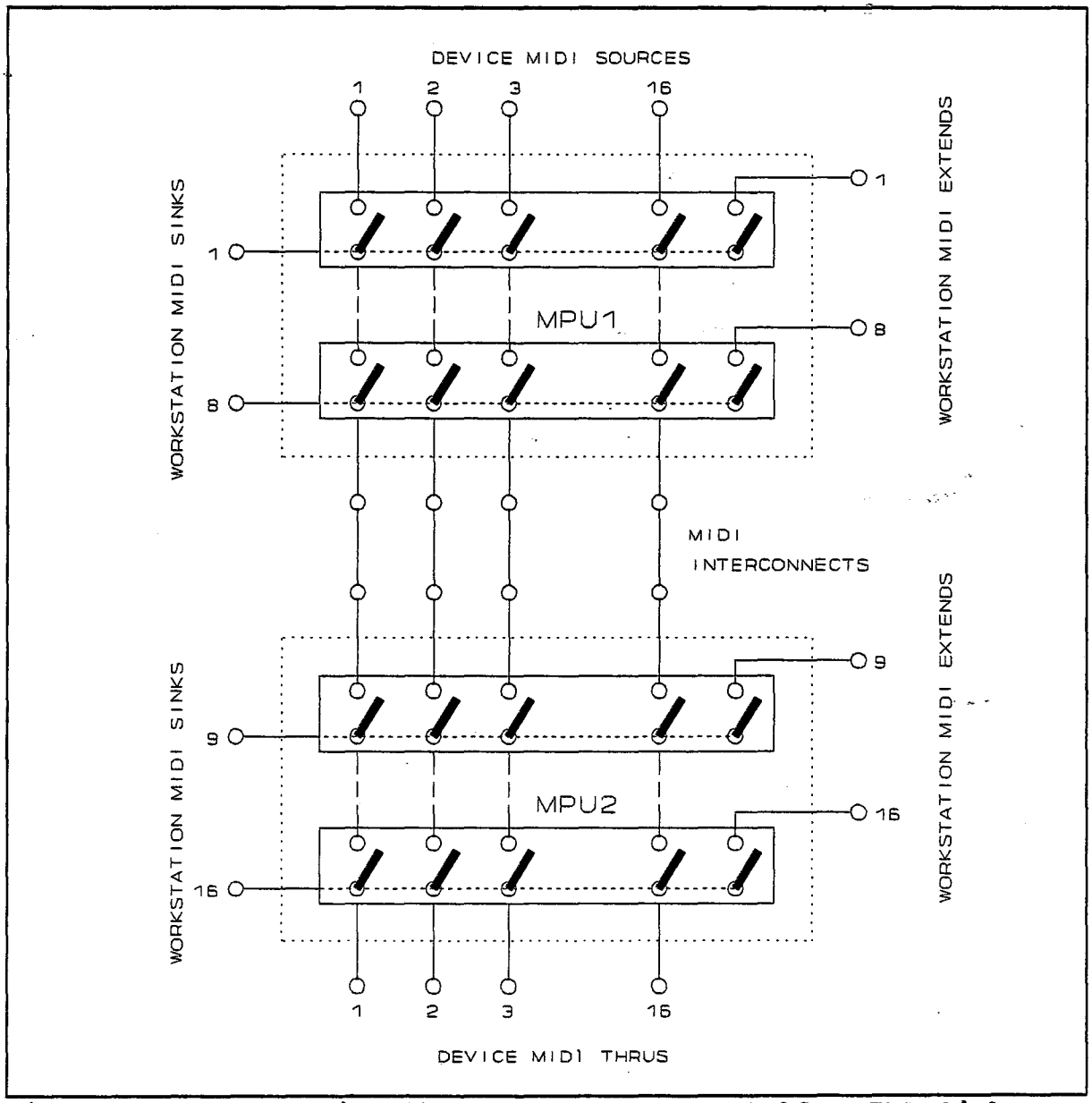

Figure 4.11 Increasing the Number of Connectable MIDI Sinks

The MPU MIDI expand inputs permit the chaining of routers of one MPU to another to cater for an increasing number of MIDI sources as illustrated in figure 4.12. The MIDI sinks of the second MPU

are connected to the MIDI extend inputs of the first MPU. This allows MIDI signals emanating from the second MPU to be transferred to the MIDI sinks of the first MPU. This is possible because the switches of the first MPU can be connected to the MIDI extend inputs. These techniques make it possible to cater for either a growth in the number of workstations and/or the number of MIDI devices within the main studio.

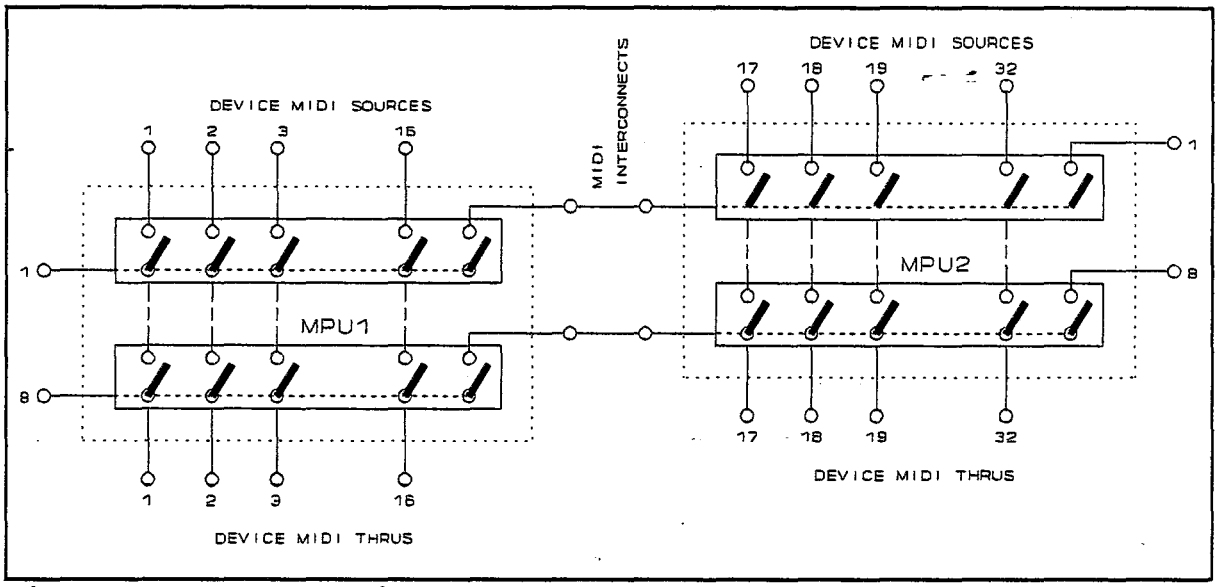

Figure 4.12 Increasing the Number of Connectable MIDI Sources

4.2.3.1 Hardware Prototyping, Implementation and Testing

The initial prototypes were evaluated using software running on an IBM AT. The software together with a small amount of hardware permitted the transfer of data from the IBM AT data bus to the hex latch on the router. MIDI messages produced by two MIDI controllers were coupled to the MIDI Inputs of the router and the MIDI Output of the router was coupled to a synthesizer. Both MIDI controllers were programmed to transmit MIDI channel voice messages on the same MIDI channel. The synthesizer was programmed to receive MIDI channel voice messages on this channel. The software running on the IBM AT permitted the input selection of the router to be altered. The MIDI controller feeding MIDI messages to the synthesizer could therefore be switched. This permitted either of the MIDI controllers, but not both, to be used to play the synthesizer. The synthesizer's audio output was used to verify that MIDI messages were correctly received by the synthesizer. This technique was used to evaluate both the workstation to device and device to workstation MIDI routers.

4.2.4 Evaluation of the Unit

The unit was evaluated using a small test network described in chapter 5. The network software automatically patches the MIDI inputs of booked devices to the correct workstation MIDI output. A workstation user can specify from which one of the booked devices MIDI is received. However, the network was not designed to test that MIDI messages had been completed before MIDI patching took place. Consequently, MIDI data corruption occurred with undesirable side effects on devices receiving the corrupted data. These included error messages, hung notes and hung devices. This problem will be cleared up in later versions of the network by methods described in section 6.2.6.

4.3 THE MICROPROCESSOR CONTROL UNIT

The aim of this section is to explain the development of a general microprocessor control unit for controlling the hardware responsible for the processing of audio signals within the APU, the routing and mixing of audio signals within the APMU, and the routing of MIDI signals within the MPU. The development of the software for each of these three applications is then discussed.

4.3.1 Description of Functionality

Internal control over the APUs, APMUs and MPUs found within the computer music network is by way of a dedicated Microprocessor Control Unit (MCU) residing in each unit. Input to the MCU is in the form of MIDI messages. The MIDI messages contain the necessary information for the control of the respective units to which they flow. The MCU minimizes the number of physical interfaces between the units and the network device server. It also reduces the demands on the processing power of the network device server.

The use of MIDI for the control of the APU, APMU and MPU prevents the formation of ground loops [1 p1]. Ground loops can give rise to control data errors. They are also responsible for the introduction of unwanted noise, usually mains hum, into audio signals. MIDI also enables APUs, APMUs and MPUs to be connected to existing MIDI networks common in most music studios nowadays.

MIDI System Exclusive messages, (hereafter abbreviated as "Exclusive" messages) are used to alter parameters within the APUs, APMUs and MPUs. An Exclusive message will contain

information identifying the respective unit, the parameters within that unit that need to be changed, and their new settings. Data is only supplied for parameters that require changing to ensure that a minimum amount of MIDI data is required to achieve the desired changes and that changes occur as quickly as possible. Exclusive messages are employed as they have no set format [1 p39]. An appropriate format can therefore be chosen to suit a specific requirement. The manufacturer's ID number, which forms part of the header of an Exclusive message is set to 7DH. This number has been reserved for research purposes [1 p40]. To prevent the unnecessary transmission of duplicate data, an End Of Exclusive (EOX) is only transmitted when an Exclusive message is required to be sent to a new unit. This allows the transmission of parameter changes to the current unit to continue without the overhead of Exclusive message headers and EOX's. To further reduce the amount of data present, on the MIDI interconnects, data is compressed. This involves placing two or more parameters into a single byte where possible. These techniques were instituted to keep the possibility of bottle-necking on the MIDI interconnects feeding the APUs, APMUs and MPUs to a minimum. The format of the MIDI messages for the APU, APMU and MPU can be found in appendix 5.

Bottle-necking on a MIDI interconnect occurs when data needing to be transmitted is produced faster than it can be transmitted because of the fixed MIDI transmission rate [1 p1]. The possibility of bottle-necking increases with each unit that is coupled to a single MIDI source. The bandwidth of the MIDI source must be shared amongst the units. If bottle-necking does occur, the only solution to the problem is to feed the units from multiple MIDI sources.

Each unit contains 2 front panel indicators, a MIDI Data Indicator (MDI) and an Error in Data Indicator (EDI). The MDI indicates the arrival of an Exclusive message destined for that particular unit. This indicator is illuminated when the correct Exclusive header has been received and is extinguished half a second after an EOX is received. This is to ensure that the MDI will always be illuminated for a period long enough so that the arrival of the shortest possible Exclusive message, a header followed by an EOX, can be observed. The EDI indicates the arrival of erroneous data. Erroneous data can take three forms, out of range data, invalid data, or unexpected end of data. An invalid data condition occurs when the data is not in the specified data set. An unexpected end of data occurs when an EOX

is received and the unit is still expecting more data. The EDI is illuminated on the detection of a data error and extinguished half a second after the receipt of the next Exclusive message header destined for this particular unit. This prevents the possibility of the EDI being rapidly extinguished by the arrival of the next Exclusive message. The front panel contains a reset button, to reset the unit to its power-up state. Both the MDI and the EDI are illuminated until the unit has achieved its power-up state.

4.3.2 Feasibility and Technology Study

Microprocessors and microcontrollers with the necessary processing power for the construction of the MCU abound. It is important that the microprocessor or microcontroller, chosen for the MCU, meet certain criteria. It should be able to run the initial APU, APMU and MPU software in real time. It should also be able to handle any future moderate changes made to that software. It should keep to a minimum the amount of external hardware required for the construction of the MCU.

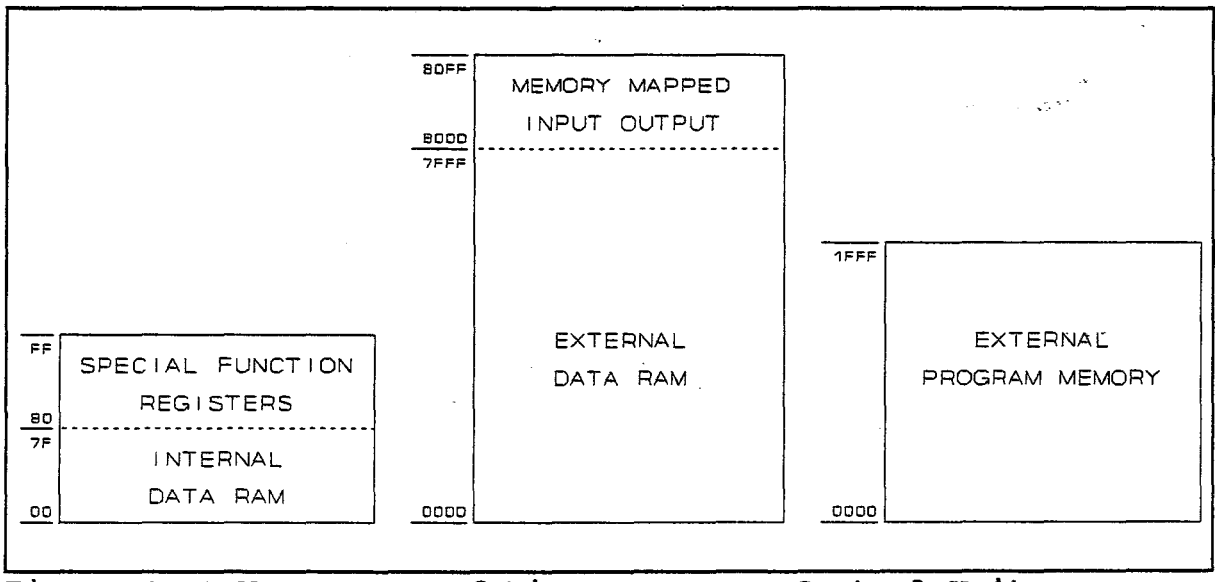

Figure 4.13 **Memory Map of Microprocessor Control Unit**

The 8031 microcontroller from the MCS-51 family of 8-bit microcontrollers was the microcontroller chosen [23 p6-1], as it has the necessary processing speed for the MCU. It has an on-chip serial port ideally suited for the receiving and transmitting of MIDI information. It has two on-chip timers for the control of the serial port and to perform other timing requirements. The 8031 also has on-chip data memory which can, with careful design and due consideration to the application, avoid the need for external Random Access Memory (RAM). External RAM can, if

necessary, be plugged into the MCU board. External program memory is required by the 8031, allowing the application to dictate the size of program memory. The ability to add external RAM and choose the size of program memory will aid in further software development and investigations. The memory map of the MCU is shown in figure 4.13 [23 p7-2].

4.3.3 The Microprocessor Control Unit Hardware

The 8031 Controller together with its peripheral devices and software is responsible for receiving and interpreting MIDI data, and determining the appropriate action to be taken within a unit.

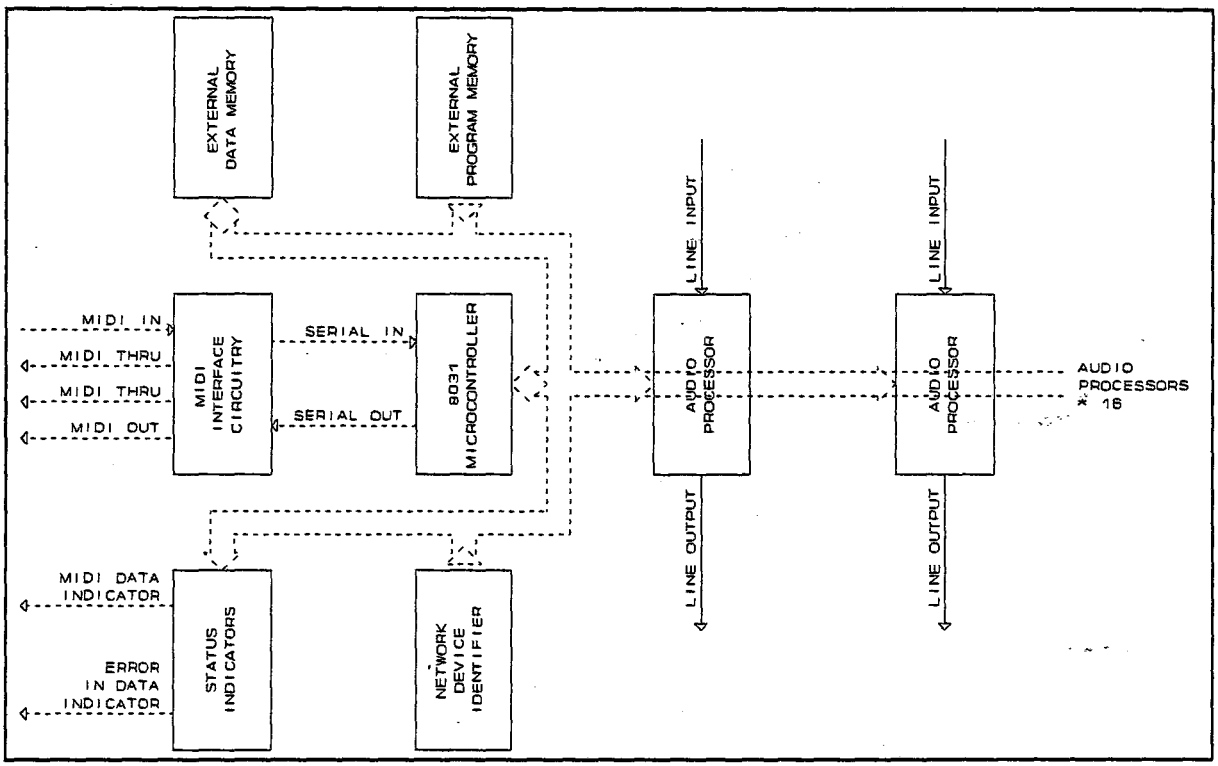

Figure 4.14 Audio Processor Unit Internal Block Diagram

The APU required 64 address decoded outputs and three, 9 bit wide, data buses to output digital data to 16 audio processors. The 64 address decoded outputs are required to address the gain, bass, midrange and treble controls of each audio processsor. All the gain controls of the audio processors are grouped together into a single bank. The equalisation controls of the audio processors are grouped into 2 banks with 8 in each. A 9 bit wide data bus is required for each bank. A data bus for each bank ensures that a fan out problem does not occur. The fan out of a logic gate is the number of logic gates it can drive [52 p287]. This number is dependent on the source current of the source gate and the sink currents of the sink gates. Capacitive coupling

between the source gate output and other signal and power paths also reduces the fan out capability of a logic gate, especially at high gate speeds. Exceeding the fan out capability of a gate can result in incorrect logic levels and therefore incorrect data transfers. An address decoded output is also required to input a 7 bit wide digital data word. This data word is generated by the network device identifier data entry switches.

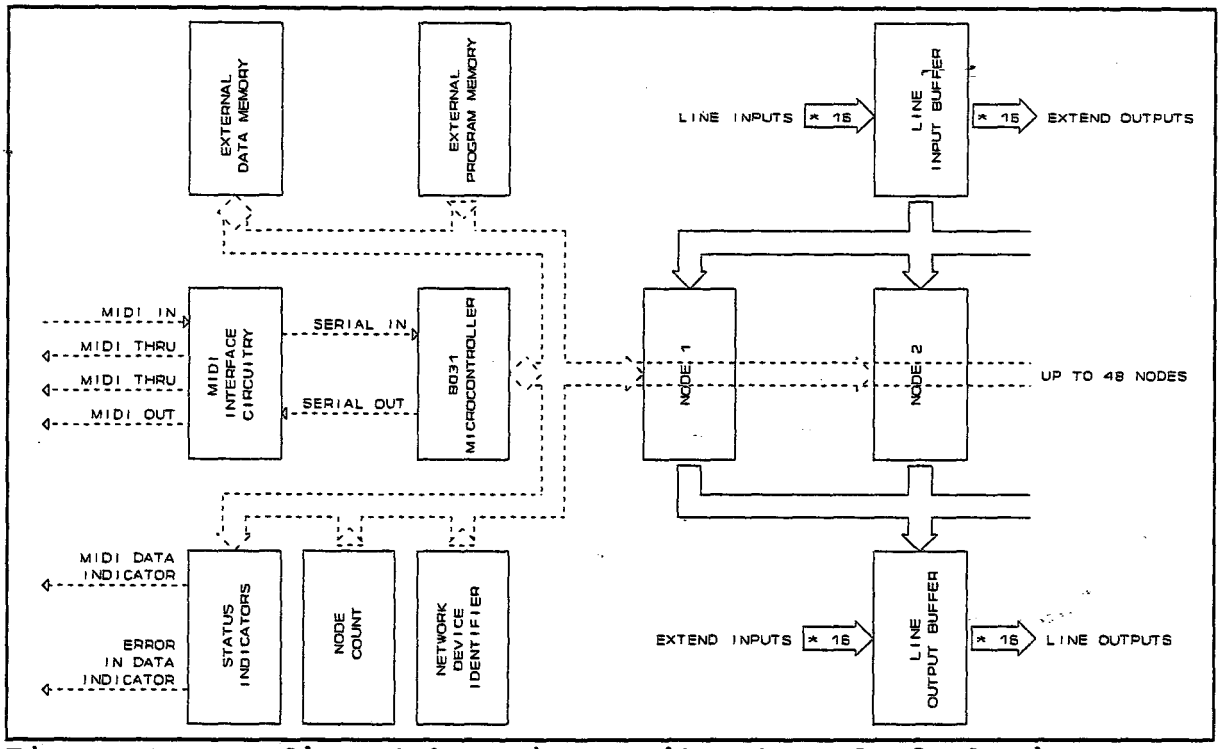

Figure 4.15 Audio Patcher/Mixer Unit Internal Block Diagram

The APMU requires 144 address decoded outputs and three, 10 bit wide, data busses to output digital data to the 48 nodes. The 144 address decoded outputs are required to address the input multiplexer, DCA, and output multiplexer for each of the 48 nodes. The 48 nodes are grouped into 3 banks with 16 nodes in each, hence the requirement for 3 data busses. Two address decoded outputs are required to input 2 digital data words. The one, a 7 bit wide digital data word and the other, a 6 bit wide digital data word. The 7 bit data word is generated by the network device identifier data entry switches. The 6 bit wide data word is generated by the node count data entry switches.

The MPU requires 24 address decoded outputs and 3 data busses to output digital data to the 24 MIDI routers. A 6 bit wide data bus is required to output data to the device-to-workstation MIDI routers. Two, 5 bit wide data busses are required to output data to the workstation-to-device MIDI routers. This is necessary

because of the limiting fan-out capabilities of the logic gates. An address decoded output is required to input a 7 bit wide digital data word. This data is generated by the network device identifier data entry switches.

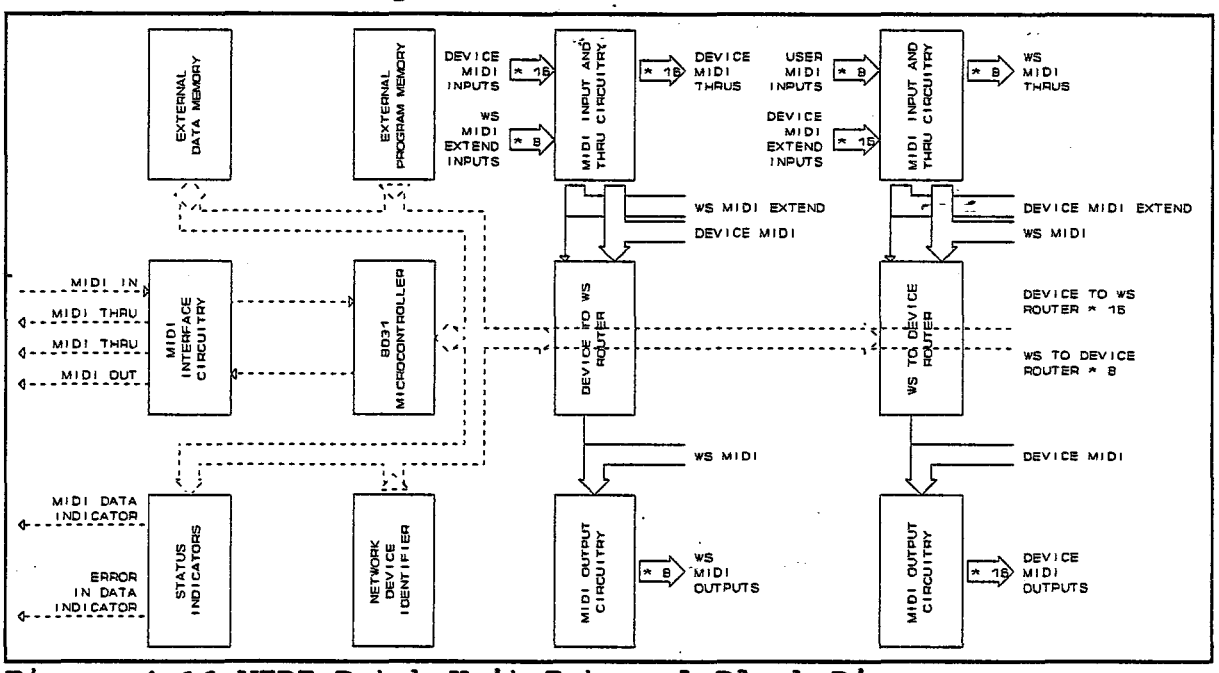

Figure 4.16 MIDI Patch Unit Internal Block Diagram

The MCU design has to cater for the limited fan out of the ICs (Integrated Circuits) used in the construction of this board as discussed above. In the case of the 8031 microcontroller, port P0 has a fan out of 8 to LS (Low-power Schottky) TTL (Transistor Transistor Logic) and on ports P1, P2, and P3 a fan out of 4 to LS TTL [23 p11-2]. LS TTL ICs have a fan out of 12 to LS TTL. An alternative, to avoid fan out limitations, is to use HCMOS (High-Speed Complementary Metal Oxide Semiconductor) ICs [71] instead of LS TTL ICs.

4.3.4 Structure of the Microprocessor Control Unit Hardware

The full circuit diagram for the MCU is shown in appendix 2.5. The circuitry is divided into 8 parts:

1) The Microcontroller.
2) The External Program and Data Memory.
3) Indicator Circuitry.
4) IO Decoding Circuitry.
5) Data Entry Circuitry.
6) Output Buffer and Latch Circuitry
7) MIDI Input Circuitry.
8) MIDI Output Circuitry.

The microcontroller circuitry shows the oscillator and reset peripheral components to the 8031. It also shows the octal latch required to obtain the additional 8 bits of the address bus. This is obtained from the output data from the data bus, Port 0 (P0) [23 p9-24].

The external program and data memory circuitry shows how an external Erasable Programmable Read Only Memory (EPROM) chip and a RAM chip are coupled to the 8031. The design shown allows the use of an 8 kilobyte EPROM and a 32 kilobyte static RAM chip. This can be modified without difficulty to handle other types and sizes of memory chips. The user is limited to 64 kilobytes of program memory and 64 kilobytes of data memory as these are the addressable memory location limitations of the microcontroller. All input and output peripheral devices must be mapped into the data memory area. This is shown in the memory map, figure 4.13.

The Input and Output (IO) decoding circuitry enables the microcontroller to address peripheral devices individually (appendix 2.5.4). This permits the 8031 to write information to, and read information from, individual peripheral devices. The initial IO decoding circuitry was built from discrete logic chips. The number of chips required to perform this operation could however be decreased by using Programmable Logic Arrays (PLAs) [41 p11].

In order to provide the 9 and 10 bit wide busses, required by the DCAs in the audio processors of the APU and the nodes of the APMU, 2 octal latches are employed (appendix 2.5.6). The bus is then created by writing 2, 8 bit wide, data words to these latches. The one latch supplies the 8 low order bits and the other, the 1 or 2 high order bits. This technique of latching also reduces the flow of data on the bus to a minimum. Only data required on the bus is placed on it by the 8031 Microcontroller. This reduces the digital interference with other processes being performed by the peripheral devices. This is important in the APU and APMU, where audio signals are being processed. Digital interference occurs due to capacitive coupling between the digital and audio signal paths, as discussed in section 4.1.2. The amount of digital interference can be further decreased by increasing the number of 16 bit busses servicing the peripheral devices. This reduces the amount of digital data on each 16 bit bus as fewer peripheral devices are serviced by the bus. Increasing the number of 16 bit busses does, however, increase the complexity of the hardware and hence the controlling

software.

The data entry circuitry permits the 8031 to read switch position settings (appendix 2.5.5). The switches supply the microcontroller with variable values which change from unit to unit. The network device identifier is a number which identifies each APU, APMU and MPU within a computer music network individually. This permits these units to be accessed individually, even if they receive the same MIDI information. The network device identifier is programmed into each unit by way of these switches. The number of nodes present within a APMU can vary from 1 to 48 depending on requirements. Switches within the APMU communicate the number of nodes present to the 8031. The circuit is designed so that a logic "1" is generated when a switch is open (the "OFF" position) and a logic "0" when closed (the "ON" position). The numbers are entered in binary form using the switches.

There are two MIDI Thrus on an APU, APMU or MPU. This allows the units to be chained together. Long chains are prevented as two units can be connected to a previous unit in the chain. This prevents the degradation of the MIDI signal typical in long chains. This problem is further alleviated by using fast opto-isolators in the MIDI input circuity [1 p1].

A component list of parts required for the construction of the MCU can be found in appendix 4.5.

4.3.5 The Audio Processor Unit Software

The aim of this section is to explain the development of the software for the MCU within the APU. The software permits the real time control of the audio processors within the APU. Control of the APU is by way of MIDI system exclusive messages.

4.3.5.1 Software Design

The chief design criteria for the software for the APU is that it should operate in real time. In order to model this real time operation the Ward and Mellor real time system development methodology was employed [27] [28] [29]. A real time system development methodology provides modelling tools to model the control aspects of real time systems. It ensures that the development of a system is well structured and documented. This has the added advantage that the system is more easily understood

by its users and those wishing to change it.

The Ward and Mellor methodology is based on the techniques described by DeMarco [45] for the derivation of data flow diagrams, and on the methods for deriving modules of instructions from the analysis of data flow graphs, described by Yourdan and Constantine [46]. It also encapsulates the Jackson JSD methodology [47], where the structure of the input and output data is first specified, and the structure of the program is derived from an analysis of this specification. The Ward and Mellor methodology derives the structure of the program from an analysis of the responses a system makes to events that occur within its environment.

The MCU receives all its real time instructions via MIDI messages. Real time response from the MCU requires that it process data, on the average, faster than the rate at which it receives MIDI bytes. MIDI bytes arrive at a maximum rate of 3125 bytes per second [1 p1].

The development methodology uses symbols for modelling the system. It also separates the essence of the system from its implementation. The essential model describes what the system must do in response to events in its environment. This model is independent of the technology employed. The description of a particular technology used in realising the system is called the implementation model. The implementation model is an elaboration of the essential model.

4.3.5.2 The Essential Model

The essential model consists of two parts, the environmental model and the behavioral model. The environmental model describes the boundary between the system and its environment, the interfaces between the two, and the events that occur within the environment to which the system must respond. The behavioral model describes how the system is required to behave in response to events.

4.3.5.3 The Environmental Model

The context schema describes the boundary between a system and its environment, and the interfaces between the two [28 p14].

The context schema for the APU is shown in figure 4.17. The

notation used is as follows:
1) Circles denote transforms. In the context schema a single transform denotes the entire system and its activities.
2) Boxes denote terminators. These are objects outside the system boundary to which the system reacts. Multiple instances of the same object are not shown.
3) Arrows denote flows, between the system and terminators. A solid line with a single headed arrow denotes time discrete data flows. A broken line with a single headed arrow denotes flows which have no content, and are either signals or commands. The direction of the arrow indicates the direction of causality.

Figure 4.17 shows the context schema for the system under development. The audio processors are regarded as being terminators in this description.

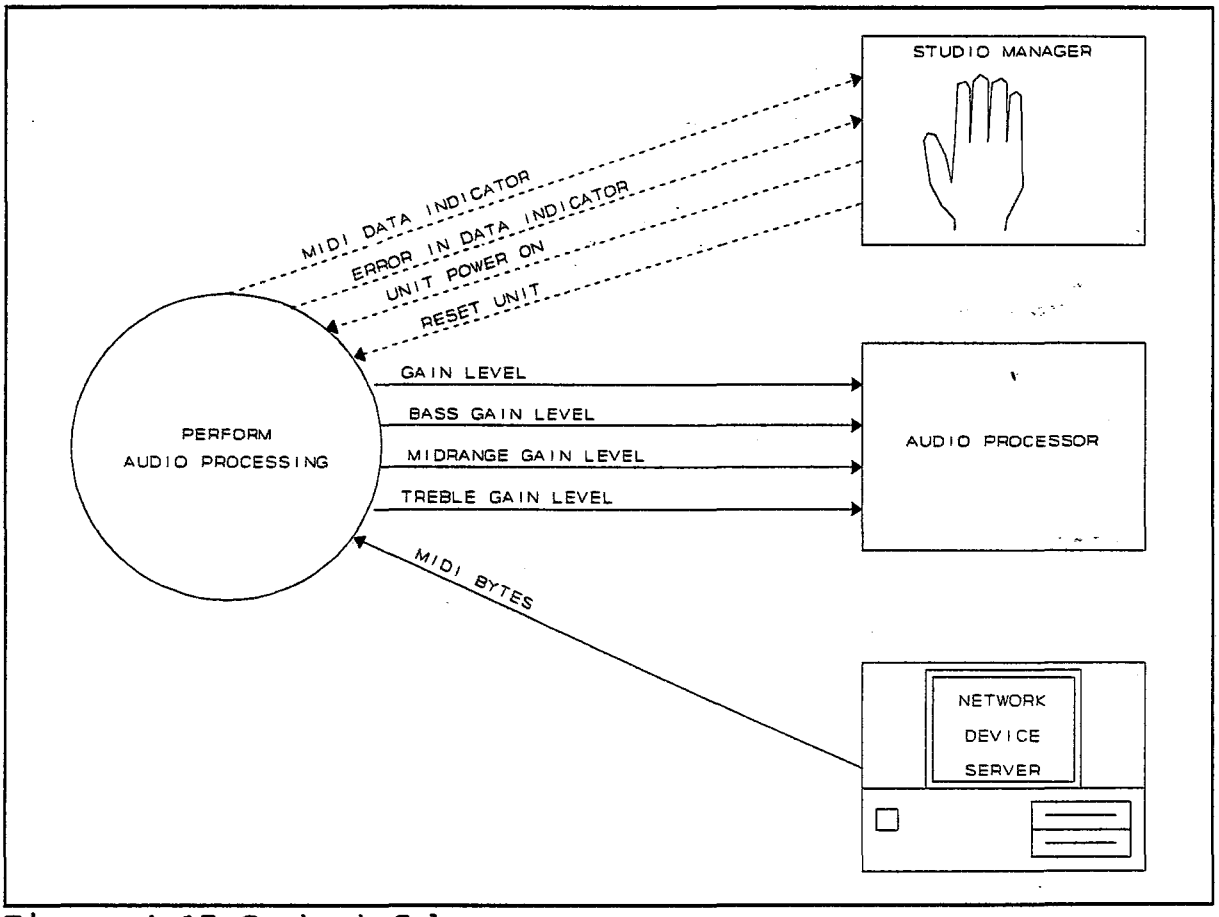

Figure 4.17 **Context Schema**

4.3.5.4 The Behavioral Model

The central feature of the environmental model is the specification of the events to which a system must respond. These external events occur in the systems environment, at some

specific point in time, and elicit a preplanned response from the system [28 p30]. The function of the behavioral model is to describe the response made by the system to each of these external events [28 p41]. This can begin with a simple event and response list.

Event and Response List

EVENT	RESPONSE
Studio manager switches Audio processor unit on.	Turn error in data and MIDI data indicators on. Set all gain levels to 0dB i.e. unity gain. Set all bass, midrange and treble gain levels to 0dB i.e. flat frequency response. Turn error in data and MIDI data indicators off.
Studio manager resets audio processor unit.	Turn error in data and MIDI data indicators on. Set all gain levels to 0dB i.e. unity gain. Set all bass, midrange and treble gain levels to 0dB i.e. flat frequency response. Turn error in data and MIDI data indicators off.
Network device server transmits a MIDI real time message.	Ignore MIDI real time message.
Network device server supplies an audio process request to this particular unit	Turn MIDI data indicator on. Update audio processors with all the valid audio processes in the request. Turn MIDI data indicator off.

Responses to events can be modelled as transforms or as state transition diagrams [28 p46]. The response of the APU's software

to incoming MIDI data is dependent on the previous MIDI data received. The next state of the system is therefore dependent on the current state of the system and the current MIDI data being received. The state transition diagram is the easiest manner to model such a response, as it describes the system in terms of states. A state represents an observable mode of behaviour [27 p64]. The name of the state is the name of the exhibited behaviour. A state is denoted by a box within the state transition diagram. The next state of the system is dependent on the current state and the condition created by the arrival of the next MIDI data. A condition causes the system to make a transition. A transition represents the movement from one state to another, and is denoted by an arrow. Action(s) are taken as the transition occurs. The states of the system can be easily derived from the sequence in which data arrives. The modelling of systems from the description of the input data is in keeping with Jackson [47].

The MIDI system exclusive message format specification for the APU is given in appendix 5.1 and the state transition diagram in appendix 6.1.2. In addition to the transitions resulting from the arrival of correct MIDI data, transitions have to be added to describe how the system behaves when erroneous conditions occur because of the arrival of incorrect MIDI data.

All systems transform inputs to outputs. A transformation schema can be used to model how a system performs these transformations [27 p41]. The transformation schema models a system as a network of activities. These activities receive and produce data and event flows. A transformation that accepts and produces only event flows is called a control transform, and is denoted by a broken circle [27 p48]. Only control transforms may enable and disable transforms within the schema [27 p51]. The state transition diagram models the behaviour of control transforms, and describes how it uses the event flows. A store within the transformation schema is denoted by two horizontal parallel lines. There should be a single data transform within the transformation schema, for each event and response.

Three data transforms were derived from the above event response list. The first two events in the list elicit the same response and were therefore modelled as a single data transform activated by two events (transform 2 of figure 6.1.1.3, in appendix 6.1.1). The third event and response is described by a single data transform which removes MIDI real time messages from the MIDI

byte stream (transform 3 of figure 6.1.1.1, in appendix 6.1.1). MIDI real time messages have high transmission priority and can interrupt MIDI system exclusive messages [1 p35]. It is therefore necessary to filter them out. This is only important for general use as in the network implementation these units will never receive MIDI real time messages.

An audio process request comprises one or more audio process(es). An audio process contains data to modify a single parameter of an audio processor, as detailed in the APU message format specification in appendix 5.1. The final event and response is modelled as a transform which updates an audio processor on the arrival of an audio process.

The remaining data transforms are derived from the state transition diagram, one for each active state. These transforms are responsible for deriving and storing data for each audio process sent to the unit. They are also responsible for detecting errors in the MIDI bytes. The transformation schema can be found in appendix 6.1.1. It has been split into three parts because of its complexity. The partitioning is not based on specific rules but rather on a logical grouping of transforms. The single control transform, described by the state transition diagram, is responsible for enabling and disabling data transforms depending on event flows and event recognition.

The best grouping of transforms, flows and stores is the one with a minimal number of interfaces [28 p62]. The understanding of one section of the essential model is made easier by minimizing its dependence on other sections. This can be done by minimizing the number of interfaces between the sections. The transformation schema for the APU represents a grouping of transforms, flows and stores which complies with this heuristic.

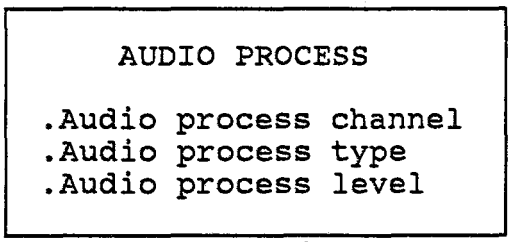

Figure 4.18 **Audio Process and Attributes**

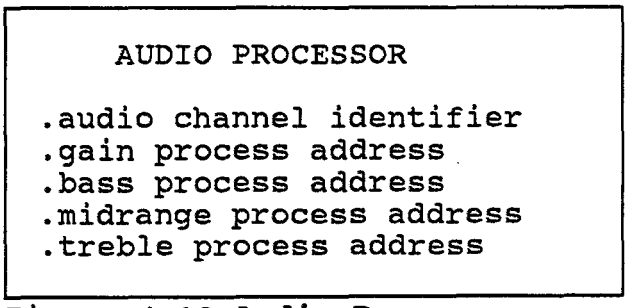

Figure 4.19 **Audio Processor and Attributes**

The model builder could put all data into a single store, or

create a multiple of simple stores, within the transformation schema. The partitioning of stores within the transformation schema of the APU was done by identifying objects within the system and its environment and their relationships [27 p108]. Two objects can be identified for the APU, an audio process and an audio processor. Their attributes are shown in figures 4.18 and 4.19.

An audio process modifies one of the audio processor's process types. In this implementation the audio process types are gain, bass gain, midrange gain and treble gain. The audio processor is identified by the audio process channel in the audio process. The audio process level is the new level for the audio process type. The location of an audio processor's audio process type within the MCU's memory map is identified by the audio process address. This is where the level data must be sent. The audio process requests received by an APU can contain errors, and can be interrupted by MIDI real time messages (appendix 5.1). The specification of audio process requests therefore includes data to identify error conditions and MIDI timing bytes. Specifications for the audio processors contain maps describing how the audio process level values must be transformed to supply the correct gain or equalisation levels, and the initial gain and equalisation levels. The data dictionary which specifies these data stores and data flows within the transformation schema can be found in appendix 6.1.4, 6.1.5 and 6.1.6.

The behaviour of a data transform within the transformation schema is inadequately specified by its name. It can be more adequately specified using pseudocode or structured english [27 p 83] [27 p85]. Both these techniques can give sub-optimal implementations due to bad procedure creation by the implementer. Other forms of specification which are less procedural offer a better alternative [27 p86]. These include graphic and tabular specifications. Another method of specifying the behaviour of data transforms is by using precondition-postcondition specifications [27 p 89]. This method relates conditions on input values to corresponding conditions on output values. This was the method chosen to specify the behaviour of data transforms within this transformation scema. A transformation can be described by one or more precondition-postcondition specifications. The precondition-postcondition specification of the data transforms within the transformation schema are shown in appendix 6.1.3. As with any method used to specify the behaviour of a data transform the data used and produced by the transformation schema must be

firstly completely specified [27 p81].

4.3.5.5 The Implementation Model

The implementation model is derived by doing a top down allocation of the essential model to the implementation technology. The main heuristic for this allocation is that it should cause minimal distortion to the essential model [29 p3]. A single processor is used to implement the APU. A processor is a person or machine capable of performing instructions and storing data [29 p6]. The entire essential model is allocated to this single processor. There is therefore no distortion of the essential model by this allocation.

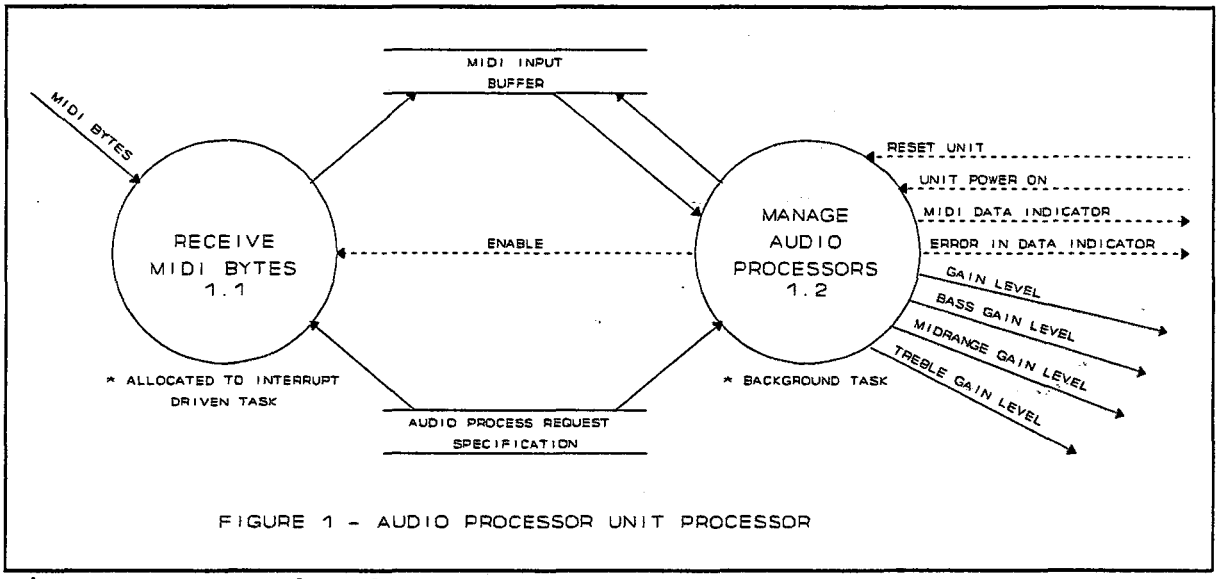

Figure 4.20 **Task Schema**

The processor is required to perform a number of tasks. A task is a set of instructions started, stopped, interrupted and resumed by the processor. A task schema describes the allocation of the essential model to these tasks. The task schema for the APU's MCU is shown in figure 4.20. It is comprised of two tasks, the one an interrupt driven task responsible for receiving MIDI bytes, and the other a background task responsible for managing the audio processors. It is necessary to have two tasks in order to queue arriving MIDI bytes. This is required since the average interval between the arrival of MIDI bytes may be less than the average time needed to process the MIDI bytes [29 p49]. This may be a problem when updating the audio processors. Processing at this point is time consuming, due to the need to look up the audio processor parameter value in a level map, and then to transfer this value to the correct audio processor. The interrupt task that receives, filters and queues the MIDI bytes is a

queuing task.

The introduction of the MIDI Input buffer required that data associated with it be specified. These additions to the data dictionary can be found in appendix 6.4.2.

A task is composed of modules. A module is a set of instructions that is activated by the control logic within a task. Only one of a task's modules may be active at a time, which precludes concurrency [29 p7]. A structure chart is used to show the modules that comprise the task and it describes their relationships. The module is modeled as a named box [29 p92]. A call to the module is modeled by an arrow pointing from the caller to the called module. The calling module becomes inactive on calling and only becomes active once the called module completes its instructions. A Shared data area between modules is modelled as a named box drawn below, and connecting, each module that shares that data. An item of data passed between modules when a call is executed is called a couple. It is modeled as an arrow with a circle at its end. The arrow indicates the direction of data flow. An open circle denotes processed data and a filled circle denotes control information that controls the logic of the receiver. The complete structure chart is diagrammed in appendix 6.1.7. The second task is divided into three diagrams because of its complexity.

All system initialisation occurs on the powering up or resetting of the unit. System initialisation includes setting the serial port to correctly receive the MIDI bytes, initialising data elements, and initialising the audio processors.

4.3.5.6 Software for Microprocessor Control Unit of the Audio Processor Unit

The software required to run the MCU resident within the APU was written on a C cross compiler [44]. The cross compiler was run on an IBM AT. After successful compilation and linking, the program was transferred to an Erasable Programmable Read Only Memory (EPROM) using an EPROM programmer attached to the IBM AT. The EPROM was then plugged into the MCU and the APU was powered up. The first step in evaluating the hardware of the unit was to ascertain whether the audio processors had been correctly initialised by the software. Audio entering an APU's audio processor should leave it unaltered, if initialisation was successful. The second part of the test was to check each audio

processor in turn, to ensure that the process types associated with each audio processor could be controlled. This ensured that there were no software or hardware problems. MIDI system exclusive messages to alter audio processor process levels were generated from software written on an IBM AT. These MIDI messages, if chosen carefully, could be used to strobe individual data and address lines that control the flow of data into the audio processors. This was ideal for tracing signal paths, as the strobing could be observed using a logic probe or an oscilloscope. Very few problems were encountered with the software. The software modelling had reduced debugging time by reducing the possibility of software errors.

The ability of the system to process data in real time was evaluated. This was done by writing software on the IBM AT that generated audio processes at the maximum MIDI rate of 3125 bytes per second. The MCU of the APU could receive, process and update the audio processors at this rate, indicating that it could operate within the real time constraints of the design. The final C program for the APU's MCU is listed in appendix 7.3.

