Musical Applications of the Atari ST's

by

R. A. Penfold

BERNARD BABANI (publishing) LTD
THE GRAMPIANS
SHEPHERDS BUSH ROAD
LONDON W6 7NF
ENGLAND

First Published — September 1988

British Library Cataloguing in Publication Data
Penfold, R. A.
Musical applications of the Atari STs.
1. Music. Applications of Atari ST micro computer systems
I. Title
780'.28'54165

ISBN 0 85934 191 7

Set from disc by Commercial Colour Press, London E.7
Printed and Bound by The Guernsey Press Co. Ltd, Channel Islands

Preface

Not so very long ago there was no clear leader in the music computer stakes, with the possible exception of the Commodore 64. However, several computers were quite popular for music applications, with the Commodore 64 being just one of several machines with a respectable range of music oriented software available, as was the Atari ST. The "64"was the most popular, but did not dominate the market. The ST gradually overhauled all the competition, including the "64". Once it gained a good lead the situation changed quite rapidly, with the ST becoming very much *the* music computer. It now has an unrivaled (but still rapidly expanding) range of available software and music add-ons. With its built-in MIDI ports, large memory, excellent graphics, high processing power, and quite moderate cost, it is perhaps the obvious choice for demanding music applications such as MIDI sequencing, and its sudden leap to total dominance is not really all that surprising.

This book is aimed at the musician who wishes