4.3.6 The Audio Patcher/Mixer Unit Software

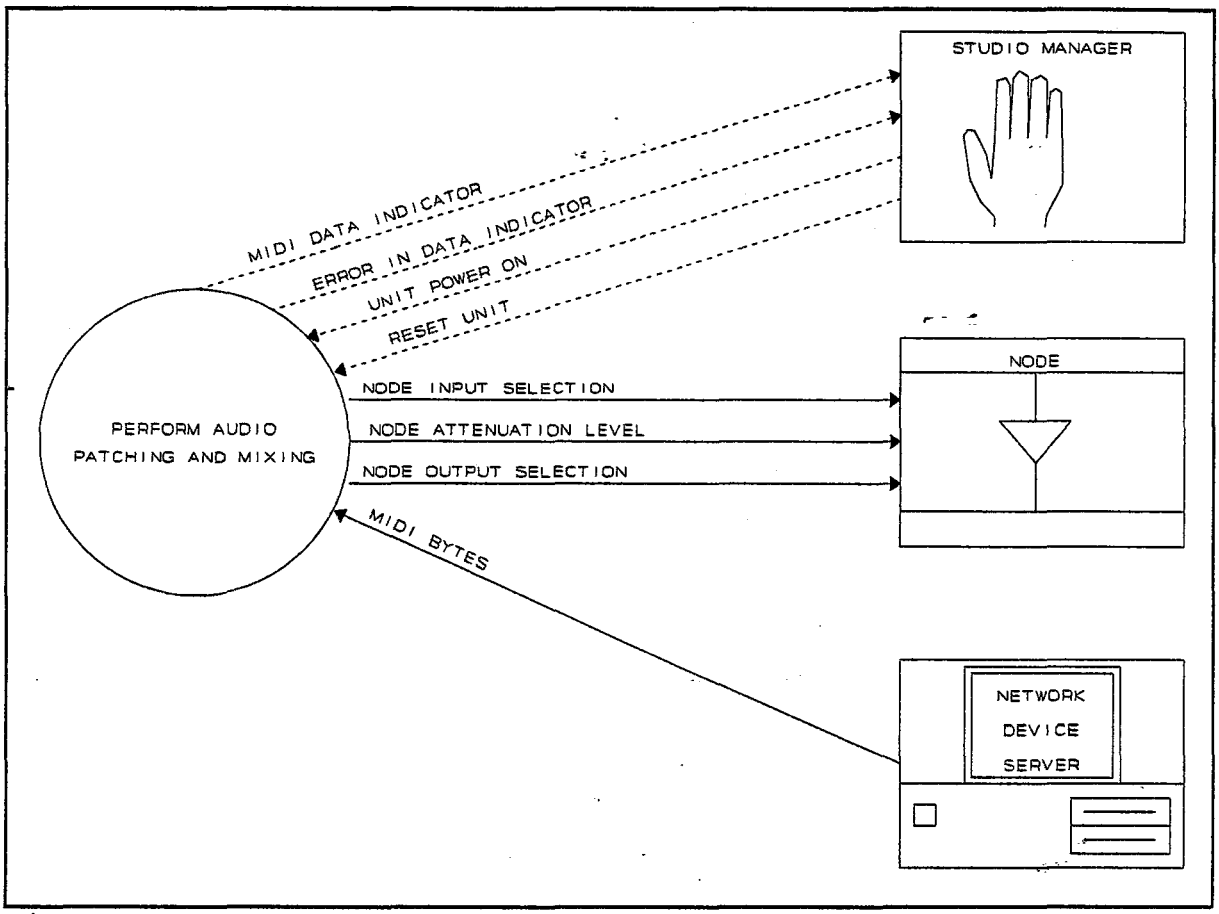

Figure 4.21 **Context Schema**

The aim of this section is to explain the development of the software for the MCU within the APMU. The software allows real time control of the audio patcher/mixer nodes within the unit. Control of the APMU is by way of MIDI system exclusive messages. The software is modelled using the same techniques used to develop the software for the APU described in section 4.3.5. The context schema is shown in figure 4.21 and the event and response list follows. The specification of the format of the MIDI system exclusive messages required to control the APMU can be found in appendix 5.2. The remainder of the software modelling descriptions can be found in appendix 6.2.

Event and Response List

EVENT	RESPONSE
Studio manager switches audio patcher/mixer unit power on.	Turn error in data and MIDI data indicators on. Disconnect all nodes and set

	all node attenuation levels to their minimum values. Turn error in data and MIDI data indicators off.
Studio manager resets audio patcher/mixer unit.	Turn error in data and MIDI data indicators on. Disconnect all nodes and set all node attenuation levels to their minimum values. Turn error in data and MIDI data indicators off.
Network device server transmits a MIDI real time message.	Ignore MIDI real time message.
Network device server transmits an audio patch request for this particular unit.	Turn MIDI data indicator on. Update nodes with all the valid audio patches in the request. Turn the MIDI Data Indicator off.

4.3.6.1 Software for the Microprocessor Control Unit of the Audio Patcher/Mixer Unit

The software was evaluated in the same manner as for the APU, as described in section 4.3.5.6. This involved generating MIDI messages using an IBM AT. The software written on the IBM AT tested that the software and the nodes were operating correctly together. This involved generating messages to connect nodes between particular audio inputs and outputs, to change the mix levels, and then to disconnect the nodes. In this manner all the nodes could be tested to ensure they could be placed at any crosspoint, that their levels could be altered correctly and that they could be successfully returned to the pool of nodes.

The software established problems with the nodes, mainly connection errors which resulted in the inability of the software to control them. The results could be detected audibly as a node could not be correctly patched or its attenuation level could not be controlled.

The APMU MCU software was also tested using MIDI messages

containing all possible error scenarios. This was to ensure that the APMU MCU illuminated the error in data indicator and then returned to receiving the MIDI data for valid audio patches. If it correctly returned to receiving MIDI data, the result could be detected audibly as node updating resumed.

The APMU MCU software was also tested to ensure that it met its real time design criteria. The updating of nodes has to occur at the rate at which audio patch requests are received. An important point regarding the 8031 serial communication is that transmit interrupts should be disabled when not required, otherwise unwanted serial interrupts occur, slowing software execution [23 p6-13].

4.3.7 The MIDI Patch Unit Software

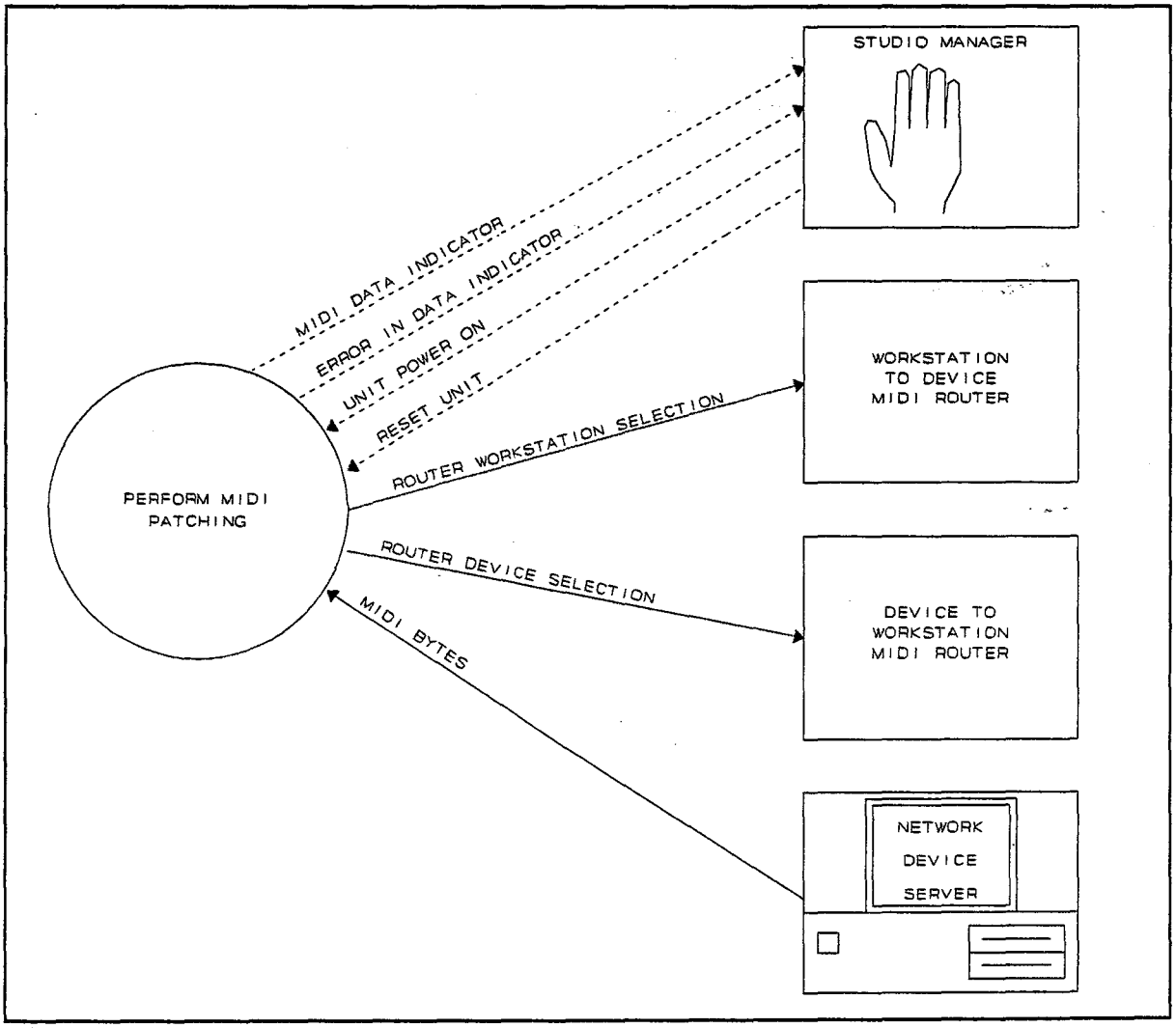

Figure 4.22 Context Schema

The aim of this section is to explain the development of the

software for the MCU within the MPU. The software allows real time control of the MIDI routers within the unit. Control of the MPU is by way of MIDI system exclusive messages. The software is modelled using the same techniques used to develop the software for the APU described in section 4.3.5. The context schema is shown in figure 4.22 and the event and response list follows. The specification of the format of the MIDI system exclusive messages required to control the MPU can be found in appendix 5.3. The remainder of the software modelling descriptions can be found in appendix 6.3.

Event and Response List

EVENT	RESPONSE
Studio manager switches MIDI patch unit on.	Turn error in data and MIDI data indicators on. Disconnect all MIDI sources from MIDI sinks. Turn error in data and MIDI data indicators off.
Studio manager resets MIDI patch unit.	Turn error in data and MIDI data indicators on. Disconnect all MIDI sources from MIDI sinks. Turn error in data and MIDI data indicators off.
Network device server transmits a MIDI real time message.	Ignore MIDI real time message.
Network device server transmits a MIDI patch request for this particular unit.	Turn the MIDI data indicator on. Update MIDI routers with all the valid MIDI patches in the request. Turn the MIDI data indicator off.

4.3.7.1 Software for the Microprocessor Control Unit of the MIDI Patch Unit

The software was developed and evaluated in the same manner as

for the APU, as described in section 4.3.5.6. This involved generating MIDI messages using an IBM AT. The software written on the IBM AT tested that the software and the MIDI routers were operating correctly together. It also tested the MPU MCU software to ensure that it met its real time design criteria. The updating of MIDI routers has to occur at the rate at which MIDI patch requests are received.

The software on the IBM AT was used to establish any existing problems with the MPU hardware. This was detectable, as problems in the hardware resulted in the inability of the MCU software to control the MIDI routers. The results could be detected audibly, as MIDI messages routed to a synthesizer were not delivered. This technique was used to ascertain that all MIDI Inputs, MIDI Outputs, MIDI extend Inputs and MIDI Thrus performed correctly on the MPU. The MPU MCU software was also evaluated using MIDI messages containing all possible error scenarios. This was to ensure that the MPU MCU illuminated the error in data indicator and then returned to receiving the MIDI data for valid MIDI patches. If it correctly returned to receiving, the result could be detected, as MIDI routing was resumed.

CHAPTER 5 - THE CONSTRUCTION OF A COMPUTER MUSIC NETWORK

This chapter describes the implementation of a minimal computer music network. It is termed "minimal" as it only offers basic facilities to a workstation user. It was primarily created to operate as a test bed for the audio processor patcher/mixer (APPM) described in section 4.1, and the MIDI patcher described in section 4.2.

The structure and operation of this computer music network is described in chapter 3. The implementation supports multiple satellite workstations. The main studio contains all of the shared resources available on the network. The network allows users at the workstations to use currently available MIDI sequencers and voice librarians to control devices in the main studio.

The facilities provided by this computer music network are a subset of those offered by the final envisaged computer music network. This implementation helped to isolate problem areas and indicate the next step to be taken in the development of the network.

5.1 Choice of an Operating System

A goal of the initial implementation was to utilise as much as possible of the software and hardware that was currently used in the single user Rhodes University Computer Music Studio. This was done to reduce the implementation time, and the amount of learning required to become a successful user of the system, as a lot of the software would be familiar. The current MIDI sequencer software runs on IBM compatibles using the IBM Disk Operating System (DOS). Other software, such as voice librarians also operate under DOS. DOS was not designed to be a multitasking system. This makes the running of multiple programs difficult and inter-program communication impossible.

One technique to overcome the limitations of DOS, is to run the network software on a computer separate from the MIDI sequencer. This permits the network software programmer to choose the operating system under which he wishes to develop the network. The extra hardware required for the test network was deemed wasteful, and other techniques were sought to overcome the limitations of DOS.

A technique has evolved to supply rudimentary multitasking abilities under DOS. Programs using this technique are termed 'Terminate and Stay Resident' (TSR) [63]. As their name implies, when these programs are run, they become resident within the run-time memory of the computer, where they stay after running to termination. They will run again at any time, after becoming resident, if a certain condition arises within the computer. Usually this condition takes the form of several keys pressed in unison, the 'hotkey'. Alternatively it may be any Input/Output device connected to the computer requiring or supplying data to the computer. The conditions for transferring control to the TSR are programmed by the writer of the TSR. The TSR assumes control by switching control from the current running program to itself. After running to termination, it retransfers control back to the original program.

The test network was written to operate under IBM DOS. The workstation network programs are TSRs. This permits the currently used MIDI sequencer to be used in conjunction with the network software at each workstation.

5.2 Implementation of the Computer Music Network Facilities

5.2.1 Booking

The resources of a computer music studio, namely the audio generators, modifiers and recorders referred to in section 2.1, are single user devices. They may only be used by a single user at any particular time. A resource must therefore be allocated to a single workstation for the time that the workstation user wishes to use it. The need to share resources amongst multiple users requires that a resource be booked at available time slots. This will provide a workstation user with sole access to that resource for those time slots.

The initial booking system is a simple system which is partially paper driven and partially computer driven. The booking of resources at available time slots occurs on paper. The connection of the resource to a workstation is performed on the workstation network computer, and is done on a simple connect and release system. This software runs on the computer music network and permits the workstation users access to the server. The server is responsible for permitting workstation access to the resources of the network. The server permits a workstation user to connect a resource to his workstation, but only if the resource is not

currently connected to another workstation. When a workstation user has completed using the resource, he simply releases it. This makes the resource available to other workstation users.

The reason for making the initial booking system so simple is twofold. Firstly, it reduces implementation time, and secondly, it helps reduce the size of the network TSR program at the workstation side. It is important to keep this program as small as possible, as it consumes valuable run time memory which could be used by the MIDI sequencer. This is a typical problem with TSR programs.

The studio manager, via the server, can book and release any resource. It is necessary for the studio manager to be able to book a resource in order to stop access to resources that may be temporarily unavailable. If a workstation user forgets to release a resource, or exceeds his time allocation for a particular resource, the studio manager can release that resource. There are no validity rules applied by the server to the studio manager. This applies to all activities supplied by this computer music network. This gives the studio manager full control over all the activities of the network.

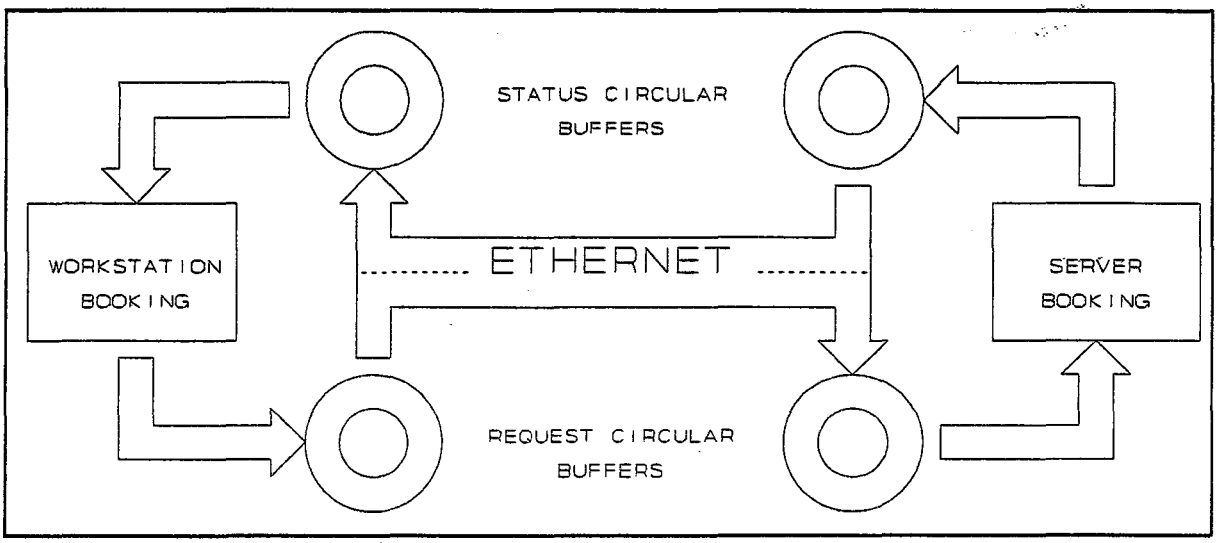

Figure 5.1 The Booking Software

Figure 5.1 illustrates how the booking software operates within the computer music network. The server booking software contains the studio manager user interface for booking and releasing devices. It also contains a booking status data structure detailing all the devices and who has current use of them. There are two circular data storage buffers associated with the server booking software. The first receives connect and release requests

from the workstation. The second circular buffer holds data which is used to inform users of the current booking status. The workstation software comprises the workstation user interface for connecting and releasing devices. It also contains the identical booking status data structure to the server. Two circular buffers allow the workstation to send connect and release requests to the server and to receive booking status data from the server.

DEVICE	USER	BOOK	COMMENTS
DX 7	-		YAMAHA DX7 SYNTHESIZER
DX 21	WS 1	X	YAMAHA DX21 SYNTHESIZER
DX 11	WS 2		YAMAHA DX11 SYNTHESIZER
D 110	WS 1	X	ROLAND D110 SYNTHESIZER
CZ 1000	SERV		CASIO CZ1000 SYNTHESIZER
S 220	-		ROLAND S220 SAMPLER
IOTA	-		IOTA FADER
SPX 900	-		YAMAHA SPX900 EFFECTS PROCESSOR
DEP 5	-		ROLAND DEP5 EFFECTS PROCESSOR

Esc for Main Menu

Figure 5.2 Booking User Interface Screen

In a typical booking sequence, the workstation user will request to connect a device via the workstation booking user interface. If the request is valid, a connect request is placed on the outgoing booking request circular buffer. A connect request is only valid if the device is not connected to another workstation user, or booked by the studio manager. The validity of the request is ascertained from the booking status data structure. Preventing invalid requests being transmitted on the LAN increases the effective transmission rate of the LAN. The circular buffer data is then removed, and sent over ethernet to the server. Ethernet is the Local Area Network (LAN) used for the transmission of non real-time control information within the computer music network, as described in chapter 3. At the server side, the data is placed on the incoming booking request circular buffer. The data is removed, and again the connect request is checked for validity, but this time against the booking status data structure held within the server. This prevents inconsistencies between the booking status data structures held within the workstation and the server. If the request is valid,

the booking status data structure within the server is altered to reflect the connection of the workstation to that device. Data detailing the booking status of all the devices is then placed on the server's outgoing booking status circular buffer. This is always done, irrespective of whether the request was valid or not. The data is then removed, and broadcast over ethernet to all the workstations. Workstations receive the data on the incoming booking status circular buffer, and use this data to update their booking status data structure and the user interface. The same sequence of events is followed when a release request is sent by a workstation.

The booking status is not only transmitted after the reception of a connect or release request. It is also transmitted at least once a second by the server. This allows workstations connecting to the network to quickly update their booking status data structure without the need to send a request. A workstation network TSR will not receive and process data from ethernet unless it is the current program running on the workstation. This is to prevent the TSR from affecting the time constraints of the MIDI sequencer. The MIDI sequencer is a program that must perform in real time. The continual transmission of booking status data ensures that when the network software becomes the current program at the workstation, the booking status data structure is quickly updated. The continual transmission of booking status data is also the only method by which the workstation users are informed of any booking operations performed by the studio manager.

Circular buffers are necessary for the reception and transmission of data due to the multiplicity of activities being performed by the workstation and server programs. The circular buffers act as temporary data storage areas. Data need not be processed immediately on reception, or transmitted immediately. Circular buffers are serviced on a round robin basis. This reduces the speed demands made on the processor. Processor speed demands can further be reduced, at the workstation, by servicing only those status circular buffers which contain status data relevant to the activity of the workstation user. The irrelevant status data is cleared from the circular buffers making way for more current status data. For example, if the current activity of the workstation user is audio recorder control, as described in section 5.2.3, only status data relevant to that audio recorder needs to be processed. Status data for other activities, such as audio mixing, described in section 5.2.2.1, need not be processed

and can be disregarded.

5.2.2 Network Device Control

The APPM discussed in section 4.1 and the MIDI patcher described in section 4.2 are essential subsystems required for the creation of a computer network. These devices are controlled by the server, which issues commands to them in response to input from the workstation users and the studio manager.

5.2.2.1 Control of the Audio Processor Patcher/Mixer

Figure 5.3 illustrates how the audio mixer software operates within the computer music network. The APPM is coupled to the server by MIDI interconnects. Audio mix requests, received by the server from the workstations, are translated into MIDI system exclusive messages to control the APPM. Section 2.2.1 describes the nature of system exclusive messages, appendix 5.1 and appendix 5.2 give the format of the system exclusive messages used to control the audio processor unit (APU) and the audio patcher/mixer unit (APMU). The APU and APMU are the building blocks of the APPM, as described in section 4.1. Circular buffers are used to receive and transmit requests and status data across the network. Outgoing MIDI data destined for the APPM is placed onto a circular buffer. The data is then removed and transmitted to the APUs and APMUs by the MIDI output device handler. Audio mixer status data is transmitted by the server on the reception of an audio mix request. Status data is also transmitted at least once a second for each APU and APMU comprising the APPM. This ensures that the computer music network TSR programs are continually updated and do not need to send update requests.

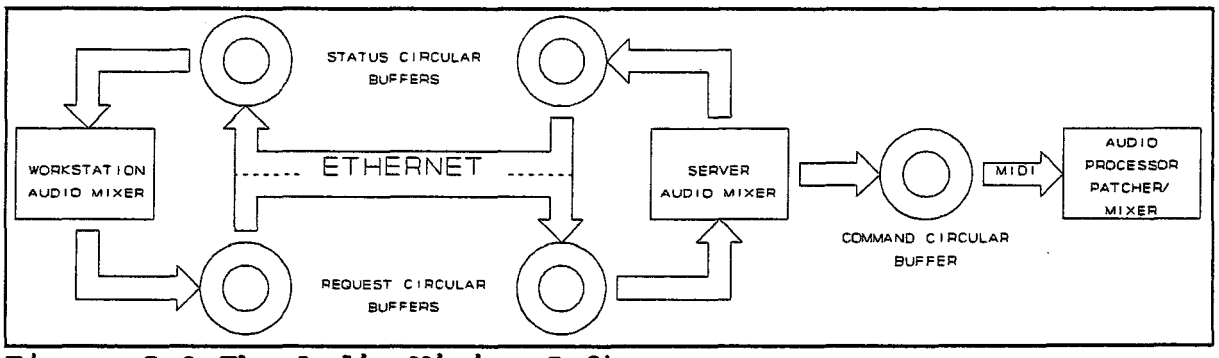

Figure 5.3 The Audio Mixing Software

Figure 5.4 illustrates the audio mixer user interface screen, the gain and equalisation settings are shown on the left and the patch points on the right. The screen scrolls, allowing the user

access to all gain and equalisation settings and patch points. When a workstation user releases a booked audio device all connected patch points associated with that audio device are disconnected. The gain and equalisation settings are also reinitialised.

Patch point edit functions permit the workstation users and the studio manager to connect audio sources to audio sinks, to then change the mix level at that patch point, and to disconnect a connected audio source and sink.

An audio mix request must be valid before the workstation will transmit it and the server will execute it. As there are two different devices comprising the APPM, there are two different sets of rules governing the validity of requests. Gain and equalisation settings can be altered only if the workstation user has booked the audio device producing audio (see figure 5.4). In order to edit a patch point the user must have booked both the audio device producing audio and the audio device to which the patch point will route audio (see figure 5.4).

GAIN dB's	BASS BOOST /CUT dB's	MID BOOST /CUT dB's	TREB BOOST /CUT dB's			AUDIO RESOURCE OUT		PATCH POINTS AND MIX LEVELS dB's				
-6.0	+0.8	-1.2	-6.8			DX 7	A	-10.0		-20.0		
-6.0	+0.8	-1.2	-6.8			DX 7	B		-10.0		-20.0	
+4.8	0.0	-12.8	-12.8			DX 21	A	-10.0		-22.4	Muted	
+4.8	0.0	-12.8	-12.8			DX 21	B		-10.0	Muted	-22.4	
+2.0	0.0	0.0	0.0			DEP 5	L			-28.0		
+2.0	0.0	0.0	0.0			DEP 5	R				-28.0	
Enter to Edit Patch Point T to Select Gain/Equ. Process Esc for Main Menu						RESOURCE		DEP 5	DEP 5	WS 1	WS 1	WS 2
						IN		L	R	L	R	L

Figure 5.4 **Audio Mixer User Interface Screen**

5.2.2.2 Control of the MIDI Patcher

The MIDI patcher software has a similar design to the APPM software, as illustrated by figure 5.5. MIDI patching differs from audio patching though, in that automation of certain

patching is possible. When a workstation user books a MIDI device, the MIDI In of that device may automatically be coupled to the MIDI Out of the workstation. This means that all MIDI devices booked by the workstation user will automatically receive MIDI from that workstation.

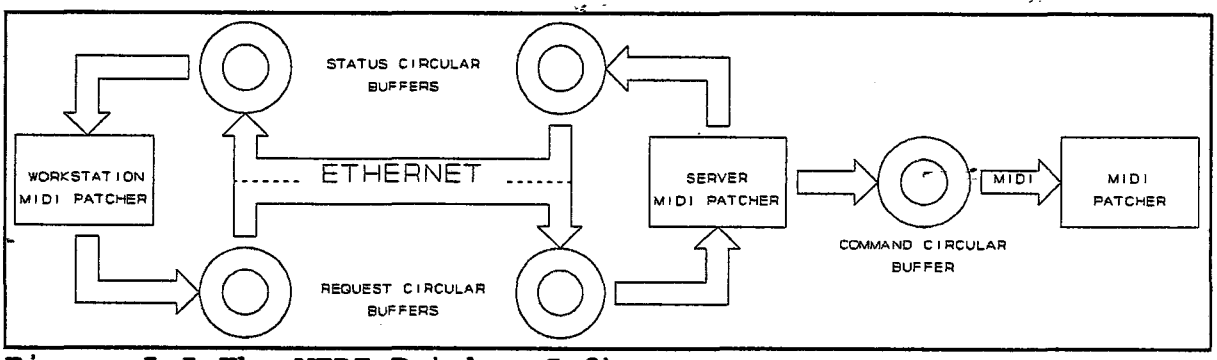

Figure 5.5 The MIDI Patcher Software

DEVICE	MIDI OUT	MIDI IN	COMMENTS
DX 7		X	Transmit Channel - Receive Channels - 6, 7
DX 21	X	X	Transmit Channel - Receive Channels - 13
DX 11		X	Transmit Channel - Receive Channels - 10
D 110			Transmit Channel - Receive Channels - 2, 3, 4, 5(R)
CZ 1000		X	Transmit Channel - Receive Channels - 1
S 220			Transmit Channel - Receive Channels - 8, 9
IOTA			Transmit Channel - Receive Channels - 16
SPX 900			Transmit Channel - Receive Channels - 14
DEP 5	-		Device has no MIDI Out Receive Channels - 15

Esc for Main Menu

Figure 5.6 MIDI Patcher User Interface Screen

The workstation user may need to receive MIDI from a booked MIDI device. This is important when the workstation sequencer must be synchronised to a video or audio recorder within the main studio, as described in section 3.8. It is necessary when receiving setup information from audio synthesizers. This two-way flow of MIDI data is also necessary for transmitting and receiving samples to and from audio samplers. To permit these operations, the workstation user can patch any MIDI Out from any booked MIDI device in the main studio to the workstation. The MIDI patcher user interface screen is shown in figure 5.6. The workstation user can patch the MIDI Out for a particular MIDI device, to the

workstation, by using the mouse or keyboard. The patch is indicated by an 'X'. The MIDI Ins are patched automatically and indicated with an 'X'. The absence of a MIDI In or Out on a device is indicated by a '-'. When a booked device is released by a workstation user, the MIDI In and the MIDI Out, if patched, are disconnected. Again the continual transmission of status data keeps users informed of MIDI patch changes.

5.2.3 Non-MIDI Device Control

Devices which are not controlled by MIDI and that do not need to be controlled in real time, can be connected to the network. Devices in this category include video recorders, audio recorders, SMPTE generators, and SMPTE to MIDI time code converters. In this implementation, there is a single SMPTE generator/SMPTE to MTC converter, coupled to an audio recorder.

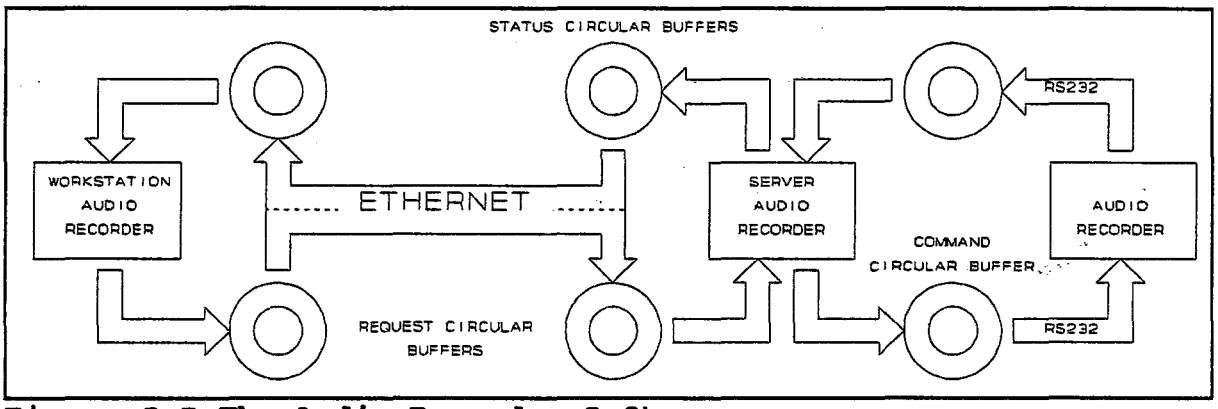

Figure 5.7 The Audio Recorder Software

The audio recorder is a Tascam 238, eight track, analogue audio recorder [20]. All the functions, apart from the shuttle control, are accessible via RS232. The audio recorder user interface was, therefore, designed to closely emulate the Tascam 238 control panel, as shown by figure 5.9. RS232 is a serial digital communications system [52 p308]. The server has the necessary RS232 hardware and interconnects to both transmit commands to, and receive status data from the audio recorder. The structure of the software required to control the audio recorder is shown in figure 5.7.

There are two problems associated with supplying network access to audio and video recorders. The first problem is that of changing the recording medium. Each network user will usually have his own collection of tapes which need to be inserted into the recorders before the workstation user can use the device. There are two solutions to this problem. The first is to allow

the workstation users access to the part of the main studio where the recorders are housed, so that they can insert and remove their own tapes. The second solution is to have a studio manager, whose responsibility it is to change tapes at the request of workstation users. These requests can be sent over the network from the workstation user to the studio manager. This solution would be possible for large commercial studios where studio technicians already perform such tasks. For smaller studios, such as at Rhodes University, the first solution is the most successful, as it does not require a studio manager. It also does not create a tape access problem. The tapes are in the possession of, and are the responsibility of, the workstation user.

The second problem with supplying network access to audio and video recorders is that of setting audio recording levels on the recorders. The record level meters that indicate the audio record levels on the audio and video recorders are remote to the workstation user. The record levels are not available via the RS232 interface on the Tascam 238. This problem can only be overcome by having a level meter at each workstation. The record level controls of the recorders should be set so that the level shown on the workstation level meter gives a true indication of the audio level being recorded on any of the recorders. To set record levels, the workstation user firstly patches the audio signal to be recorded to his level meter. The mix levels are then adjusted on the APPM so that the audio signal level does not exceed that specified by the recorder.

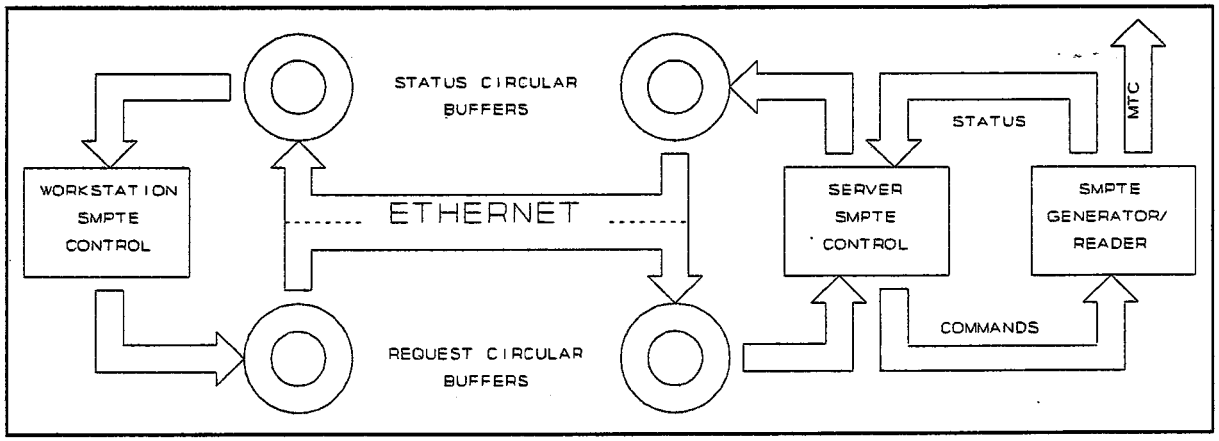

Figure 5.8 The SMPTE Control Software

The SMPTE generator/SMPTE to MTC converter, is part of a MIDI card that is resident within the server. Its function is to achieve synchronisation between the workstation sequencer and the audio recorder, as described in section 3.8. The SMPTE generator is used to stripe a single audio track on the audio recorder. The

SMPTE to MTC converter produces MTC when reading that SMPTE track. A MIDI interconnect communicates the MTC to the MIDI patch bay, from which it can be sent to the workstations. The SMPTE control software is illustrated in figure 5.8. Commands are relayed directly from the server to the SMPTE generator/SMPTE to MTC converter. There is no need to place commands in a circular buffer, as the card is resident within the server and commands are received instantly. Only the most recent SMPTE value read from the card is kept by the server, the rest are disregarded. Consequently there is no need for a status circular buffer to hold received status data.

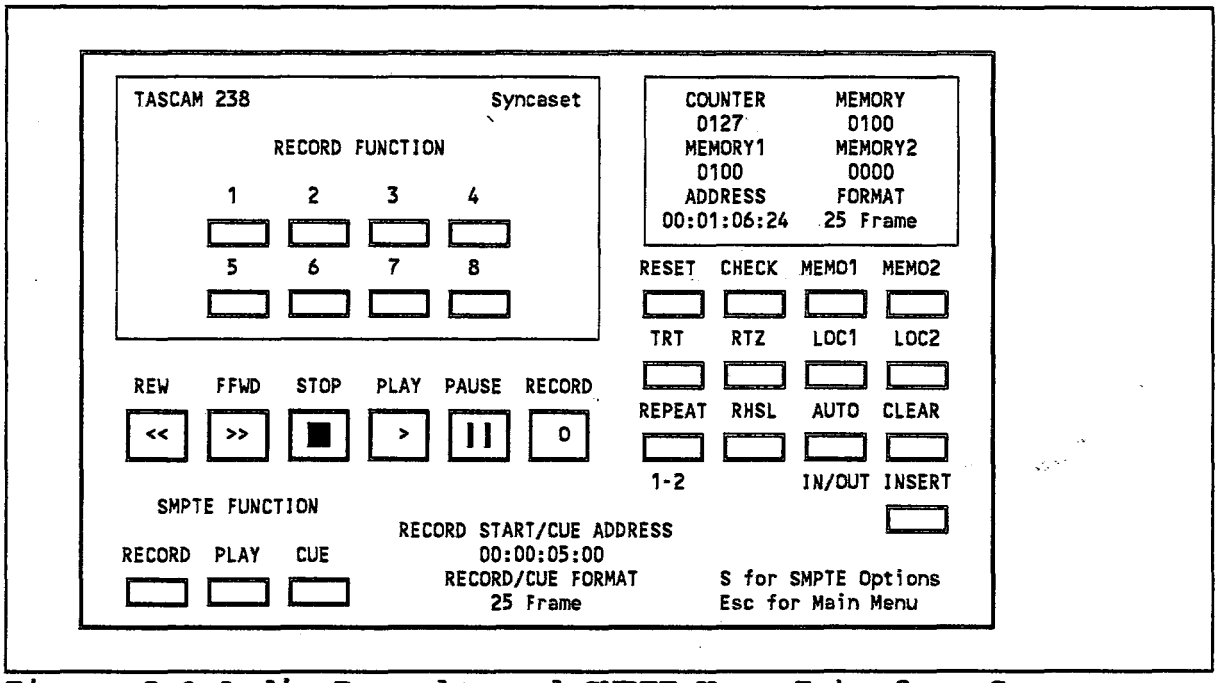

Figure 5.9 Audio Recorder and SMPTE User Interface Screen

The type of SMPTE control supplied to the user is shown in the audio recorder and SMPTE user interface screen shown in figure 5.9. The user can set the SMPTE generator start time as well as the SMPTE format, for recording onto the eighth track of the audio recorder. It also offers a cue feature. This is a special case of the SMPTE read (playback) option which will stop the audio recorder when a particular SMPTE time is read. The read option produces MTC from the SMPTE being read from the eighth track of the audio recorder.

5.3 Evaluation of the Computer Music Network

The computer music network software performed well as a test bed for the APPM and MIDI patcher. It showed how the units comprising

the two systems fitted into the network and how control over the two systems could be achieved. It also confirmed the feasibility of sharing the resources of the main studio in this way. For a user, there are both positive and negative aspects to using a workstation on the network rather than a typical studio. These are discussed below. The intention is that the negative aspects will be addressed in future network implementations.

5.3.1 Ease of Use

The response from the LAN was fast enough to give the workstation users the impression that their commands were executed immediately. It is important to give the user a feeling of direct access to the studio, as this will ensure better support for the network.

The automated MIDI patching, and the ability for the workstation user to select the device from which he is receiving MIDI, using the MIDI patching software, offers a much simpler method than the original manual MIDI patching technique in the single user studio.

While providing complete access to all the normal studio controls through a single computer may be deemed desirable, and will lead to greater productivity, this is not always the case. The user interface that a computer may offer may be more difficult to employ than the original hands on control offered by directly interfacing with the device concerned.

This was indeed the case with changing gain, equalisation and mix levels on the APPM. The current network software only permits keyboard and mouse control over these levels, and then it is on a one-at-a-time basis. A conventional audio mixer is far quicker and easier to operate. A solution to the problem is to use data entry slider type MIDI controllers as described in section 6.2.5.1.

The APPM and the user interface provide the network users with an audio mixer which is very easy to understand. The patching/mixing facilities allow for any audio source to be connected to any audio sink. The two dimensional matrix patcher/mixer user interface the workstation users employ to create mix scenarios can clearly be understood just by looking at the screen (see figure 5.4). This is in stark contrast to some conventional audio mixers, which are provided with internal block

diagrams to explain the connectivity they provide.

Slower access to the function keys, and the inability to use the shuttle function are the only complications that remote control of the Tascam 238 audio recorder introduce, when compared to direct control. The setting of record levels is complicated by not having visible access to the audio recorder, as discussed earlier. The SMPTE generator/SMPTE to MTC converter is computer controlled, so there is no change experienced in the user interface.

5.3.2 The TSR Approach

The TSR approach allows both the MIDI sequencer and the workstation network software to be on the same computer. The user therefore only interacts with a single computer. This is easier than interacting with two computers, which would occur if the MIDI sequencer and the workstation network software were on different computers.

The TSR approach poses too many limitations on the computer music network to be a viable option for further network development. The fact that the TSR program cannot communicate with the MIDI sequencer on the workstation computer is the cause of most of the problems.

The first problem is that no mix level automation is possible. The lack of communication between the TSR program and the MIDI sequencer prevents MIDI data generated by the MIDI sequencer from being used by the network software to generate audio mix requests.

A further problem related to the lack of communication between the TSR program and the MIDI sequencer is that MIDI patching can take place while the MIDI sequencer is running. The MIDI patcher is termed 'dumb', as discussed in section 4.2.2. This means that it is unaware of the MIDI information on the MIDI connections it is rerouting. The result is that MIDI messages being transmitted at the time of rerouting will be corrupted. Of course, a user should not really be doing patching while the MIDI sequencer is running.

Another problem arising from the simultaneous use of the TSR and the MIDI sequencer is that the TSR program replaces the MIDI sequencer at the time the TSR is invoked. This means that any

data arriving at the MIDI sequencer is no longer processed. In our situation the TSR does not entirely deactivate the MIDI sequencer software when it becomes active. The routines for receiving MIDI data continue to run. If these routines are receiving large amounts of MIDI data while the TSR is active, the MIDI receive buffers overflow. When the TSR becomes active while the MIDI sequencer is receiving MIDI time code from the SMPTE to MTC converter, the MIDI sequencer's receive buffers overflow quickly, due to the large amount of MIDI data being communicated. The MIDI sequencer software then has to be reset on returning from the TSR. This is not a great problem but it may have other effects on different MIDI sequencers (the MIDI sequencer used in this project was Voyetra's Sequencer Plus Gold).

The underlying cause of the above problems is that the workstation network TSR software and the MIDI sequencer are two independent non-communicating programs. Techniques which allow the network software and the MIDI sequencer to be active simultaneously at the workstation, and for the programs to communicate with one another, are described in sections 6.2.5.2 and 6.2.6. These techniques provide solutions to the problems of mix level automation and the corruption of MIDI data during rerouting.

CHAPTER 6 - CONCLUSION

Previous chapters have described the construction of a computer music network. It was designed to solve the problems arising from the shared usage of a single user studio, in particular the problems associated with the Rhodes University Computer Music Studio. It is now necessary to evaluate the extent to which the computer music network has solved these problems. It is also necessary to evaluate how well the audio processor patcher/mixer (APPM) and MIDI patcher perform their designated roles within this network. Pointers are provided to the future development of the network.

6.1 Solving the Problems of Using a Shared Studio Facility

The computer music network achieves the goals for which it was designed. It provides multiple users with simultaneous access to the facilities provided by a computer music studio. It solves many of the problems associated with sharing a single user computer music studio.

6.1.1 Improved Resource Utilisation

As discussed in section 2.5.2, the problem with a single user studio is that not all the resources of the studio are utilised at any one time. A user will only use a subset of the audio generators, modifiers and recorders available to him. Access to an unused resource is difficult, and can usually only be achieved by physically disconnecting and relocating it. This task is time consuming and would interfere with the user of the main studio. A relocated resource will also require certain ancillary devices in order to operate. An audio mixer would be required to mix audio signals. Equipment allowing the user to listen to the audio mix would also need to be provided. A MIDI sequencer and MIDI controller would be essential to provide access to MIDI controlled devices. These facilities are usually provide by creating smaller satellite studios around the main studio.

The network provides shared access without the need for the user to relocate or physically alter the interconnects to a resource. The network provides access to the resources by allowing the patching of resource interconnects directly to the workstation user. The audio processing, patching and mixing facilities offered by the APPM (section 4.1), and the MIDI patching facilities (section 4.2) offered by the MIDI patcher, are

controlled by a network server. These patching facilities are easily accessible at the workstation, via a computer. Patching occurs instantaneously and in the case of some MIDI interconnects, automatically. This saves time and therefore improves resource utilisation.

The APPM provides mixing features that exceed those provided by conventional audio mixers, providing a greater degree of flexibility to the users of the network. The matrix nature of this audio mixer allows the user to create any mix scenario, without the need to change interconnects. This greater flexibility improves resource utilisation.

6.1.2 Reduced Technical Understanding

A new studio user must learn correct studio and equipment usage before he can successfully start to create compositions. The less that is required of the user to learn, the better. The computer music network seeks to minimise the amount of learning the user must do, in order to create compositions successfully.

The matrix representation of the audio mixing within the computer music network, as described in section 4.1.5.1, greatly simplifies the complexity of current computer music studios. The user interface shows from where the audio signal originates and to where it is destined. The simplified internal structure of the APPM, when compared to conventional audio mixers, and also the absence of external audio patch bays, make audio patching and mixing far easier.

Each workstation provides the same facilities to the user. A user therefore only has to learn how to use the facilities provided by a single workstation in order to have access to the resources of the main studio. It does not require him to learn how to relocate and reconnect resources. It does not require him to learn how to use smaller satellite studios in order to use relocated equipment.

6.1.3 Addition of Resources

In a conventional studio, the inputs and outputs of new resources are usually made available on patch bays. The problem with conventional audio mixers is that they have a finite number of inputs and outputs and cannot be connected to every available resource. As described in section 2.1.5, audio patch bays have

evolved to solve this problem. Audio patch bays are not the best solution as they provide added connectivity but not in a clear intuitive manner.

Congruent and easy addition of resources is possible with our network implementation. The APPM is expandable to cater for an increasing number of audio sources and sinks as described in section 4.1.1. The expansion effectively increases the mixing and patching capabilities.

A MIDI patch bay is necessary to allow MIDI sequencers to transmit and receive MIDI to and from MIDI controllable resources, as described in section 2.2.6. Studios grow with time and a MIDI patch bay must provide for an increasing number of MIDI interconnects, otherwise it will need to be replaced.

The MIDI patcher is fully expandable to cater for the addition of MIDI controllable devices and workstations to the network, as described in section 4.2.1. This ensures that the MIDI patch bay will not need to be replaced but will grow with the network.

6.2 Future Developments

Although the software developed for the minimal computer music network was only intended to be a test bed for the APPM and MIDI patcher, it will act as a start system for more advanced computer music networks that will be written at Rhodes University in the future. The software solved many of the problems that would have been encountered in the development of these future networks. These include the remote control of audio mixing and MIDI patching, the remote control of the Tascam 238 audio recorder and the SMPTE generator/SMPTE to MTC converter. These more advanced networks will initially provide more powerful software capable of supplying more of the facilities required by the computer music network. Later improvements will also include improving the hardware facilities and technology. This section discusses some of these software and hardware improvements.

6.2.1 Online Help

An online help system would help a workstation user to come to grips with the concepts of a computer music network. It would also help him to fully utilise the facilities provided by the network and the resources that it contains. This feature would be provided by software resident in the workstation network

computer.

6.2.2 Booking

The minimal computer music network booking system, for studio resources, was a simple book and release system as described in section 5.2.1. It requires the workstation user to make his initial booking on paper. Only when he comes to use the booked resources, at the specified time, can he do any rerouting to achieve control of those booked resources. The problem with this system is that the user cannot reroute control of a resource if another user currently has control. The resource must first be released by its current user before the rerouting of control is possible. A fully computerised booking system would automatically reroute control of resources at the specified times, solving this problem.

6.2.3 Save and Recall

The minimal computer music network implementation does not supply the user with the ability to save and recall studio setup information. Including this facility would allow a user to recall gain, equalisation and mix levels which had previously been set. The initialisation phase would be greatly shortened, and consequently the effective utilisation time of studio resources would be improved. The remote control feature of the APPM permits the implementation of the save and recall option, as the server can change any gain, equalisation and mix level settings. The initial settings for a composition could be saved either on the server or on the workstation network computer for later recall.

6.2.4 Addition of Resources

At a software level, a facility to add and remove resources attached to the network is required. This would cater both for an expanding and changing computer music network. The software would allow for the addition of an audio processor unit (APU) or audio patcher/mixer unit (APMU) to the APPM, the addition of a MIDI patcher unit (MPU) to the MIDI patcher, the addition of a device to the main studio, and the addition of a new workstation.

6.2.5 Audio Processor Patcher/Mixer Control

The APPM provides for all the routing and mixing requirements of the computer music network. There are a few inherent problems

with the APPM, both as a stand alone unit and as a resource within the minimal network. These problems were envisaged at the conception of the APPM and future solutions devised.

6.2.5.1 The User Interface

The main problem with the APPM is not one of features but one of control. The workstation mixer user interface does not provide the type of control that a conventional audio mixer offers. Better control over the gain, equalisation and mix levels of the APPM can be achieved by using data entry slider type MIDI controllers, such as the J.L Cooper FaderMaster [60]. A data entry slider can be assigned to control a single level or a group of levels. This provides the same physical interface as a normal audio mixer, while retaining the greater flexibility of the APPM.

6.2.5.2 Audio Mix Automation

One can envisage, on a LAN with real time communication attributes, that the mix levels on the APPM could be changed in real time. MIDI time code produced by a workstation MIDI sequencer could be used by the workstation network software to generate synchronised audio mix requests. The requests could be sent, via the LAN, to the server. The server could then issue the appropriate mix commands to the APPM. Another approach would be for the MIDI sequencer to communicate MIDI control changes to the workstation network software. They could then be translated to audio mix requests. There is also the possibility of automating gain and equalisation levels.

Direct access, by workstation users, to the APPM is undesirable as the integrity of the flow of audio within the network would be at risk. The network software must maintain control to ensure only legal operations are performed. What is required is a technique by which MIDI time code and the MIDI control change messages generated by the workstation MIDI sequencers can be used by the network to generate audio mix requests. This translation must be performed by the network software, to ensure that only legal transformations occur. Only the audio produced from devices booked by a workstation user must be processed.

One such possible solution is illustrated in figure 6.1. The workstation MIDI sequencer and the network software are separated into two different computers. This allows both of them to be active at the same time. The workstation network computer has the

necessary MIDI hardware and interconnects to receive MIDI from the workstation MIDI sequencer. The workstation user can specify, with the software present on the network computer, how MIDI time code and/or MIDI controller change messages are used to generate audio mix requests.

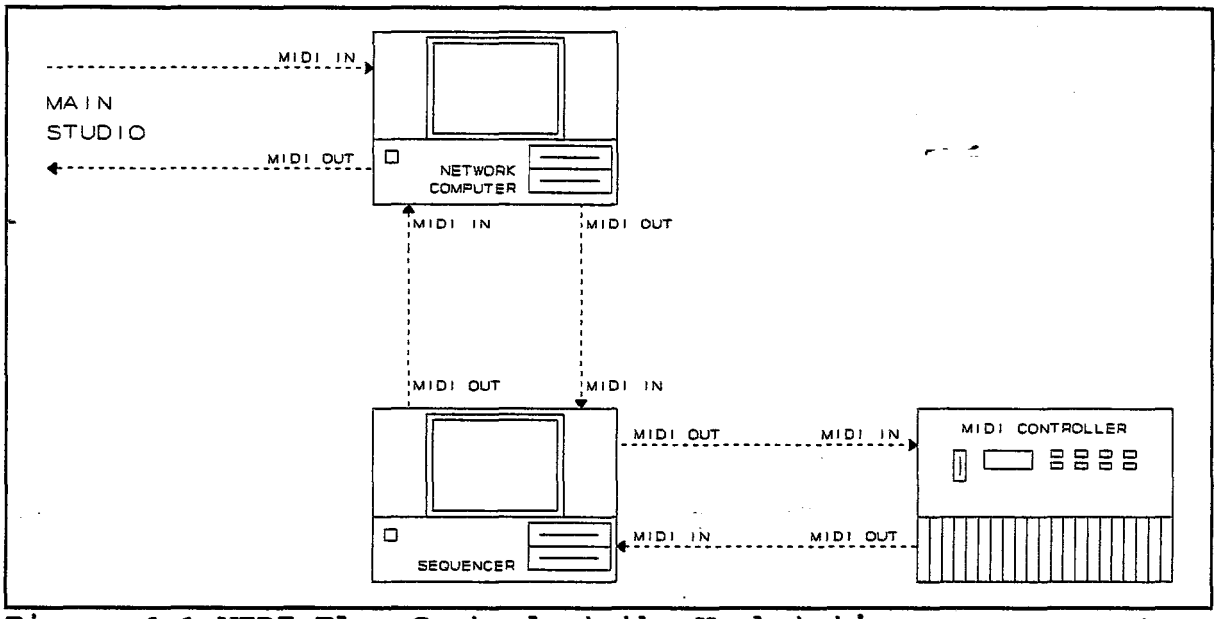

Figure 6.1 MIDI Flow Control at the Workstation

There are other advantages in separating the network software from the MIDI sequencer at the workstation side. Firstly the workstation user can use any MIDI sequencer at the workstation, not only those provided. The network software can also be developed under any operating system. This is important, as an operating system providing the multitasking and network support required by this project can be chosen. It can also be used to overcome the problem of MIDI data corruption by the MIDI patcher. This technique is discussed in section 6.2.6.

The problem with having two computers is that interaction becomes difficult as the user keeps having to change computer. One possible solution to this problem is to incorporate the software for a MIDI sequencer into the workstation computer music network software. The MIDI sequencer software would be able to issue audio mix requests over the network. Writing the software for a complete MIDI sequencer is a time consuming task. With the number of excellent MIDI sequencers on the market, it would be preferable to let the network control coreside with a sequencer. Multitasking operating systems and MIDI sequencers that operate under them are available. These would permit MIDI sequencers and the network software to coreside. Furthermore, software now

exists for these operating systems, that allow applications to communicate MIDI information between each other, by way of virtual MIDI ports which can be created between different programs. An example of such software is Apple's MIDI Manager [64]. This would provide the necessary audio mix automation and MIDI switching, to prevent MIDI data corruption while rerouting, without the need to have two computers or to write a MIDI sequencer.

Obviously audible delays caused by the inability of the network to rapidly relay the audio mix requests to the APPM server, must not occur. Apart from non real-time LANs which could cause transmission delays, there is also the problem of bottle-necks when it comes to the transmission of vast amounts of data on serial communication interconnects. Data may be delayed or lost if it cannot be transmitted fast enough. This delay in transmission could occur in three areas. On the MIDI interconnect between the workstation MIDI sequencer and the workstation network computer, on the LAN, and on the MIDI interconnect between the server and the APPM. Preventing MIDI data bottle-necks between the workstation MIDI sequencer and the network computer can only be prevented by keeping the number of MIDI controllers controlled by the MIDI sequencer low enough, so as not to exceed the transmission rate of MIDI. Bottle-necking on the LAN can be prevented by not transmitting every audio mix request. This works on the assumption that periodic updates of mix levels will provide acceptable results.

The use of volume control messages to control the level of audio signals being produced by devices is not a feasible solution to the problem of audio mix automation. Devices such as audio recorders do not provide the ability to control their output levels via MIDI. Often an audio source is required to feed more than a single audio sink. Varying the audio signal level at source will affect the signal being received by all the sinks. This is undesirable in certain mix scenarios.

MIDI controllable audio faders such as Iota systems' Midi-Fader [2] could be connected to the APPM to provide a resource that could be booked to provide mix automation. However, increasing the size of the APPM to provide for the increased number of audio inputs and outputs, is not the ideal situation, especially when the fader still does not provide the facility to automate mixing at every cross point within the matrix.

6.2.5.3 Limited Audio Processing Features

The limited audio processing features offered by the system can be improved by either including audio effects units into the APPM, as described in section 4.1.4.1, or by adding MIDI controllable audio effects units to the main studio that the workstation user must book to use.

6.2.6 MIDI Communication

The MIDI patcher developed for routing MIDI information in this network overcomes the problem of transmitting MIDI information over non real-time networks, such as ethernet. It ensures that MIDI data is transmitted in real time and that no audible delays occur.

A problem with the current MIDI patcher is the fact that it is a 'dumb' router. It takes no cognisance of the fact that MIDI data is present on the MIDI interconnects at the time of rerouting, leading to MIDI data corruption. Simply warning the user not to do any MIDI routing while the MIDI sequencer is running may not be good enough, because of the human factor. Stopping the flow of MIDI messages into and out of the workstation MIDI sequencer, while rerouting, provides a full solution to the problem of MIDI data corruption. This will prevent MIDI devices from receiving corrupt MIDI messages, as there will be no MIDI data being transmitted to them, at the time of routing. The workstation MIDI sequencer will not receive any MIDI messages during rerouting and will therefore receive no corrupt MIDI data. The network software, which controls MIDI rerouting, must control the flow of MIDI messages between the MIDI sequencer and the MIDI devices, in order for this to occur. Incorporating the MIDI sequencer software into the network software is one technique which allows the network software to control the flow of MIDI messages. This should be avoided for reasons described in section 6.2.5.2. Another possible solution is to use MIDI management software such as Apple's MIDI manager [64]. The workstation network software can then control the transfer of MIDI messages through virtual MIDI ports between the workstation MIDI sequencer and itself, and through actual MIDI ports between itself and the MIDI devices.

Another method which can be used to prevent MIDI data corruption and/or the reception of corrupt MIDI data is to employ the configuration shown in figure 6.1. The workstation network

computer has the MIDI hardware for receiving MIDI data from two sources and for transmitting MIDI data to two sinks. This hardware receives the MIDI serial data and converts it to parallel data for processing by the computer. It also performs the opposite role of converting parallel data to serial MIDI data. The workstation network computer acts as an intelligent MIDI switch. It can effectively isolate the workstation MIDI sequencer from the main studio MIDI interconnects while MIDI rerouting is occurring. This will prevent MIDI data corruption occurring to data being transmitted by the MIDI sequencer. It will also stop any corrupt MIDI data being received by the MIDI sequencer.

6.2.7 Recorder Control

It is not desirable to provide all the control features of an audio or video recorder to a workstation. Many of the features offered by these devices are redundant when one considers their operation within the computer music network. For example, if all video or audio tapes used within the video and audio recorders are striped with SMPTE, no other form of positional data is required to effectively access the material on those tapes. The 'tape counter' which provides a tape elapsed indication [17 p407], becomes redundant, as does any other function associated with the counter. SMPTE striping a tape gives each point on that tape a unique SMPTE time, something the tape counter does not. This means that locating tape positions by using SMPTE is far more accurate, and easy to duplicate. A generic interface to audio and video recorders would be useful.

6.2.8 Analogue verses Digital

All the audio processing, recording and transmission within this network implementation was done in the analogue domain. The trend nowadays is towards doing all audio processing, recording and transmission within the digital domain. It offers a far greater degree of processing control [24 p136] and noise immunity [24 p1]. The basic concepts do not change, nor does the topology of the network, if the audio processing is done in the digital domain.

The audio processing, patching and mixing will need to be done externally to the network, due to the large amounts of data and the specialised processing required. An all digital APPM will still need to offer a matrix-type mixer to achieve the mixing

requirements required by the network. Plans already exist for the construction of an APMU in which all the mixing is done in the digital domain. The unit will provide all the features of its analogue counterpart and the ability to communicate audio in digital form as well as analogue.

Centralised processing, patching and mixing of digital audio will reduce the amount of interconnects required to communicate this data between the workstation and the main studio. Resources will therefore still need to be centralised, as described in section 3.3, to make audio available to the APPM. The processed digital audio can be fed directly to the workstations using the AES/EBU standard [39]. Alternatively, because of the reduced amount of digital audio data requiring distribution, it is feasible to use high speed real time networks, such as FDDI [51 p166], to transfer this data between the main studio and the workstations. Networks such as SonicNet [65 p74] use FDDI for the transmission of digital audio but do not embody the concept of centralised audio processing and resources to reduce the amount of digital audio data needing to be transferred. The concept of a main studio with satellite workstations makes a fully digital implementation far more feasible.

The introduction of a real time network, such as FDDI, will make MIDI LAN transmissions possible. This will permit the implementation of the concept of the MIDI server as proposed by Buxton [30] and described in section 3.1. Ideally there will eventually only be a single interconnect carrying control data and performance data between the main studio and each workstation.

APPENDIX 1 - HARDWARE SPECIFICATIONS

TABLE OF CONTENTS

1.1 Electrical Characteristics of the Audio Processor Unit

Line Input:	RCA type, unbalanced
Input Impedance:	10k ohms
Nominal Input Level:	-10 dBV (0.3V)
Minimum Input Level:	-26 dBV (0.05V)
Maximum Input Level:	+20 dBV (10V)
Equaliser:	
Low:-	
Frequency:	0 Hz - 225 Hz
Boost/Cut:	+12.6 dB to -12.8 dB
Mid:-	
Frequency:	225 Hz - 5 kHz
Boost/Cut:	+12.6 dB to -12.8 dB
High:-	
Frequency:	5 kHz - 20kHz
Boost/Cut:	+12.6 dB to -12.8 dB
Line Output:	
Output Impedance:	100 ohms
Minimum Load Impedance:	2k ohms
Nominal Load Impedance:	10k ohms
Nominal Output Level:	-10 dBV (0.3V)
Maximum Output Level:	+20 dBV (10V)

1.1.1 MIDI Gain Level Verses Actual dB Gain

VALUE	dB	VALUE	dB	VALUE	dB	VALUE	dB
127	+20.0	95	+7.2	63	-5.6	31	-18.4
126	+19.6	94	+6.8	62	-6.0	30	-18.8
125	+19.2	93	+6.4	61	-6.4	29	-19.2
124	+18.8	92	+6.0	60	-6.8	28	-19.6
123	+18.4	91	+5.6	59	-7.2	27	-20.0
122	+18.0	90	+5.2	58	-7.6	26	-20.4
121	+17.6	89	+4.8	57	-8.0	25	-20.8
120	+17.2	88	+4.4	56	-8.4	24	-21.2
119	+16.8	87	+4.0	55	-8.8	23	-21.6
118	+16.4	86	+3.6	54	-9.2	22	-22.0
117	+16.0	85	+3.2	53	-9.6	21	-22.4
116	+15.6	84	+2.8	52	-10.0	20	-22.8
115	+15.2	83	+2.4	51	-10.4	19	-23.2
114	+14.8	82	+2.0	50	-10.8	18	-23.6
113	+14.4	81	+1.6	49	-11.2	17	-24.0
112	+14.0	80	+1.2	48	-11.6	16	-24.4
111	+13.6	79	+0.8	47	-12.0	15	-24.8
110	+13.2	78	+0.4	46	-12.4	14	-25.2
109	+12.8	77	+0.0	45	-12.8	13	-25.6
108	+12.4	76	-0.4	44	-13.2	12	-26.0
107	+12.0	75	-0.8	43	-13.6	11	-26.4
106	+11.6	74	-1.2	42	-14.0	10	-26.8
105	+11.2	73	-1.6	41	-14.4	9	-27.2
104	+10.8	72	-2.0	40	-14.8	8	-27.6
103	+10.4	71	-2.4	39	-15.2	7	-28.0
102	+10.0	70	-2.8	38	-15.6	6	-28.4
101	+9.6	69	-3.2	37	-16.0	5	-28.8
100	+9.2	68	-3.6	36	-16.4	4	-29.2
99	+8.8	67	-4.0	35	-16.8	3	-29.6
98	+8.4	66	-4.4	34	-17.2	2	-30.0
97	+8.0	65	-4.8	33	-17.6	1	-30.4
96	+7.6	64	-5.2	32	-18.0	0	-30.8

1.1.2 MIDI Equalisation Level Verses Actual dB Boost/Cut

VALUE	dB	VALUE	dB	VALUE	dB	VALUE	dB
127	+12.6	95	+6.2	63	-0.2	31	-6.6
126	+12.4	94	+6.0	62	-0.4	30	-6.8
125	+12.2	93	+5.8	61	-0.6	29	-7.0
124	+12.0	92	+5.6	60	-0.8	28	-7.2
123	+11.8	91	+5.4	59	-1.0	27	-7.4
122	+11.6	90	+5.2	58	-1.2	26	-7.6
121	+11.4	89	+5.0	57	-1.4	25	-7.8
120	+11.2	88	+4.8	56	-1.6	24	-8.0
119	+11.0	87	+4.6	55	-1.8	23	-8.2
118	+10.8	86	+4.4	54	-2.0	22	-8.4
117	+10.6	85	+4.2	53	-2.2	21	-8.6
116	+10.4	84	+4.0	52	-2.4	20	-8.8
115	+10.2	83	+3.8	51	-2.6	19	-9.0
114	+10.0	82	+3.6	50	-2.8	18	-9.2
113	+9.8	81	+3.4	49	-3.0	17	-9.4
112	+9.6	80	+3.2	48	-3.2	16	-9.6
111	+9.4	79	+3.0	47	-3.4	15	-9.8
110	+9.2	78	+2.8	46	-3.6	14	-10.0
109	+9.0	77	+2.6	45	-3.8	13	-10.2
108	+8.8	76	+2.4	44	-4.0	12	-10.4
107	+8.6	75	+2.2	43	-4.2	11	-10.6
106	+8.4	74	+2.0	42	-4.4	10	-10.8
105	+8.2	73	+1.8	41	-4.6	9	-11.0
104	+8.0	72	+1.6	40	-4.8	8	-11.2
103	+7.8	71	+1.4	39	-5.0	7	-11.4
102	+7.6	70	+1.2	38	-5.2	6	-11.6
101	+7.4	69	+1.0	37	-5.4	5	-11.8
100	+7.2	68	+0.8	36	-5.6	4	-12.0
99	+7.0	67	+0.6	35	-5.8	3	-12.2
98	+6.8	66	+0.4	34	-6.0	2	-12.4
97	+6.6	65	+0.2	33	-6.2	1	-12.6
96	+6.4	64	0	32	-6.4	0	-12.8

1.2 Performance Characteristics of the Audio Processor Unit

Signal-to-Noise Ratio:	70 dB
Total Harmonic Distortion (THD):	0.01%, 20 Hz - 20 KHz
Intermodulation Distortion (IMD):	0.04%, 20 Hz - 20 kHz
Frequency Response:-	
Any Input to Any Output:	20 Hz - 25 kHz +- 1.0 dB
Cross-Talk (1kHz):	80dB

1.3 Electrical Characteristics of the Audio Patcher/Mixer Unit

Line Input:	RCA type, unbalanced
Input Impedance:	10k ohms
Nominal Input Level:	-10 dBV (0.3V)
Maximum Input Level:	+20 dBV (10V)
Extend Input:	RCA type, unbalanced
Input Impedance:	10k ohms
Nominal Input Level:	-10 dBV (0.3V)
Maximum Input Level:	+20 dBV (10V)
Line Output:	RCA type, unbalanced
Output Impedance:	22 ohms
Minimum Load Impedance:	500 ohms
Nominal Load Impedance:	1K ohms
Nominal Output Level:	-10 dBV (0.3V)
Maximum Output Level:	+20 dBV (10V)
Extend Output:	RCA type, unbalanced
Output Impedance:	100 ohms
Minimum Load Impedance:	2K ohms
Nominal Load Impedance:	10K ohms
Nominal Output Level:	-10 dBV (0.3V)
Maximum Output Level:	+20 dBV (10V)

Attenuation at 1KHz: 0 to -100dB

Attenuation Step Size:-

0 to -12.0 dB:	0.2 dB
-12.0 to -24.0 dB:	0.4 dB
-24.0 to -36.0 dB:	0.8 dB
-36.0 to -52.0 dB:	1.6 dB
-52.0 to -68.0 dB:	3.2 dB
-68.0 to -101.6 dB:	4.8 dB

1.3.1 Mix Attenuation Level Map (decibel form)

VALUE	dB	VALUE	dB	VALUE	dB	VALUE	dB
127	0	95	-6.4	63	-13.6	31	-28.8
126	-0.2	94	-6.6	62	-14.0	30	-29.6
125	-0.4	93	-6.8	61	-14.4	29	-30.4
124	-0.6	92	-7.0	60	-14.8	28	-31.2
123	-0.8	91	-7.2	59	-15.2	27	-32.0
122	-1.0	90	-7.4	58	-15.6	26	-32.8
121	-1.2	89	-7.6	57	-16.0	25	-33.6
120	-1.4	88	-7.8	56	-16.4	24	-34.4
119	-1.6	87	-8.0	55	-16.8	23	-35.2
118	-1.8	86	-8.2	54	-17.2	22	-36.0
117	-2.0	85	-8.4	53	-17.6	21	-37.6
116	-2.2	84	-8.6	52	-18.0	20	-39.2
115	-2.4	83	-8.8	51	-18.4	19	-40.8
114	-2.6	82	-9.0	50	-18.8	18	-42.4
113	-2.8	81	-9.2	49	-19.2	17	-44.0
112	-3.0	80	-9.4	48	-19.6	16	-45.6
111	-3.2	79	-9.6	47	-20.0	15	-47.2
110	-3.4	78	-9.8	46	-20.4	14	-48.8
109	-3.6	77	-10.0	45	-20.8	13	-50.4
108	-3.8	76	-10.2	44	-21.2	12	-52.0
107	-4.0	75	-10.4	43	-21.6	11	-55.2
106	-4.2	74	-10.6	42	-22.0	10	-58.4
105	-4.4	73	-10.8	41	-22.4	9	-61.6
104	-4.6	72	-11.0	40	-22.8	8	-64.8
103	-4.8	71	-11.2	39	-23.2	7	-68.0
102	-5.0	70	-11.4	38	-23.6	6	-72.8
101	-5.2	69	-11.6	37	-24.0	5	-77.6
100	-5.4	68	-11.8	36	-24.8	4	-82.4
99	-5.6	67	-12.0	35	-25.6	3	-87.2
98	-5.8	66	-12.4	34	-26.4	2	-92.0
97	-6.0	65	-12.8	33	-27.2	1	-96.8
96	-6.2	64	-13.2	32	-28.0	0	-101.6

1.4 Performance Characteristics of the Audio Patcher/Mixer Unit

Signal-to-Noise Ratio:	70 dB
Total Harmonic Distortion (THD):	0.01%, 20 Hz - 20 KHz
Intermodulation Distortion (IMD):	0.01%, 20 Hz - 20 kHz
Frequency Response:-	
Any Input to Any Output:	20 Hz - 25 kHz +- 0.5 dB
Cross-Talk (1kHz):	80dB

APPENDIX 2 - CIRCUIT DIAGRAMS

TABLE OF CONTENTS

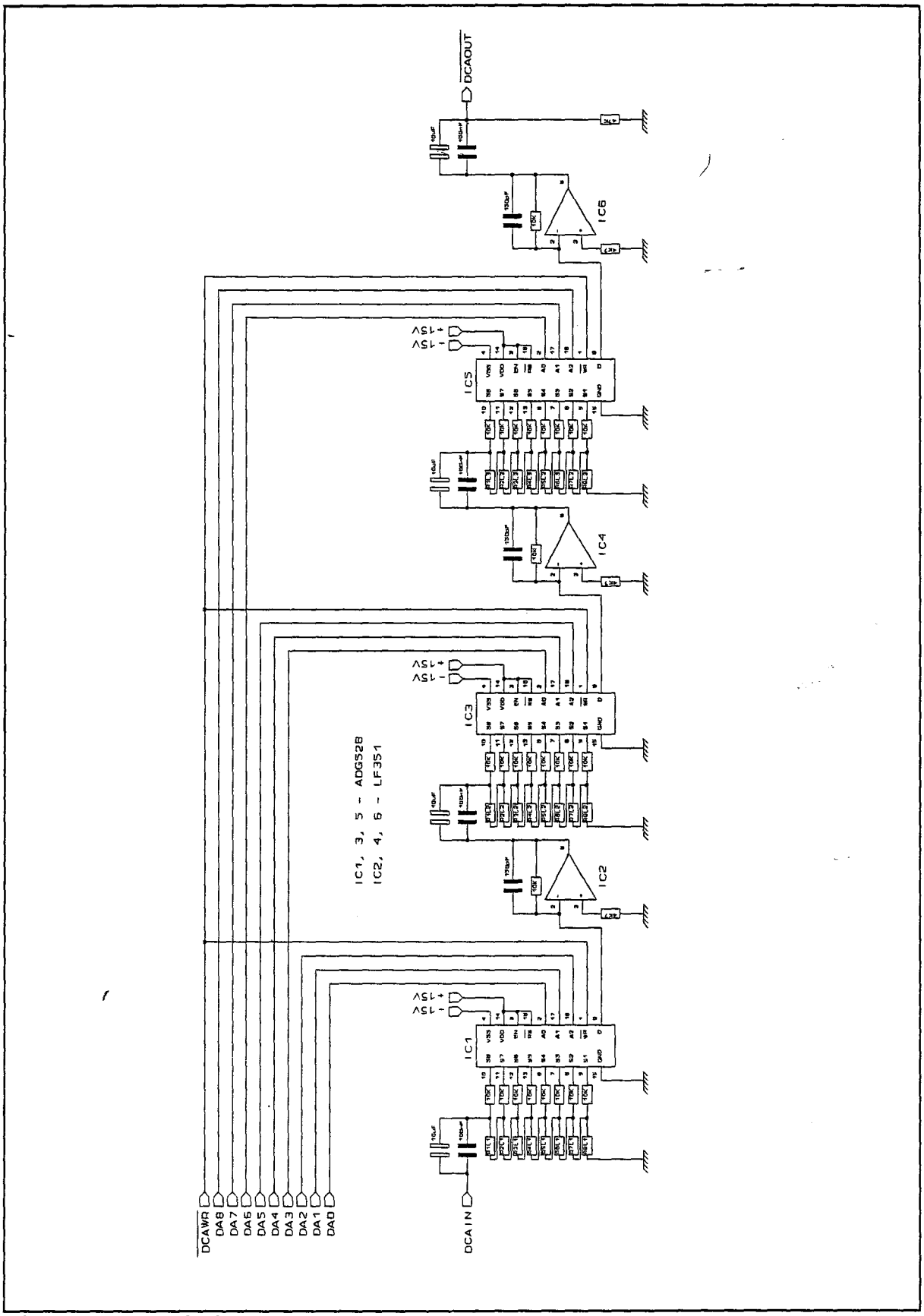

Figure 2.1 Digitally Controlled Attenuator

2.2 The Audio Processor Unit's Audio Processing Circuitry

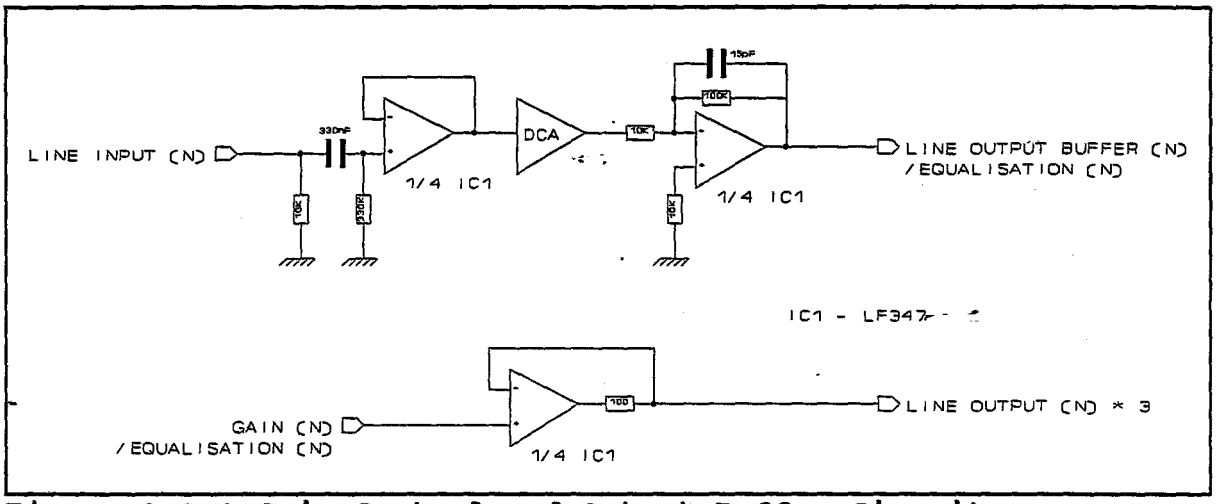

Figure 2.2.1 Gain Control and Output Buffer Circuitry

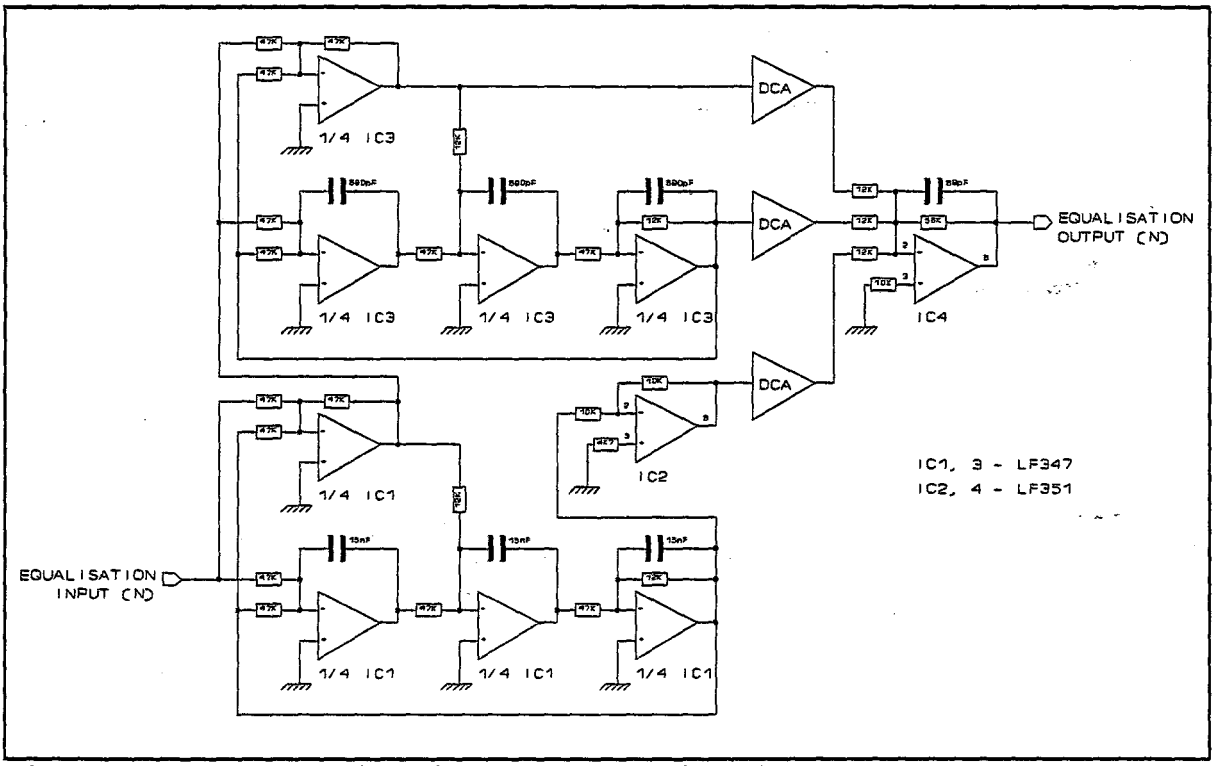

Figure 2.2.2 Equalisation Control Circuitry

2.2.3 Audio Channel Processors Port Addresses (binary form)

CHANNEL	GAIN PROCESS ADDRESS
1	10000000 00000011
2	10000000 00000100
3	10000000 00000101
4	10000000 00000110
5	10000000 00000111
6	10000000 00001000
7	10000000 00001001
8	10000000 00001010
9	10000000 00001011
10	10000000 00001100
11	10000000 00001101
12	10000000 00001110
13	10000000 00001111
14	10000000 00010000
15	10000000 00010001
16	10000000 00010010

CHANNEL	BASS PROCESS ADDRESS	MIDRANGE PROCESS ADDRESS	TREBLE PROCESS ADDRESS
1	10000000 01000011	10000000 01000100	10000000 01000101
2	10000000 01000110	10000000 01000111	10000000 01001000
3	10000000 01001001	10000000 01001010	10000000 01001011
4	10000000 01001100	10000000 01001101	10000000 01001110
5	10000000 01001111	10000000 01010000	10000000 01010001
6	10000000 01010010	10000000 01010011	10000000 01010100
7	10000000 01010101	10000000 01010110	10000000 01010111
8	10000000 01011000	10000000 01011001	10000000 01011010
9	10000000 10000011	10000000 10000100	10000000 10000101
10	10000000 10000110	10000000 10000111	10000000 10001000
11	10000000 10001001	10000000 10001010	10000000 10001011
12	10000000 10001100	10000000 10001101	10000000 10001110
13	10000000 10001111	10000000 10010000	10000000 10010001
14	10000000 10010010	10000000 10010011	10000000 10010100
15	10000000 10010101	10000000 10010110	10000000 10010111
16	10000000 10011000	10000000 10011001	10000000 10011010

Most Significant Byte Register for Gain controls
resides at address : 10000000 00000001

Least Significant Byte Register for Gain controls
resides at address : 10000000 00000010

Most Significant Byte Register for Equalisation controls
channels 1-8 resides at address : 10000000 01000001

Least Significant Byte Register for Equalisation controls
channels 1-8 resides at address : 10000000 01000010

Most Significant Byte Register for Equalisation controls
channels 9-16 resides at address : 10000000 10000001

Least Significant Byte Register for Equalisation controls
channels 9-16 resides at address : 10000000 10000010

2.3 The Audio Patcher/Mixer Unit's Audio Routing and Mixing Circuitry

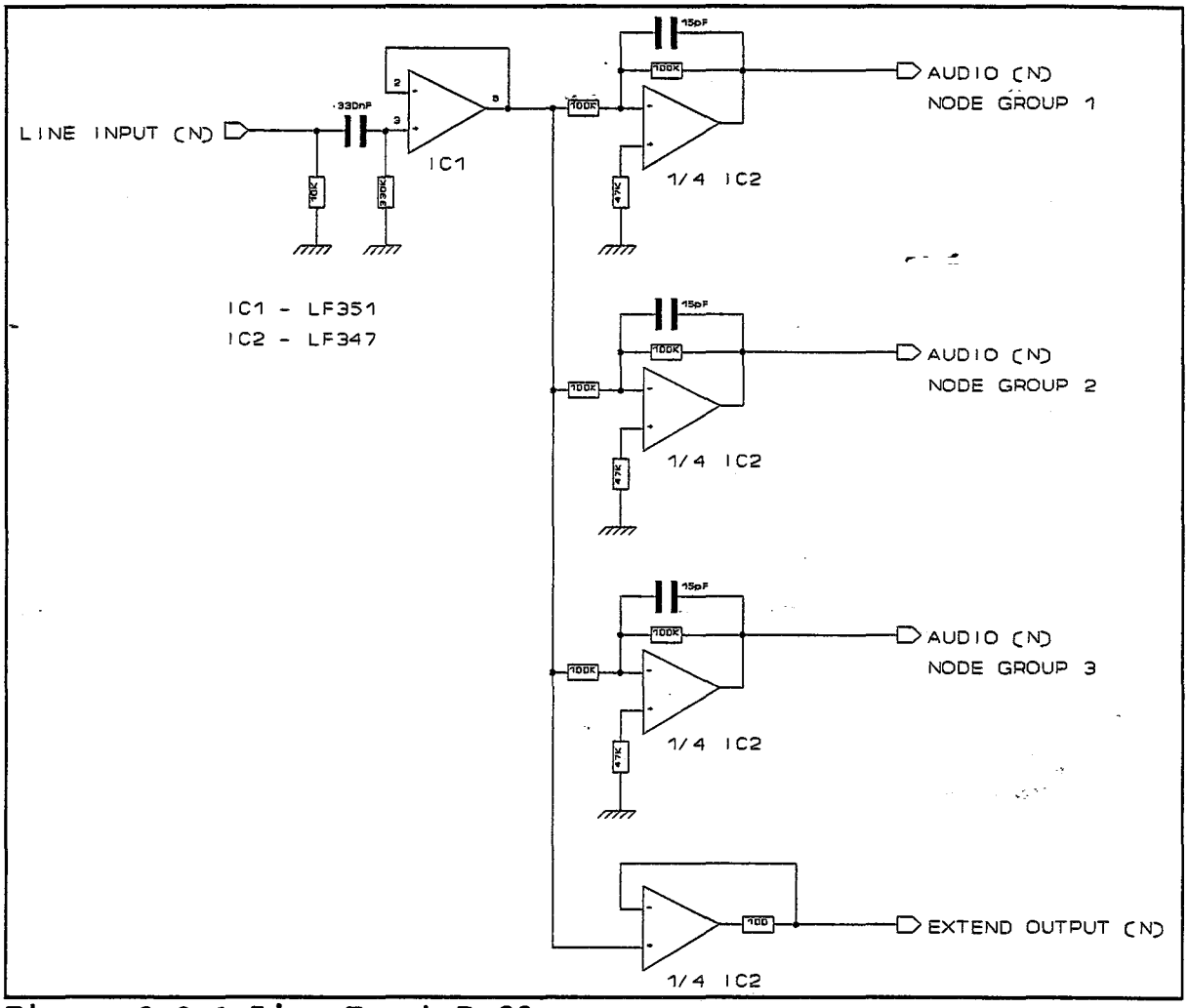

Figure 2.3.1 Line Input Buffer

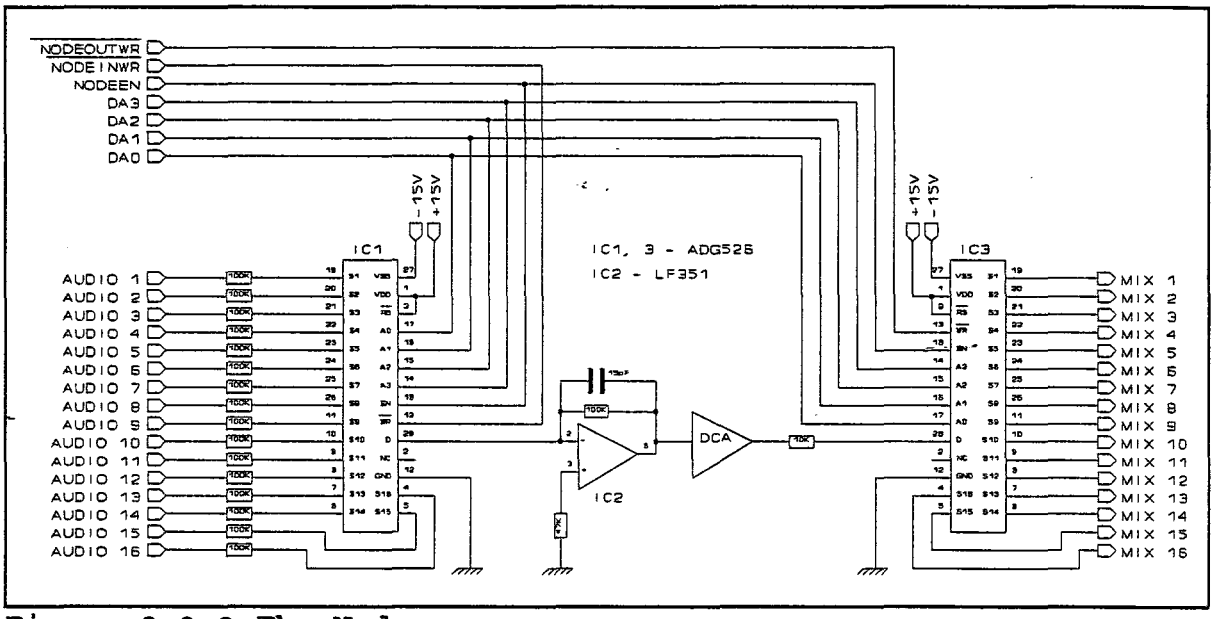

Figure 2.3.2 The Node

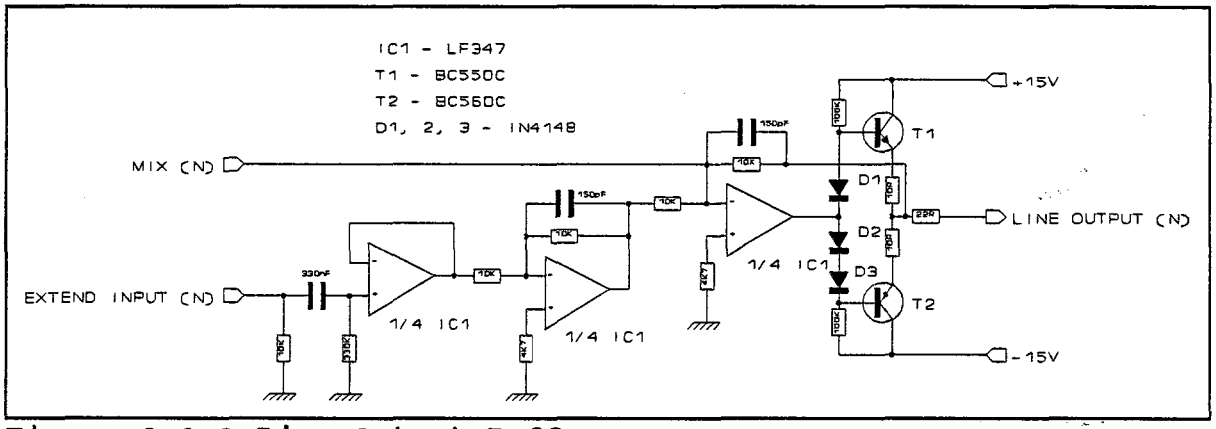

Figure 2.3.3 Line Output Buffer

2.3.4 Node Port Addresses (binary form)

NODE	AUDIO PATCH INPUT ADDRESS	DCA ADDRESS	AUDIO PATCH OUTPUT ADDRESS
1	10000000 00000011	10000000 00000100	10000000 00000101
2	10000000 00000110	10000000 00000111	10000000 00001000
3	10000000 00001001	10000000 00001010	10000000 00001011
4	10000000 00001100	10000000 00001101	10000000 00001110
5	10000000 00001111	10000000 00010000	10000000 00010001
6	10000000 00010010	10000000 00010011	10000000 00010100
7	10000000 00010101	10000000 00010110	10000000 00010111
8	10000000 00011000	10000000 00011001	10000000 00011010
9	10000000 00011011	10000000 00011100	10000000 00011101
10	10000000 00011110	10000000 00011111	10000000 00100000
11	10000000 00100001	10000000 00100010	10000000 00100011
12	10000000 00100100	10000000 00100101	10000000 00100110
13	10000000 00100111	10000000 00101000	10000000 00101001
14	10000000 00101010	10000000 00101011	10000000 00101100
15	10000000 00101101	10000000 00101110	10000000 00101111
16	10000000 00110000	10000000 00110001	10000000 00110010
17	10000000 01000011	10000000 01000100	10000000 01000101
18	10000000 01000110	10000000 01000111	10000000 01001000
19	10000000 01001001	10000000 01001010	10000000 01001011
20	10000000 01001100	10000000 01001101	10000000 01001110
21	10000000 01001111	10000000 01010000	10000000 01010001
22	10000000 01010010	10000000 01010011	10000000 01010100
23	10000000 01010101	10000000 01010110	10000000 01010111
24	10000000 01011000	10000000 01011001	10000000 01011010
25	10000000 01011011	10000000 01011100	10000000 01011101
26	10000000 01011110	10000000 01011111	10000000 01100000
27	10000000 01100001	10000000 01100010	10000000 01100011
28	10000000 01100100	10000000 01100101	10000000 01100110
29	10000000 01100111	10000000 01101000	10000000 01101001
30	10000000 01101010	10000000 01101011	10000000 01101100
31	10000000 01101101	10000000 01101110	10000000 01101111
32	10000000 01110000	10000000 01110001	10000000 01110010

33	10000000 10000011	10000000 10000100	10000000 10000101
34	10000000 10000110	10000000 10000111	10000000 10001000
35	10000000 10001001	10000000 10001010	10000000 10001011
36	10000000 10001100	10000000 10001101	10000000 10001110
37	10000000 10001111	10000000 10010000	10000000 10010001
38	10000000 10010010	10000000 10010011	10000000 10010100
39	10000000 10010101	10000000 10010110	10000000 10010111
40	10000000 10011000	10000000 10011001	10000000 10011010
41	10000000 10011011	10000000 10011100	10000000 10011101
42	10000000 10011110	10000000 10011111	10000000 10100000
43	10000000 10100001	10000000 10100010	10000000 10100011
44	10000000 10100100	10000000 10100101	10000000 10100110
45	10000000 10100111	10000000 10101000	10000000 10101001
46	10000000 10101010	10000000 10101011	10000000 10101100
47	10000000 10101101	10000000 10101110	10000000 10101111
48	10000000 10110000	10000000 10110001	10000000 10110010

Most Significant Byte Register for Nodes 1-16
resides at address : 10000000 00000001

Least Significant Byte Register for Nodes 1-16
resides at address : 10000000 00000010

Most Significant Byte Register for Nodes 16-32
resides at address : 10000000 01000001

Least Significant Byte Register for Nodes 16-32
resides at address : 10000000 01000010

Most Significant Byte Register for Nodes 32-48
resides at address : 10000000 10000001

Least Significant Byte Register for Nodes 32-48
resides at address : 10000000 10000010

2.4 The MIDI Patch Unit's MIDI Routing Circuitry

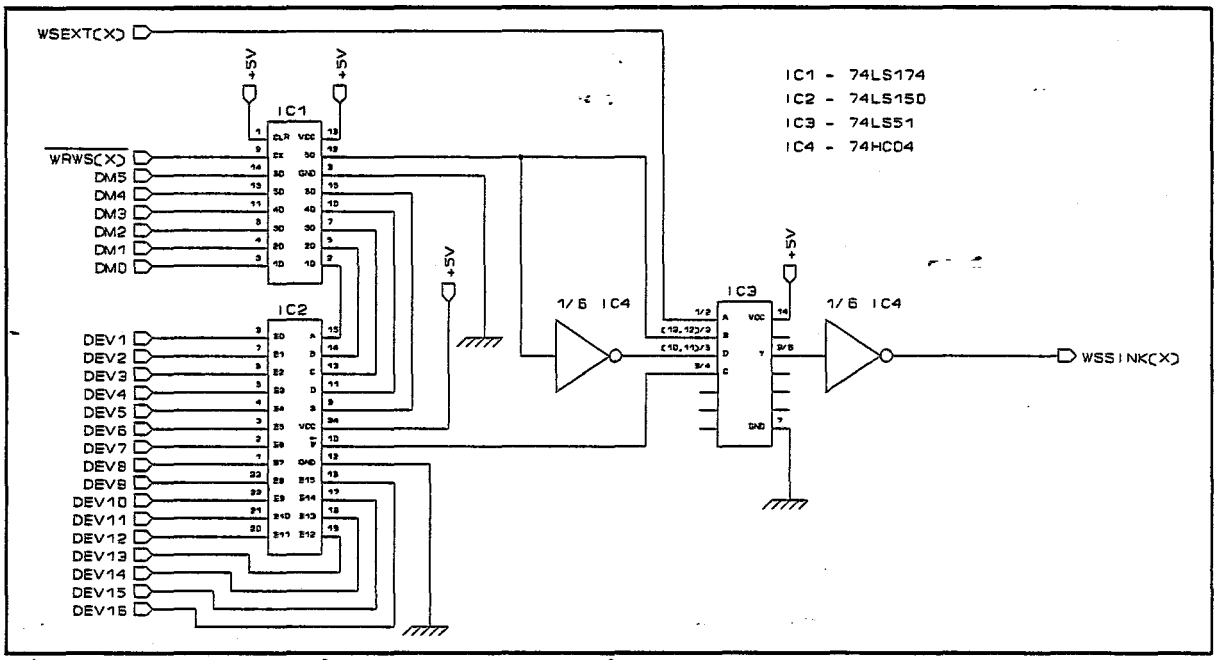

Figure 2.4.1 Device to Workstation MIDI Router

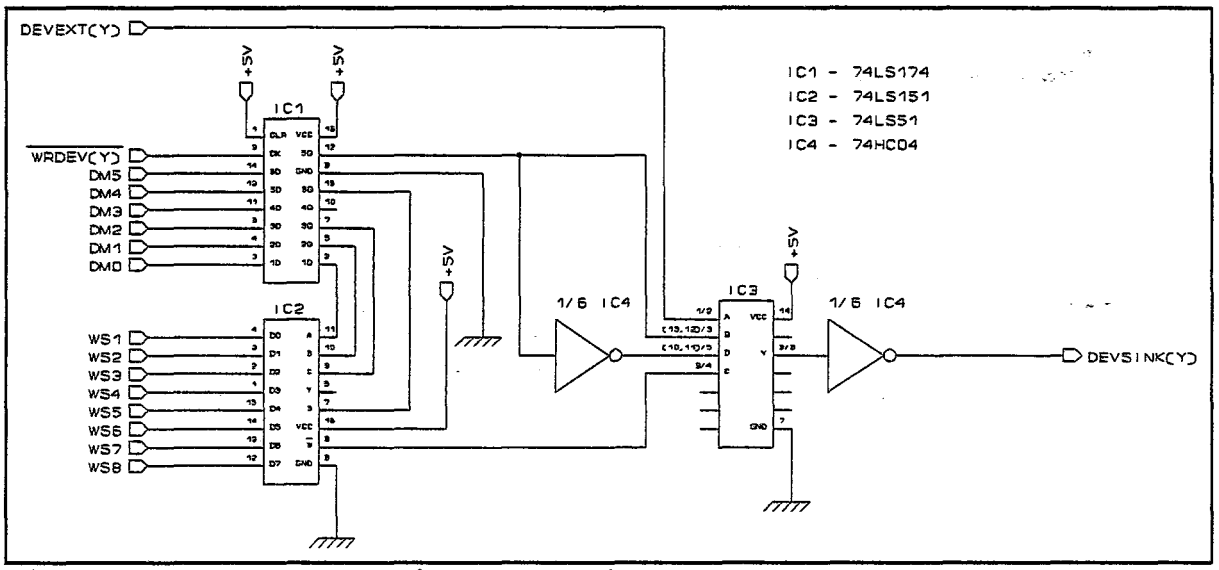

Figure 2.4.2 Workstation to Device MIDI Router

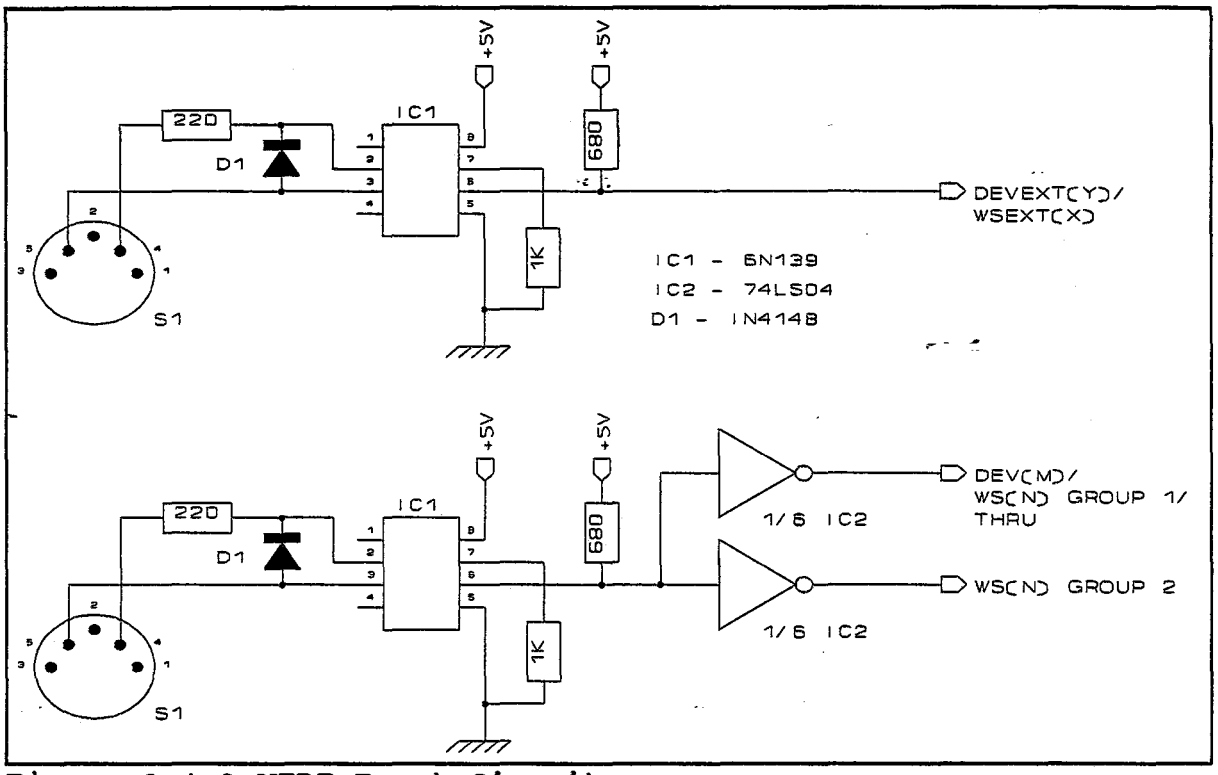

Figure 2.4.3 MIDI Input Circuitry

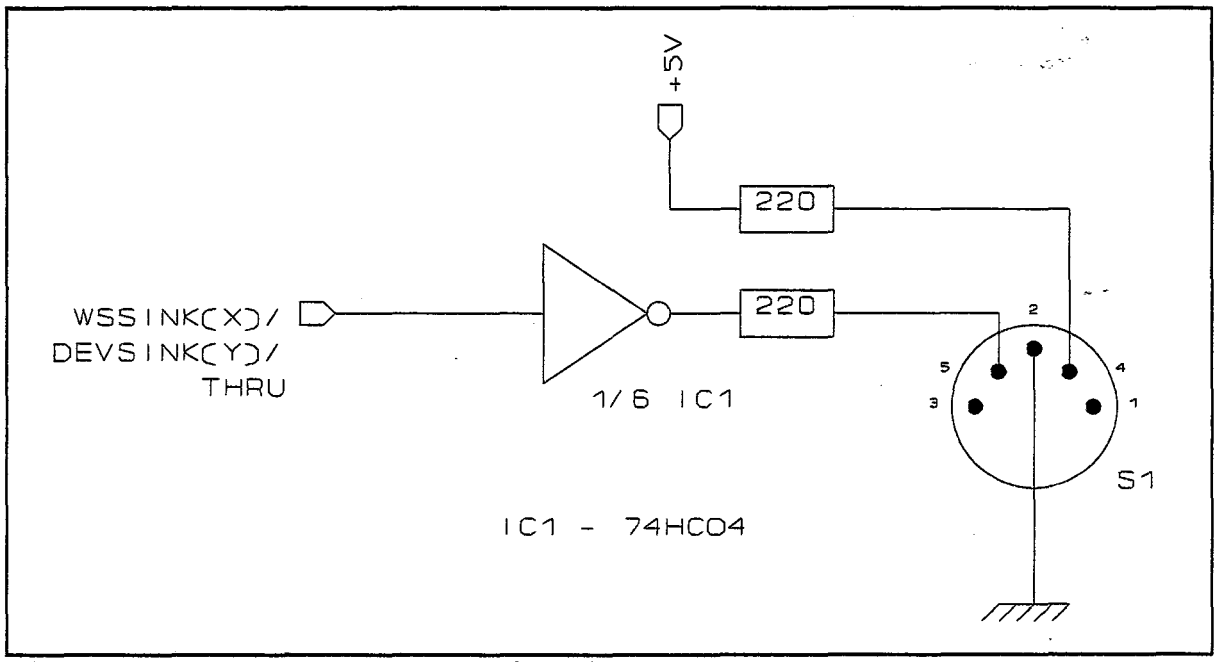

Figure 2.4.4 MIDI Output Circuitry

2.4.5 MIDI Router Port Addresses (binary form)

WORKSTATION	WORKSTATION TO DEVICE MIDI ROUTER ADDRESS
1	10000000 00000011
2	10000000 00000100
3	10000000 00000101
4	10000000 00000110
5	10000000 00000111
6	10000000 00001000
7	10000000 00001001
8	10000000 00001010
9	10000000 01000011
10	10000000 01000100
11	10000000 01000101
12	10000000 01000110
13	10000000 01000111
14	10000000 01001000
15	10000000 01001001
16	10000000 01001010

DEVICE	DEVICE TO WORKSTATION MIDI ROUTER ADDRESS
1	10000000 10000011
2	10000000 10000100
3	10000000 10000101
4	10000000 10000110
5	10000000 10000111
6	10000000 10001000
7	10000000 10001001
8	10000000 10001010

Register for Workstation to Device MIDI Routers 1-8
resides at address : 10000000 00000001

Register for Workstation to Device MIDI Routers 9-16
resides at address : 10000000 01000001

Register for Device to Workstation MIDI Routers
resides at address : 10000000 10000001

2.5 The Microprocessor Control Unit

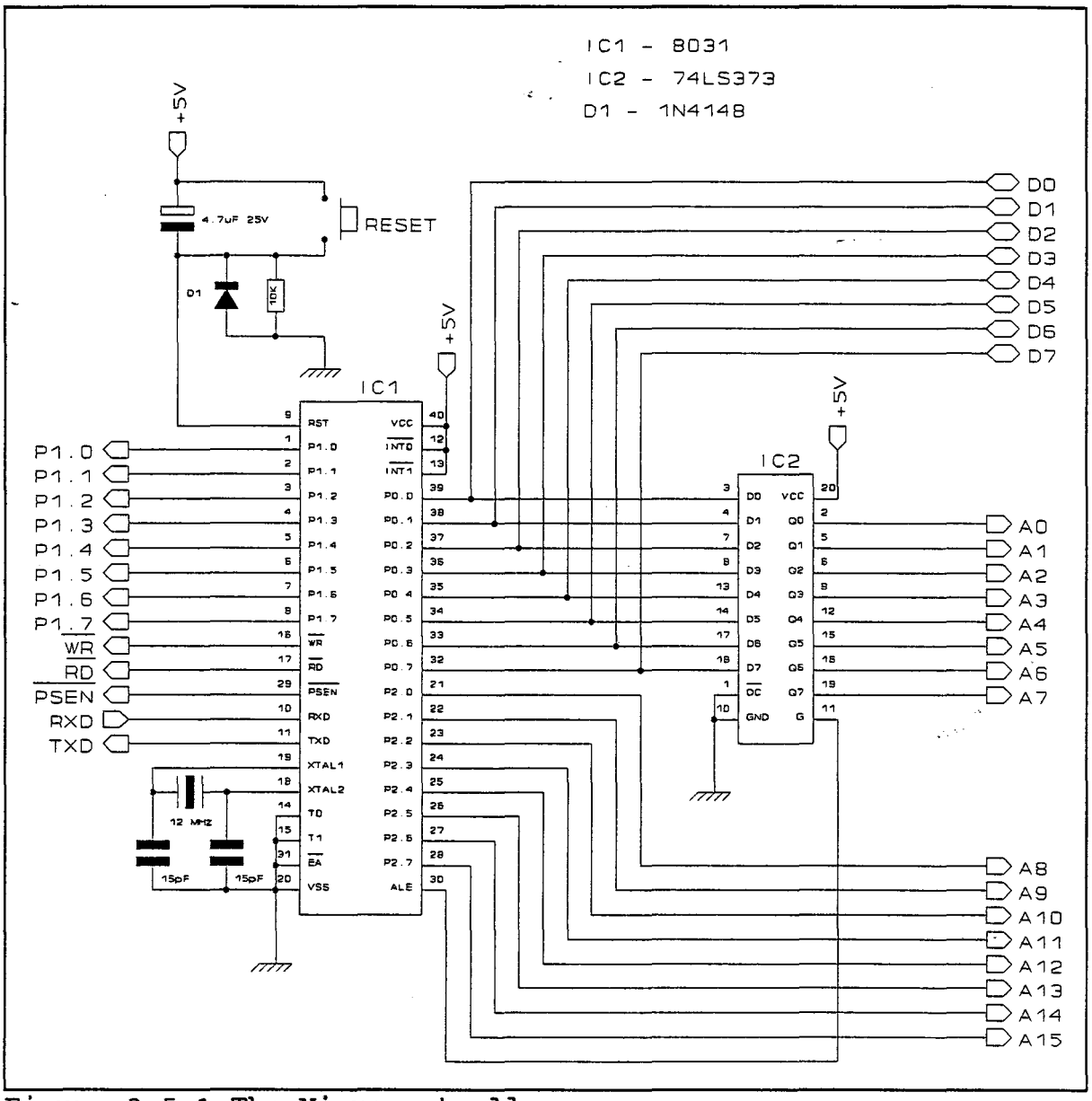

Figure 2.5.1 The Microcontroller

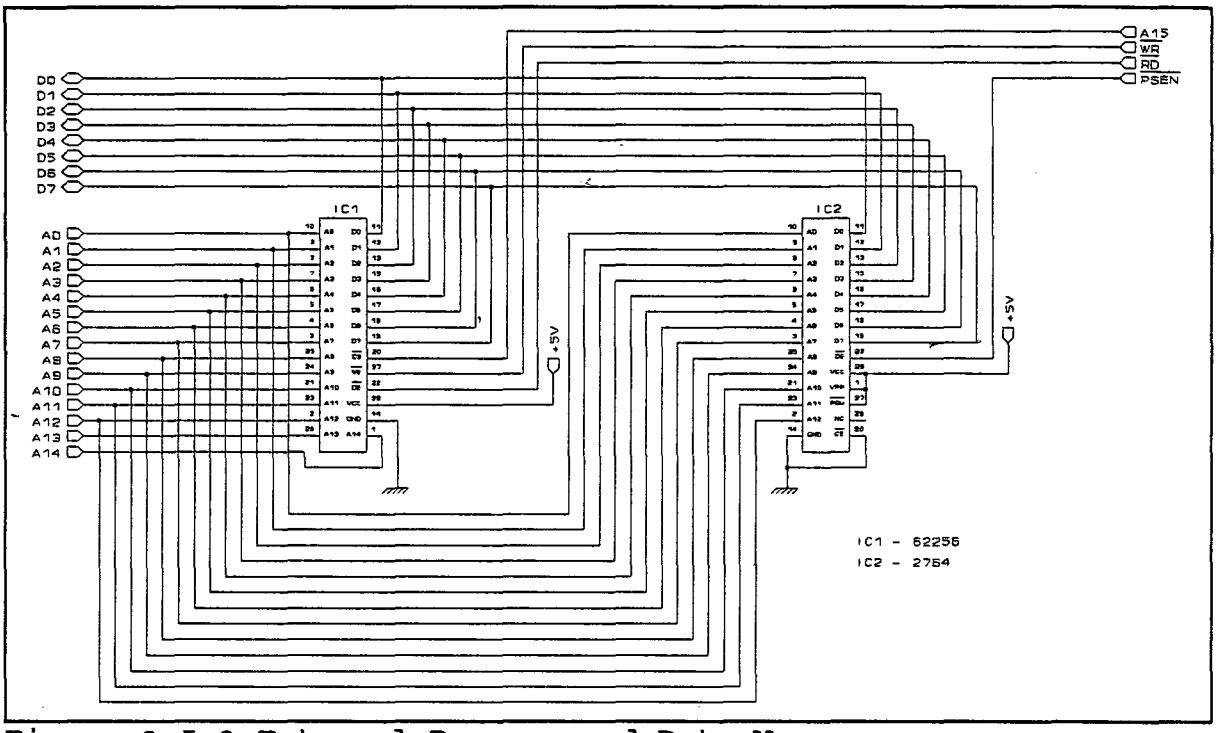

Figure 2.5.2 External Program and Data Memory

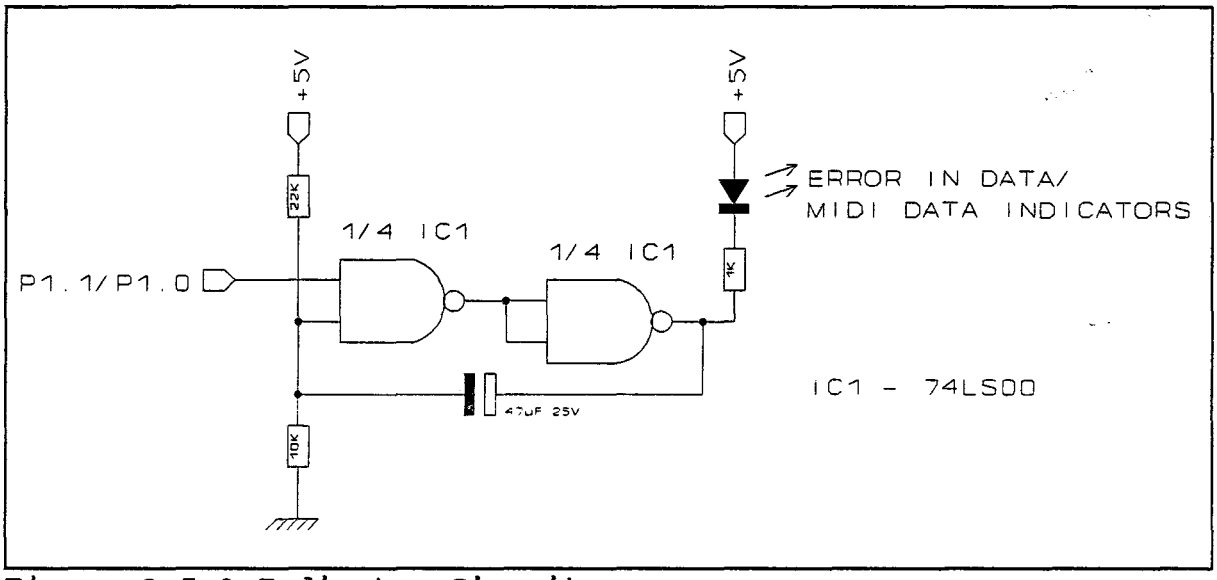

Figure 2.5.3 Indicator Circuitry

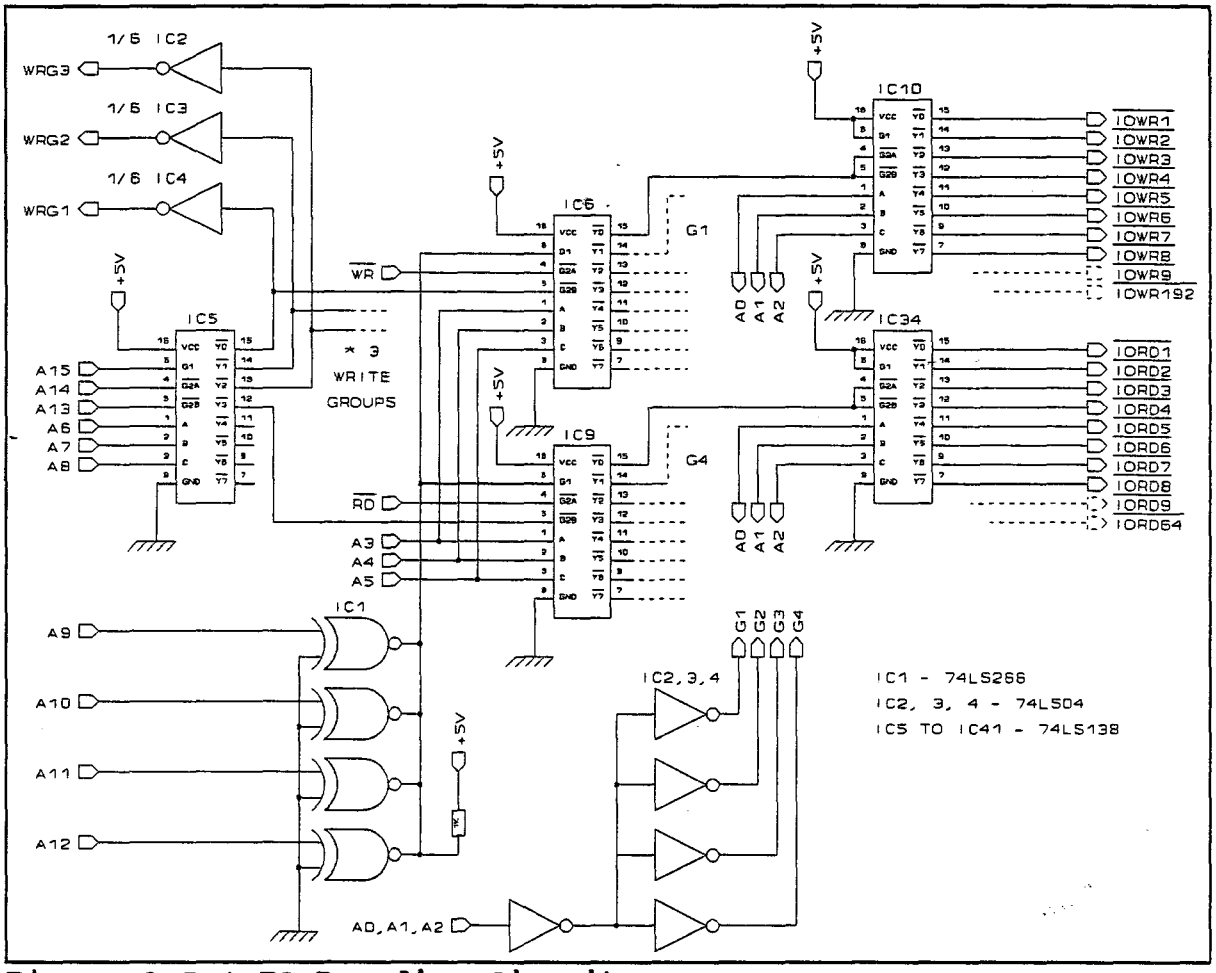

Figure 2.5.4 IO Decoding Circuitry

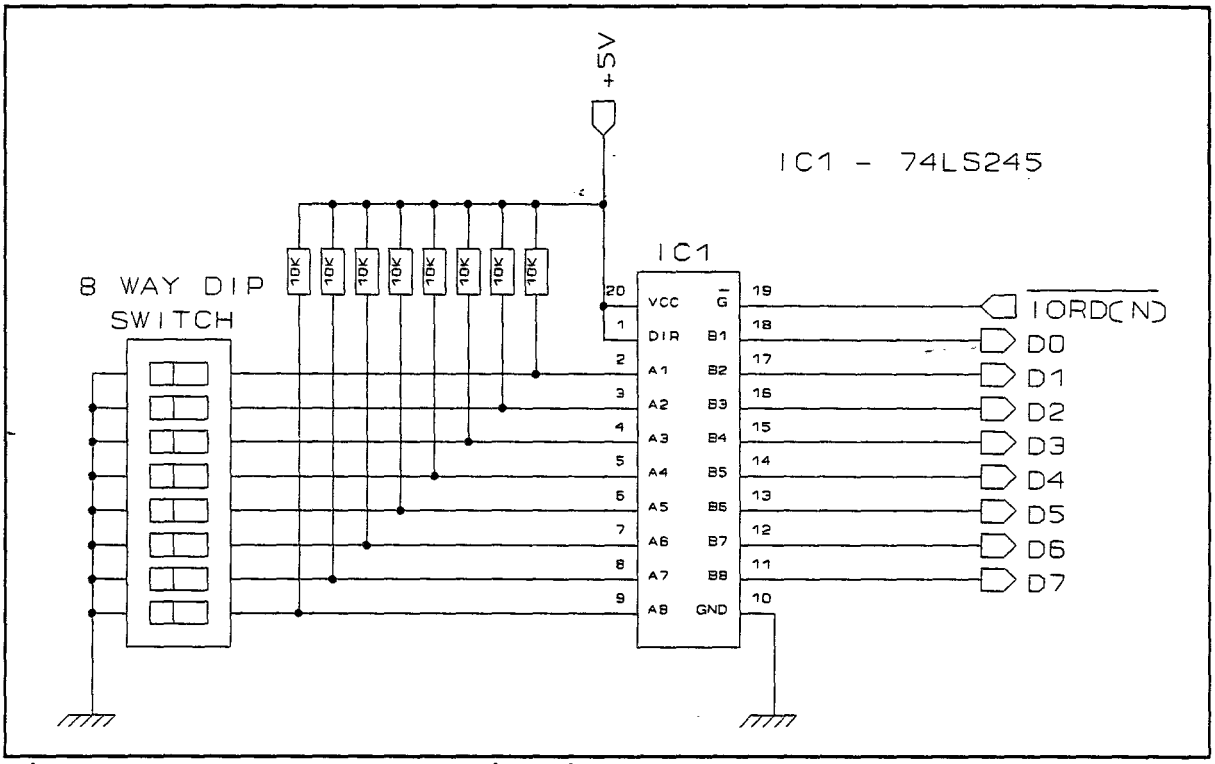

Figure 2.5.5 Data Entry Circuitry

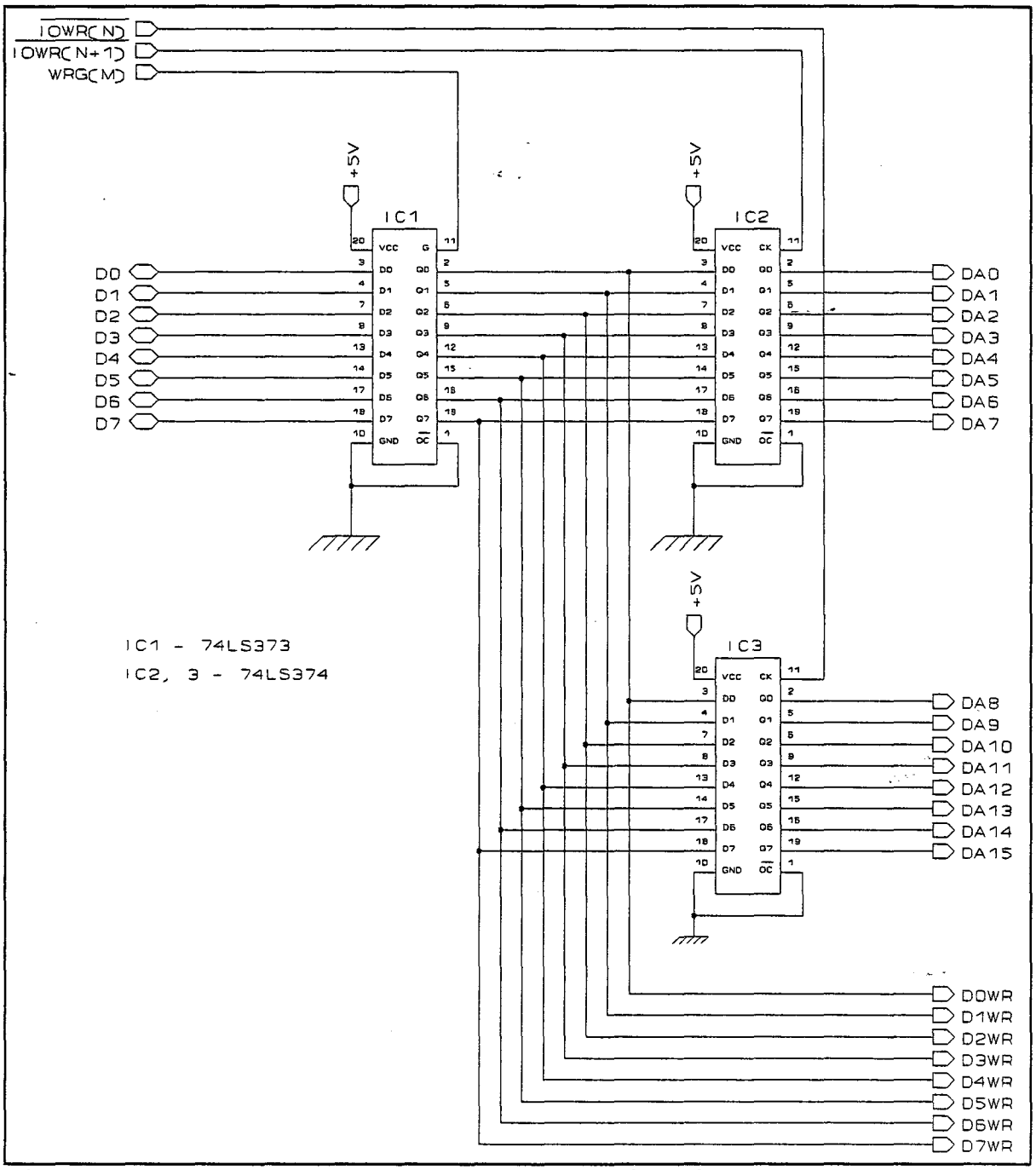

Figure 2.5.6 Output Buffer and Latch Circuitry

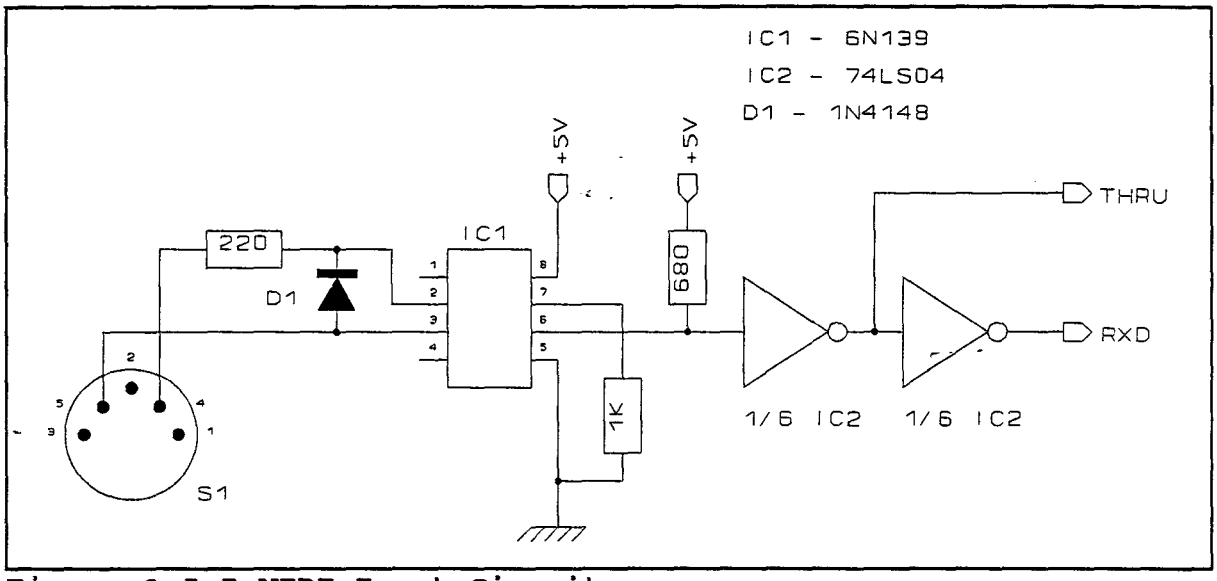

Figure 2.5.7 MIDI Input Circuitry

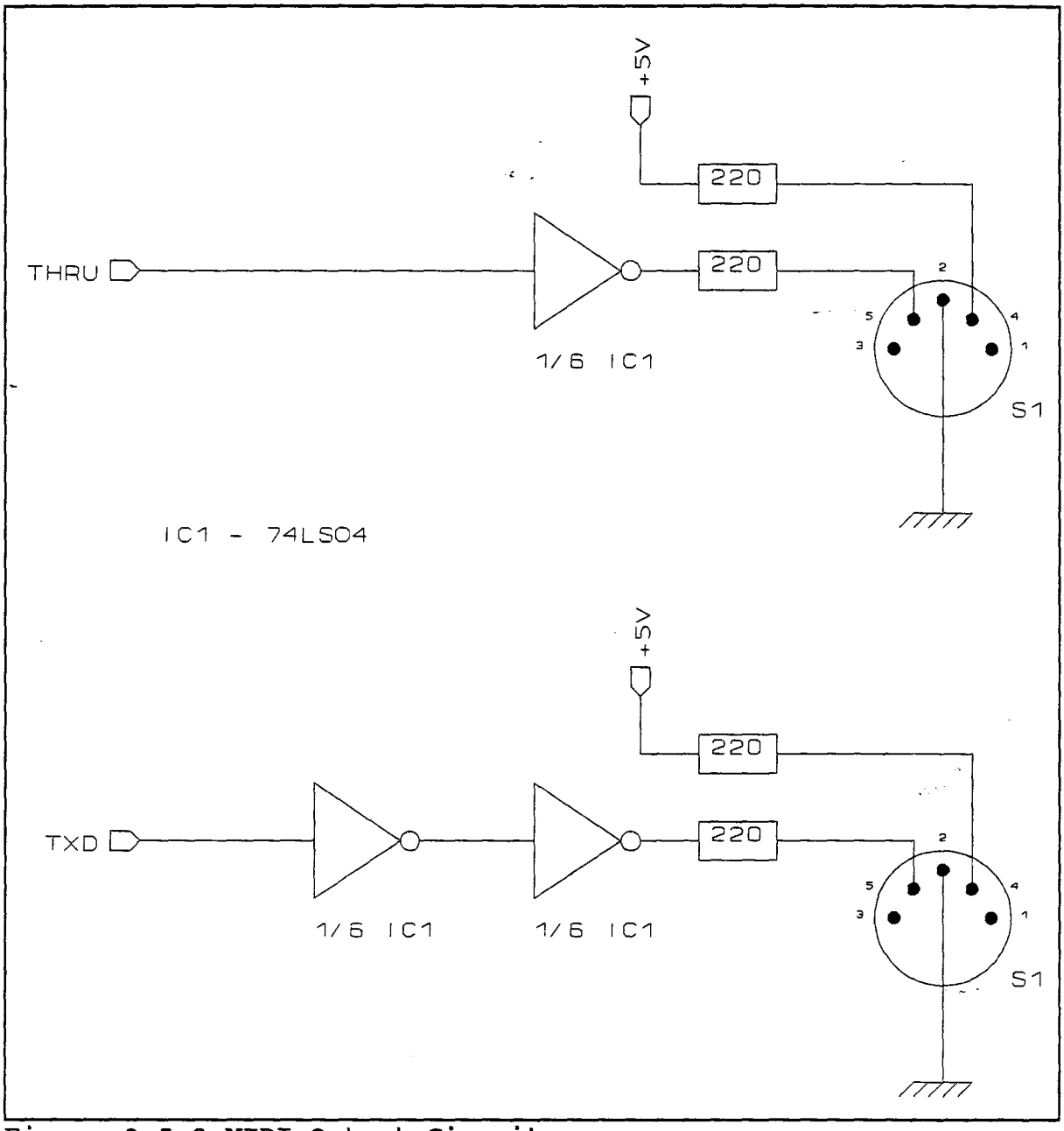

Figure 2.5.8 MIDI Output Circuitry

APPENDIX 3 - CIRCUIT BOARDS

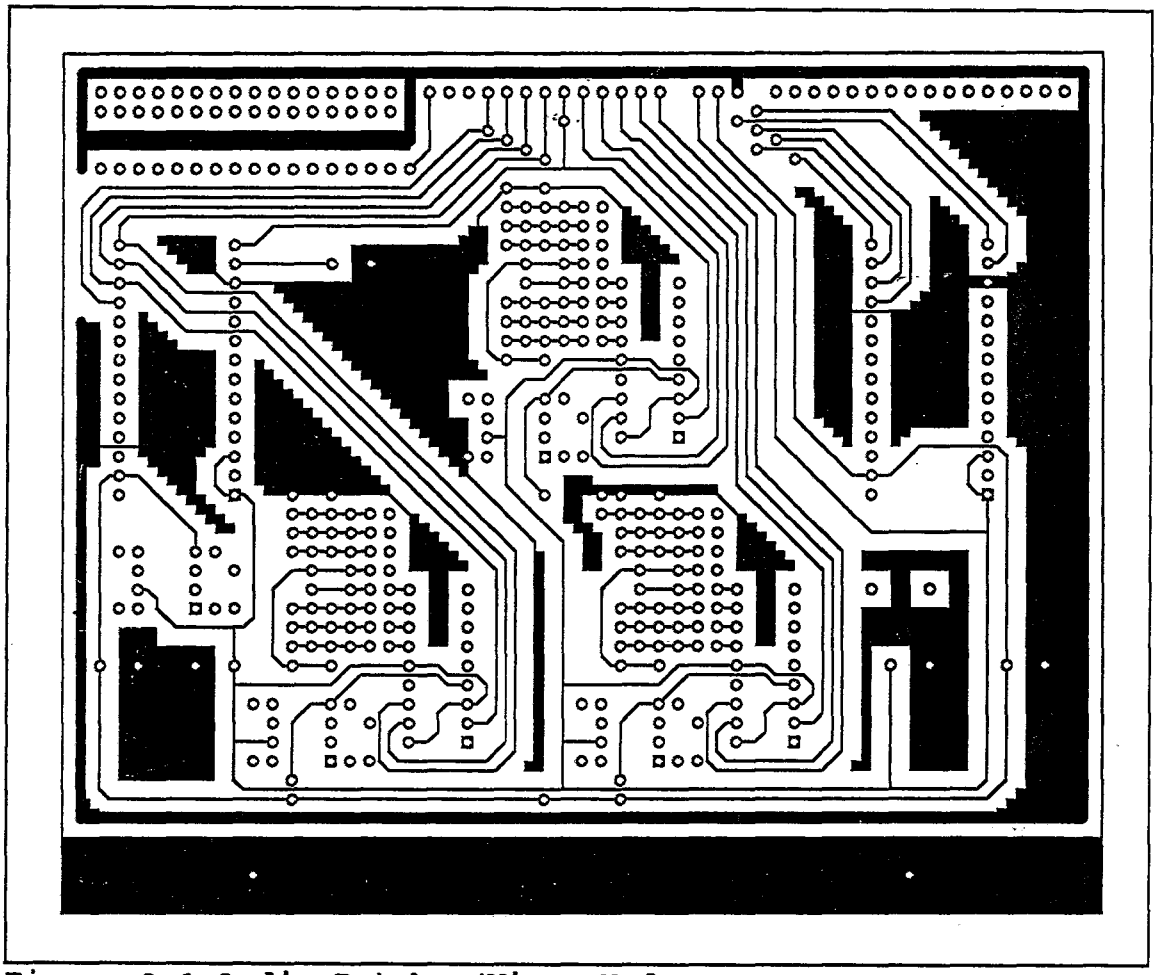

Figure 3.1 Audio Patcher/Mixer Node
Printed Circuit Board,
Component Side

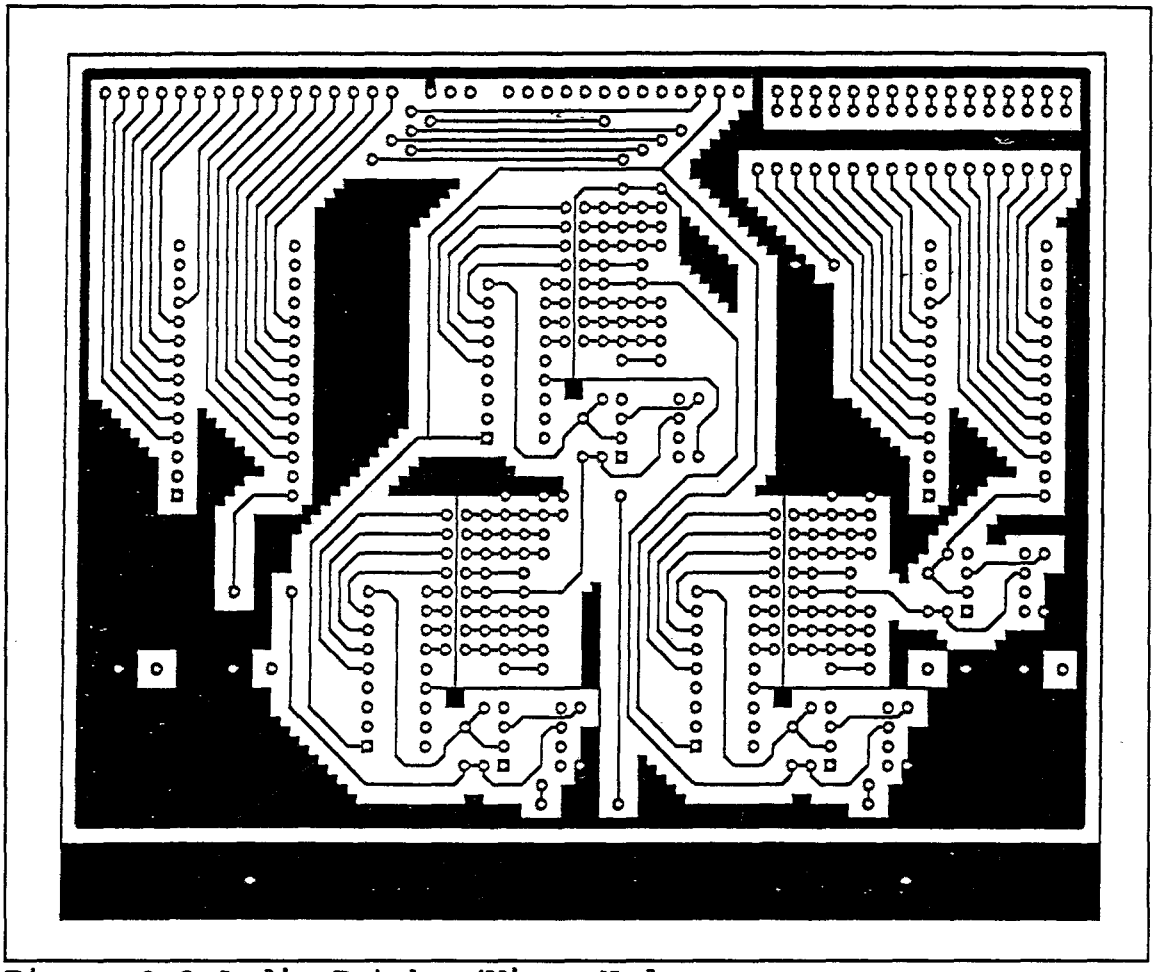

Figure 3.2 Audio Patcher/Mixer Node
Printed Circuit Board,
Solder Side

APPENDIX 4 - COMPONENT PART LISTS

TABLE OF CONTENTS

4.1 The Digitally Controlled Attenuator (DCA)

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

R1L1	120//3K3	R1L2	1K5//2K7	R1L3	10K//27K
R2L1	120//4K7	R2L2	1K2//6K8	R2L3	2K2//100K
R3L1	150//560	R3L2	1K2//8K2	R3L3	560//1K8
R4L1	120//33K	R4L2	1K5//3K3	R4L3	100//1K5
R5L1	220//270	R5L2	1K5//2K7	R5L3	22//680
R6L1	220//270	R6L2	1K//6K8	R6L3	8R2//12
R7L1	150//680	R7L2	1K5//1K5	R7L3	2R2//2R2
R8L1	10K//100K	R8L2	4K7//12K	R8L3	1//1//1

2 * 100K	1 * 47K	1 * 33K	1 * 27K
1 * 12K	28 * 10K	1 * 8K2	2 * 6K8
5 * 4K7	2 * 3K3	2 * 2K7	1 * 2K2
1 * 1K8	6 * 1K5	2 * 1K2	1 * 1K
2 * 680	2 * 560	2 * 270	2 * 220
2 * 150	3 * 120	1 * 100	1 * 22
1 * 12	1 * 8R2	2 * 2R2	3 * 1

Capacitors

3 * 150pF 50VDC Disc Ceramic
4 * 100nF 50VDC Mylar
4 * 10uF 25VDC Nonpolar Electrolytic

Semiconductors

3 * ADG528AKN CMOS Latched 8 Channel Analog Multiplexer (IC1,3,5)
[25 p7-49]
3 * LF351N BiFET Operational Amplifier (IC2,4,6)
[57 p2-38]

4.2 The Audio Processor Unit's Audio Processing Circuitry

4.2.1 Gain Control Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 100	3 * 10K	1 * 100K	1 * 330K

Capacitors

1 * 15pF 50VDC Disc Ceramic

1 * 330nF 50VDC Mylar

Semiconductors

1 * LF347N Quad BiFET Operational Amplifier (IC1)
[57 p2-14]

4.2.2 Equalisation Control Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 4K7 3 * 10K 7 * 12K 14 * 47K 1 * 56K

Capacitors

1 * 68pF 50VDC Disc Ceramic
3 * 680pF 50VDC Disc Ceramic
3 * 15nF 50VDC Mylar

Semiconductors

2 * LF347N Quad BiFET Operational Amplifier (IC1,3)
2 * LF351N BiFET Operational Amplifier (IC2,4)

4.3 The Audio Patcher/Mixer Unit's Audio Routing and Mixing Circuitry

4.3.1 The Line Input Buffer

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 100 1 * 10K 3 * 47K 6 * 100K 1 * 330K

Capacitors

3 * 15pF 50VDC Disc Ceramic
1 * 330nF 50VDC Mylar

Semiconductors

1 * LF351N BiFET Operational Amplifier (IC1)
1 * LF347N Quad BiFET Operational Amplifier (IC2)

4.3.2 The Node

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 10K 1 * 47K 17 * 100K

Capacitors

1 * 15pF 50VDC Disc Ceramic

Semiconductors

2 * ADG526AKN CMOS Latched 16 Channel Analog Multiplexer (IC1,3) [25 p7-41]
1 * LF351N BiFET Operational Amplifier (IC2)

4.3.3 The Line Output Buffer

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

2 * 10 1 * 22 2 * 4K7 5 * 10K 2 * 100K 1 * 330K

Capacitors

2 * 150pF 50VDC Disc Ceramic
1 * 330nF 50VDC Mylar

Semiconductors

1 * LF347N BiFET Operational Amplifier (IC1)
1 * BC550C PNP Silicon Transistor (T1)
1 * BC560C NPN Silicon Transistor (T2)
3 * 1N4148 Silicon Diodes (D1,2,3)

4.4 The MIDI Patch Unit's MIDI Routing Circuitry

4.4.1 Device to Workstation MIDI Router

Semiconductors

1 * 74LS174 (IC1) [58 p7-242]
1 * 74LS150 (IC2) [58 p7-147]
1 * 74LS51 (IC3) [58 p5-17]
1 * 74HC04 (IC4) [58 p5-7]

4.4.2 Workstation to Device MIDI Router

Semiconductors

1 * 74LS174 (IC1)
1 * 74LS151 (IC2) [58 p7-147]
1 * 74LS51 (IC3)
1 * 74HCO4 (IC4)

4.4.3 MIDI Input Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 220 1 * 680 1 * 1K

Semiconductors

1 * 6N139 (IC1) [53 p25]
1 * 74LS04 (IC2) [58 p5-7]
1 * 1N4148 Silicon Diodes (D1)

Miscellaneous

1 * 5 Pin DIN Female Chassis mount plug (S1)

4.4.4 MIDI Output Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

2 * 220

Semiconductors

1 * 74HC04 (IC1)

Miscellaneous

1 * 5 Pin DIN Female Chassis mount plug (S1)

4.5 The Microprocessor Control Unit (MCU)

4.5.1 The Microcontroller

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 10K

Capacitors

2 * 15pF 50VDC Disc Ceramic
1 * 4.7uF 25VDC Axial Electrolytic

Semiconductors

1 * 8031 12 MHz Microcontroller (IC1) [23 p6-1]
1 * 74LS373 (IC2) [58 p7-496]
1 * 1N4148 Silicon Diodes (D1)

Miscellaneous

1 * 12 MHz Crystal
1 * Push to Make chassis mounting push button

4.5.2 The External Program and Data Memory

1 * 62256-12 32KByte Static RAM (IC1) [59 p234]
1 * 27C64Q-20 8KByte CMOS EPROM (IC2) [59 p482]

4.5.3 Indicator Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

2 * 1K 2 * 10K 2 * 22K

Capacitors

2 * 47uF 25VDC Axial Electrolytic

Semiconductors

1 * 74LS00 (IC1) [58 p5-6]
1 * 5mm LED red (Error in Data Indicator)
1 * 5mm LED green (MIDI Data Indicator)

4.5.4 IO Decoding Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 1K

Semiconductors

1 * 74LS266 (IC1) [58 p5-65]
3 * 74LS04 (IC2,3,4)
37 * 74LS138 (IC5 - IC41) [58 p7-128]

4.5.5 Data Entry Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

8 * 10K

Semiconductors

1 * 74LS245 (IC1) [58 p7-359]

Miscellaneous

1 * 8 Way DIP Switch

4.5.6 Output Buffer and Latch Circuitry

Semiconductors

1 * 74LS373 (IC1)
2 * 74LS374 (IC2,3) [58 p7-496]

4.5.7 MIDI Input Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

1 * 220 1 * 680 1 * 1K

Semiconductors

1 * 6N139 (IC1)
1 * 74LS04 (IC2)
1 * 1N4148 Silicon Diodes (D1)

Miscellaneous

1 * 5 Pin DIN Female Chassis mount plug (S1)

4.5.8 MIDI Output Circuitry

Resistors/ohms (all 0.25W 5% Tolerance Carbon Resistors)

6 * 220

Semiconductors

1 * 74LS04 (IC1)

Miscellaneous

3 * 5 Pin DIN Female Chassis mount plug (S1)

APPENDIX 5 - MIDI SYSTEM EXCLUSIVE MESSAGE FORMAT

5.1 Packet Format Specification for Audio Processor Unit

```
audio process request = * A request to alter the audio
                          processing on specified channels on
                          a specified audio processor unit *
                      = sysex start + network device +
                        network device identifier +
                        0{audio process} + sysex end

sysex start           = * value : 11110000, type : binary *

network device        = * This is a device type identifier in
                          the network (this number is reserved
                          for research purposes [1]) *
                      = * value : 01111101, type : binary *

network device        = * network device number identifying a
identifier                particular unit *
                      = * values : 00000000 - 01111111, type :
                          binary *

audio process         = * Alter the specified process being
                          performed on a specified channel *
                      = audio process channel +
                        audio process type +
                        audio process level

audio process channel = * choice of sixteen channels
                          (occupies the same byte as the
                          audio process type) *
                      = * values : 00000000 - 00001111, type :
                          binary *

audio process type    = * (occupies the same byte as the
                          audio process channel) *
                      = [gain process ¦ bass process ¦
                         midrange process ¦ treble process]

gain process          = * value : 00010000, type : binary *

bass process          = * value : 00100000, type : binary *

midrange process      = * value : 00110000, type : binary *
```

```
treble process          = * value : 01000000, type : binary *

audio process level     = * See Appendix 6.1.5 and 6.1.6 *
                        = * values : 00000000 - 01111111, type :
                            binary *

sysex end (EOX)         = * specific value is 11110111 (binary)
                            [1 p16] *
                          * values : 10000000 - 11110111, type :
                            binary *
```

5.2 Packet Format Specification for Audio Patcher/Mixer Unit

```
audio patch request     = * A request to connect, disconnect and
                            alter mix levels between specified
                            inputs and specified outputs on a
                            specified Audio Patcher/Mixer Unit *
                        = sysex start + network device +
                          network device identifier +
                          0{audio patch} + sysex end

sysex start             = * value : 11110000, type : binary *

network device          = * This is a device type identifier in
                            the network (this number is reserved
                            for research purposes [1]) *
                        = * value : 01111101, type : binary *

network device          = * network device number identifying a
identifier                  particular unit *
                        = * values : 00000000 - 01111111, type :
                            binary *

audio patch             = * A connection, disconnection or a
                            change of mix level between a
                            specified input and a specified
                            output (two or three bytes long) *
                        = audio patch input number +
                          audio patch output number +
                          audio patch type

audio patch input       = * choice of sixteen inputs *
number                  = * values : 00000000 - 00001111, type :
                            binary *
```

```
audio patch output number    = * choice of sixteen outputs
                                 (occupies the same byte as part of
                                 the audio patch type) *
                             = * values : 00000000 - 00001111, type :
                                 binary *

audio patch type             = * audio patch type to perform *
                             = [connect patch + mix level¦
                                change level patch + mix level¦
                                disconnect patch]

connect patch                = * connect a specified input to a
                                 specified output and set the mix
                                 level (occupies the same byte as the
                                 audio patch output number) *
                             = * value : 00010000, type : binary *

change level patch           = * change mix level only
                                 (occupies the same byte as the
                                 audio patch output number) *
                             = * value : 00100000, type : binary *

disconnect patch             = * disconnect a specified input from
                                 a specified output and set
                                 the mix level to its minimum
                                 (occupies the same byte as the
                                 audio patch output number) *
                             = * value : 01000000, type : binary *

mix level                    = * See Appendix 6.2.5 *
                             = * values : 00000000 - 01111111, type :
                                 binary *

sysex end (EOX)              = * specific value is 11110111 (binary)
                                 [1 p16] *
                               * values : 10000000 - 11110111, type :
                                 binary *
```

5.3 Packet Format Specification for MIDI Patch Unit

```
MIDI patch request      = * A request to connect specified MIDI
                            devices to specified workstations on
                            a specified MIDI Patch Unit *
                        = sysex start + network device +
                          network device identifier +
                          0{MIDI patch} + sysex end

sysex start             = * value : 11110000, type : binary *

network device          = * This is a device type identifier in
                            the network (this number is reserved
                            for research purposes [1]) *
                        = * value : 01111101, type : binary *

network device          = * network device number identifying a
identifier                  particular unit *
                        = * values : 00000000 - 01111111, type :
                            binary *

MIDI patch              = * A connection between a specified
                            MIDI device and a specified
                            workstation (two bytes long) *
                        = workstation number + MIDI direction +
                          device number

workstation number      = * choice of eight workstations
                            (occupies the same byte as the
                             MIDI direction) *
                        = * [0000000 - 00000111 ¦ external],
                             type : binary *

external                = * select MIDI input from external
                            patch bay *
                        = * value : 00010000, type : binary *

MIDI direction          = * directional flow of MIDI information
                            (occupies the same byte as the
                            workstation number) *
                        = [workstation to device ¦ device to
                          workstation]

workstation to device = * value : 00100000, type : binary *
```

```
device to workstation = * value : 01000000, type : binary *

device number         = * choice of sixteen MIDI devices *
                      = * values : [00000000 - 00001111 |
                          external], type : binary *

sysex end (EOX)       = * specific value is 11110111 (binary)
                          [1 p16] *
                        * values : 10000000 - 11110111, type :
                          binary *
```

APPENDIX 6 - SOFTWARE DESIGN

TABLE OF CONTENTS

6.1 The Audio Processor Unit

6.1.1 Transformation Schema

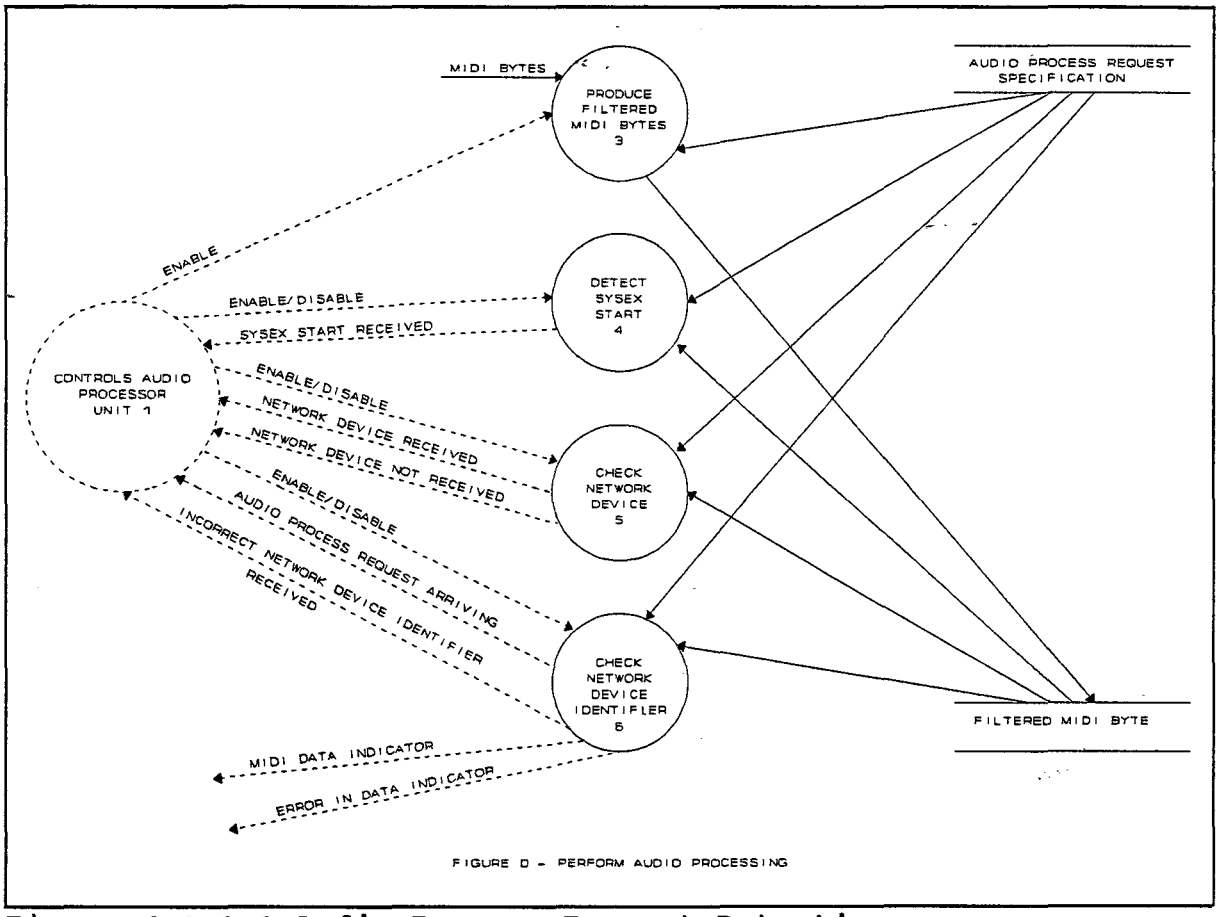

Figure 6.1.1.1 Audio Process Request Detection

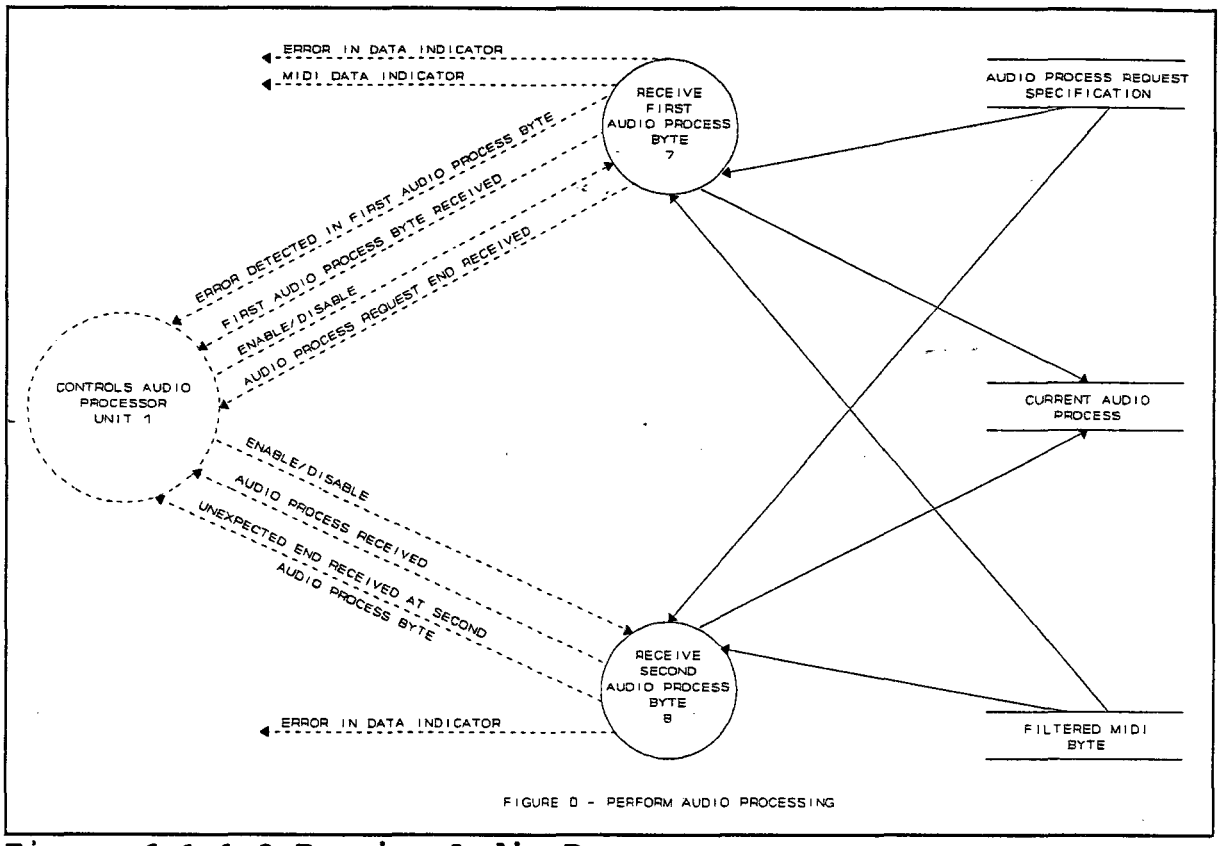

Figure 6.1.1.2 Receive Audio Process

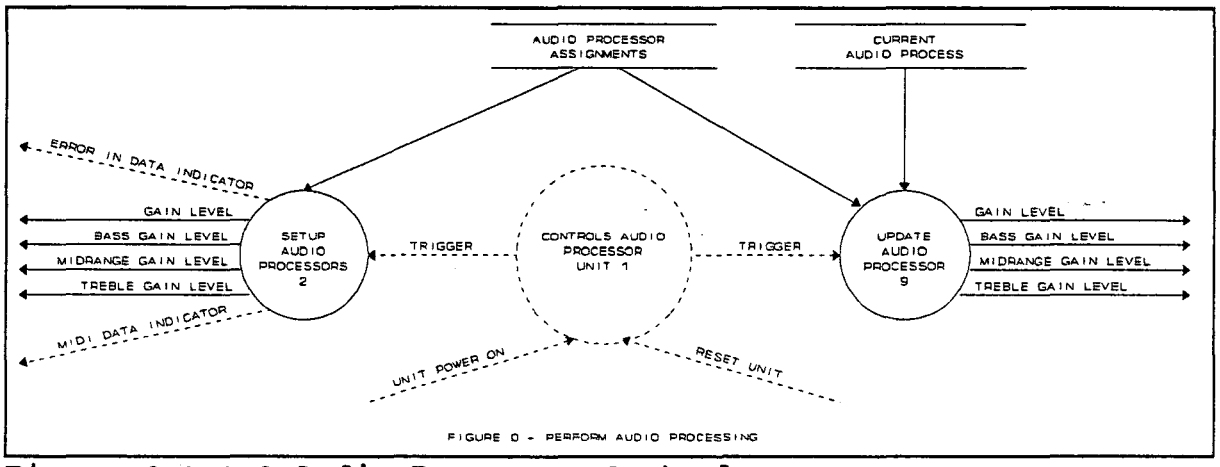

Figure 6.1.1.3 Audio Processor Control

6.1.2 State Transition Diagram

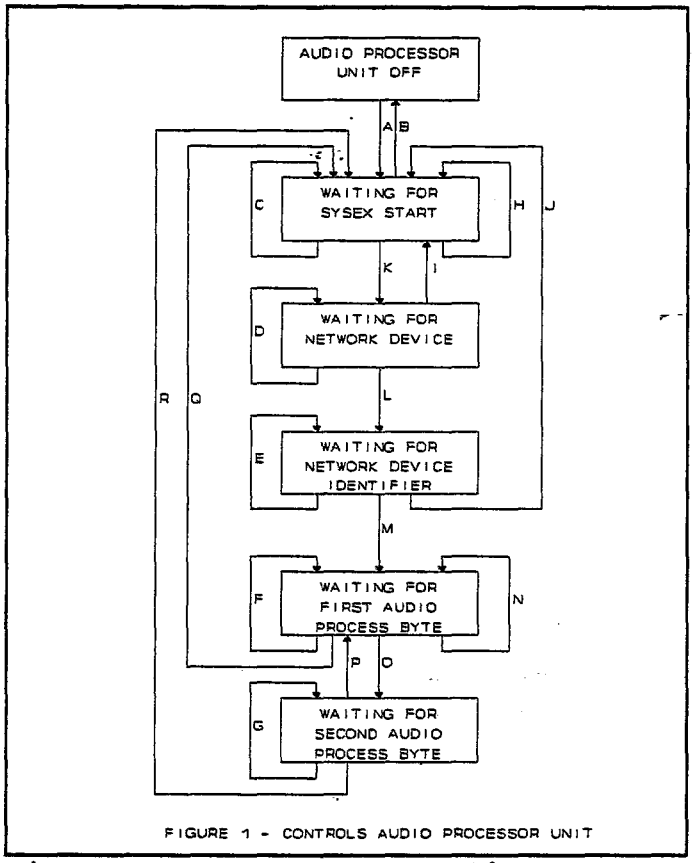

Figure 6.1.2 Controls Audio Processor Unit

1 A) Studio Manager switches unit power on
In order do:
    Trigger "Setup Audio Processors"
    Enable "Produce Filtered MIDI Bytes"
    Enable "Detect Sysex Start"
1 B) Studio Manager switches unit power off
1 C) - G) MIDI Real Time Message arrives
1 H) Studio Manager resets unit
In order do:
    Trigger "Setup Audio Processors"
    Enable "Produce Filtered MIDI Bytes"
    Enable "Detect Sysex Start"
1 I) Network Device not received
In order do:
    Disable "Check Network Device"
    Enable "Detect Sysex Start"
1 J) Incorrect Network Device Identifier received
In order do:
    Disable "Check Network Device Identifier"
    Enable "Detect Sysex Start"

1 K) Sysex Start received
In order do:
    Disable "Detect Sysex Start"
    Enable "Check Network Device"
1 L) Network Device received
In order do:
    Disable "Check Network Device"
    Enable "Check Network Device Identifier"
1 M) Audio Process Request arriving
In order do:
    Disable "Check Network Device Identifier"
    Enable "Receive First Audio Process Byte"
1 N) Error detected in First Audio Process Byte
1 O) First Audio Process Byte received
In order do:
    Disable "Receive First Audio Process Byte"
    Enable "Receive Second Audio Process Byte"
1 P) Audio Process received
In order do:
    Disable "Receive Second Audio Process Byte"
    Trigger "Update Audio Processor"
    Enable "Receive First Audio Process Byte"
1 Q) Audio Process Request End received
In order do:
    Disable "Receive First Audio Process Byte"
    Enable "Receive Sysex Start"
1 R) Unexpected End received at Second Audio Process Byte
In order do:
    Disable "Receive Second Audio Process Byte"
    Enable "Receive Sysex Start"

6.1.3 Transformation Specifications

2 SETUP AUDIO PROCESSORS
Precondition 1
    None
Postcondition 1
    MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = ON
and GAIN LEVEL = INITIAL GAIN LEVEL
    in GAIN LEVEL MAP
    produced for GAIN PROCESS ADDRESS
and BASS GAIN LEVEL = INITIAL EQUALISATION LEVEL
    in EQUALISATION LEVEL MAP
    produced for BASS PROCESS ADDRESS

```
and   MIDRANGE GAIN LEVEL = INITIAL EQUALISATION LEVEL
      in EQUALISATION LEVEL MAP
      produced for MIDRANGE PROCESS ADDRESS
and   TREBLE GAIN LEVEL = INITIAL EQUALISATION LEVEL
      in EQUALISATION LEVEL MAP
      produced for TREBLE PROCESS ADDRESS
      from AUDIO CHANNEL PROCESSOR PORT ADDRESSES
      in AUDIO PROCESSOR ASSIGNMENTS
      for all AUDIO CHANNEL IDENTIFIERs
and   MIDI DATA INDICATOR = OFF
and   ERROR IN DATA INDICATOR = OFF

3 PRODUCE FILTERED MIDI BYTES
Precondition 1
      MIDI BYTE occurs
and   MIDI BYTE is less than MIDI REAL TIME MESSAGE
Postcondition 1
      FILTERED MIDI BYTE is produced
Precondition 2
      MIDI BYTE occurs
and   MIDI BYTE is greater or equal to MIDI REAL TIME MESSAGE
Postcondition 2
      None

4 DETECT SYSEX START
Precondition 1
      FILTERED MIDI BYTE occurs
and   FILTERED MIDI BYTE is equal to SYSEX START
Postcondition 1
      SYSEX START RECEIVED occurs
Precondition 2
      FILTERED MIDI BYTE occurs
and   FILTERED MIDI BYTE is not equal to SYSEX START
Postcondition 2
      None

5 CHECK NETWORK DEVICE
Precondition 1
      FILTERED MIDI BYTE occurs
and   FILTERED MIDI BYTE is equal to NETWORK DEVICE
Postcondition 1
      NETWORK DEVICE RECEIVED occurs
Precondition 2
      FILTERED MIDI BYTE occurs
and   FILTERED MIDI BYTE is not equal to NETWORK DEVICE
```

Postcondition 2
NETWORK DEVICE NOT RECEIVED occurs

6 CHECK NETWORK DEVICE IDENTIFIER
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is equal to NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 1
MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = OFF
and AUDIO PROCESS REQUEST ARRIVING occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is not equal to
NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 2
INCORRECT NETWORK DEVICE IDENTIFIER RECEIVED occurs

7 RECEIVE FIRST AUDIO PROCESS BYTE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PROCESS TYPE MASK
is equal to GAIN PROCESS
Postcondition 1
CURRENT AUDIO PROCESS = GAIN PROCESS
CURRENT AUDIO PROCESS CHANNEL =
FILTERED MIDI BYTE & AUDIO PROCESS CHANNEL MASK
and FIRST AUDIO PROCESS BYTE RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PROCESS TYPE MASK
is equal to BASS PROCESS
Postcondition 2
CURRENT AUDIO PROCESS = BASS PROCESS
CURRENT AUDIO PROCESS CHANNEL =
FILTERED MIDI BYTE & AUDIO PROCESS CHANNEL MASK
and FIRST AUDIO PROCESS BYTE RECEIVED occurs
Precondition 3
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PROCESS TYPE MASK

```
     is equal to MIDRANGE PROCESS
Postcondition 3
     CURRENT AUDIO PROCESS = MIDRANGE PROCESS
     CURRENT AUDIO PROCESS CHANNEL =
     FILTERED MIDI BYTE & AUDIO PROCESS CHANNEL MASK
and  FIRST AUDIO PROCESS BYTE RECEIVED occurs
Precondition 4
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & AUDIO PROCESS TYPE MASK
     is equal to TREBLE PROCESS
Postcondition 4
     CURRENT AUDIO PROCESS = TREBLE PROCESS
     CURRENT AUDIO PROCESS CHANNEL =
     FILTERED MIDI BYTE & AUDIO PROCESS CHANNEL MASK
and  FIRST AUDIO PROCESS BYTE RECEIVED occurs
Precondition 5
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & AUDIO PROCESS TYPE MASK
     is not equal to GAIN PROCESS
and  is not equal to BASS PROCESS
and  is not equal to MIDRANGE PROCESS
and  is not equal to TREBLE PROCESS
Postcondition 5
     ERROR IN DATA INDICATOR = ON
and  ERROR DETECTED IN FIRST AUDIO PROCESS BYTE occurs
Precondition 6
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 6
     MIDI DATA INDICATOR = OFF
and  AUDIO PROCESS REQUEST END RECEIVED occurs

8 RECEIVE SECOND AUDIO PROCESS BYTE
Precondition 1
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
Postcondition 1
     CURRENT AUDIO PROCESS LEVEL = FILTERED MIDI BYTE
and  AUDIO PROCESS RECEIVED occurs
Precondition 2
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is greater than or equal to SYSEX END
```

Postcondition 2

MIDI DATA INDICATOR = OFF
and ERROR IN DATA INDICATOR = ON
and UNEXPECTED END RECEIVED AT SECOND AUDIO PROCESS BYTE
occurs

9 UPDATE AUDIO PROCESSOR

Precondition 1

CURRENT AUDIO PROCESS TYPE contains GAIN PROCESS

Postcondition 1

GAIN LEVEL = CURRENT AUDIO PROCESS LEVEL
in GAIN LEVEL MAP
produced for GAIN PROCESS ADDRESS
from AUDIO CHANNEL PROCESSOR PORT ADDRESSES
in AUDIO PROCESSOR ASSIGNMENTS
for AUDIO CHANNEL IDENTIFIER
corresponding to CURRENT AUDIO PROCESS CHANNEL

Precondition 2

CURRENT AUDIO PROCESS TYPE contains BASS PROCESS

Postcondition 2

BASS GAIN LEVEL = CURRENT AUDIO PROCESS LEVEL
in EQUALISATION LEVEL MAP
produced for BASS PROCESS ADDRESS
from AUDIO CHANNEL PROCESSOR PORT ADDRESSES
in AUDIO PROCESSOR ASSIGNMENTS
for AUDIO CHANNEL IDENTIFIER
corresponding to CURRENT AUDIO PROCESS CHANNEL

Precondition 3

CURRENT AUDIO PROCESS TYPE contains MIDRANGE PROCESS

Postcondition 3

MIDRANGE GAIN LEVEL = CURRENT AUDIO PROCESS LEVEL
in EQUALISATION LEVEL MAP
produced for MIDRANGE PROCESS ADDRESS
from AUDIO CHANNEL PROCESSOR PORT ADDRESSES
in AUDIO PROCESSOR ASSIGNMENTS
for AUDIO CHANNEL IDENTIFIER
corresponding to CURRENT AUDIO PROCESS CHANNEL

Precondition 4

CURRENT AUDIO PROCESS TYPE contains TREBLE PROCESS

Postcondition 4

TREBLE GAIN LEVEL = CURRENT AUDIO PROCESS LEVEL
in EQUALISATION LEVEL MAP
produced for TREBLE PROCESS ADDRESS
from AUDIO CHANNEL PROCESSOR PORT ADDRESSES
in AUDIO PROCESSOR ASSIGNMENTS

for AUDIO CHANNEL IDENTIFIER
corresponding to CURRENT AUDIO PROCESS CHANNEL

6.1.4 Data Dictionary

```
audio channel identifier = * choice of 16 channels *
                           * values : 00000000 - 00001111, type :
                             binary *

audio channel processor port addresses = @audio channel identifier +
                                         gain process address +
                                         bass process address +
                                         midrange process address +
                                         treble process address
                                         * See Appendix 2.2.3 *

audio process channel mask = * occupies the lower four bits *
                             * value : 00001111, type : binary *

audio process received = **

audio process request arriving = * signal indicating audio processes
                                   arriving for this unit *

audio process request end = **

audio process request specification = * specification of information in
                                        an audio process request *
                                      = sysex start + network device +
                                        network device identifier address +
                                        MIDI real time messages + sysex end +
                                        gain process + bass process +
                                        midrange process + treble process +
                                        audio process channel mask +
                                        audio process type mask

audio process type mask = * occupies the upper four bits *
                          * value : 11110000, type : binary *

audio processor assignments = initial gain level +
                              initial equalisation level +
                              equalisation level map +
                              gain level map +
                              1{audio channel processor port
                              addresses}16
```

```
bass gain level        = * +12.6 - -12.8 dBs *
                       = * values : 00000000 00000000 -
                           00000001 11111111, type : binary *

bass process           = * value : 00100000, type : binary *

bass process address   = * the port address for the bass
                           control of a particular channel *
                       = * values ; 10000000 00000000 -
                           10000000 11111111, type binary *

current audio process  = current audio process channel +
                         current audio process type +
                         current audio process level

current audio process  = * choice of sixteen inputs *
channel                = * values : 00000000 - 00001111, type :
                           binary *

current audio process  = * choice of 128 levels *
level                  = * values : 00000000 - 01111111, type :
                           binary *

current audio process  = [gain process ¦ bass process ¦
type                      midrange process ¦ treble process]

equalisation level     = * +12.6 - -12.8 dBs *
                       = * values : 00000000 00000000 -
                           00000001 11111111, type : binary *

equalisation level     = * process level to equalisation level
map                        map *
                       = 1{equalisation level}128
                         * See Appendix 6.1.6 *

error detected in first audio process byte = **

error in data          = * indicates the occurrence of an error
indicator                  state within the unit *
                       = * values : [on ¦ off] *

filtered MIDI byte     = * contains no MIDI real time messages *
                         * values : 00000000 - 11110111, type :
                           binary *
```

```
first audio patch byte received = **

gain level              = * +20.0 - -30.8 dBs *
                        = * values : 00000000 00000000 -
                            00000001 11111111, type : binary *

gain level map          = * process level to gain level map *
                        = 1{gain level}128
                          * See Appendix 6.1.5 *

gain process            = * value : 00010000, type : binary *

gain process address    = * the port address for the gain
                            control of a particular channel *
                        = * values : 10000000 00000000 -
                            10000000 11111111, type : binary *

incorrect network device identifier received = **

initial gain level      = * index into gain level map giving
                            unity gain *
                        = * value : 77, type : byte *

initial equalisation    = * index into equalisation level map
level                       giving flat frequency response *
                        = * value : 64, type : byte *

midrange gain level     = * +12.6 - -12.8 dBs *
                        = * values : 00000000 00000000 -
                            00000001 11111111, type : binary *

midrange process        = * value : 00110000, type : binary *

midrange process        = * the port address for the midrange
address                     control of a particular channel *
                        = * values : 10000000 00000000 -
                            10000000 11111111, type binary *

MIDI byte               = * values : 00000000 - 11111111, type :
                            binary *

MIDI bytes              = 1{MIDI byte}

MIDI data indicator     = * indicates the receival of an audio
                            process request via MIDI *
```

```
                        = * values : [on | off] *

MIDI real time messages = * value : 11111000, type : binary *

network device          = * This is a device type identifier in
                            the network *
                        = * value : 01111101, type : binary *

network device          = * network device number identifying a
identifier                  particular unit *
                        = * values : 00000000 - 01111111
                            type : binary *

network device          = * the port address to obtain the
identifier address          network device identifier *
                        = * value : 10000000 11000001, type :
                            binary *

network device not received = **

network device received = **

reset unit      = * signal to initiate unit reinitialisation *

sysex end               = * value : 10000000, type : binary *

sysex start             = * value : 11110000, type : binary *

sysex start received    = **

treble gain level       = * +12.6 - -12.8 dBs *
                        = * values : 00000000 00000000 -
                            00000001 11111111, type : binary *

treble process          = * value : 01000000, type : binary *

treble process          = * the port address for the treble
address                     control of a particular channel *
                        = * values : 10000000 00000000 -
                            10000000 11111111, type binary *

unexpected end received at second audio process byte = **

unit power on   = * signal to initiate unit initialisation *
```

6.1.5 Gain Level Map (binary form for gain controls)

VALUE	BINARY	VALUE	BINARY	VALUE	BINARY	VALUE	BINARY
127	111111111	95	110111111	63	101111111	31	100111111
126	111111101	94	110111101	62	101111101	30	100111101
125	111111011	93	110111011	61	101111011	29	100111011
124	111111001	92	110111001	60	101111001	28	100111001
123	111110111	91	110110111	59	101110111	27	100110110
122	111110101	90	110110101	58	101110101	26	100110101
121	111110011	89	110110011	57	101110011	25	100110011
120	111110001	88	110110001	56	101110001	24	100110001
119	111101111	87	110101111	55	101101111	23	100101111
118	111101101	86	110101101	54	101101101	22	100101101
117	111101011	85	110101011	53	101101011	21	100101011
116	111101001	84	110101001	52	101101001	20	100101001
115	111100111	83	110100111	51	101100111	19	100100111
114	111100101	82	110100101	50	101100101	18	100100101
113	111100011	81	110100011	49	101100011	17	100100011
112	111100001	80	110100001	48	101100001	16	100100001
111	111011111	79	110011111	47	101011111	15	100011111
110	111011101	78	110011101	46	101011101	14	100011101
109	111011011	77	110011011	45	101011011	13	100011011
108	111011001	76	110011001	44	101011001	12	100011001
107	111010111	75	110010111	43	101010111	11	100010111
106	111010101	74	110010101	42	101010101	10	100010101
105	111010011	73	110010011	41	101010011	9	100010011
104	111010001	72	110010001	40	101010001	8	100010001
103	111001111	71	110001111	39	101001111	7	100001111
102	111001101	70	110001101	38	101001101	6	100001101
101	111001011	69	110001011	37	101001011	5	100001011
100	111001001	68	110001001	36	101001001	4	100001001
99	111000111	67	110000111	35	101000111	3	100000111
98	111000101	66	110000101	34	101000101	2	100000101
97	111000011	65	110000011	33	101000011	1	100000011
96	111000001	64	110000001	32	101000001	0	100000001

6.1.6 Equalisation Level Map (binary form for equalisation controls)

VALUE	BINARY	VALUE	BINARY	VALUE	BINARY	VALUE	BINARY
127	111111111	95	111011111	63	110111111	31	110011111
126	111111110	94	111011110	62	110111110	30	110011110
125	111111101	93	111011101	61	110111101	29	110011101
124	111111100	92	111011100	60	110111100	28	110011100
123	111111011	91	111011011	59	110111011	27	110011011
122	111111010	90	111011010	58	110111010	26	110011010
121	111111001	89	111011001	57	110111001	25	110011001
120	111111000	88	111011000	56	110111000	24	110011000
119	111110111	87	111010111	55	110110111	23	110010111
118	111110110	86	111010110	54	110110110	22	110010110
117	111110101	85	111010101	53	110110101	21	110010101
116	111110100	84	111010100	52	110110100	20	110010100
115	111110011	83	111010011	51	110110011	19	110010011
114	111110010	82	111010010	50	110110010	18	110010010
113	111110001	81	111010001	49	110110001	17	110010001
112	111110000	80	111010000	48	110110000	16	110010000
111	111101111	79	111001111	47	110101111	15	110001111
110	111101110	78	111001110	46	110101110	14	110001110
109	111101101	77	111001101	45	110101101	13	110001101
108	111101100	76	111001100	44	110101100	12	110001100
107	111101011	75	111001011	43	110101011	11	110001011
106	111101010	74	111001010	42	110101010	10	110001010
105	111101001	73	111001001	41	110101001	9	110001001
104	111101000	72	111001000	40	110101000	8	110001000
103	111100111	71	111000111	39	110100111	7	110000111
102	111100110	70	111000110	38	110100110	6	110000110
101	111100101	69	111000101	37	110100101	5	110000101
100	111100100	68	111000100	36	110100100	4	110000100
99	111100011	67	111000011	35	110100011	3	110000011
98	111100010	66	111000010	34	110100010	2	110000010
97	111100001	65	111000001	33	110100001	1	110000001
96	111100000	64	111000000	32	110100000	0	110000000

6.1.7 Structure Chart

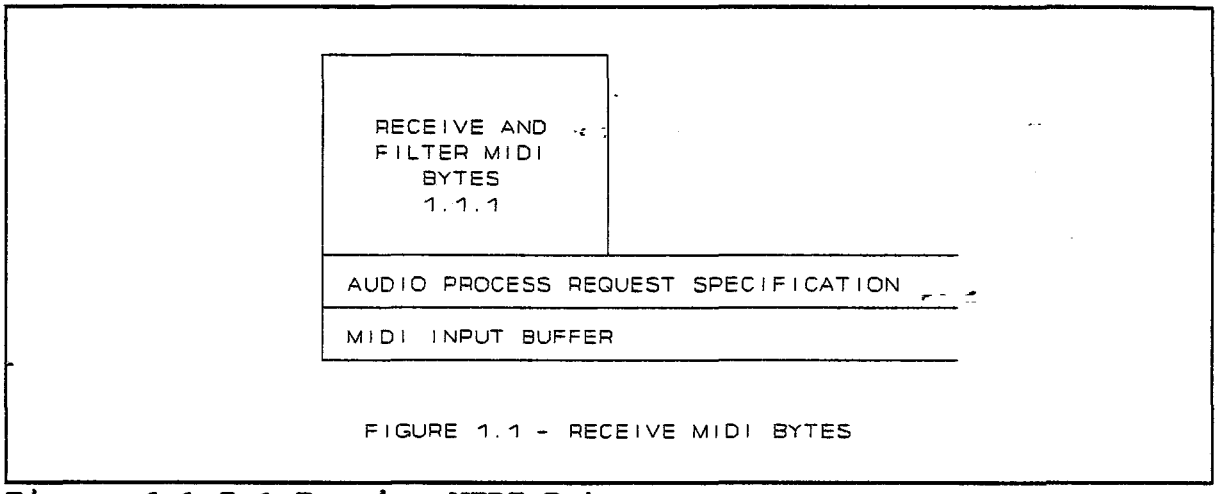

Figure 6.1.7.1 Receive MIDI Bytes

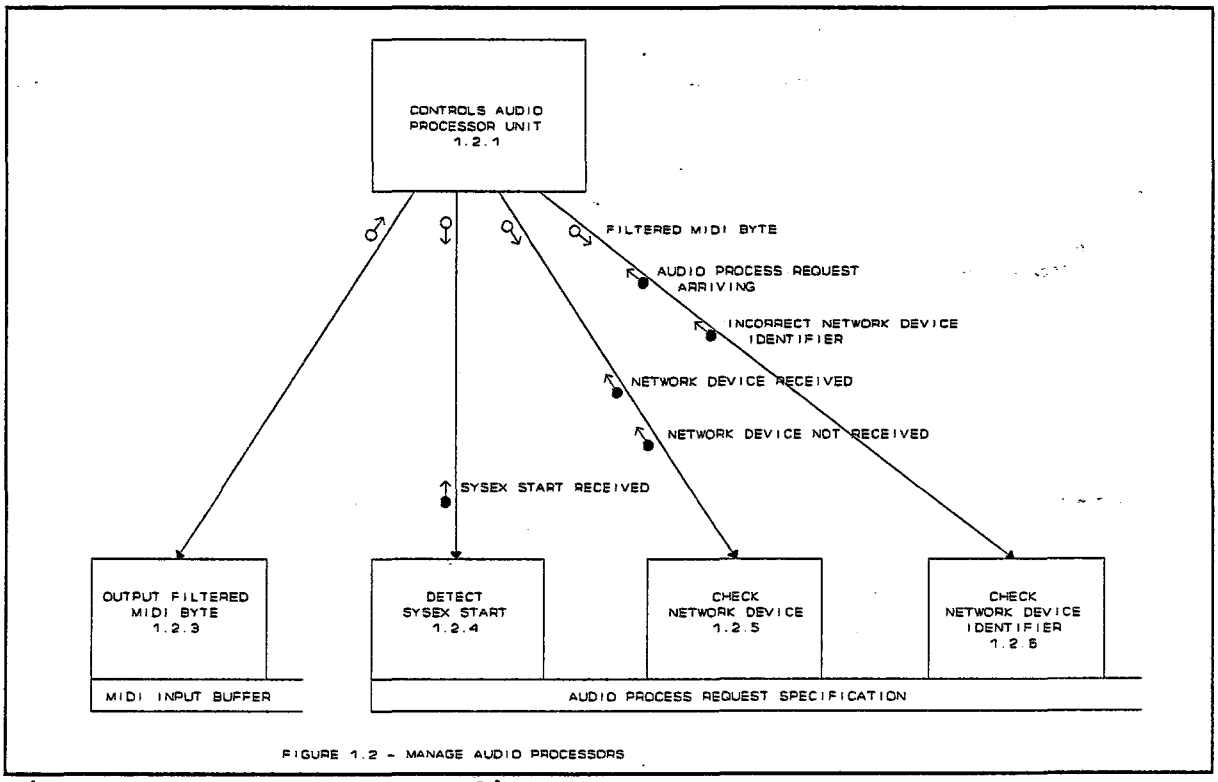

Figure 6.1.7.2 Manage Audio Processors

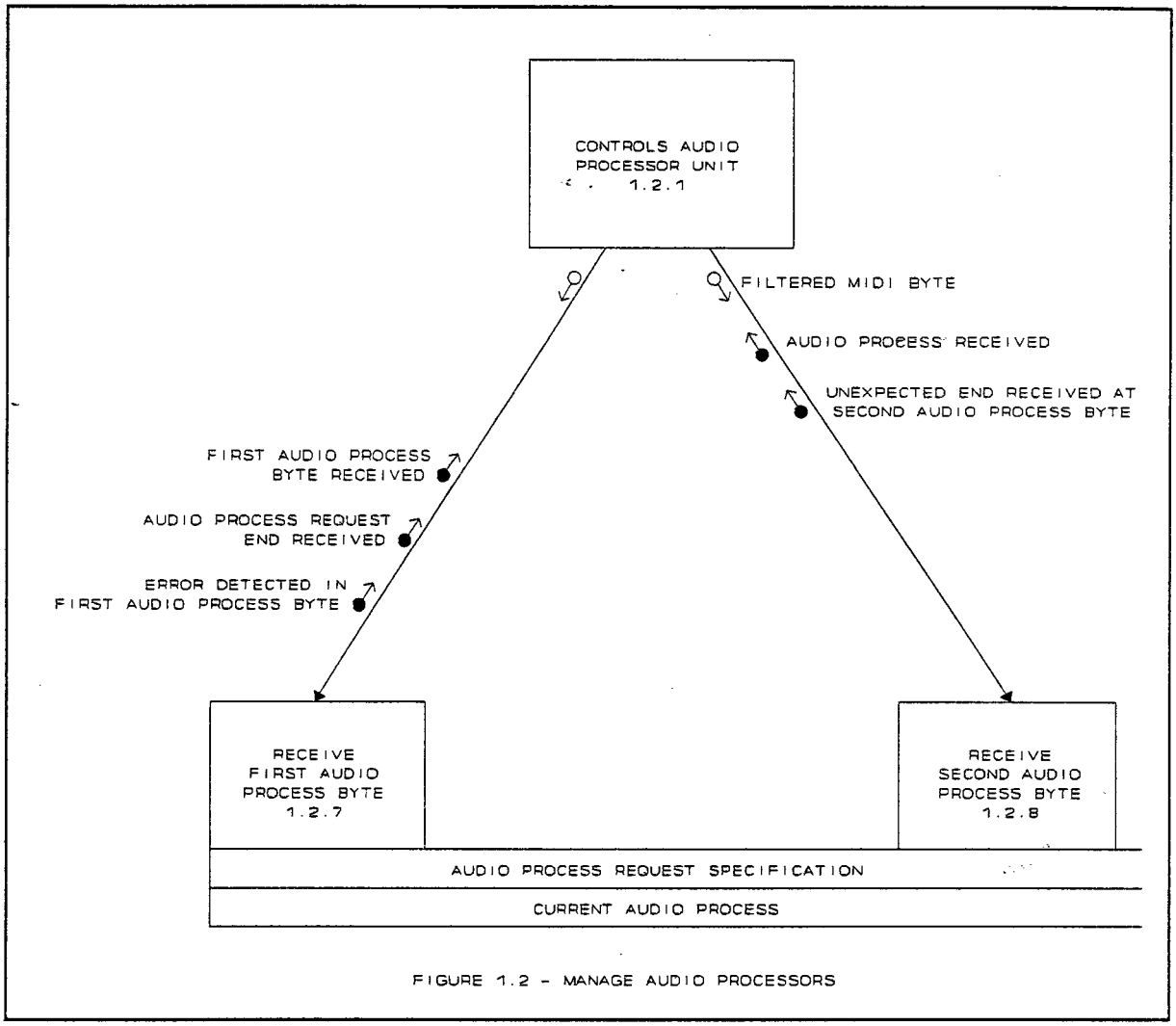

Figure 6.1.7.3 Manage Audio Processors

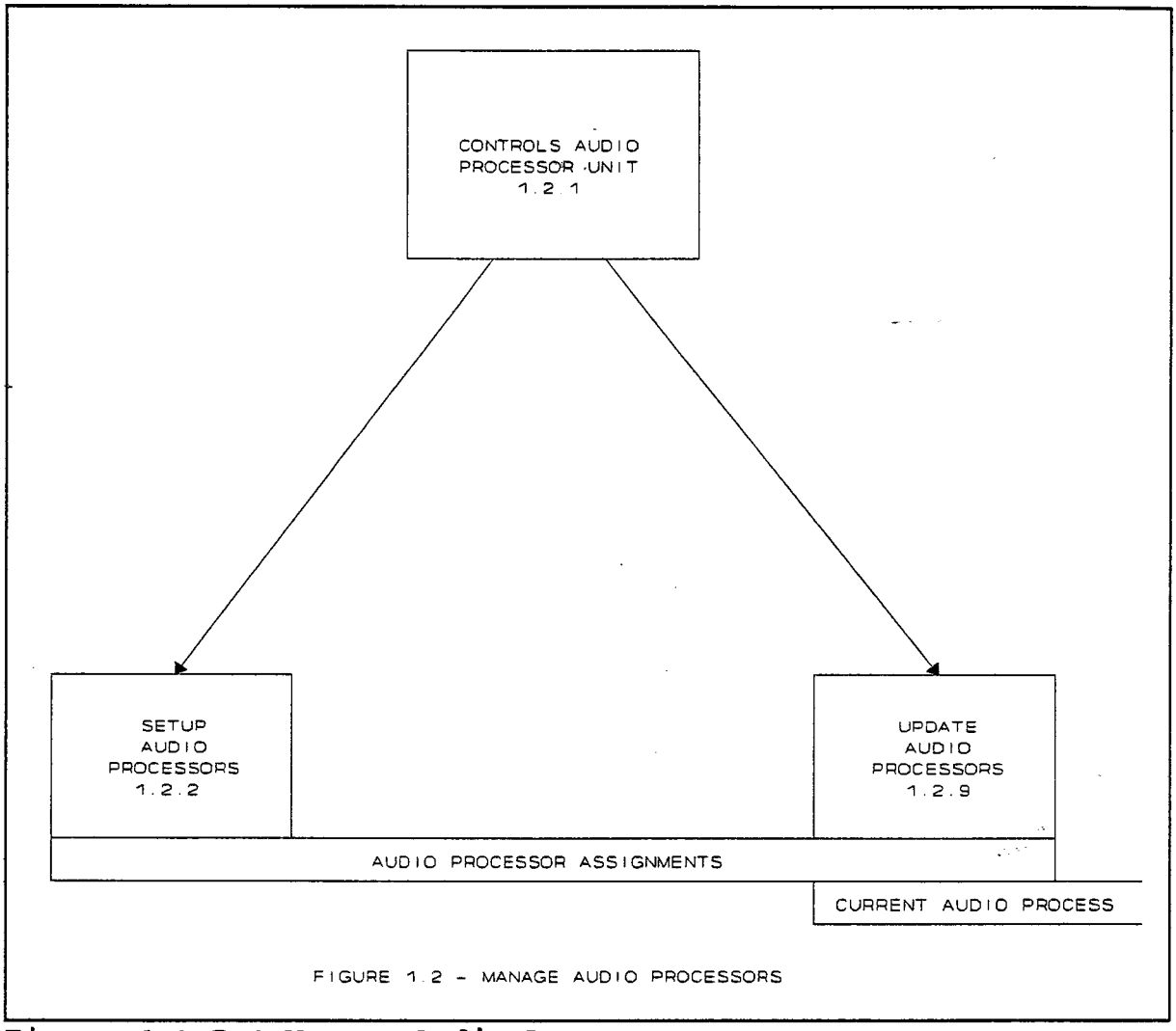

Figure 6.1.7.4 Manage Audio Processors

6.2 The Audio Patcher/Mixer Unit

6.2.1 Transformation Schema

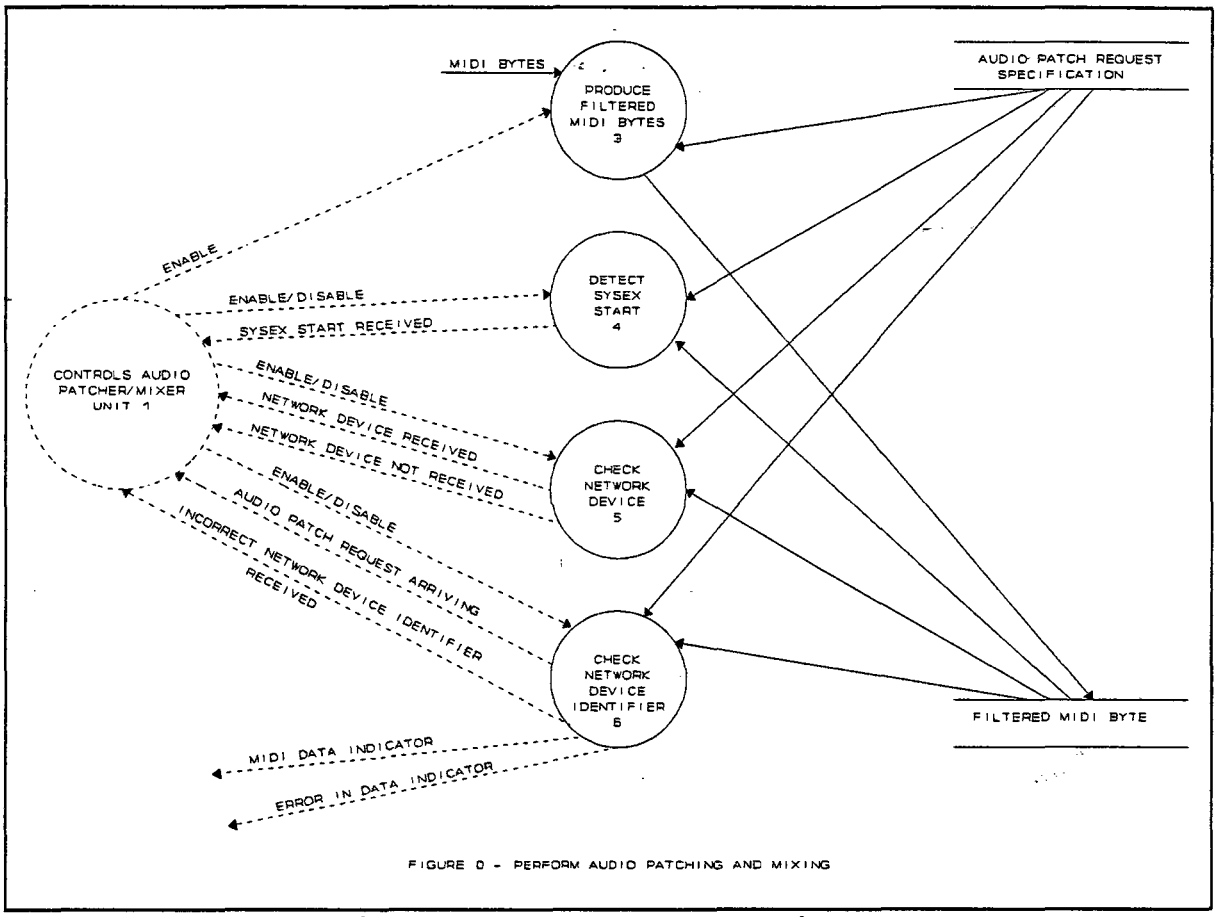

Figure 6.2.1.1 Audio Patch Request Detection

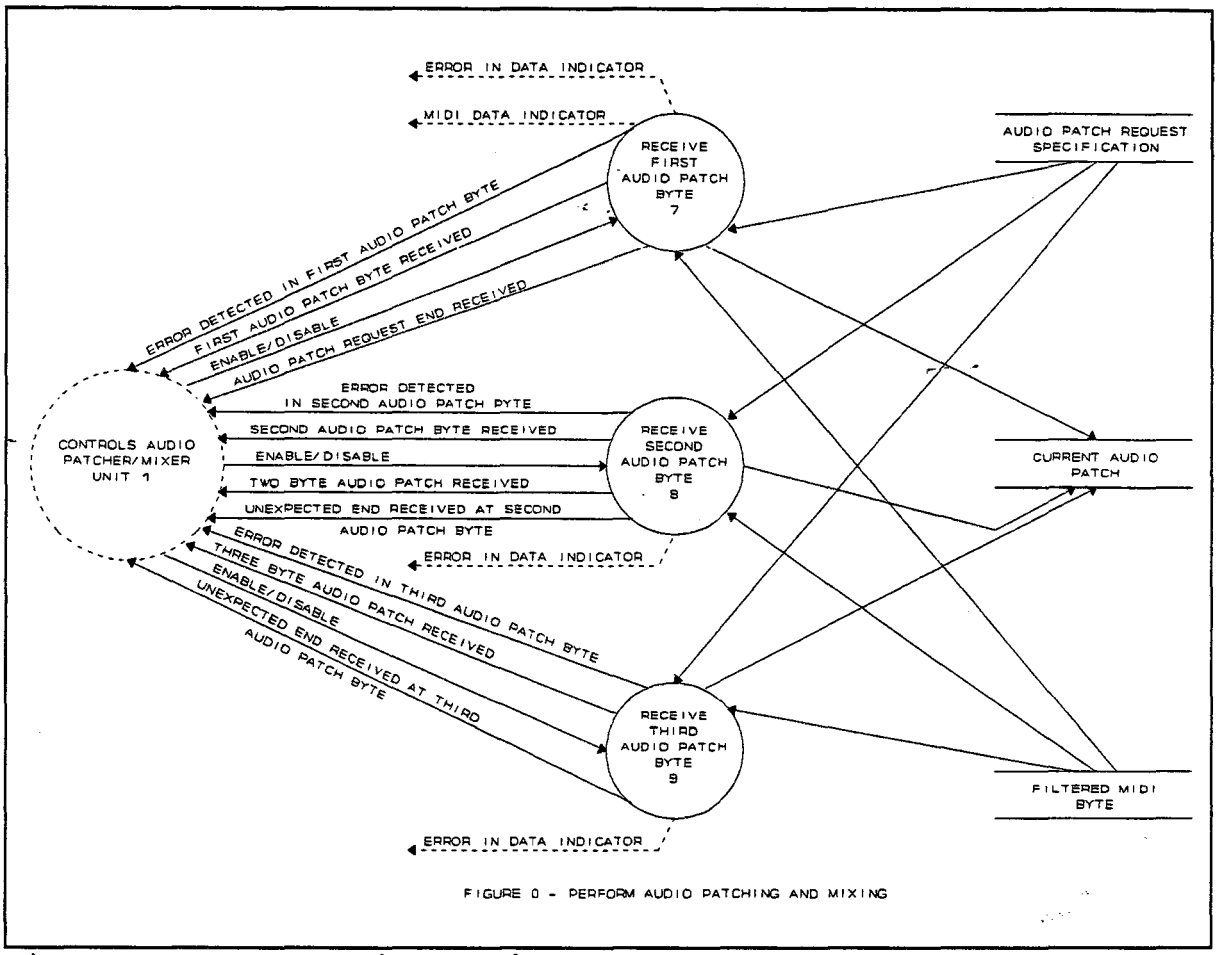

Figure 6.2.1.2 Receive Audio Patch

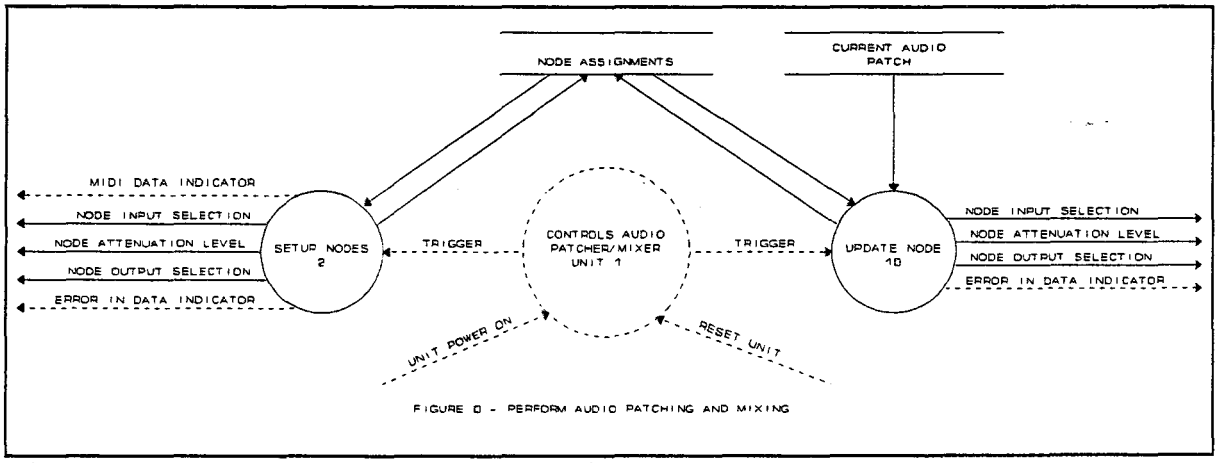

Figure 6.2.1.3 Audio Patcher/Mixer Node Control

6.2.2 State Transition Diagram

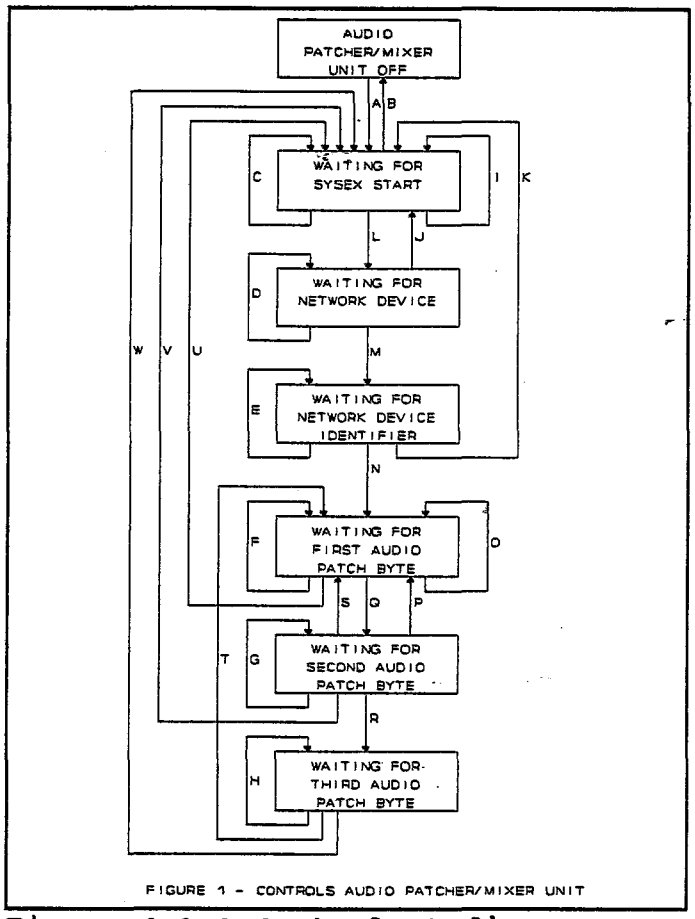

Figure 6.2.2 Controls Audio Patcher/Mixer Unit

1 A) Studio Manager unit power on
In order do:
    Trigger "Setup Nodes"
    Enable "Produce Filtered MIDI Bytes"
    Enable "Detect Sysex Start"
1 B) Studio Manager switches unit power off
1 C) - H) MIDI Real Time Message arrives
1 I) Studio Manager resets unit
In order do:
    Trigger "Setup Nodes"
    Enable "Produce Filtered MIDI Bytes"
    Enable "Detect Sysex Start"
1 J) Network Device not received
In order do:
    Disable "Check Network Device"
    Enable "Detect Sysex Start"

1 K) Incorrect Network Device Identifier received
In order do:
Disable "Check Network Device Identifier"
Enable "Detect Sysex Start"
1 L) Sysex Start received
In order do:
Disable "Detect Sysex Start"
Enable "Check Network Device"
1 M) Network Device received
In order do:
Disable "Check Network Device"
Enable "Check Network Device Identifier"
1 N) Audio Patch Request arriving
In order do:
Disable "Check Network Device Identifier"
Enable "Receive First Audio Patch Byte"
1 O) Error detected in First Audio Patch Byte
1 P) Error detected in Second Audio Patch Byte
In order do:
Disable "Receive Second Audio Patch Byte"
Enable "Receive First Audio Patch Byte"
1 Q) First Audio Patch Byte received
In order do:
Disable "Receive First Audio Patch Byte"
Enable "Receive Second Audio Patch Byte"
1 R) Second Audio Patch Byte received
In order do:
Disable "Receive Second Audio Patch Byte"
Enable "Receive Third Audio Patch Byte"
1 S) Two Byte Audio Patch received
In order do:
Disable "Receive Second Audio Patch Byte"
Trigger "Update Node"
Enable "Receive First Audio Patch Byte"
1 T) Three Byte Audio Patch received
In order do:
Disable "Receive Third Audio Patch Byte"
Trigger "Update Node"
Enable "Receive First Audio Patch Byte"
1 U) Audio Patch Request End received
In order do:
Disable "Receive First Audio Patch Byte"
Enable "Receive Sysex Start"

1 V) Unexpected End received at Second Audio Patch Byte
In order do:
    Disable "Receive Second Audio Patch Byte"
    Enable "Receive Sysex Start"
1 W) Unexpected End received at Third Audio Patch Byte
In order do:
    Disable "Receive Second Audio Patch Byte"
    Enable "Receive Sysex Start"

6.2.3 Transformation Specifications

2 SETUP NODES
Precondition 1
    None
Postcondition 1
    MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = ON
and NODE INPUT SELECTION = DISABLE
    produced for AUDIO PATCH INPUT ADDRESS
and NODE ATTENUATION LEVEL = MINIMUM ATTENUATION LEVEL
    produced for DCA ADDRESS
and NODE OUTPUT SELECTION = DISABLE
    produced for AUDIO PATCH OUTPUT ADDRESS from
    NODE PORT ADDRESSES in NODE ASSIGNMENTS
    for all NODE IDENTIFIERs
and DISCONNECTED NODES contain all NODE IDENTIFIERs
    less than or equal to NODE COUNT
    produced from NODE COUNT ADDRESS
and all CONNECTED NODES contain EMPTY
and MIDI DATA INDICATOR = OFF
and ERROR IN DATA INDICATOR = OFF

3 PRODUCE FILTERED MIDI BYTES
Precondition 1
    MIDI BYTE occurs
and MIDI BYTE is less than MIDI REAL TIME MESSAGE
Postcondition 1
    FILTERED MIDI BYTE is produced
Precondition 2
    MIDI BYTE occurs
and MIDI BYTE is greater or equal to MIDI REAL TIME MESSAGE
Postcondition 2
    None

4 DETECT SYSEX START
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is equal to SYSEX START
Postcondition 1
SYSEX START RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is not equal to SYSEX START
Postcondition 2
None

5 CHECK NETWORK DEVICE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is equal to NETWORK DEVICE
Postcondition 1
NETWORK DEVICE RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is not equal to NETWORK DEVICE
Postcondition 2
NETWORK DEVICE NOT RECEIVED occurs

6 CHECK NETWORK DEVICE IDENTIFIER
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is equal to NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 1
MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = OFF
AUDIO PATCH REQUEST ARRIVING occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is not equal to
NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 2
INCORRECT NETWORK DEVICE IDENTIFIER RECEIVED occurs

7 RECEIVE FIRST AUDIO PATCH BYTE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END

and FILTERED MIDI BYTE is less than AUDIO PATCH INPUT LIMIT
Postcondition 1
     CURRENT AUDIO PATCH INPUT NUMBER = FILTERED MIDI BYTE
and FIRST AUDIO PATCH BYTE RECEIVED occurs
Precondition 2
     FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE is greater than or equal to
     AUDIO PATCH INPUT LIMIT
and FILTERED MIDI BYTE is less than SYSEX END
Postcondition 2
     ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN FIRST AUDIO PATCH BYTE occurs
Precondition 3
     FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 3
     MIDI DATA INDICATOR = OFF
and AUDIO PATCH REQUEST END RECEIVED occurs

8 RECEIVE SECOND AUDIO PATCH BYTE
Precondition 1
     FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
     is equal to CONNECT PATCH
Postcondition 1
     CURRENT AUDIO PATCH OUTPUT NUMBER =
     FILTERED MIDI BYTE & AUDIO PATCH OUTPUT MASK
and CURRENT AUDIO PATCH TYPE =
     FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
and SECOND AUDIO PATCH BYTE RECEIVED occurs
Precondition 2
     FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
     is equal to CHANGE LEVEL PATCH
Postcondition 2
     CURRENT AUDIO PATCH OUTPUT NUMBER =
     FILTERED MIDI BYTE & AUDIO PATCH OUTPUT MASK
and CURRENT AUDIO PATCH TYPE =
     FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
and SECOND AUDIO PATCH BYTE RECEIVED occurs
Precondition 3
     FILTERED MIDI BYTE occurs

and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
is equal to DISCONNECT PATCH
Postcondition 3
CURRENT AUDIO PATCH OUTPUT NUMBER =
FILTERED MIDI BYTE & AUDIO PATCH OUTPUT MASK
and CURRENT AUDIO PATCH TYPE =
FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
and TWO BYTE AUDIO PATCH RECEIVED occurs
Precondition 4
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & AUDIO PATCH TYPE MASK
is not equal to DISCONNECT PATCH
and is not equal to CONNECT PATCH
and is not equal to CHANGE LEVEL PATCH
Postcondition 4
ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN SECOND AUDIO PATCH BYTE occurs
Precondition 5
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 5
MIDI DATA INDICATOR = OFF
and ERROR IN DATA INDICATOR = ON
and UNEXPECTED END RECEIVED AT SECOND AUDIO PATCH BYTE occurs

9 RECEIVE THIRD AUDIO PATCH BYTE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
Postcondition 1
CURRENT AUDIO PATCH MIX LEVEL = FILTERED MIDI BYTE
and THREE BYTE AUDIO PATCH RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 2
MIDI DATA INDICATOR = OFF
and ERROR IN DATA INDICATOR = ON
and UNEXPECTED END RECEIVED AT THIRD AUDIO PATCH BYTE occurs

10 UPDATE NODE
Precondition 1
CURRENT AUDIO PATCH TYPE contains CONNECT PATCH

```
      and corresponding EMPTY CONNECTED NODE where
      AUDIO PATCH INPUT NUMBER is equal to
      CURRENT AUDIO PATCH INPUT NUMBER
and   AUDIO PATCH OUTPUT NUMBER is equal to
      CURRENT AUDIO PATCH OUTPUT NUMBER
and   nonEMPTY DISCONNECTED NODE available
Postcondition 1
      NODE INPUT SELECTION = CURRENT AUDIO PATCH INPUT NUMBER +
      ENABLE produced for AUDIO PATCH INPUT ADDRESS
and   NODE ATTENUATION LEVEL = CURRENT AUDIO PATCH MIX LEVEL
      in MIX ATTENUATION LEVEL MAP produced for DCA ADDRESS
and   NODE OUTPUT SELECTION =
      CURRENT AUDIO PATCH OUTPUT NUMBER + ENABLE
      produced for AUDIO PATCH OUTPUT ADDRESS
      from NODE PORT ADDRESSES
      in NODE ASSIGNMENTS for NODE IDENTIFIER
      corresponding to nonEMPTY DISCONNECTED NODE
and   CONNECTED NODE = NODE IDENTIFIER
      of nonEMPTY DISCONNECTED NODE
      where AUDIO PATCH INPUT NUMBER is equal to
      CURRENT AUDIO PATCH INPUT NUMBER
and   AUDIO PATCH OUTPUT NUMBER is equal to
      CURRENT AUDIO PATCH OUTPUT NUMBER
and   DISCONNECTED NODE = EMPTY
Precondition 2
      CURRENT AUDIO PATCH TYPE contains CONNECT PATCH
and   no nonEMPTY DISCONNECTED NODE available
Postcondition 2
      ERROR IN DATA INDICATOR = ON
Precondition 3
      CURRENT AUDIO PATCH TYPE contains CONNECT PATCH
      and corresponding nonEMPTY CONNECTED NODE where
      AUDIO PATCH INPUT NUMBER is equal to
      CURRENT AUDIO PATCH INPUT NUMBER
and   AUDIO PATCH OUTPUT NUMBER is equal to
      CURRENT AUDIO PATCH OUTPUT NUMBER
Postcondition 3
      ERROR IN DATA INDICATOR = ON
Precondition 4
      CURRENT AUDIO PATCH TYPE contains CHANGE LEVEL PATCH
      and corresponding nonEMPTY CONNECTED NODE where
      AUDIO PATCH INPUT NUMBER is equal to
      CURRENT AUDIO PATCH INPUT NUMBER
and   AUDIO PATCH OUTPUT NUMBER is equal to
      CURRENT AUDIO PATCH OUTPUT NUMBER
```

Postcondition 4

NODE ATTENUATION LEVEL = CURRENT AUDIO PATCH MIX LEVEL
in MIX ATTENUATION LEVEL MAP produced for DCA ADDRESS
from NODE PORT ADDRESSES in NODE ASSIGNMENTS
for NODE IDENTIFIER corresponding to CONNECTED NODE where
AUDIO PATCH INPUT NUMBER is equal to
CURRENT AUDIO PATCH INPUT NUMBER
and AUDIO PATCH OUTPUT NUMBER is equal to
CURRENT AUDIO PATCH OUTPUT NUMBER

Precondition 5

CURRENT AUDIO PATCH TYPE contains CHANGE LEVEL PATCH
and corresponding EMPTY CONNECTED NODE where
AUDIO PATCH INPUT NUMBER is equal to
CURRENT AUDIO PATCH INPUT NUMBER
and AUDIO PATCH OUTPUT NUMBER is equal to
CURRENT AUDIO PATCH OUTPUT NUMBER

Postcondition 5

ERROR IN DATA INDICATOR = ON

Precondition 6

CURRENT AUDIO PATCH TYPE contains DISCONNECT PATCH
and corresponding nonEMPTY CONNECTED NODE where
AUDIO PATCH INPUT NUMBER is equal to
CURRENT AUDIO PATCH INPUT NUMBER
and AUDIO PATCH OUTPUT NUMBER is equal to
CURRENT AUDIO PATCH OUTPUT NUMBER

Postcondition 6

NODE INPUT SELECTION = DISABLE
produced for AUDIO PATCH INPUT ADDRESS
and NODE ATTENUATION LEVEL = MINIMUM ATTENUATION LEVEL
produced for DCA ADDRESS
and NODE OUTPUT SELECTION = DISABLE
produced for AUDIO PATCH OUTPUT ADDRESS
from NODE PORT ADDRESSES in NODE ASSIGNMENTS
for NODE IDENTIFIER corresponding to CONNECTED NODE where
AUDIO PATCH INPUT NUMBER is equal to
CURRENT AUDIO PATCH INPUT NUMBER
and AUDIO PATCH OUTPUT NUMBER is equal to
CURRENT AUDIO PATCH OUTPUT NUMBER
and EMPTY DISCONNECTED NODE = NODE IDENTIFIER for
corresponding CONNECTED NODE where
AUDIO PATCH INPUT NUMBER is equal to
CURRENT AUDIO PATCH INPUT NUMBER
and AUDIO PATCH OUTPUT NUMBER is equal to
CURRENT AUDIO PATCH OUTPUT NUMBER
and CONNECTED NODE = EMPTY where

```
     AUDIO PATCH INPUT NUMBER is equal to
     CURRENT AUDIO PATCH INPUT NUMBER
and  AUDIO PATCH OUTPUT NUMBER is equal to
     CURRENT AUDIO PATCH OUTPUT NUMBER
```

Precondition 7

```
     CURRENT AUDIO PATCH TYPE contains DISCONNECT PATCH
     and corresponding EMPTY CONNECTED NODE where
     AUDIO PATCH INPUT NUMBER is equal to
     CURRENT AUDIO PATCH INPUT NUMBER
and  AUDIO PATCH OUTPUT NUMBER is equal to
     CURRENT AUDIO PATCH OUTPUT NUMBER
```

Postcondition 7

```
     ERROR IN DATA INDICATOR = ON
```

6.2.4 Data Dictionary

```
audio patch input address     = * the port address for the input
                                  selector of a particular node *
                              = * values : 10000000 00000000 -
                                  10000000 11111111, type : binary *

audio patch input limit       = * sixteen inputs only *
                              = * value : 16, type : byte *

audio patch input number      = * choice of sixteen inputs *
                              = * values : 0 - 15, type : byte *

audio patch output address    = * the port address for the output
                                  selector of a particular node *
                              = * values : 10000000 00000000 -
                                  10000000 11111111, type : binary *

audio patch output mask       = * occupies the lower four bits *
                              = * value : 00001111, type : binary *

audio patch output number     = * choice of sixteen outputs *
                              = * values : 0 - 15, type : byte *

audio patch request arriving  = * signal indicating audio patches
                                  arriving for this unit *

audio patch request end = **

audio patch request specification = * specification of information in
                                     an audio patch request *
```

```
                          = sysex start + network device +
                            network device identifier address +
                            MIDI real time messages + sysex end +
                            connect patch + change level patch +
                            disconnect patch +
                            audio patch input limit +
                            audio patch output mask +
                            audio patch type mask

audio patch type mask     = * occupies the upper four bits *
                          = * value : 11110000, type : binary *

change level patch        = * audio patch type *
                          = * value : 00100000, type : binary *

connect patch             = * audio patch type *
                          = * value : 00010000, type : binary *

connected node            = node number +
                            @audio input number +
                            @audio output number

connected nodes           = {connected node}48

current audio patch       = current audio patch input number +
                            current audio patch output number +
                            current audio patch type +
                            current audio patch mix level

current audio patch       = * choice of sixteen inputs *
input number              = * values : 0 - 15, type : byte *

current audio patch       = * choice of 128 mix levels *
mix level                 = * values: 0 - 127, type : byte *

current audio patch       = * choice of sixteen outputs  *
output number             = * values : 0 - 15, type : byte *

current audio patch       = [disconnect patch ¦ connect patch ¦
type                        change level patch]

DCA address               = * the port address for the DCA of
                              a particular node *
                          = * values : 10000000 00000000 -
                              10000000 11111111, type : binary *
```

```
disable                    = * to disable audio input selector
                             and/or audio output selector *
                           = * value : 00000000 00000000, type :
                             binary *

disconnect patch           = * audio patch type *
                           = * value : 01000000, type : binary *

disconnected node          = node number

disconnected nodes         = {disconnected node}48

empty                      = * empty entry in connected nodes
                             and disconnected nodes *
                           = * value : 48, type : byte *

enable                     = * to enable audio input selector
                             and/or audio output selector *
                           = * value : 00000010 00000000, type :
                             binary *

error detected in first audio patch byte = **

error detected in second audio patch byte = **

error in data              = * indicates the occurrence of an error
indicator                    state within the unit *
                           = * values : [on | off] *

filtered MIDI byte         = * contains no MIDI real time messages *
                             * values : 0000000 - 11110111, type :
                               binary *

first audio patch byte received = **

incorrect network device identifier received = **

minimum attenuation        = * -102.2 dBs *
level                      = * value : 00000000 00000000, type :
                             binary *

MIDI byte                  = * values : 0000000 - 11111111, type :
                             binary *

MIDI bytes                 = 1{MIDI byte}
```

```
MIDI data indicator    = * indicates the receival of an audio
                         patch request via MIDI *
                       = * values : [on | off] *

MIDI real time messages = * value : 11111000, type : binary *

mix attenuation level = * mix to attenuation level map *
map                   = 1{node attenuation level}128
                        * see Appendix 6.2.5 for values *

network device        = * This is a device type identifier in
                         the network *
                       = * value : 01111101, type : binary *

network device        = * network device number identifying a
identifier               particular unit *
                       = * values : 00000000 - 01111111,
                         type : binary *

network device        = * the port address to obtain the
identifier address       network device identifier
                       = * value : 10000000 11000001, type :
                         binary *

network device not received = **

network device received = **

node assignments      = disable + enable +
                        minimum attenuation level +
                        node count address +
                        disconnected nodes +
                        connected nodes +
                        mix attenuation level map +
                        1{node port addresses}48

node attenuation      = * 0.0 - -102.2 dBs *
level                 = * values : [00000000 00000000 -
                         00000001 11111111], type : binary *

node count            = * number of nodes within the unit *
                      = * values : 1 - 48, type : byte *

node count address    = * the port address to obtain the node
                         count *
```

```
                          = * value : 10000000 11000010, type :
                            binary *

node identifier           = * 48 possible nodes *
                            * values : 0 - 47, type : byte *

node input selection      = * values : [0000010 00000000 -
                            00000010 00001111 ¦ disable],
                            type : binary *

node number               = * values : [0 - 47 ¦ empty], type :
                            byte *

node output selection     = * values : [0000010 00000000 -
                            00000010 00001111 ¦ disable],
                            type : binary *

node port addresses       = @node identifier +
                            audio patch input address +
                            DCA address +
                            audio patch output address
                            * see Appendix 2.3.4 for values *

reset unit          = * signal to initiate unit reinitialisation *

second audio patch byte received = **

sysex end                 = * value : 10000000, type : binary *

sysex start               = * value : 11110000, type : binary *

sysex start received      = **

three byte audio patch received = **

two byte audio patch received = **

unexpected end received at second audio patch byte = **

unexpected end received at third audio patch byte = **

unit power on       = * signal to initiate unit initialisation *
```

6.2.5 Attenuation Level Map (binary form for nodes)

VALUE	BINARY	VALUE	BINARY	VALUE	BINARY	VALUE	BINARY
127	111111111	95	111011111	63	110111011	31	101101111
126	111111110	94	111011110	62	110111001	30	101101011
125	111111101	93	111011101	61	110110111	29	101100111
124	111111100	92	111011100	60	110110101	28	101100011
123	111111011	91	111011011	59	110110011	27	101011111
122	111111010	90	111011010	58	110110001	26	101011011
121	111111001	89	111011001	57	110101111	25	101010111
120	111111000	88	111011000	56	110101101	24	101010011
119	111110111	87	111010111	55	110101011	23	101001111
118	111110110	86	111010110	54	110101001	22	101001011
117	111110101	85	111010101	53	110100111	21	101000011
116	111110100	84	111010100	52	110100101	20	100111011
115	111110011	83	111010011	51	110100011	19	100110011
114	111110010	82	111010010	50	110100001	18	100101011
113	111110001	81	111010001	49	110011111	17	100100011
112	111110000	80	111010000	48	110011101	16	100011011
111	111101111	79	111001111	47	110011011	15	100010011
110	111101110	78	111001110	46	110011001	14	100001011
109	111101101	77	111001101	45	110010111	13	100000011
108	111101100	76	111001100	44	110010101	12	011111011
107	111101011	75	111001011	43	110010011	11	011101011
106	111101010	74	111001010	42	110010001	10	011011011
105	111101001	73	111001001	41	110001111	9	011001011
104	111101000	72	111001000	40	110001101	8	010111011
103	111100111	71	111000111	39	110001011	7	010101011
102	111100110	70	111000110	38	110001001	6	010010011
101	111100101	69	111000101	37	110000111	5	001111011
100	111100100	68	111000100	36	110000011	4	001100011
99	111100011	67	111000011	35	101111111	3	001001011
98	111100010	66	111000001	34	101111011	2	000110011
97	111100001	65	110111111	33	101110111	1	000011011
96	111100000	64	110111101	32	101110011	0	000000011

6.2.6 Task Schema

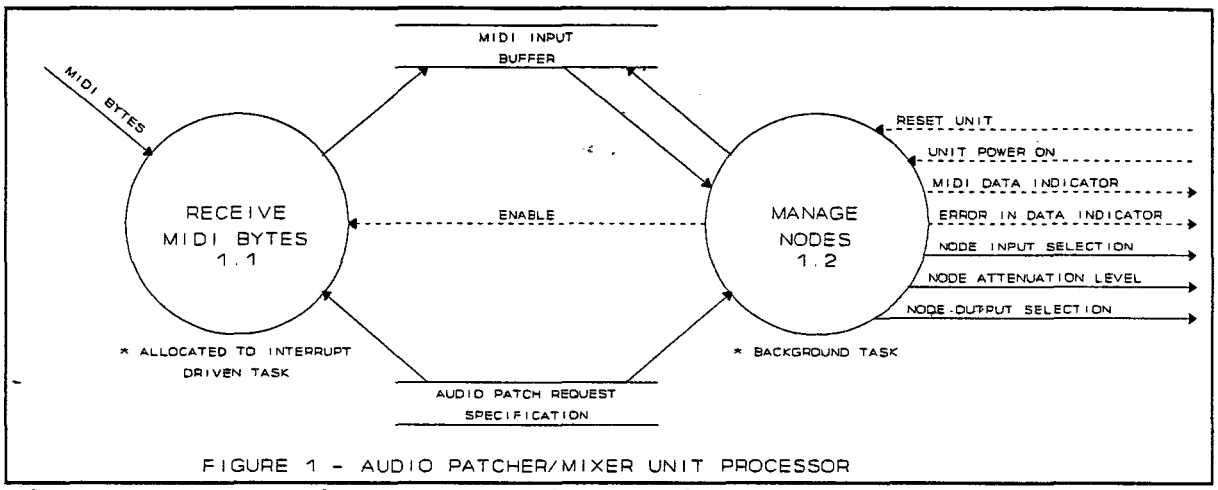

Figure 6.2.6 Audio Patcher/Mixer Unit Processor

6.2.7 Structure Chart

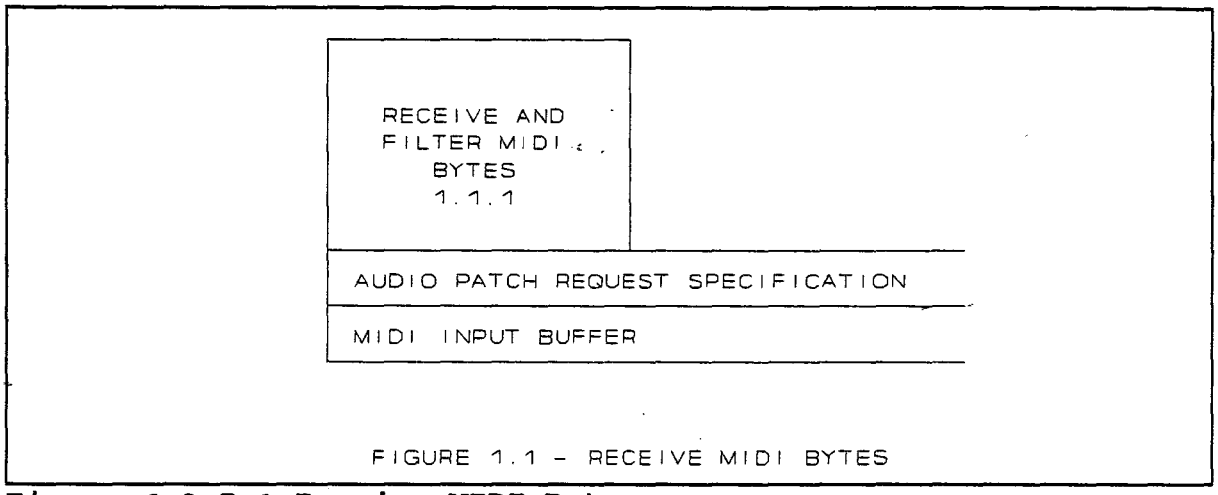

Figure 6.2.7.1 Receive MIDI Bytes

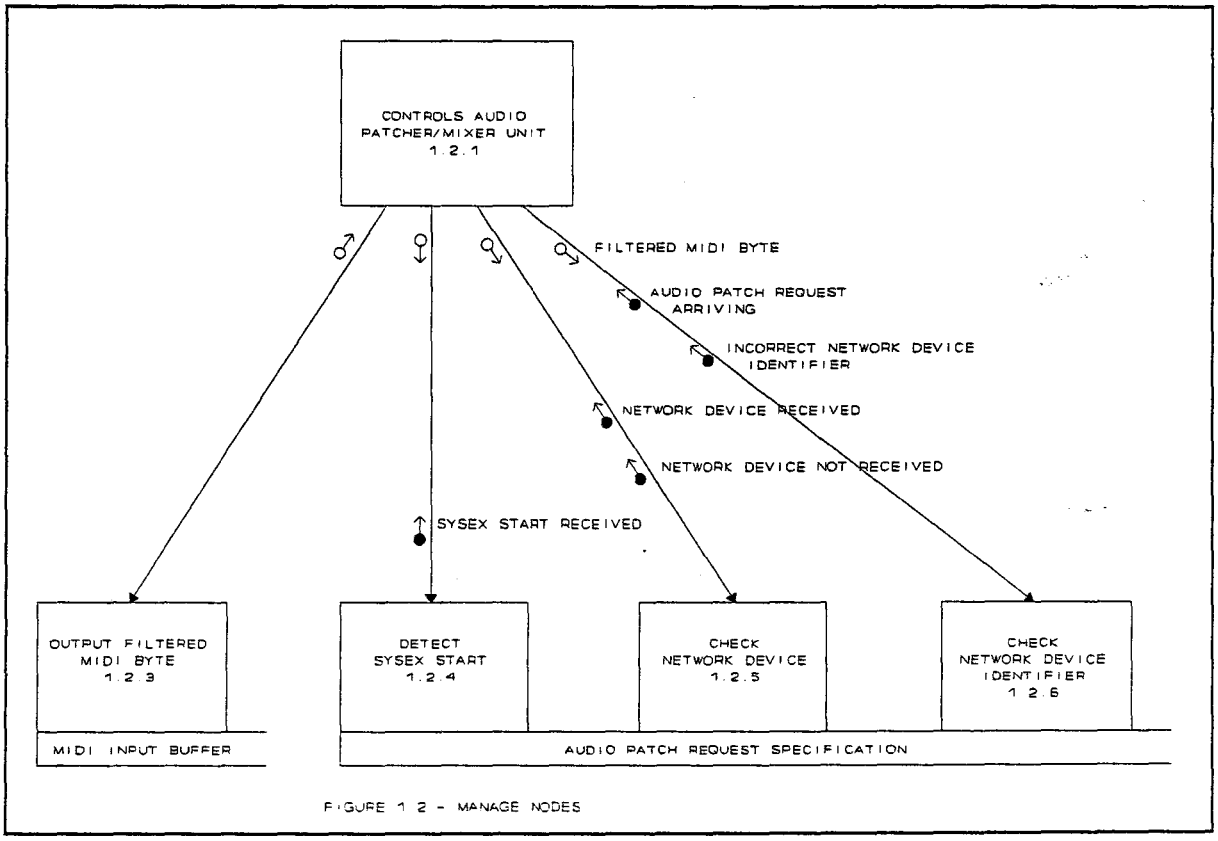

Figure 6.2.7.2 Manage Nodes

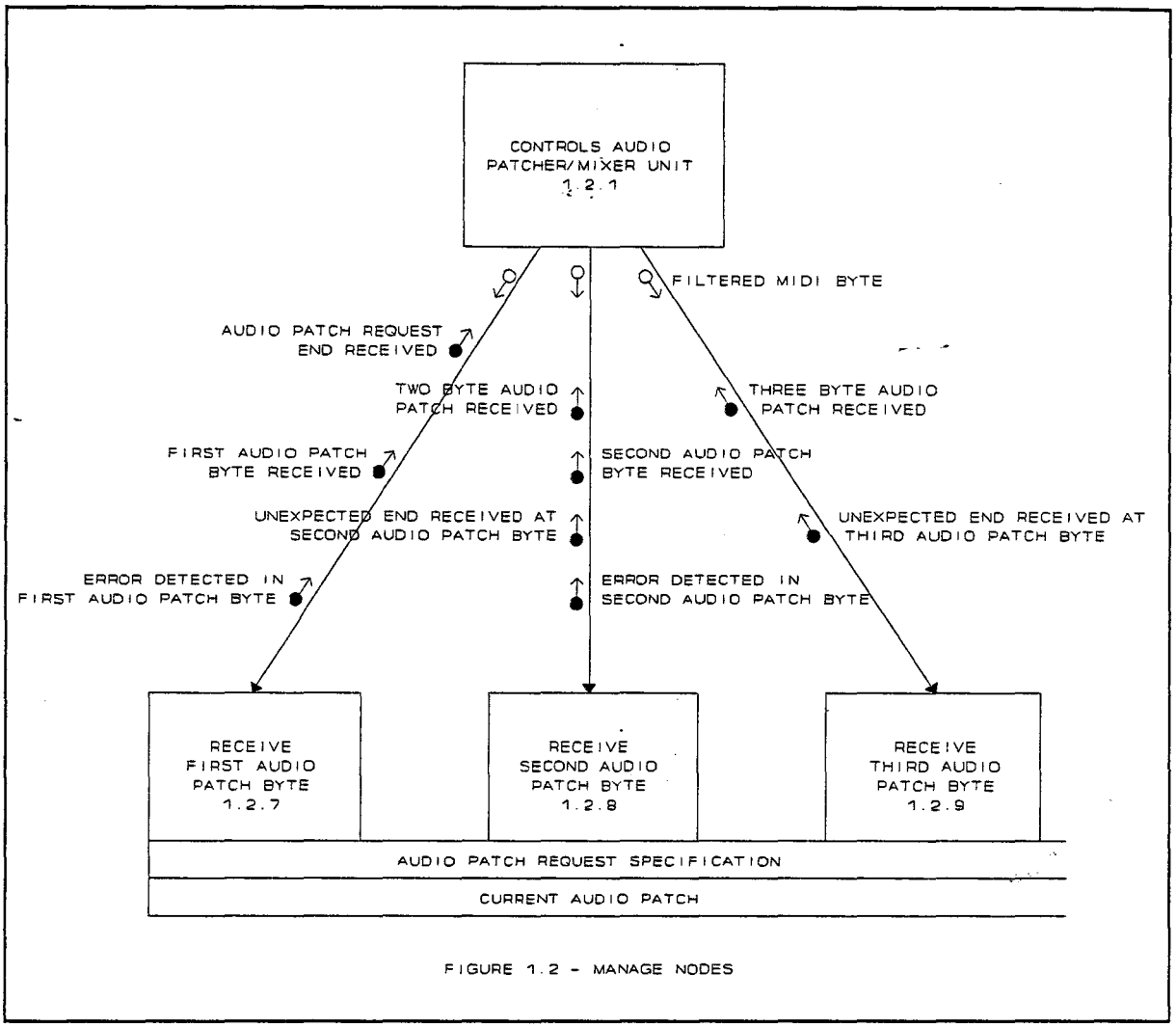

Figure 6.2.7.3 Manage Nodes

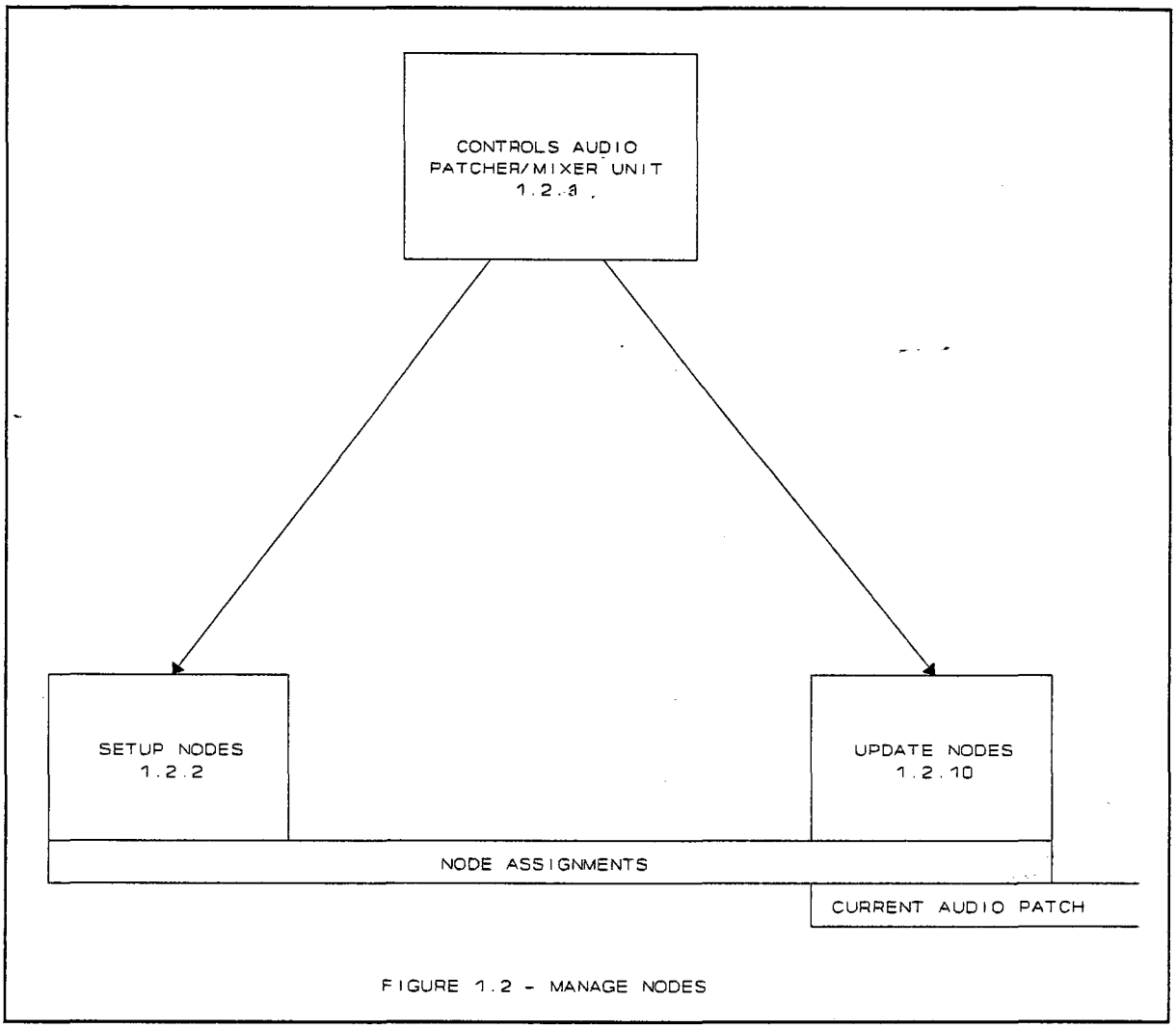

Figure 6.2.7.4 Manage Nodes

6.3 The MIDI Patch Unit

6.3.1 Transformation Schema

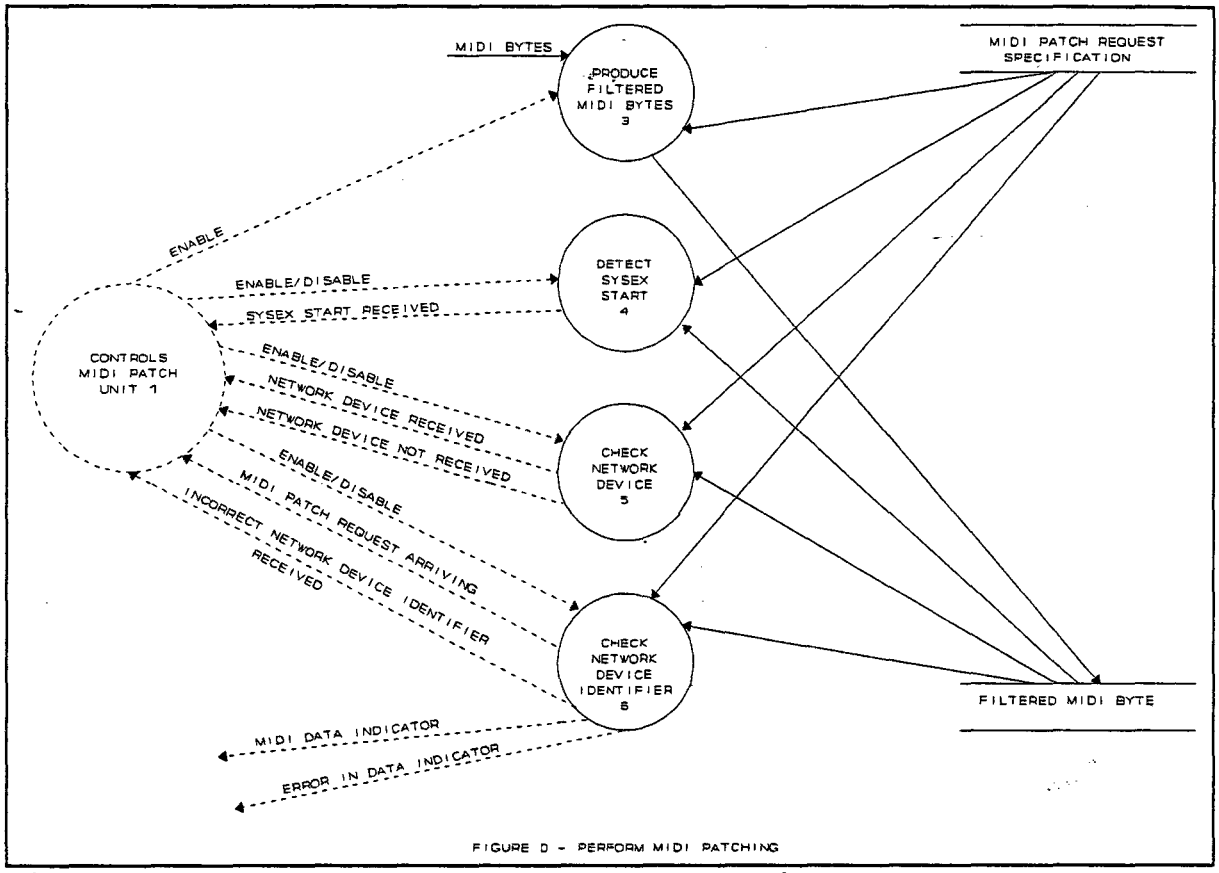

Figure 6.3.1.1 MIDI Patch Request Detection

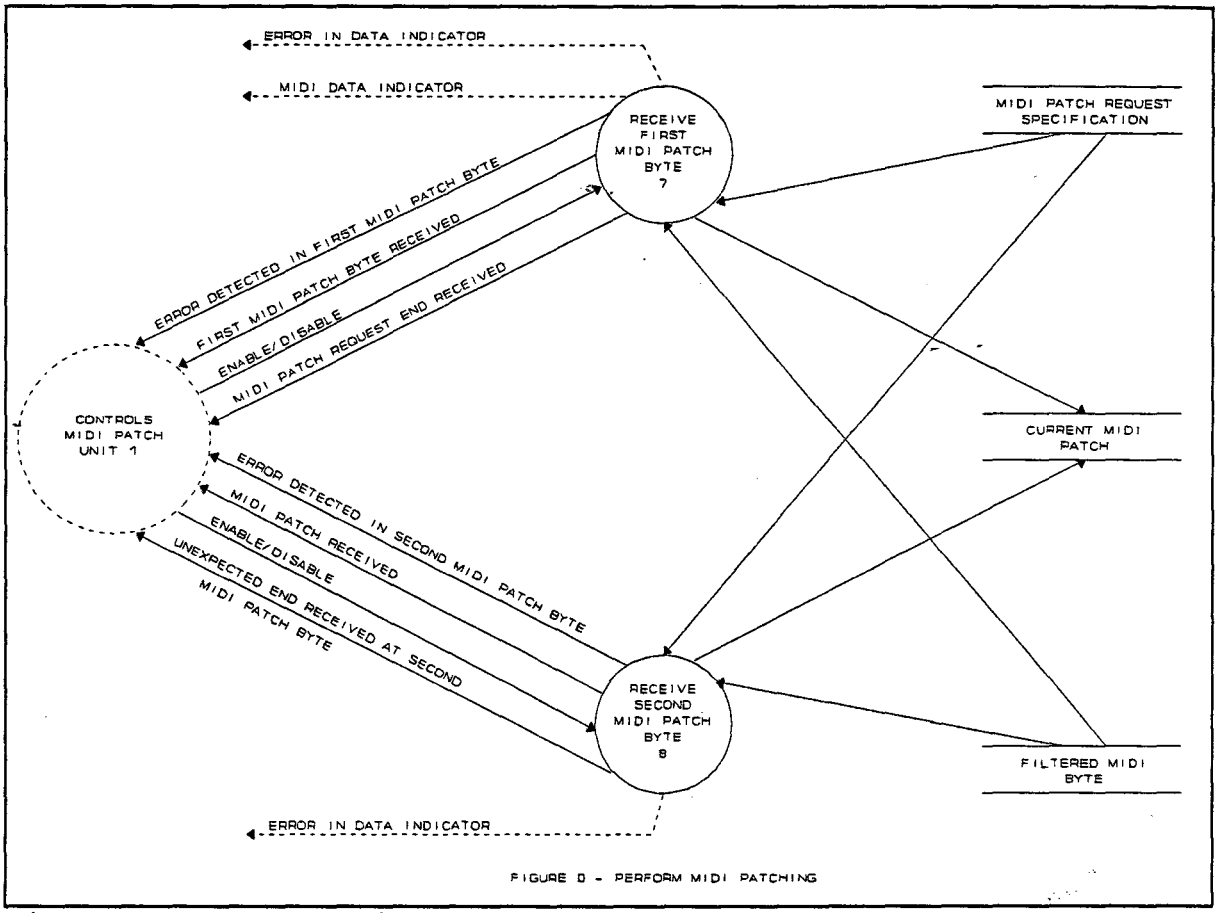

Figure 6.3.1.2 **Receive MIDI Patch**

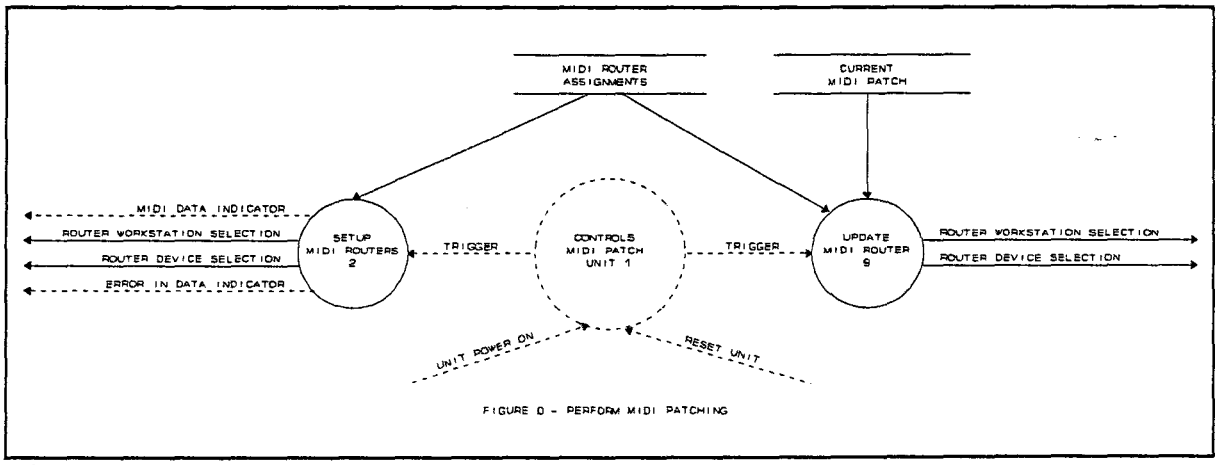

Figure 6.3.1.3 **MIDI Router Control**

6.3.2 State Transition Diagram

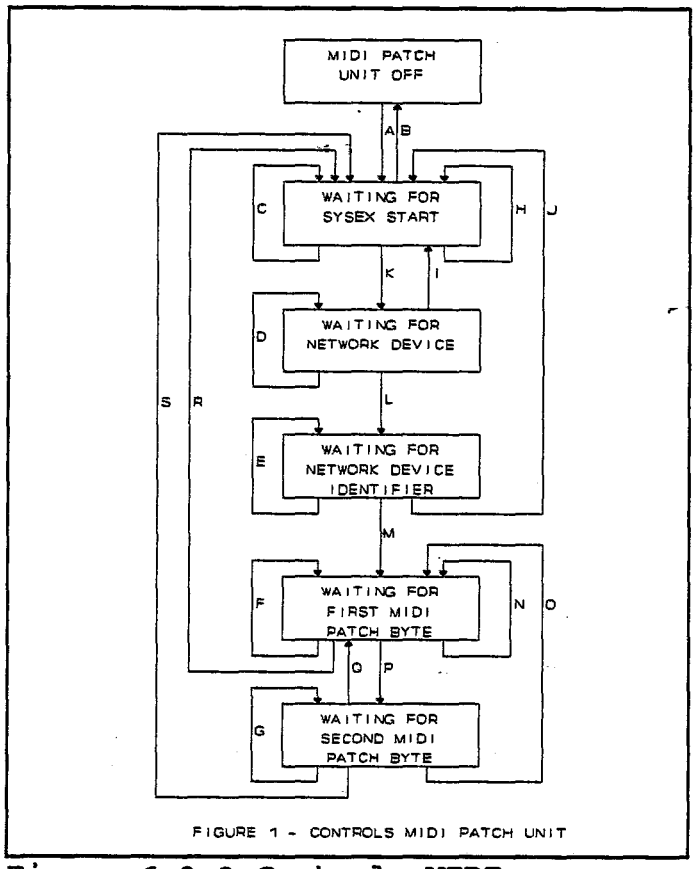

Figure 6.3.2 Controls MIDI Patch Unit

1 A) Studio Manager switches unit power on
In order do:
Trigger "Setup MIDI Routers"
Enable "Produce Filtered MIDI Bytes"
Enable "Detect Sysex Start"
1 B) Studio Manager switches unit power off
1 C) - G) MIDI Real Time Message arrives
1 H) Studio Manager resets unit
In order do:
Trigger "Setup MIDI Routers"
Enable "Produce Filtered MIDI Bytes"
Enable "Detect Sysex Start"
1 I) Network Device not received
In order do:
Disable "Check Network Device"
Enable "Detect Sysex Start"
1 J) Incorrect Network Device Identifier received
In order do:
Disable "Check Network Device Identifier"
Enable "Detect Sysex Start"

1 K) Sysex Start received
In order do:
Disable "Detect Sysex Start"
Enable "Check Network Device"
1 L) Network Device received
In order do:
Disable "Check Network Device"
Enable "Check Network Device Identifier"
1 M) MIDI Patch Request arriving
In order do:
Disable "Check Network Device Identifier"
Enable "Receive First MIDI Patch Byte"
1 N) Error detected in First MIDI Patch Byte
1 O) Error detected in Second MIDI Patch Byte
Disable "Receive Second MIDI Patch Byte"
Enable "Receive First Audio Process Byte"
1 P) First MIDI Patch Byte received
In order do:
Disable "Receive First MIDI Patch Byte"
Enable "Receive Second MIDI Patch Byte"
1 Q) MIDI Patch received
In order do:
Disable "Receive Second MIDI Patch Byte"
Trigger "Update MIDI Router"
Enable "Receive First MIDI Patch Byte"
1 R) MIDI Patch Request End received
In order do:
Disable "Receive First MIDI Patch Byte"
Enable "Receive Sysex Start"
1 S) Unexpected End received at Second MIDI Patch Byte
In order do:
Disable "Receive Second MIDI Patch Byte"
Enable "Receive Sysex Start"

6.3.3 Transformation Specifications

2 SETUP MIDI ROUTERS
Precondition 1
None.
Postcondition 1
MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = ON
ROUTER WORKSTATION SELECTION = EXTERNAL
produced for WORKSTATION TO DEVICE MIDI ROUTER ADDRESS
from MIDI ROUTER ASSIGNMENTS

```
     for all WORKSTATION TO DEVICE MIDI ROUTER IDENTIFIERs
and  ROUTER DEVICE SELECTION = EXTERNAL
     produced for DEVICE TO WORKSTATION MIDI ROUTER ADDRESS
     from MIDI ROUTER ASSIGNMENTS
     for all DEVICE TO WORKSTATION MIDI ROUTER IDENTIFIERs
and  MIDI DATA INDICATOR = OFF
and  ERROR IN DATA INDICATOR = OFF

3 PRODUCE FILTERED MIDI BYTES
Precondition 1
     MIDI BYTE occurs
and  MIDI BYTE is less than MIDI REAL TIME MESSAGE
Postcondition 1
     FILTERED MIDI BYTE is produced
Precondition 2
     MIDI BYTE occurs
and  MIDI BYTE is greater or equal to MIDI REAL TIME MESSAGE
Postcondition 2
     None

4 DETECT SYSEX START
Precondition 1
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is equal to SYSEX START
Postcondition 1
     SYSEX START RECEIVED occurs
Precondition 2
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is not equal to SYSEX START
Postcondition 2
     None

5 CHECK NETWORK DEVICE
Precondition 1
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is equal to NETWORK DEVICE
Postcondition 1
     NETWORK DEVICE RECEIVED occurs
Precondition 2
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is not equal to NETWORK DEVICE
Postcondition 2
     NETWORK DEVICE NOT RECEIVED occurs
```

6 CHECK NETWORK DEVICE IDENTIFIER
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is equal to NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 1
MIDI DATA INDICATOR = ON
and ERROR IN DATA INDICATOR = OFF
MIDI PATCH REQUEST ARRIVING occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is not equal to
NETWORK DEVICE IDENTIFIER
produced from NETWORK DEVICE IDENTIFIER ADDRESS
Postcondition 2
INCORRECT NETWORK DEVICE IDENTIFIER RECEIVED occurs

7 RECEIVE FIRST MIDI PATCH BYTE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to WORKSTATION TO DEVICE
and FILTERED MIDI BYTE & EXTERNAL
is equal to ZERO
and FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
is less than MIDI PATCH WORKSTATION LIMIT
Postcondition 1
CURRENT MIDI PATCH DIRECTION =
FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
and CURRENT MIDI WORKSTATION =
FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
and FIRST MIDI PATCH BYTE RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to DEVICE TO WORKSTATION
and FILTERED MIDI BYTE & EXTERNAL
is equal to ZERO
and FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
is less than MIDI PATCH WORKSTATION LIMIT
Postcondition 2
CURRENT MIDI PATCH DIRECTION =
FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK

```
and  CURRENT MIDI WORKSTATION =
     FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
and  FIRST MIDI PATCH BYTE RECEIVED occurs
Precondition 3
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
     is equal to WORKSTATION TO DEVICE
and  FILTERED MIDI BYTE & EXTERNAL
     is equal to EXTERNAL
and  FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
     is equal to ZERO
Postcondition 3
     CURRENT MIDI PATCH DIRECTION =
     FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
and  CURRENT MIDI WORKSTATION = EXTERNAL
and  FIRST MIDI PATCH BYTE RECEIVED occurs
Precondition 4
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
     is equal to DEVICE TO WORKSTATION
and  FILTERED MIDI BYTE & EXTERNAL
     is equal to EXTERNAL
Postcondition 4
     ERROR IN DATA INDICATOR = ON
and  ERROR DETECTED IN FIRST MIDI PATCH BYTE occurs
Precondition 5
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
     is equal to WORKSTATION TO DEVICE
and  FILTERED MIDI BYTE & EXTERNAL
     is equal to EXTERNAL
and  FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
     is not equal to ZERO
Postcondition 5
     ERROR IN DATA INDICATOR = ON
and  ERROR DETECTED IN FIRST MIDI PATCH BYTE occurs
Precondition 6
     FILTERED MIDI BYTE occurs
and  FILTERED MIDI BYTE is less than SYSEX END
and  FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
     is not equal to WORKSTATION TO DEVICE
and  is not equal to DEVICE TO WORKSTATION
```

Postcondition 6
ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN FIRST MIDI PATCH BYTE occurs
Precondition 7
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 7
MIDI DATA INDICATOR = OFF
and MIDI PATCH REQUEST END RECEIVED occurs

& RECEIVE SECOND MIDI PATCH BYTE
Precondition 1
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to ZERO
and FILTERED MIDI BYTE & EXTERNAL
is equal to ZERO
Postcondition 1
CURRENT MIDI DEVICE =
FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
and MIDI PATCH RECEIVED occurs
Precondition 2
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to ZERO
and FILTERED MIDI BYTE & EXTERNAL
is not equal to ZERO
and FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
is equal to ZERO
and CURRENT MIDI PATCH DIRECTION is equal to
DEVICE TO WORKSTATION
Postcondition 2
CURRENT MIDI DEVICE = EXTERNAL
and MIDI PATCH RECEIVED occurs
Precondition 3
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is not equal to ZERO
Postcondition 3
ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN SECOND MIDI PATCH BYTE occurs

Precondition 4
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to ZERO
and FILTERED MIDI BYTE & EXTERNAL
is not equal to ZERO
and CURRENT MIDI PATCH DIRECTION is equal to
WORKSTATION TO DEVICE
Postcondition 4
ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN SECOND MIDI PATCH BYTE occurs
Precondition 5
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is less than SYSEX END
and FILTERED MIDI BYTE & MIDI PATCH DIRECTION MASK
is equal to ZERO
and FILTERED MIDI BYTE & EXTERNAL
is not equal to ZERO
and FILTERED MIDI BYTE & MIDI PATCH WORKSTATION/DEVICE MASK
is not equal to ZERO
Postcondition 5
ERROR IN DATA INDICATOR = ON
and ERROR DETECTED IN SECOND MIDI PATCH BYTE occurs
Precondition 6
FILTERED MIDI BYTE occurs
and FILTERED MIDI BYTE is greater than or equal to SYSEX END
Postcondition 6
MIDI DATA INDICATOR = OFF
and ERROR IN DATA INDICATOR = ON
and UNEXPECTED END RECEIVED AT SECOND MIDI PATCH BYTE occurs

9 UPDATE MIDI ROUTER
Precondition 1
CURRENT MIDI PATCH DIRECTION contains
WORKSTATION TO DEVICE
Postcondition 1
ROUTER WORKSTATION SELECTION =
CURRENT MIDI PATCH WORKSTATION
produced for WORKSTATION TO DEVICE MIDI ROUTER ADDRESS
from MIDI ROUTER ASSIGNMENTS
for WORKSTATION TO DEVICE MIDI ROUTER IDENTIFIER
corresponding to CURRENT MIDI PATCH DEVICE
Precondition 2
CURRENT MIDI PATCH DIRECTION contains

DEVICE TO WORKSTATION

Postcondition 2

ROUTER DEVICE SELECTION = CURRENT MIDI PATCH DEVICE
produced for DEVICE TO WORKSTATION MIDI ROUTER ADDRESS
from MIDI ROUTER ASSIGNMENTS
for DEVICE TO WORKSTATION MIDI ROUTER IDENTIFIER
corresponding to CURRENT MIDI PATCH WORKSTATION

6.3.4 Data Dictionary

```
current MIDI patch      = current MIDI patch workstation +
                          current MIDI patch direction +
                          current MIDI patch device

current MIDI patch      = * choice of sixteen local devices *
device                    * values [00000000 - 00001111 ¦
                          external], type : binary *

current MIDI patch      = * values : [device to workstation ¦
direction                 workstation to device] *

current MIDI patch      = * choice of eight local workstations *
workstation               * values : [00000000 - 00000111 ¦
                          external], type : binary *

device to workstation = * value : 01000000, type : binary *

device to workstation = * router port address *
MIDI router address   = * values : 10000000 00000000 -
                          00000001 11111111, type : binary *

device to workstation  = * choice of eight routers *
MIDI router identifier = * value : 00000000 - 00000111, type :
                           binary *

error detected in first MIDI patch byte = **

error detected in second MIDI patch byte = **

error in data           = * indicates the occurrence of an error
indicator                   state within the unit *
                        = * values : [on ¦ off] *

external                = * select MIDI input from external
                            MIDI Patch Unit *
```

```
                        = * value : 00010000, type : binary *

filtered MIDI byte      = * contains no MIDI real time messages *
                          * values : 00000000 - 11110111, type :
                            binary *

first audio patch byte received = **

MIDI byte               = * values : 00000000 - 11111111, type :
                            binary *

MIDI bytes              = 1{MIDI byte}

MIDI data indicator     = * indicates the receival of a MIDI
                            patch request via MIDI *
                        = * values : [on | off] *

MIDI patch direction mask = * value : 01100000, type :
                              binary *

MIDI patch received     = **

MIDI patch request      = * signal indicating MIDI patches
arriving                    arriving for this unit *

MIDI patch request end = **

MIDI patch request      = * specification of information in
specification                a MIDI patch request *
                        = sysex start + network device +
                          network device identifier address +
                          MIDI real time messages + sysex end +
                          external + workstation to device
                          device to workstation
                          MIDI patch direction mask +
                          MIDI patch workstation/device mask +
                          MIDI patch workstation limit + zero

MIDI patch workstation/device = * value : 00001111, type :
mask                            binary *

MIDI patch              = * Maximum of eight workstations *
workstation limit       = * value : 00000111, type : binary *

MIDI real time messages = * value : 11111000, type : binary *
```

```
MIDI router assignments      = external + 1{workstation to device
                               MIDI router address}16 +
                               1{device to workstation MIDI router
                               address}8
                               * See Appendix 2.4.5 *

network device               = * This is a device type identifier in
                               the network *
                             = * value : 01111101, type : binary *

network device identifier    = * network device number identifying a
                               particular unit *
                             = * values : 00000000 - 01111111
                               type : binary *

network device identifier address = * the port address to obtain the
                               network device identifier *
                             = * value : 10000000 11000001, type :
                               binary *

network device not received = **

network device received = **

reset unit      = * signal to initiate unit reinitialisation *

router device selection      = * device selection for device to
                               workstation routers *
                               * values [00000000 - 00001111 |
                               external], type : binary *

router workstation selection = * workstation selection for
                               workstation to device routers *
                               * values [00000000 - 00000111 |
                               external], type : binary *

sysex end                    = * value : 10000000, type : binary *

sysex start                  = * value : 11110000, type : binary *

sysex start received  = **

unexpected end received at second audio process byte = **

unit power on   = * signal to initiate unit initialisation *
```

```
workstation to device = * value : 00100000, type : binary *

workstation to device = * router port address *
MIDI router address   = * values : 10000000 00000000 -
                          00000001 11111111, type : binary *

workstation to device  = * choice of sixteen routers *
MIDI router identifier = * value : 00000000 - 00001111, type :
                           binary *

zero                   = * value : 00000000, type : binary *
```

6.3.5 Task Schema

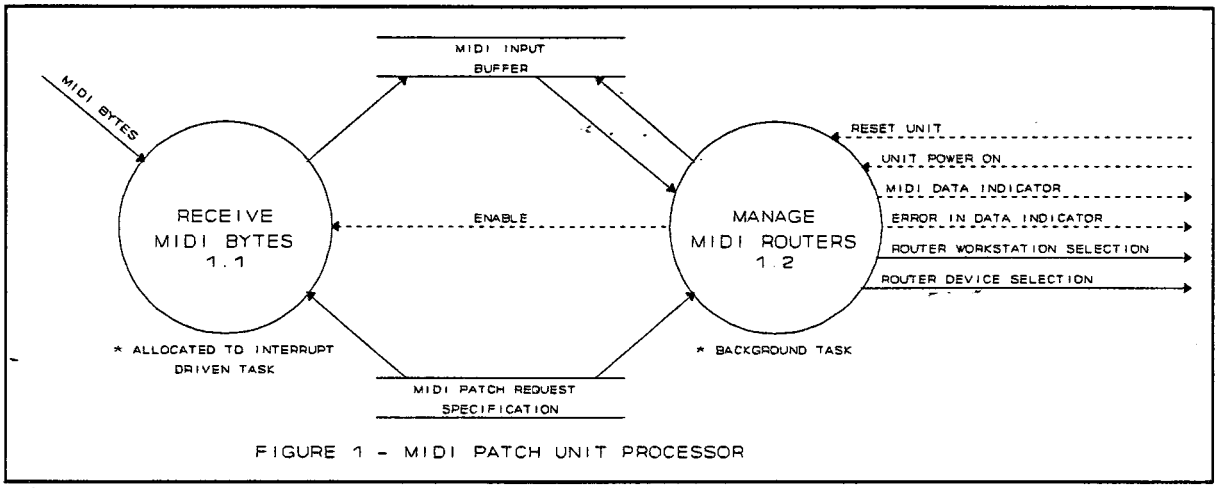

Figure 6.3.5 MIDI Patch Unit Processor

6.3.6 Structure Chart

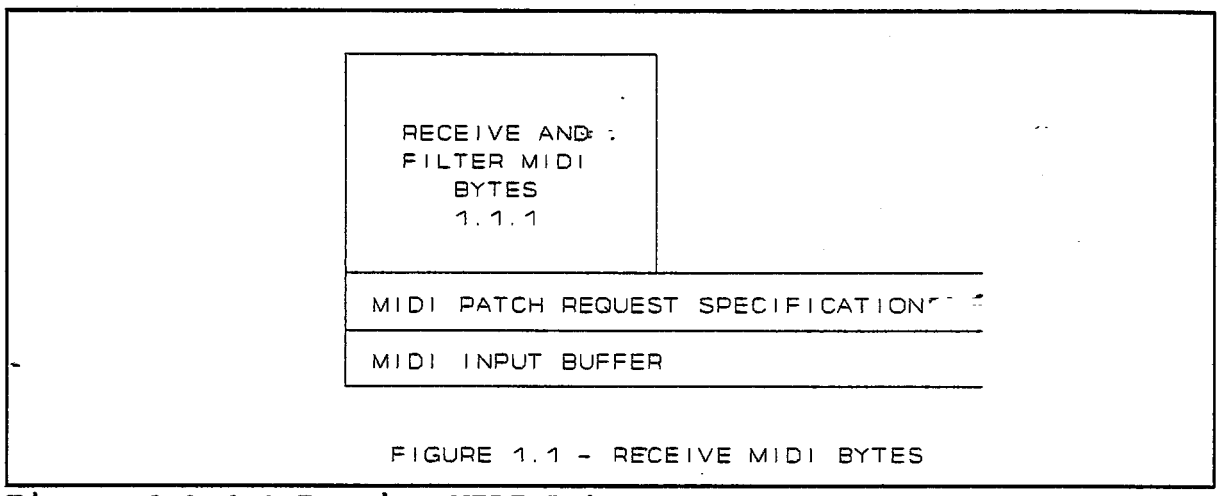

Figure 6.3.6.1 Receive MIDI Bytes

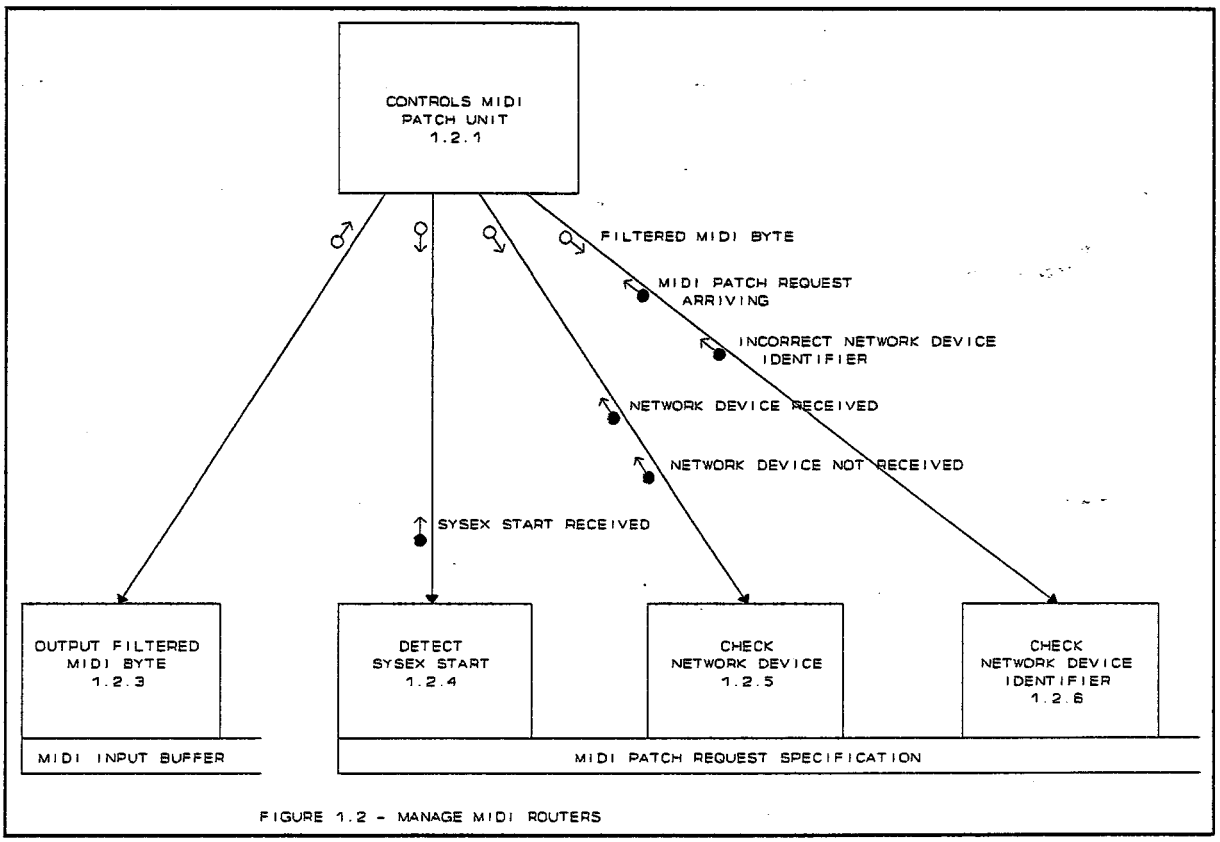

Figure 6.3.6.2 Manage MIDI Routers

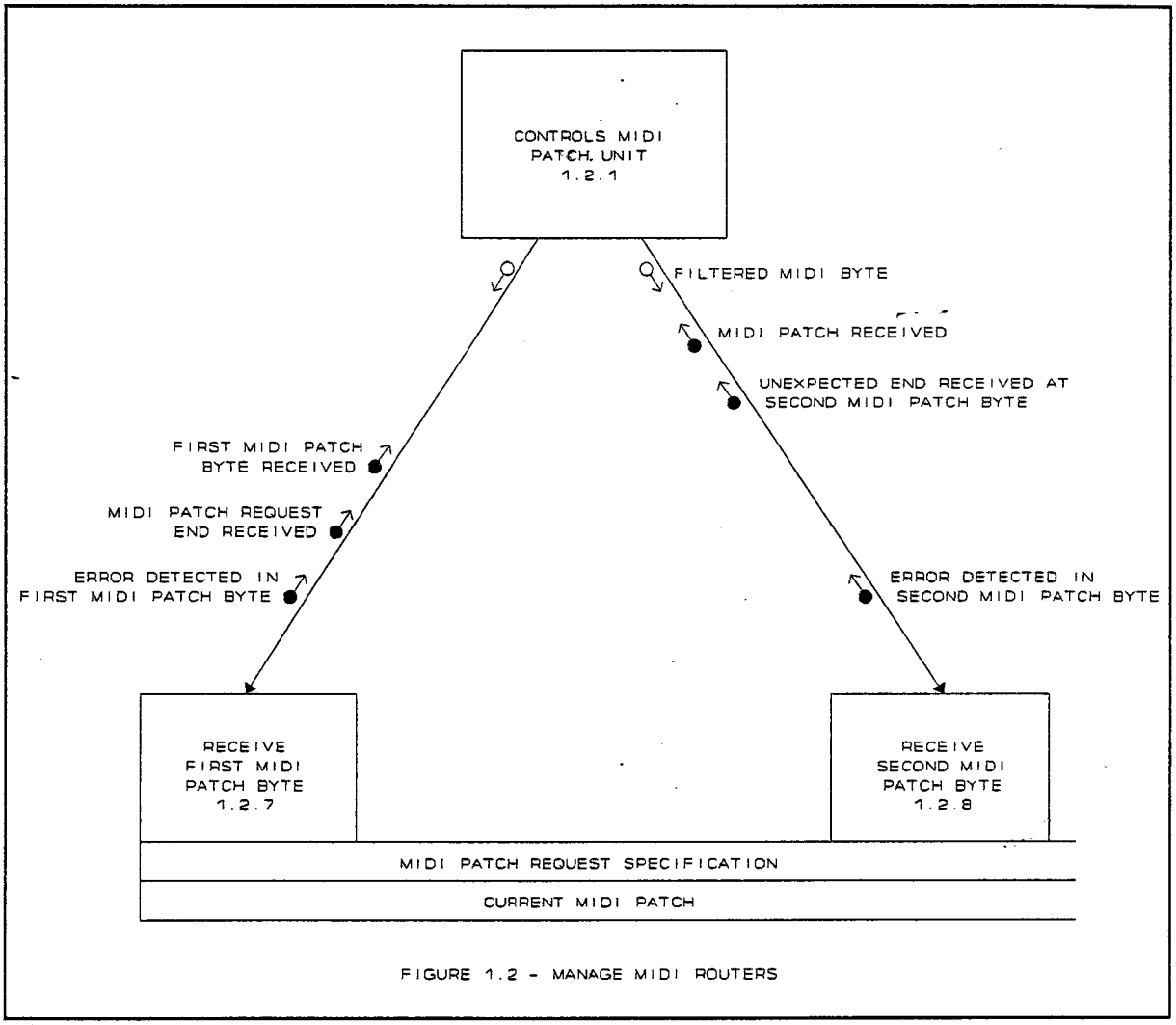

Figure 6.3.6.3 **Manage MIDI Routers**

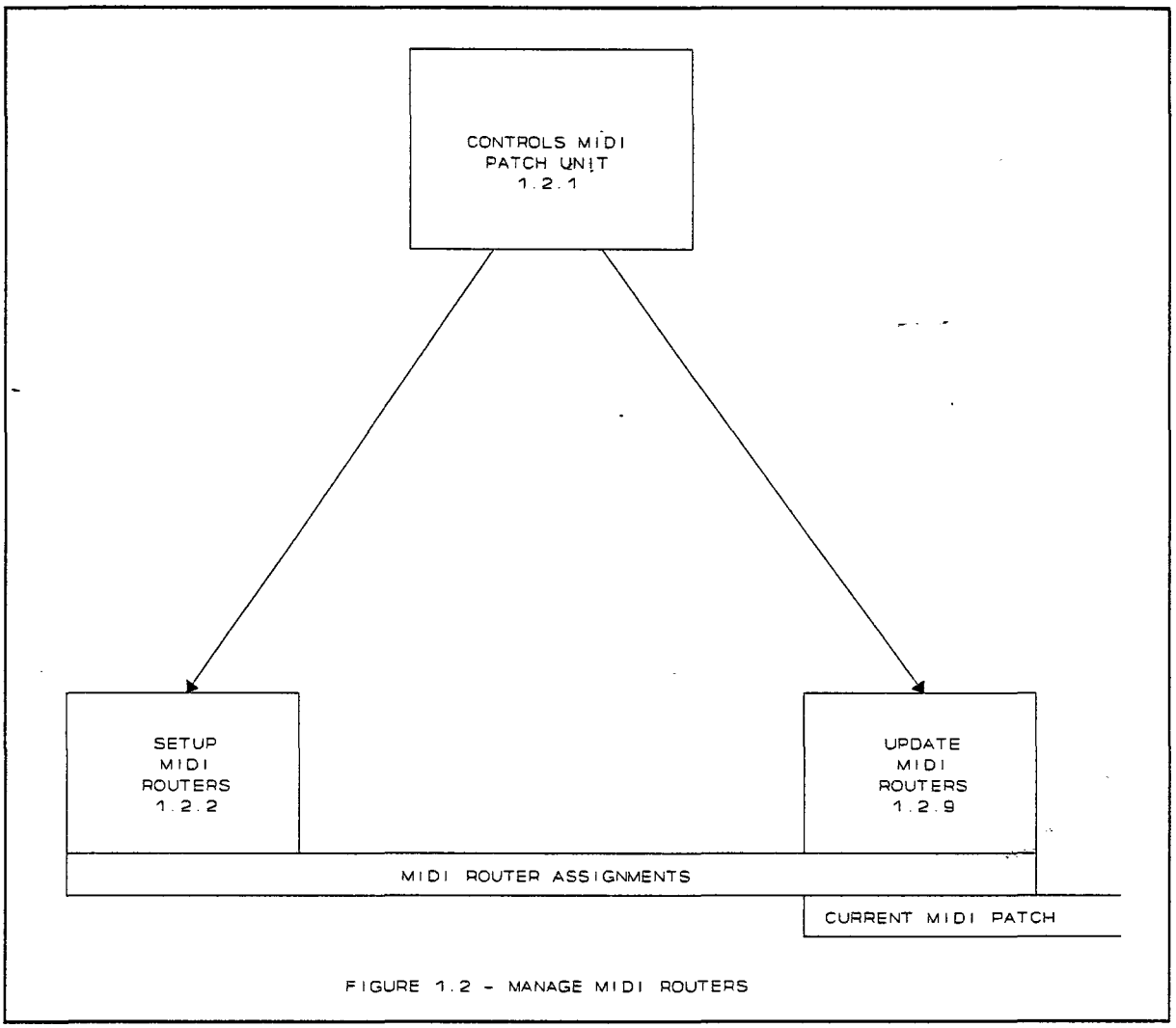

Figure 6.3.6.4 Manage MIDI Routers

6.4 The MIDI Input Buffer

6.4.1 Module Specifications

1.1.1 RECEIVE AND FILTER MIDI BYTES
Precondition 1
MIDI BYTE is less than MIDI REAL TIME MESSAGE
Postcondition 1
MIDI INPUT BUFFER ENTRY = MIDI BYTE of CIRCULAR BUFFER pointed to by MIDI INPUT BUFFER HEAD
and MIDI INPUT BUFFER HEAD points to next MIDI INPUT BUFFER ENTRY of CIRCULAR BUFFER
and MIDI INPUT BUFFER AVAILABLE ENTRIES incremented
Precondition 2
MIDI BYTE is greater than or equal to MIDI REAL TIME MESSAGE
Postcondition 2
None

1.2.3 OUTPUT FILTERED MIDI BYTES
Precondition 1
MIDI INPUT BUFFER AVAILABLE ENTRIES is greater than 0 occurs
Postcondition 1
FILTERED MIDI BYTE = MIDI INPUT BUFFER ENTRY of CIRCULAR BUFFER pointed to by MIDI INPUT BUFFER TAIL
and MIDI INPUT BUFFER TAIL points to next MIDI INPUT BUFFER ENTRY of CIRCULAR BUFFER
and MIDI INPUT BUFFER AVAILABLE ENTRIES decremented

6.4.2 Additions to Data Dictionary

```
MIDI input buffer head      = * pointer to head circular buffer *
                            = * values : 0 - 3, type : byte *

MIDI input buffer tail      = * pointer to circular buffer *
                            = * values : 0 - 3, type : byte *

circular buffer             = {MIDI input buffer entry}4

MIDI input buffer available entries
                            = * number of available filtered MIDI
                              bytes on the circular buffer *
                            = * values : 0 - 4, type : byte *
```

```
MIDI input buffer      = * single entry in circular buffer *
entry                    * values : 00000000 - 11110111,
                           type : binary *

MIDI input buffer      = circular buffer +
                         MIDI input buffer head +
                         MIDI input buffer tail +
                         MIDI input buffer available entries
```

APPENDIX 7 - SOFTWARE LISTINGS

TABLE OF CONTENTS

7.1 Computer Aided DCA Design Program

```
/*-------------------------------------------------------------------
 * DCA.C Program to calculate resistor values in
 *       Digitally Controlled Attenuator (DCA)
 * Written by : A.J.Wilks
 *       date : 27/03/1990
 *-------------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>

#define max_num_ladders 10 /* The maximum number of division ladders */
#define max_divisions 100 /* The maximum number of divisions in a ladder */

float Ra [max_divisions], Rb, Rc, Rd;      /* Resistor values required for calculations */
float R [max_num_ladders] [max_divisions]; /* Array containing final values */
float required_atten, step_size_atten;
float atten, atten_level, x;          /* Attenuation specifics */
int ladder, ladder_num;               /* Ladder specifics */
int chain_divisions, chain_num;       /* Division in ladder specifics */
FILE *output_file;

/*--- Local Prototypes ---*/

void main (int, char *[]);
void input_dca_specs (void);
void calculate_resistors (int);
void print_resistors (int);

/*--- Main Program to read, calculate and write data ---*/

void main (int argc, char *argv [])
 {
  if (argc != 2)
    printf ("\nOutput File? Usage : DCA OUTPUTFILE.EXTENSION\n");
  else
   {
    output_file = fopen (argv [1], "w");
    input_dca_specs ();
    for (ladder = 0; ladder < ladder_num; ladder++)
     {
      calculate_resistors (ladder);
      print_resistors (ladder);
```

```
    }
   fclose (output_file);
  }
 }

/*--- Prompt user to enter DCA specifications ---*/

void input_dca_specs (void)
 {
  do
   {
    printf ("Digitally Controlled Attenuator (DCA) Design Aid.\n\n");
    do
     {
      printf ("Required Attenuation Level in dBs : ");
      scanf ("%f", &required_atten);
     }
    while (required_atten >= 0);       /* Enter required attenuation */
    do
     {
      printf ("Attenuation step size in dBs : ");
      scanf ("%f", &step_size_atten);
     }
    while (step_size_atten >= 0);      /* Enter required attenuation step size */
    do
     {
      printf ("Number of divisions per ladder : ");
      scanf ("%d", &chain_divisions);
     }
    while (chain_divisions <= 1);      /* Enter number of divisions per ladder */
    do
     {
      printf ("Number of ladders in DCA : ");
      scanf ("%d", &ladder_num);
     }
    while (ladder_num <= 0);           /* Enter number of ladders in DCA */
    atten = (float) pow (chain_divisions, ladder_num) * step_size_atten;
    if ((atten - step_size_atten) > required_atten)
      printf ("Cannot meet required specifications!\n\n");
   }
  while ((atten - step_size_atten) > required_atten);
  do
   {
    printf ("Resistor Rc value for division chains in kOhms : ");
    scanf ("%f", &Rc);
   }
  while (Rc <= 0);                     /* Enter Rc value */
  do
   {
    printf ("Resistor Rd (Ra + Rb) value for division chains in kOhms : ");
    scanf ("%f", &Rd);
   }
  while (Rd <= 0);                     /* Enter Rd value */
  printf ("\n");
  fprintf (output_file,"Digitally Controlled Attenuator (DCA) Design Aid.\n\n");
  fprintf (output_file,"Total Attenuation Level = %3.5f dBs\n", (atten - step_size_atten));
  fprintf (output_file,"Number of divisions per Ladder = %d\n", chain_divisions);
  fprintf (output_file,"Number of Ladders in DCA = %d\n", ladder_num);
  fprintf (output_file,"Resistor Rc value for division chains = %3.3f kOhms\n", Rc);
  fprintf (output_file,"Resistor Rd (Ra + Rb) value for division chains = %3.3f kOhms\n\n", Rd);
```

```
}

/*--- Calculate resistor values for a ladder of the DCA ---*/

void calculate_resistors (int ladder)
 {
  atten_level = (float) atten/(pow (chain_divisions, (ladder_num - ladder)));
  for (chain_num = 0; chain_num < (chain_divisions - 1); chain_num++)
   {
    x = (float) pow(10,(((chain_num + 1) * atten_level)/20));
    Rb = (Rd - Rc/x + sqrt (((Rc/x - Rd) * (Rc/x - Rd)) + 4 * Rc * Rd))/2;
    Ra [chain_num] = Rd - Rb;
    if (chain_num == 0)
      R [ladder] [chain_num] = Ra [chain_num];
    else
      R [ladder] [chain_num] = Ra [chain_num] - Ra [chain_num - 1];
   }
  R [ladder] [(chain_divisions - 1)] = Rb;  /* Final resistor in network */
 }

/*--- Print resistor values for a ladder of the DCA ---*/

void print_resistors (int ladder)
 {
  fprintf (output_file,"Attenuation Level per division for Ladder %d (L%d) =
                        %2.5f dBs\n", (ladder + 1), (ladder + 1), atten_level);
  fprintf (output_file,"Resistor values for division chain:\n\n");
  for (chain_num = 0; chain_num < chain_divisions; chain_num++)
    fprintf (output_file,"R%2d = %2.5f kOhms\n", (chain_num + 1), R [ladder] [chain_num]);
  fprintf (output_file,"\n");
 }
```

7.2 Program Output for DCA Resistor Values

```
Digitally Controlled Attenuator (DCA) Design Aid.

Total Attenuation Level = -102.20000 dBs
Number of divisions per Ladder = 8
Number of Ladders in DCA = 3
Resistor Rc value for division chains = 10.000 kOhms
Resistor Rd (Ra + Rb) value for division chains = 10.000 kOhms

Attenuation Level per division for Ladder 1 (L1) = -0.20000 dBs
Resistor values for division chain:

R 1 = 0.11579 kOhms
R 2 = 0.11708 kOhms
R 3 = 0.11834 kOhms
R 4 = 0.11956 kOhms
R 5 = 0.12074 kOhms
R 6 = 0.12187 kOhms
R 7 = 0.12296 kOhms
R 8 = 9.16368 kOhms
```

Attenuation Level per division for Ladder 2 (L2) = -1.60000 dBs
Resistor values for division chain:

R 1 = 0.96031 kOhms
R 2 = 1.02187 kOhms
R 3 = 1.04808 kOhms
R 4 = 1.02920 kOhms
R 5 = 0.96422 kOhms
R 6 = 0.86323 kOhms
R 7 = 0.74352 kOhms
R 8 = 3.36957 kOhms

Attenuation Level per division for Ladder 3 (L3) = -12.80000 dBs
Resistor values for division chain:

R 1 = 7.25267 kOhms
R 2 = 2.19515 kOhms
R 3 = 0.43051 kOhms
R 4 = 0.09405 kOhms
R 5 = 0.02131 kOhms
R 6 = 0.00487 kOhms
R 7 = 0.00111 kOhms
R 8 = 0.00033 kOhms

7.3 The Audio Process Unit Software

```
/*-------------------------------------------------------------------------
 * APU51.C
 * Program to Perform Audio Processing.
 * This program resides upon an EPROM within the Audio Processor Unit.
 * Written by : A.J.Wilks
 *      Date : 20/9/1991
 *-------------------------------------------------------------------------*/

#include "reg51.h"

/*--- General Definitions ---*/

#define byte                    unsigned char
#define MIDI_byte               SBUF       /* MIDI byte in serial buffer */
#define XBYTE                   ((byte *) 0x20000L)

#define num_audio_channels      16

#define OFF                     1
#define ON                      0

#define FALSE                   0
#define TRUE                    1

/*--- Audio Process Request Specification ---*/
```

```
#define sysex_start                0xF0
#define network_device             0x7D     /* Reserved for research */
#define network_device_identifier_address  0x80C1
#define MIDI_real_time_messages    0xF8
#define sysex_end                  0x80
#define gain_process               0x10
#define bass_process               0x20
#define midrange_process           0x30
#define treble_process             0x40
#define audio_process_channel_mask 0x0F
#define audio_process_type_mask    0xF0

/*--- Audio Processor Assignments ---*/

#define initial_gain_level         77
#define initial_equalisation_level 64

/*--- MIDI Input Buffer ---*/

#define CIRC_BUFFER_SIZE         0x04
static data struct
 {
  byte entry [CIRC_BUFFER_SIZE];  /* Circular buffer array */
  byte head;                      /* Head of circular buffer */
  byte tail;                      /* Tail of circular buffer */
  byte available_entries;         /* number of available entries*/
 } MIDI_input_buffer;

/*--- Indicators ---*/

sbit MIDI_data_indicator = P1^0;     /* Hardware-Port for MIDI Data Indicator */
sbit error_in_data_indicator = P1^1; /* Hardware-Port for Error in Data Indicator */

/*--- Local Prototypes ---*/

void main (void);

void main () using 1
 {
  static data byte audio_channel_identifier;

  static data struct
   {
    byte channel;
    byte type;
    byte level;
   } current_audio_process;

  static data byte filtered_MIDI_byte;

  static data byte network_device_identifier;

  /*--- Gain Level Map ---*/

  static code unsigned int gain_level [128] =
   {
    0x0101,0x0103,0x0105,0x0107,0x0109,0x010B,0x010D,0x010F,
    0x0111,0x0113,0x0115,0x0117,0x0119,0x011B,0x011D,0x011F,
    0x0121,0x0123,0x0125,0x0127,0x0129,0x012B,0x012D,0x012F,
    0x0131,0x0133,0x0135,0x0137,0x0139,0x013B,0x013D,0x013F,
```

```
  0x0141,0x0143,0x0145,0x0147,0x0149,0x014B,0x014D,0x014F,
  0x0151,0x0153,0x0155,0x0157,0x0159,0x015B,0x015D,0x015F,
  0x0161,0x0163,0x0165,0x0167,0x0169,0x016B,0x016D,0x016F,
  0x0171,0x0173,0x0175,0x0177,0x0179,0x017B,0x017D,0x017F,
  0x0181,0x0183,0x0185,0x0187,0x0189,0x018B,0x018D,0x018F,
  0x0191,0x0193,0x0195,0x0197,0x0199,0x019B,0x019D,0x019F,
  0x01A1,0x01A3,0x01A5,0x01A7,0x01A9,0x01AB,0x01AD,0x01AF,
  0x01B1,0x01B3,0x01B5,0x01B7,0x01B9,0x01BB,0x01BD,0x01BF,
  0x01C1,0x01C3,0x01C5,0x01C7,0x01C9,0x01CB,0x01CD,0x01CF,
  0x01D1,0x01D3,0x01D5,0x01D7,0x01D9,0x01DB,0x01DD,0x01DF,
  0x01E1,0x01E3,0x01E5,0x01E7,0x01E9,0x01EB,0x01ED,0x01EF,
  0x01F1,0x01F3,0x01F5,0x01F7,0x01F9,0x01FB,0x01FD,0x01FF,
 };

/*--- Equalisation Level Map ---*/

static code unsigned int equalisation_level [128] =
 {
  0x0180,0x0181,0x0182,0x0183,0x0184,0x0185,0x0186,0x0187,
  0x0188,0x0189,0x018A,0x018B,0x018C,0x018D,0x018E,0x018F,
  0x0190,0x0191,0x0192,0x0193,0x0194,0x0195,0x0196,0x0197,
  0x0198,0x0199,0x019A,0x019B,0x019C,0x019D,0x019E,0x019F,
  0x01A0,0x01A1,0x01A2,0x01A3,0x01A4,0x01A5,0x01A6,0x01A7,
  0x01A8,0x01A9,0x01AA,0x01AB,0x01AC,0x01AD,0x01AE,0x01AF,
  0x01B0,0x01B1,0x01B2,0x01B3,0x01B4,0x01B5,0x01B6,0x01B7,
  0x01B8,0x01B9,0x01BA,0x01BB,0x01BC,0x01BD,0x01BE,0x01BF,
  0x01C0,0x01C1,0x01C2,0x01C3,0x01C4,0x01C5,0x01C6,0x01C7,
  0x01C8,0x01C9,0x01CA,0x01CB,0x01CC,0x01CD,0x01CE,0x01CF,
  0x01D0,0x01D1,0x01D2,0x01D3,0x01D4,0x01D5,0x01D6,0x01D7,
  0x01D8,0x01D9,0x01DA,0x01DB,0x01DC,0x01DD,0x01DE,0x01DF,
  0x01E0,0x01E1,0x01E2,0x01E3,0x01E4,0x01E5,0x01E6,0x01E7,
  0x01E8,0x01E9,0x01EA,0x01EB,0x01EC,0x01ED,0x01EE,0x01EF,
  0x01F0,0x01F1,0x01F2,0x01F3,0x01F4,0x01F5,0x01F6,0x01F7,
  0x01F8,0x01F9,0x01FA,0x01FB,0x01FC,0x01FD,0x01FE,0x01FF,
 };

static data byte *level_byte;

/*--- Audio Channel Processor Port Addresses ---*/

static code unsigned int gain_process_address [num_audio_channels] =
 {
  0x8003,0x8004,0x8005,0x8006,0x8007,0x8008,0x8009,0x800A,
  0x800B,0x800C,0x800D,0x800E,0x800F,0x8010,0x8011,0x8012
 };
static code unsigned int bass_process_address [num_audio_channels] =
 {
  0x8043,0x8046,0x8049,0x804C,0x804F,0x8052,0x8055,0x8058,
  0x8083,0x8086,0x8089,0x808C,0x808F,0x8092,0x8095,0x8098
 };
static code unsigned int midrange_process_address [num_audio_channels] =
 {
  0x8044,0x8047,0x804A,0x804D,0x8050,0x8053,0x8056,0x8059,
  0x8084,0x8087,0x808A,0x808D,0x8090,0x8093,0x8096,0x8099
 };
static code unsigned int treble_process_address [num_audio_channels] =
 {
  0x8045,0x8048,0x804B,0x804E,0x8051,0x8054,0x8057,0x805A,
  0x8085,0x8088,0x808B,0x808E,0x8091,0x8094,0x8097,0x809A
 };
```

```
static code unsigned int MSB_reg_gain     = 0x8001;
static code unsigned int LSB_reg_gain     = 0x8002;
static code unsigned int MSB_reg_eq_1to8  = 0x8041;
static code unsigned int LSB_reg_eq_1to8  = 0x8042;
static code unsigned int MSB_reg_eq_9to16 = 0x8081;
static code unsigned int LSB_reg_eq_9to16 = 0x8082;

/*--- Control flags ---*/

static data byte audio_process_received;
static data byte audio_process_request_arriving;
static data byte audio_process_request_end_received;
static data byte error_detected_in_process_byte;    /* For first byte */
static data byte first_audio_process_byte_received;
static data byte incorrect_network_device_identifier_received;
static data byte network_device_not_received;
static data byte network_device_received;
static data byte sysex_start_received;
static data byte unexpected_end_received;           /* For second byte */
static data byte reset_unit;                        /* Unit reset on power up */

P1 = 0xFF;                                          /* Output Port */

MIDI_data_indicator = ON; /* 1.2.2 */
error_in_data_indicator = ON; /* 1.2.2 */

/*--- Initialise control flags ---*/

audio_process_received = FALSE;
audio_process_request_arriving = FALSE;
audio_process_request_end_received = FALSE;
error_detected_in_process_byte = FALSE;
first_audio_process_byte_received = FALSE;
incorrect_network_device_identifier_received = FALSE;
network_device_not_received = FALSE;
network_device_received = FALSE;
sysex_start_received = FALSE;
unexpected_end_received = FALSE;
reset_unit = TRUE;

network_device_identifier = XBYTE [network_device_identifier_address];

/*--- Set gain to unity gain and equalisation flat 1.2.2 ---*/

level_byte = (byte *) &gain_level [initial_gain_level];
XBYTE [MSB_reg_gain] = *level_byte;
level_byte++;
XBYTE [LSB_reg_gain] = *level_byte;
level_byte = (byte *) &equalisation_level [initial_equalisation_level];
XBYTE [MSB_reg_eq_1to8] = *level_byte;
XBYTE [MSB_reg_eq_9to16] = *level_byte;
level_byte++;
XBYTE [LSB_reg_eq_1to8] = *level_byte;
XBYTE [LSB_reg_eq_9to16] = *level_byte;
for (audio_channel_identifier = 0; audio_channel_identifier < num_audio_channels;
     audio_channel_identifier++)
 {
  /*--- Strobe correct Ports ---*/

  XBYTE [gain_process_address [audio_channel_identifier]] = 0x00;
```

```
  XBYTE [bass_process_address [audio_channel_identifier]] = 0x00;
  XBYTE [midrange_process_address [audio_channel_identifier]] = 0x00;
  XBYTE [treble_process_address [audio_channel_identifier]] = 0x00;
 }

/*--- Initialise circular buffer ---*/

MIDI_input_buffer.head = 0;
MIDI_input_buffer.tail = 0;
MIDI_input_buffer.available_entries = 0;

/*--- Setup serial port control hardware (31250 BAUD @12MHZ) ---*/

TH1 =  0xFF;     /* 31250 Baud @ 12Mhz */
TMOD = 0x20;     /* Auto reload Timer 1 */
TR1 = 1;         /* Start Timer 1 */
SCON = 0x50;     /* 10 Bits */

ES = 1;          /* Enable Serial Interrupts */
EA = 1;          /* Global Interrupt enable */

MIDI_data_indicator = OFF; /* 1.2.2 */
error_in_data_indicator = OFF; /* 1.2.2 */

while (1)
 {
  /*--- Detect Sysex Start 1.2.4 ---*/

  if (reset_unit || unexpected_end_received ||
      audio_process_request_end_received || network_device_not_received ||
      incorrect_network_device_identifier_received)
   {
    reset_unit = FALSE;
    unexpected_end_received = FALSE;
    audio_process_request_end_received = FALSE;
    network_device_not_received = FALSE;
    incorrect_network_device_identifier_received = FALSE;
    do
     {
      /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

      while (!(MIDI_input_buffer.available_entries));
      filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
      MIDI_input_buffer.tail++;
      if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
        MIDI_input_buffer.tail = 0;
      MIDI_input_buffer.available_entries--;

      if (filtered_MIDI_byte == sysex_start)
        sysex_start_received = TRUE;
     }
    while (!sysex_start_received);
   }

  /*--- Check Network Device 1.2.5 ---*/

  if (sysex_start_received)
   {
    sysex_start_received = FALSE;
```

```
  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte == network_device)
    network_device_received = TRUE;
  else
    network_device_not_received = TRUE;
 }

/*--- Check Network Device Identifier 1.2.6 ---*/

if (network_device_received)
 {
  network_device_received = FALSE;

  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte == network_device_identifier)
   {
    audio_process_request_arriving = TRUE;
    MIDI_data_indicator = ON;
    error_in_data_indicator = OFF;
   }
  else
    incorrect_network_device_identifier_received = TRUE;
 }

/*--- Receive First Audio Process Byte 1.2.7 ---*/

if (audio_process_request_arriving ||
    audio_process_received ||
    error_detected_in_process_byte)
 {
  audio_process_request_arriving = FALSE;
  audio_process_received = FALSE;
  error_detected_in_process_byte = FALSE;

  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte < sysex_end)
```

```
    {
     current_audio_process.type = filtered_MIDI_byte & audio_process_type_mask;
     current_audio_process.channel = filtered_MIDI_byte &
                                     audio_process_channel_mask;
     if (current_audio_process.type == gain_process)
       first_audio_process_byte_received = TRUE;
     else if ((current_audio_process.type == bass_process) ||
              (current_audio_process.type == midrange_process) ||
              (current_audio_process.type == treble_process))
       first_audio_process_byte_received = TRUE;
     else
      {
       error_detected_in_process_byte = TRUE;
       error_in_data_indicator = ON;
      }
    }
   else
    {
     audio_process_request_end_received = TRUE;
     MIDI_data_indicator = OFF;
    }
  }

 /*--- Receive Second Audio Process Byte 1.2.8 ---*/

 if (first_audio_process_byte_received)
  {
   first_audio_process_byte_received = FALSE;

   /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

   while (!(MIDI_input_buffer.available_entries));
   filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
   MIDI_input_buffer.tail++;
   if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
     MIDI_input_buffer.tail = 0;
   MIDI_input_buffer.available_entries--;

   if (filtered_MIDI_byte < sysex_end)
    {
     current_audio_process.level = filtered_MIDI_byte;
     audio_process_received = TRUE;
    }
   else
    {
     unexpected_end_received = TRUE;
     error_in_data_indicator = ON;
     MIDI_data_indicator = OFF;
    }
  }

 /*--- Update Audio Processor 1.2.9 ---*/

 if (audio_process_received)
  {
   audio_channel_identifier = current_audio_process.channel;
   if (current_audio_process.type == gain_process)
    {
     level_byte = (byte *) &gain_level [current_audio_process.level];
     XBYTE [MSB_reg_gain] = *level_byte;
```

```
      level_byte++;
      XBYTE [LSB_reg_gain] = *level_byte;
      XBYTE [gain_process_address [audio_channel_identifier]] = 0x00;
     }
    else if (current_audio_process.type == bass_process)
     {
      if (audio_channel_identifier < 8) /* Split in two */
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_1to8] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_1to8] = *level_byte;
       }
      else
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_9to16] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_9to16] = *level_byte;
       }
      XBYTE [bass_process_address [audio_channel_identifier]] = 0x00;
     }
    else if (current_audio_process.type == midrange_process)
     {
      if (audio_channel_identifier < 8) /* Split in two */
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_1to8] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_1to8] = *level_byte;
       }
      else
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_9to16] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_9to16] = *level_byte;
       }
      XBYTE [midrange_process_address [audio_channel_identifier]] = 0x00;
     }
    else /* treble process */
     {
      if (audio_channel_identifier < 8) /* Split in two */
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_1to8] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_1to8] = *level_byte;
       }
      else
       {
        level_byte = (byte *) &equalisation_level [current_audio_process.level];
        XBYTE [MSB_reg_eq_9to16] = *level_byte;
        level_byte++;
        XBYTE [LSB_reg_eq_9to16] = *level_byte;
       }
      XBYTE [treble_process_address [audio_channel_identifier]] = 0x00;
     }
   }
}
```

```
}       /* End of main () */

void receive_and_filter_MIDI_bytes () interrupt 4 using 2  /* 1.1.1 */
{
 if (MIDI_byte < MIDI_real_time_messages)
  {
   MIDI_input_buffer.entry [MIDI_input_buffer.head] = MIDI_byte;
   MIDI_input_buffer.head++;
   if (MIDI_input_buffer.head == CIRC_BUFFER_SIZE)
     MIDI_input_buffer.head = 0;
   MIDI_input_buffer.available_entries++;
  }
 RI = 0;       /* End of Interrupt */
}
```

7.4 The Audio Patcher/Mixer Unit Software

```
/*-------------------------------------------------------------------------
 * APMU51.C
 * Program to Perform Audio Patching and Mixing.
 * This program resides upon an EPROM within the Audio Patcher/Mixer Unit.
 * Written by : A.J.Wilks
 *       Date : 19/9/1991
 *-------------------------------------------------------------------------*/

#include "reg51.h"

/*--- General Definitions ---*/

#define byte                    unsigned char
#define MIDI_byte               SBUF       /* MIDI byte in serial buffer */
#define XBYTE                   ((byte *) 0x20000L)

#define max_number_nodes        48

#define OFF                     1
#define ON                      0

#define FALSE                   0
#define TRUE                    1

#define EMPTY                   max_number_nodes

/*--- Audio Patch Request Specification ---*/

#define sysex_start             0xF0
#define network_device          0x7D       /* Reserved for research */
#define network_device_identifier_address  0x80C1
#define MIDI_real_time_messages  0xF8
#define sysex_end               0x80
#define connect_patch           0x10
#define change_level_patch      0x20
#define disconnect_patch        0x40
#define audio_patch_input_limit  16
#define audio_patch_output_mask  0x0F
#define audio_patch_type_mask    0xF0

/*--- Node Assignments ---*/
```

```
#define DISABLE                 0x00
#define ENABLE                  0x02
#define node_count_address      0x80C2

/*--- MIDI Input Buffer ---*/

#define CIRC_BUFFER_SIZE        0x04
static data struct
 {
  byte entry [CIRC_BUFFER_SIZE];      /* Circular buffer array */
  byte head;                          /* Head of circular buffer */
  byte tail;                          /* Tail of circular buffer */
  byte available_entries;             /* number of available entries*/
 } MIDI_input_buffer;

/*--- Indicators ---*/

sbit MIDI_data_indicator = P1^0;     /* Hardware-Port for MIDI Data Indicator */
sbit error_in_data_indicator = P1^1; /* Hardware-Port for Error in Data Indicator */

/*--- Local Prototypes ---*/

void main (void);

void main () using 1
 {
  static byte connected_node [16] [16];
  static data byte audio_patch_input_number, audio_patch_output_number;

  static byte disconnected_node [max_number_nodes];
  static data byte available_disconnected_node_ptr; /* implimented as a stack */

  static data byte node_identifier;

  static data struct
   {
    byte input_number;
    byte output_number;
    byte type;
    byte mix_level;
   } current_audio_patch;

  static data byte filtered_MIDI_byte;

  static data byte node_count;

  static data byte network_device_identifier;

  /*--- Mix Attenuation Level Map ---*/

  static code unsigned int attenuation_level [128] =
   {
    0x0003,0x001B,0x0033,0x004B,0x0063,0x007B,0x0093,0x00AB,
    0x00BB,0x00CB,0x00DB,0x00EB,0x00FB,0x0103,0x010B,0x0113,
    0x011B,0x0123,0x012B,0x0133,0x013B,0x0143,0x014B,0x014F,
    0x0153,0x0157,0x015B,0x015F,0x0163,0x0167,0x016B,0x016F,
    0x0173,0x0177,0x017B,0x017F,0x0183,0x0187,0x0189,0x018B,
    0x018D,0x018F,0x0191,0x0193,0x0195,0x0197,0x0199,0x019B,
    0x019D,0x019F,0x01A1,0x01A3,0x01A5,0x01A7,0x01A9,0x01AB,
    0x01AD,0x01AF,0x01B1,0x01B3,0x01B5,0x01B7,0x01B9,0x01BB,
```

```
  0x01BD,0x01BF,0x01C1,0x01C3,0x01C4,0x01C5,0x01C6,0x01C7,
  0x01C8,0x01C9,0x01CA,0x01CB,0x01CC,0x01CD,0x01CE,0x01CF,
  0x01D0,0x01D1,0x01D2,0x01D3,0x01D4,0x01D5,0x01D6,0x01D7,
  0x01D8,0x01D9,0x01DA,0x01DB,0x01DC,0x01DD,0x01DE,0x01DF,
  0x01E0,0x01E1,0x01E2,0x01E3,0x01E4,0x01E5,0x01E6,0x01E7,
  0x01E8,0x01E9,0x01EA,0x01EB,0x01EC,0x01ED,0x01EE,0x01EF,
  0x01F0,0x01F1,0x01F2,0x01F3,0x01F4,0x01F5,0x01F6,0x01F7,
  0x01F8,0x01F9,0x01FA,0x01FB,0x01FC,0x01FD,0x01FE,0x01FF
 };

static code unsigned int minimum_attenuation_level = 0x0000;

static data byte *attenuation_level_byte;

/*--- Node Port Addresses ---*/

static code unsigned int audio_patch_input_address [max_number_nodes] =
 {
  0x8003,0x8006,0x8009,0x800C,0x800F,0x8012,0x8015,0x8018,
  0x801B,0x801E,0x8021,0x8024,0x8027,0x802A,0x802D,0x8030,
  0x8043,0x8046,0x8049,0x804C,0x804F,0x8052,0x8055,0x8058,
  0x805B,0x805E,0x8061,0x8064,0x8067,0x806A,0x806D,0x8070,
  0x8083,0x8086,0x8089,0x808C,0x808F,0x8092,0x8095,0x8098,
  0x809B,0x809E,0x80A1,0x80A4,0x80A7,0x80AA,0x80AD,0x80B0
 };
static code unsigned int DCA_address [max_number_nodes] =
 {
  0x8004,0x8007,0x800A,0x800D,0x8010,0x8013,0x8016,0x8019,
  0x801C,0x801F,0x8022,0x8025,0x8028,0x802B,0x802E,0x8031,
  0x8044,0x8047,0x804A,0x804D,0x8050,0x8053,0x8056,0x8059,
  0x805C,0x805F,0x8062,0x8065,0x8068,0x806B,0x806E,0x8071,
  0x8084,0x8087,0x808A,0x808D,0x8090,0x8093,0x8096,0x8099,
  0x809C,0x809F,0x80A2,0x80A5,0x80A8,0x80AB,0x80AE,0x80B1
 };
static code unsigned int audio_patch_output_address [max_number_nodes] =
 {
  0x8005,0x8008,0x800B,0x800E,0x8011,0x8014,0x8017,0x801A,
  0x801D,0x8020,0x8023,0x8026,0x8029,0x802C,0x802F,0x8032,
  0x8045,0x8048,0x804B,0x804E,0x8051,0x8054,0x8057,0x805A,
  0x805D,0x8060,0x8063,0x8066,0x8069,0x806C,0x806F,0x8072,
  0x8085,0x8088,0x808B,0x808E,0x8091,0x8094,0x8097,0x809A,
  0x809D,0x80A0,0x80A3,0x80A6,0x80A9,0x80AC,0x80AF,0x80B2
 };
static code unsigned int MSB_reg_nodes_1to16  = 0x8001;
static code unsigned int LSB_reg_nodes_1to16  = 0x8002;
static code unsigned int MSB_reg_nodes_17to32 = 0x8041;
static code unsigned int LSB_reg_nodes_17to32 = 0x8042;
static code unsigned int MSB_reg_nodes_33to48 = 0x8081;
static code unsigned int LSB_reg_nodes_33to48 = 0x8082;

/*--- Control flags ---*/

static data byte audio_patch_received; /* For two and three byte audio patches */
static data byte audio_patch_request_arriving;
static data byte audio_patch_request_end_received;
static data byte error_detected_in_patch_byte; /* For first and second bytes */
static data byte first_audio_patch_byte_received;
static data byte incorrect_network_device_identifier_received;
static data byte network_device_not_received;
static data byte network_device_received;
```

```
static data byte second_audio_patch_byte_received;
static data byte sysex_start_received;
static data byte unexpected_end_received;      /* For second and third bytes */
static data byte reset_unit;                   /* Unit reset on power up */

P1 = 0xFF;                                     /* Output Port */

MIDI_data_indicator = ON; /* 1.2.2 */
error_in_data_indicator = ON; /* 1.2.2 */

/*--- Initialise control flags ---*/

audio_patch_received = FALSE;
audio_patch_request_arriving = FALSE;
audio_patch_request_end_received = FALSE;
error_detected_in_patch_byte = FALSE;
first_audio_patch_byte_received = FALSE;
incorrect_network_device_identifier_received = FALSE;
network_device_not_received = FALSE;
network_device_received = FALSE;
second_audio_patch_byte_received = FALSE;
sysex_start_received = FALSE;
unexpected_end_received = FALSE;
reset_unit = TRUE;

node_count = XBYTE [node_count_address];

network_device_identifier = XBYTE [network_device_identifier_address];

/*--- Disconnect all node inputs and outputs and set attenuation level to minimum
      1.2.2 ---*/

for (node_identifier = 0; node_identifier < max_number_nodes; node_identifier++)
 {
  if (node_identifier < 16) /* Nodes in groups of 16 */
   {
    attenuation_level_byte = (byte *) &minimum_attenuation_level;
    XBYTE [MSB_reg_nodes_1to16] = *attenuation_level_byte;
    attenuation_level_byte++;
    XBYTE [LSB_reg_nodes_1to16] = *attenuation_level_byte;
    XBYTE [DCA_address [node_identifier]] = 0x00; /* Strobe correct Port */
    XBYTE [MSB_reg_nodes_1to16] = DISABLE;
    XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
    XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
   }
  else if (node_identifier < 32)
   {
    attenuation_level_byte = (byte *) &minimum_attenuation_level;
    XBYTE [MSB_reg_nodes_17to32] = *attenuation_level_byte;
    attenuation_level_byte++;
    XBYTE [LSB_reg_nodes_17to32] = *attenuation_level_byte;
    XBYTE [DCA_address [node_identifier]] = 0x00;
    XBYTE [MSB_reg_nodes_17to32] = DISABLE;
    XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
    XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
   }
  else
   {
    attenuation_level_byte = (byte *) &minimum_attenuation_level;
    XBYTE [MSB_reg_nodes_33to48] = *attenuation_level_byte;
```

```
    attenuation_level_byte++;
    XBYTE [LSB_reg_nodes_33to48] = *attenuation_level_byte;
    XBYTE [DCA_address [node_identifier]] = 0x00;
    XBYTE [MSB_reg_nodes_33to48] = DISABLE;
    XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
    XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
   }
 }

/*--- Allocate all nodes to disconnected nodes 1.2.2 ---*/

for (available_disconnected_node_ptr = 0; available_disconnected_node_ptr <
     node_count; available_disconnected_node_ptr++)
  disconnected_node [available_disconnected_node_ptr] =
  available_disconnected_node_ptr;
available_disconnected_node_ptr = 0;
for (audio_patch_input_number = 0; audio_patch_input_number <
     audio_patch_input_limit; audio_patch_input_number++)
  for (audio_patch_output_number = 0; audio_patch_output_number <
       audio_patch_input_limit; audio_patch_output_number++)
    connected_node [audio_patch_input_number] [audio_patch_output_number] = EMPTY;

/*--- Initialise circular buffer ---*/

MIDI_input_buffer.head = 0;
MIDI_input_buffer.tail = 0;
MIDI_input_buffer.available_entries = 0;

/*--- Setup serial port control hardware (31250 BAUD @12MHZ) ---*/

TH1 =  0xFF;     /* 31250 Baud @ 12Mhz */
TMOD = 0x20;     /* Auto reload Timer 1 */
TR1 = 1;         /* Start Timer 1 */
SCON = 0x50;     /* 10 Bits */

ES = 1;          /* Enable Serial Interrupts */
EA = 1;          /* Global Interrupt enable */

MIDI_data_indicator = OFF; /* 1.2.2 */
error_in_data_indicator = OFF; /* 1.2.2 */

while (1)
 {
  /*--- Detect Sysex Start 1.2.4 ---*/

  if (reset_unit || unexpected_end_received ||
      audio_patch_request_end_received || network_device_not_received ||
      incorrect_network_device_identifier_received)
   {
    reset_unit = FALSE;
    unexpected_end_received = FALSE;
    audio_patch_request_end_received = FALSE;
    network_device_not_received = FALSE;
    incorrect_network_device_identifier_received = FALSE;
    do
     {
      /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

      while (!(MIDI_input_buffer.available_entries));
      filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
```

```
    MIDI_input_buffer.tail++;
    if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.tail = 0;
    MIDI_input_buffer.available_entries--;

    if (filtered_MIDI_byte == sysex_start)
      sysex_start_received = TRUE;
   }
  while (!sysex_start_received);
 }

/*--- Check Network Device 1.2.5 ---*/

if (sysex_start_received)
 {
  sysex_start_received = FALSE;

  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/
   /

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte == network_device)
    network_device_received = TRUE;
  else
    network_device_not_received = TRUE;
 }

/*--- Check Network Device Identifier 1.2.6 ---*/

if (network_device_received)
 {
  network_device_received = FALSE;

  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte == network_device_identifier)
   {
    audio_patch_request_arriving = TRUE;
    MIDI_data_indicator = ON;
    error_in_data_indicator = OFF;
   }
  else
    incorrect_network_device_identifier_received = TRUE;
 }

/*--- Receive First Audio Patch Byte 1.2.7 ---*/

if (audio_patch_request_arriving ||
```

```
    audio_patch_received ||
    error_detected_in_patch_byte)
  {
   audio_patch_request_arriving = FALSE;
   audio_patch_received = FALSE;
   error_detected_in_patch_byte = FALSE;

   /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

   while (!(MIDI_input_buffer.available_entries));
   filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
   MIDI_input_buffer.tail++;
   if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
     MIDI_input_buffer.tail = 0;
   MIDI_input_buffer.available_entries--;

   if (filtered_MIDI_byte < sysex_end)
    {
     if (filtered_MIDI_byte < audio_patch_input_limit)
      {
       current_audio_patch.input_number = filtered_MIDI_byte;
       first_audio_patch_byte_received = TRUE;
      }
     else
      {
       error_detected_in_patch_byte = TRUE;
       error_in_data_indicator = ON;
      }
    }
   else
    {
     audio_patch_request_end_received = TRUE;
     MIDI_data_indicator = OFF;
    }
  }

 /*--- Receive Second Audio Patch Byte 1.2.8 ---*/

 if (first_audio_patch_byte_received)
  {
   first_audio_patch_byte_received = FALSE;

   /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

   while (!(MIDI_input_buffer.available_entries));
   filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
   MIDI_input_buffer.tail++;
   if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
     MIDI_input_buffer.tail = 0;
   MIDI_input_buffer.available_entries--;

   if (filtered_MIDI_byte < sysex_end)
    {
     current_audio_patch.type = filtered_MIDI_byte & audio_patch_type_mask;
     current_audio_patch.output_number = filtered_MIDI_byte &
                                         audio_patch_output_mask;
     if (current_audio_patch.type == change_level_patch)
       second_audio_patch_byte_received = TRUE;
     else if (current_audio_patch.type == connect_patch)
       second_audio_patch_byte_received = TRUE;
```

```
      else if (current_audio_patch.type == disconnect_patch)
        audio_patch_received = TRUE;
      else
       {
        error_detected_in_patch_byte = TRUE;
        error_in_data_indicator = ON;
       }
     }
    else
     {
      unexpected_end_received = TRUE;
      error_in_data_indicator = ON;
      MIDI_data_indicator = OFF;
     }
   }

  /*--- Receive Third Audio Patch Byte 1.2.9 ---*/

  if (second_audio_patch_byte_received)
   {
    second_audio_patch_byte_received = FALSE;

    /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

    while (!(MIDI_input_buffer.available_entries));
    filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
    MIDI_input_buffer.tail++;
    if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.tail = 0;
    MIDI_input_buffer.available_entries--;
    if (filtered_MIDI_byte < sysex_end)
     {
      current_audio_patch.mix_level = filtered_MIDI_byte;
      audio_patch_received = TRUE;
     }
    else
     {
      unexpected_end_received = TRUE;
      error_in_data_indicator = ON;
      MIDI_data_indicator = OFF;
     }
   }

  /*--- Update Node 1.2.10 ---*/

  if (audio_patch_received)
   {
    if (current_audio_patch.type == change_level_patch)
     {
      if ((node_identifier = connected_node [current_audio_patch.input_number]
                             [current_audio_patch.output_number]) != EMPTY)
       {
        if (node_identifier < 16) /* Nodes in groups of 16 */
         {
          attenuation_level_byte = (byte *) &attenuation_level
                                   [current_audio_patch.mix_level];
          XBYTE [MSB_reg_nodes_1to16] = *attenuation_level_byte;
          attenuation_level_byte++;
          XBYTE [LSB_reg_nodes_1to16] = *attenuation_level_byte;
          XBYTE [DCA_address [node_identifier]] = 0x00;
```

```
    }
   else if (node_identifier < 32)
    {
     attenuation_level_byte = (byte *) &attenuation_level
                              [current_audio_patch.mix_level];
     XBYTE [MSB_reg_nodes_17to32] = *attenuation_level_byte;
     attenuation_level_byte++;
     XBYTE [LSB_reg_nodes_17to32] = *attenuation_level_byte;
     XBYTE [DCA_address [node_identifier]] = 0x00;
    }
   else
    {
     attenuation_level_byte = (byte *) &attenuation_level
                              [current_audio_patch.mix_level];
     XBYTE [MSB_reg_nodes_33to48] = *attenuation_level_byte;
     attenuation_level_byte++;
     XBYTE [LSB_reg_nodes_33to48] = *attenuation_level_byte;
     XBYTE [DCA_address [node_identifier]] = 0x00;
    }
  }
  else
    error_in_data_indicator = ON;
 }
else if (current_audio_patch.type == connect_patch)
 {
  if ((connected_node [current_audio_patch.input_number]
       [current_audio_patch.output_number] == EMPTY) &&
     (available_disconnected_node_ptr < node_count))
   {
    node_identifier = disconnected_node [available_disconnected_node_ptr];
    if (node_identifier < 16) /* Nodes in groups of 16 */
     {
      XBYTE [MSB_reg_nodes_1to16] = ENABLE;
      XBYTE [LSB_reg_nodes_1to16] = current_audio_patch.input_number;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
      XBYTE [LSB_reg_nodes_1to16] = current_audio_patch.output_number;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      attenuation_level_byte = (byte *) &attenuation_level
                               [current_audio_patch.mix_level];
      XBYTE [MSB_reg_nodes_1to16] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_1to16] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
     }
    else if (node_identifier < 32)
     {
      XBYTE [MSB_reg_nodes_17to32] = ENABLE;
      XBYTE [LSB_reg_nodes_17to32] = current_audio_patch.input_number;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
      XBYTE [LSB_reg_nodes_17to32] = current_audio_patch.output_number;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      attenuation_level_byte = (byte *) &attenuation_level
                               [current_audio_patch.mix_level];
      XBYTE [MSB_reg_nodes_17to32] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_17to32] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
     }
    else
     {
```

```
      XBYTE [MSB_reg_nodes_33to48] = ENABLE;
      XBYTE [LSB_reg_nodes_33to48] = current_audio_patch.input_number;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
      XBYTE [LSB_reg_nodes_33to48] = current_audio_patch.output_number;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      attenuation_level_byte = (byte *) &attenuation_level
                               [current_audio_patch.mix_level];
      XBYTE [MSB_reg_nodes_33to48] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_33to48] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
     }
    connected_node [current_audio_patch.input_number]
                   [current_audio_patch.output_number] = node_identifier;
    available_disconnected_node_ptr++;
   }
  else
    error_in_data_indicator = ON;
 }
else    /* disconnect patch */
 {
  if ((node_identifier = connected_node [current_audio_patch.input_number]
       [current_audio_patch.output_number]) != EMPTY)
   {
    if (node_identifier < 16) /* Nodes in groups of 16 */
     {
      attenuation_level_byte = (byte *) &minimum_attenuation_level;
      XBYTE [MSB_reg_nodes_1to16] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_1to16] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
      XBYTE [MSB_reg_nodes_1to16] = DISABLE;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
     }
    else if (node_identifier < 32)
     {
      attenuation_level_byte = (byte *) &minimum_attenuation_level;
      XBYTE [MSB_reg_nodes_17to32] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_17to32] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
      XBYTE [MSB_reg_nodes_17to32] = DISABLE;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
     }
    else
     {
      attenuation_level_byte = (byte *) &minimum_attenuation_level;
      XBYTE [MSB_reg_nodes_33to48] = *attenuation_level_byte;
      attenuation_level_byte++;
      XBYTE [LSB_reg_nodes_33to48] = *attenuation_level_byte;
      XBYTE [DCA_address [node_identifier]] = 0x00;
      XBYTE [MSB_reg_nodes_33to48] = DISABLE;
      XBYTE [audio_patch_output_address [node_identifier]] = 0x00;
      XBYTE [audio_patch_input_address [node_identifier]] = 0x00;
     }
    available_disconnected_node_ptr--;
    disconnected_node [available_disconnected_node_ptr] =
    connected_node [current_audio_patch.input_number]
```

```
                        [current_audio_patch.output_number];
          connected_node [current_audio_patch.input_number]
                        [current_audio_patch.output_number] = EMPTY;
         }
        else
          error_in_data_indicator = ON;
       }
     }
   }
 }       /* End of main () */

void receive_and_filter_MIDI_bytes () interrupt 4 using 2  /* 1.1.1 */
 {
  if (MIDI_byte < MIDI_real_time_messages)
   {
    MIDI_input_buffer.entry [MIDI_input_buffer.head] = MIDI_byte;
    MIDI_input_buffer.head++;
    if (MIDI_input_buffer.head == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.head = 0;
    MIDI_input_buffer.available_entries++;
   }
  RI = 0;       /* End of Interrupt */
 }
```

7.5 The MIDI Patch Unit Software

```
/*-------------------------------------------------------------------------
 * MPU51.C
 * Program to Perform MIDI Patching.
 * This program resides upon an EPROM within the MIDI Patch Unit.
 * Written by : A.J.Wilks
 *       Date : 20/9/1991
 *-------------------------------------------------------------------------*/

#include "reg51.h"

/*--- General Definitions ---*/

#define byte                    unsigned char
#define MIDI_byte               SBUF      /* MIDI byte in serial buffer */
#define XBYTE                   ((byte *) 0x20000L)

#define num_workstation_to_device_routers  16
#define num_device_to_workstation_routers  8

#define OFF                     1
#define ON                      0

#define FALSE                   0
#define TRUE                    1

/*--- MIDI Patch Request Specification ---*/

#define sysex_start                         0xF0
#define network_device                      0x7D /* Reserved for research */
#define network_device_identifier_address   0x80C1
#define MIDI_real_time_messages             0xF8
#define sysex_end                           0x80
#define external                            0x10
```

```
#define workstation_to_device               0x20
#define device_to_workstation               0x40
#define MIDI_patch_direction_mask           0xE0
#define MIDI_patch_workstation_device_mask  0x0F
#define MIDI_patch_workstation_limit        0x08
#define zero                                0x00

/*--- MIDI Input Buffer ---*/

#define CIRC_BUFFER_SIZE        0x04
static data struct
 {
  byte entry [CIRC_BUFFER_SIZE];  /* Circular buffer array */
  byte head;                      /* Head of circular buffer */
  byte tail;                      /* Tail of circular buffer */
  byte available_entries;         /* number of available entries*/
 } MIDI_input_buffer;

/*--- Indicators ---*/

sbit MIDI_data_indicator = P1^0;       /* Hardware-Port for MIDI Data Indicator */
sbit error_in_data_indicator = P1^1;   /* Hardware-Port for Error in Data Indicator */

/*--- Local Prototypes ---*/

void main (void);

void main () using 1
 {
  static data byte workstation_to_device_router_identifier;
  static data byte device_to_workstation_router_identifier;

  static data struct
   {
    byte workstation;
    byte direction;
    byte device;
   } current_MIDI_patch;

  static data byte filtered_MIDI_byte;

  static data byte network_device_identifier;

  /*--- Workstation to Device MIDI Router Port Addresses ---*/

  static code unsigned int workstation_to_device_router_address
                           [num_workstation_to_device_routers] =
   {
    0x8003,0x8004,0x8005,0x8006,0x8007,0x8008,0x8009,0x800A,
    0x8043,0x8044,0x8045,0x8046,0x8047,0x8048,0x8049,0x804A
   };

  /*--- Device to Workstation MIDI Router Port Addresses ---*/

  static code unsigned int device_to_workstation_router_address
                           [num_device_to_workstation_routers] =
   {
    0x8083,0x8084,0x8085,0x8086,0x8087,0x8088,0x8089,0x808A,
   };
  static code unsigned int reg_1to8_workstation_to_device_routers  = 0x8001;
```

```
static code unsigned int reg_9to16_workstation_to_device_routers = 0x8041;
static code unsigned int reg_device_to_workstation_routers       = 0x8081;

/*--- Control flags ---*/

static data byte error_detected_in_patch_byte; /* For first and second bytes */
static data byte first_MIDI_patch_byte_received;
static data byte incorrect_network_device_identifier_received;
static data byte MIDI_patch_received;
static data byte MIDI_patch_request_arriving;
static data byte MIDI_patch_request_end_received;
static data byte network_device_not_received;
static data byte network_device_received;
static data byte sysex_start_received;
static data byte unexpected_end_received;          /* For second byte */
static data byte reset_unit;                       /* Unit reset on power up */

P1 = 0xFF;                                         /* Output Port */

MIDI_data_indicator = ON; /* 1.2.2 */
error_in_data_indicator = ON; /* 1.2.2 */

/*--- Initialise control flags ---*/

error_detected_in_patch_byte = FALSE;
first_MIDI_patch_byte_received = FALSE;
incorrect_network_device_identifier_received = FALSE;
MIDI_patch_received = FALSE;
MIDI_patch_request_arriving = FALSE;
MIDI_patch_request_end_received = FALSE;
network_device_not_received = FALSE;
network_device_received = FALSE;
sysex_start_received = FALSE;
unexpected_end_received = FALSE;
reset_unit = TRUE;

network_device_identifier = XBYTE [network_device_identifier_address];

/*--- Set MIDI routers to external 1.2.2 ---*/

XBYTE [reg_1to8_workstation_to_device_routers]  = external;
XBYTE [reg_9to16_workstation_to_device_routers] = external;
XBYTE [reg_device_to_workstation_routers] = external;
for (workstation_to_device_router_identifier = 0;
     workstation_to_device_router_identifier < num_workstation_to_device_routers;
     workstation_to_device_router_identifier++)
  XBYTE [workstation_to_device_router_address
         [workstation_to_device_router_identifier]] = 0x00;
for (device_to_workstation_router_identifier = 0;
     device_to_workstation_router_identifier < num_device_to_workstation_routers;
     device_to_workstation_router_identifier++)
  XBYTE [device_to_workstation_router_address
         [device_to_workstation_router_identifier]] = 0x00;

/*--- Initialise circular buffer ---*/

MIDI_input_buffer.head = 0;
MIDI_input_buffer.tail = 0;
MIDI_input_buffer.available_entries = 0;
```

```
/*--- Setup serial port control hardware (31250 BAUD @12MHZ) ---*/

TH1 =  0xFF;     /* 31250 Baud @ 12Mhz */
TMOD = 0x20;     /* Auto reload Timer 1 */
TR1 = 1;         /* Start Timer 1 */
SCON = 0x50;     /* 10 Bits */

ES = 1;          /* Enable Serial Interrupts */
EA = 1;          /* Global Interrupt enable */

MIDI_data_indicator = OFF; /* 1.2.2 */
error_in_data_indicator = OFF; /* 1.2.2 */

while (1)
 {
  /*--- Detect Sysex Start 1.2.4 ---*/

  if (reset_unit || unexpected_end_received ||
      MIDI_patch_request_end_received || network_device_not_received ||
      incorrect_network_device_identifier_received)
   {
    reset_unit = FALSE;
    unexpected_end_received = FALSE;
    MIDI_patch_request_end_received = FALSE;
    network_device_not_received = FALSE;
    incorrect_network_device_identifier_received = FALSE;
    do
     {
      /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

      while (!(MIDI_input_buffer.available_entries));
      filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
      MIDI_input_buffer.tail++;
      if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
        MIDI_input_buffer.tail = 0;
      MIDI_input_buffer.available_entries--;

      if (filtered_MIDI_byte == sysex_start)
        sysex_start_received = TRUE;
     }
    while (!sysex_start_received);
   }

  /*--- Check Network Device 1.2.5 ---*/

  if (sysex_start_received)
   {
    sysex_start_received = FALSE;

    /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

    while (!(MIDI_input_buffer.available_entries));
    filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
    MIDI_input_buffer.tail++;
    if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.tail = 0;
    MIDI_input_buffer.available_entries--;

    if (filtered_MIDI_byte == network_device)
      network_device_received = TRUE;
```

```
    else
      network_device_not_received = TRUE;
   }

  /*--- Check Network Device Identifier 1.2.6 ---*/

  if (network_device_received)
   {
    network_device_received = FALSE;

    /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

    while (!(MIDI_input_buffer.available_entries));
    filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
    MIDI_input_buffer.tail++;
    if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.tail = 0;
    MIDI_input_buffer.available_entries--;

    if (filtered_MIDI_byte == network_device_identifier)
     {
      MIDI_patch_request_arriving = TRUE;
      MIDI_data_indicator = ON;
      error_in_data_indicator = OFF;
     }
    else
      incorrect_network_device_identifier_received = TRUE;
   }

  /*--- Receive First MIDI Patch Byte 1.2.7 ---*/

  if (MIDI_patch_request_arriving ||
      MIDI_patch_received ||
      error_detected_in_patch_byte)
   {
    MIDI_patch_request_arriving = FALSE;
    MIDI_patch_received = FALSE;
    error_detected_in_patch_byte = FALSE;

    /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

    while (!(MIDI_input_buffer.available_entries));
    filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
    MIDI_input_buffer.tail++;
    if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.tail = 0;
    MIDI_input_buffer.available_entries--;

    if (filtered_MIDI_byte < sysex_end)
     {
      current_MIDI_patch.direction = filtered_MIDI_byte &
                                     MIDI_patch_direction_mask;
      if ((current_MIDI_patch.direction == workstation_to_device) ||
          (current_MIDI_patch.direction == device_to_workstation))
       {
        current_MIDI_patch.workstation = filtered_MIDI_byte &
                                         MIDI_patch_workstation_device_mask;
        if (filtered_MIDI_byte & external == zero)
         {
          if (current_MIDI_patch.workstation < MIDI_patch_workstation_limit)
```

```
            first_MIDI_patch_byte_received = TRUE;
          else
           {
            error_detected_in_patch_byte = TRUE;
            error_in_data_indicator = ON;
           }
         }
        else
         {
          if (current_MIDI_patch.workstation == zero)
           {
            current_MIDI_patch.workstation = external;
            first_MIDI_patch_byte_received = TRUE;
           }
          else
           {
            error_detected_in_patch_byte = TRUE;
            error_in_data_indicator = ON;
           }
         }
       }
      else
       {
        error_detected_in_patch_byte = TRUE;
        error_in_data_indicator = ON;
       }
     }
    else
     {
      MIDI_patch_request_end_received = TRUE;
      MIDI_data_indicator = OFF;
     }
  }

/*--- Receive Second MIDI Patch Byte 1.2.8 ---*/

if (first_MIDI_patch_byte_received)
 {
  first_MIDI_patch_byte_received = FALSE;

  /*--- Output Filtered MIDI Bytes 1.2.3 ---*/

  while (!(MIDI_input_buffer.available_entries));
  filtered_MIDI_byte = MIDI_input_buffer.entry [MIDI_input_buffer.tail];
  MIDI_input_buffer.tail++;
  if (MIDI_input_buffer.tail == CIRC_BUFFER_SIZE)
    MIDI_input_buffer.tail = 0;
  MIDI_input_buffer.available_entries--;

  if (filtered_MIDI_byte < sysex_end)
   {
    if ((filtered_MIDI_byte & MIDI_patch_direction_mask) == zero)
     {
      current_MIDI_patch.device = filtered_MIDI_byte &
                                  MIDI_patch_workstation_device_mask;
      if ((filtered_MIDI_byte & external) == zero)
        MIDI_patch_received = TRUE;
      else if ((current_MIDI_patch.device == zero) &&
               (current_MIDI_patch.direction == device_to_workstation))
       {
```

```
            current_MIDI_patch.device = external;
            MIDI_patch_received = TRUE;
           }
          else
           {
            error_detected_in_patch_byte = TRUE;
            error_in_data_indicator = ON;
           }
         }
        else
         {
          error_detected_in_patch_byte = TRUE;
          error_in_data_indicator = ON;
         }
       }
      else
       {
        unexpected_end_received = TRUE;
        error_in_data_indicator = ON;
        MIDI_data_indicator = OFF;
       }
     }

    /*--- Update MIDI Router 1.2.9 ---*/

    if (MIDI_patch_received)
     {
      if (current_MIDI_patch.direction == workstation_to_device)
       {
        workstation_to_device_router_identifier = current_MIDI_patch.workstation;
        if (workstation_to_device_router_identifier < 8) /* Split in two */
          XBYTE [reg_1to8_workstation_to_device_routers] = current_MIDI_patch.device;
        else
          XBYTE [reg_9to16_workstation_to_device_routers] =
          current_MIDI_patch.device;
        XBYTE [workstation_to_device_router_address
              [workstation_to_device_router_identifier]] = 0x00;
       }
      else /* device to workstation patch */
       {
        device_to_workstation_router_identifier = current_MIDI_patch.device;
        XBYTE [reg_device_to_workstation_routers] = current_MIDI_patch.workstation;
        XBYTE [device_to_workstation_router_address
              [device_to_workstation_router_identifier]] = 0x00;
       }
     }
   }
 }       /* End of main () */

void receive_and_filter_MIDI_bytes () interrupt 4 using 2  /* 1.1.1 */
 {
  if (MIDI_byte < MIDI_real_time_messages)
   {
    MIDI_input_buffer.entry [MIDI_input_buffer.head] = MIDI_byte;
    MIDI_input_buffer.head++;
    if (MIDI_input_buffer.head == CIRC_BUFFER_SIZE)
      MIDI_input_buffer.head = 0;
    MIDI_input_buffer.available_entries++;
   }
  RI = 0;}      /* End of Interrupt */
```

APPENDIX 8 - GLOSSARY

A/D Converter - Analogue to digital converter, it converts analogue signals to digital equivalents.
AES/EBU - Audio Engineering Society/ European Broadcast Union, a term usually used to describe a specification for the transmission of single, dual and stereo digital audio signals.
APPM - Audio Processor Patcher Mixer, a system developed within this thesis for the real time processing, routing and mixing of audio signals.
APMU - Audio Patcher/Mixer Unit, the unit responsible for the routing and mixing of audio signals within the APPM.
APU - Audio Processor Unit, the unit responsible for gain and equalisation control within the APPM.
ATM - Asynchronous Transfer Mode, a high speed LAN.
Arconet - Artist's Computer Network, a computer music network proposal.
audio device - a device responsible for generating, processing or recording audio.
audio generator - an audio device responsible for the generation of sound.
audio interconnects - connections between audio devices to carry audio signals.
audio mixer - an audio modifier which allows audio signals to be merged.
audio modifier - an audio device responsible for the manipulation of sound produced by audio generators.
audio patch bay - an audio modifier which allows easy rerouting of audio signals between audio devices.
audio patcher/mixer node - the building block of the APMU.
audio recorder - an audio device responsible for storing of sound.
audio track - the physical area on the recording medium where sound is recorded.
behavioral model - describes how a system behaves in response to events.
computer music network - a network to provide users shared access to the resources of a computer music studio.
computer music studio - a music studio in which computers are used to generate, process and record sound.
context schema - describes the boundary between a system and its environment, and the interfaces between the two.
D/A converter - Digital to Analogue converter, it converts digital signals to analogue equivalents.
DCA - Digitally controlled attenuator, developed in this thesis

for the purpose of audio level control within the APPM.
DOS - Disk Operating System, an operating system for IBM personal computers and compatibles.
EPROM - Erasable Programmable Read Only Memory, is non-volatile data memory.
EOX - End Of Exclusive, the end of a MIDI System Exclusive message.
environmental model - describes the boundary between the system and its environment, the interfaces between the two, and the events in the environment to which the system must respond.
essential model - describes what the system should must do in response to events in its environment.
ethernet - a medium speed LAN.
FDDI - Fibre Distributed Data Interface, a high speed LAN.
graphic equaliser - divides the audio spectrum into several, set centre frequency, set bandwidth, boost/cut controls.
HCMOS - High-speed Complementary Metal Oxide Semiconductor, a family of logic devices.
Implementation model - is derived by doing a top down allocation of the essential model to the implementation technology.
LAN - Local Area Network, allows the communication of data between nodes. Usually the total length of the network will not exceed 2 kms.
LTC SMPTE - Longitudinal Time Code, is the audio form of SMPTE.
LS - Low-power Schottky, a high speed, low power requirement form of TTL.
MADI - Multichannel Audio Digital Interface, a standard for the transmission of multiple digital audio channels down a single link.
MCU - Microprocessor Control Unit, a microcontroller based unit within the APU, APMU and MPU used to control their hardware.
MIDI - Musical Instrument Digital Interface, a communication specification which permits controllers and devices within a computer music studio to communicate with one another.
MIDI controller - a device capable of generating MIDI events for the control of devices fitted with the MIDI interface.
MIDI In - Input port for a MIDI signal.
MIDI merging - the merging of two or more MIDI links into a single MIDI link.
MIDI Out - Output port for a MIDI signal.
MIDI patch bay - a device which allows easy MIDI rerouting and possibly MIDI merging, filtering, transformation or transposing.
MIDI Patcher - a system developed within this thesis for the real time routing of MIDI.
MIDI sequencer - a device capable of recording and replaying

sequences of MIDI control information.
MIDI thru box - a device which replicates a MIDI signal, providing many MIDI Outs from a single MIDI In.
MIDI track - a single sequence of MIDI control information within a MIDI sequencer.
MIDI transformation - the specific alteration of MIDI messages.
MIDI transposing - a form of MIDI transformation, in which the pitch of MIDI note information is altered.
MMC - MIDI Machine Control, used by MIDI sequencers to control recorders.
MPU - MIDI Patch Unit, the units that are used to build the MIDI Patcher.
MTC - MIDI Time Code, is used to synchronise a MIDI sequencer to a recorder.
main studio - place where the shared resources of this computer music network implementation are located.
Medialink - Lone Wolf's protocol for the real time distribution of media on a LAN.
module - a set of instructions that is activated by the control logic within a task.
node - see "audio patcher/mixer node".
operational amplifier - a low power integrated amplifier.
opto-isolators - devices used in the reception of MIDI.
PCB - Printed Circuit Board.
parametric equaliser - allows control over the centre frequency, bandwidth and boost/cut within the equaliser.
precondition-postcondition specifications - relates conditions on input values to corresponding conditions on output values.
processor - is a person or machine capable of performing instructions and storing data.
RAM - Random Access Memory, is volatile data memory.
RS232 - a digital serial communication specification used for data transfer.
RS422 - a digital serial communication specification used for data transfer.
SMPTE - Society of Motion Picture and Television Engineers, a time code specification which is used to synchronise audio to audio, or audio to video.
sampler - an audio generator which allows the repeated replay of an existing sampled sound of short duration.
SonicNet - a network implementation that uses FDDI for the real time distribution of digital audio.
state - an observable mode of behaviour.
synthesiser - an audio generator which uses various synthesis techniques to generate audio waveforms.

TTL - Transistor Transistor Logic, a family of logic devices.
TSR - Terminate and Stay Resident, a technique by which two programs can run under DOS together.
task - a set of instructions, started, stopped, interrupted and resumed by the processor.
transition - movement from one state to another.
VCA - Voltage Controlled Amplifier, device used for the automated level control of audio signals in audio mixers.
VITC SMPTE - Vertical Interval Time Code, is a form of SMPTE that is inserted into the video signal.
workstation - a place from where a user can book and utilise the sharable resources in the main studio.

APPENDIX 9 - REFERENCES

[1] MIDI Manufacturers Association,
MIDI 1.0 Detailed Specification Document Version 4.1.
International MIDI Association (January 1989).

[2] Iota Systems,
MIDI-Fader Operations Manual.
Iota Systems (1987).

[3] Lone Wolf,
MidiTap User Manual.
Lone Wolf, Inc (1990).

[4] Hertig B.,
The Last Noise Reduction Article, Part 1.
Electronic Musician (October 1991) p50-59.

[5] Hertig B.,
The Last Noise Reduction Article, Part 2.
Electronic Musician (November 1991) p46-56.

[6] Hertig B.,
The Future of Analog Recording.
Electronic Musician (May 1992) p39-44.

[7] Gross P.,
Synthesis Techniques for the 1990s and Beyond.
Electronic Musician (February 1990) p62-74.

[8] Brighton N.,
The Patch Bay.
Electronic Musician (May 1992) p88-92.

[9] Cochran C.F.,
Samplers Made Simple.
Electronic Musician (March 1991) p74-81.

[10] Wilkinson S.,
Sequencing Made Easy, Part 1.
Electronic Musician (March 1992) p68-71.

[11] Wilkinson S.,
Sequencing Made Easy, Part 2.
Electronic Musician (April 1992) p65-70.

[12] MIDI Machine Control Ratified.
the IMA Bulletin 9(1) (Summer 1992).

[13] Phillips D.,
Secrets of Synchronisation.
Electronic Musician (July 1991) p84-87.

[14] Ciarca S.,
Build an Audio-and-Video Multiplexer.
Byte 11(2) (February 1986) 86-99.

[15] Cipher Digital, Inc,
Time Code Handbook.
Cipher Digital, Inc (1987).

[16] Lehrman P.D.,
Decoding SMPTE.
Electronic Musician 7(4) (April 1991) p62-70.

[17] Woram J.M.,
Sound Recording Handbook.
Howard W.Sams & Company (1989).

[18] Phillips D.,
From the Top: How Sequencers Work.
Electronic Musician 7(4) (April 1991) p86-91.

[19] Giffard T.,
All_Solid_State_Preamplifier - Part 2.
Elektor Electronics 16(174) (January 1990) 27-32.

[20] TEAC Corporation,
TASCAM TEAC Professional Division,
238 Syncaset Owner's Manual.
TEAC Corporation (1988).

[21] Roland Corporation,
Digital Effects Processor DEP-5 Owner's Manual.
Roland Corporation (1986).

[22] Yamaha Corporation,
SPX900 Professional Effect Processor Operation Manual.
Yamaha Corporation (1989).

[23] Intel,
Microcontroller Handbook.
Intel (1984).

[24] Watkinson J.,
The Art of Digital Audio.
Focal Press (1988).

[25] Analog Devices
Data Conversion Products Databook.
Analog Devices (1988)

[26] Oppenheimer L.,
Intone MIDI Maestro Audio and MIDI Patch Bay.
Electronic Musician (March 1991) p94-97.

[27] Ward P.T. & Mellor S.J.,
Structured development of Real-Time Systems,
Volume 1: Introduction & Tools.
Yourdan Press (1985).

[28] Ward P.T. & Mellor S.J.,
Structured development of Real-Time Systems,
Volume 2: Essential Modeling Techniques.
Yourdan Press (1985).

[29] Ward P.T. & Mellor S.J.,
Structured development of Real-Time Systems,
Volume 3: Implementation Modeling Techniques.
Yourdan Press (1986).

[30] Buxton W.,
Masters and Slaves Verses Democracy: MIDI and
Local Area Networks.
Proceedings of the AES 5th International Conference.

[31] Westfall L.,
The Local Area Network: MIDI's Next Step?.
Electronic Musician (November 1989) p64-119.

[32] Allik K., Dunne S., Mulder R.,
ArcoNet: A Proposal for a Standard Network
for Communication and Control In Real-Time Performance.
*Proceedings of the International
Computer Music Conference 1986.*

[33] Hurtig B.,
The Future of Analog.
Electronic Musician (May 1992) p39-46.

[34] Quark LRM2 MIDI Link.
Music Technology (November 1986) p28.

[35] 360 Systems Audio Matrix 16
Music Tecnology (April 1989) p14.

[36] AM-16 Series Audio Crosspoint Switchers.
360 Systems.

[37] Aikin J. & Milano D.,
MIDI-Controlled Digital Mixer.
Keyboard (August 1987) p110-146.

[38] Aikin J.,
Motu Mixer 7s.
Keyboard (May 1991) p124-126.

[39] Borish J.
Keeping it Digital.
Electronic Musician (October 1990) p60-69.

[40] Massey H.,
Secrets of the Yamaha SY77.
Keyboard (May 1991) p64-78.

[41] Borwick J.,
Sound Recording Handbook, Third Edition.
Oxford University Press (1987).

[42] Short K.,
Microprocessors and Programmed Logic.
Prentice Hall, Inc. (1981).

[43] TEAC Corporation,
TASCAM TEAC Professional Division
200 Series Mixing Consoles.
TEAC Corporation (1987).

[44] Franklin Software,
C-Compiler-51, User's Guide.
Franklin Software, Inc. (1989).

[45] DeMarco T.,
Structured Analysis & System Specification.
Yourdon Press. (1979).

[46] Yourdan E., Constantine L.,
Structured Design.
Yourdon Press. (1979).

[47] Jackson M.,
System Development.
Prentice Hall International, Inc. (1983).

[48] Metts A.,
Voyetra Sequencer Plus Gold 4.0 (IBM).
Electronic Musician (July 1991) p91-95.

[49] Osborne A., Kane J.,
An Introduction to Microcomputers Vol.3,
Some Real Support Devices.
A Osborne & Associates Inc. (1978).

[50] Foss R., Wilks A.,
A Network Approach to the Problem of Sharing
Music Studio Resources.
Proceedings of the International
Computer Music Conference 1990.

[51] Tanenbaum A.,
Computer Networks, Second Edition.
Prentice Hall International, Inc. (1988).

[52] Malmstadt H., Enke C., Crouch S.,
Electronics and Instrumentation for Scientists.
The Benjamin/Cummings Publishing Company, Inc. (1981).

[53] Optoelectronics Division, General Instruments,
Catalog of Optoelectronic Products 1983.
General Instruments.

[54] MIDI Time Code, The Linking of MIDI and SMPTE.
the IMA Bulletin 3(6) (July 1986).

[55] Kubicky J.,
A MIDI Project, A MIDI interface
with software for the IBM PC.
BYTE 11(6) (June 1986).

[56] MIDI Manufacturers Association,
MIDI Machine Control 1.0.
International MIDI Association (January 1992).

[57] National Semiconductor Corporation,
Linear Databook 1, Rev. 1.
National Semiconductor Corporation (1988).

[58] Texas Instruments,
The TTL Data Book, Volume 1.
Texas Instruments (1985).

[59] Hitachi Ltd.,
IC Memory Products.
Hitachi Ltd. (1986).

[60] O'Donnell B.,
J.L.Cooper FaderMaster.
Electronic Musician (November 1989) p75,76.

[61] Berlin H.M.,
Design of Active Filters, with Experiments.
Howard W.Sams & Co., Inc (1978) p175-.

[62] Orr T.,
Semi-Professional Mixer.
Practical Electronics (October 1982) p30-37.

[63] Stevens A.,
Writing TSRs.
Computer Language (February 1988).

[64] Apple Developer Products,
MIDI Management Tools, Developer Notes.
Apple Computer, Inc. (August 1989).

[65] Wilkinson S.,
Sound all Around.
Electronic Musician (October 1993) p70-76.

[66] Cubase.
Electronic Musician (October 1993) p127.

[67] Alles A.,
ATM in Private Networking.
Hughes LAN Systems (1993).

[68] Performance for the Long Run.
Richmond Sound Design Ltd (1993).

[69] Bernardini N., Otto P.,
An Interactive System for Sound Location.
Proceedings of the International Computer Music Conference 1989.

[70] Harris L., Scott B.,
A Single-Chip Stereo Volume Control.
Crystal Semiconductor Corporation Digital Audio Products Data Book (January 1994) p8-203.

[71] National Semiconductor Corporation,
CMOS Logic Databook.
National Semiconductor Corporation (1988) p2-3.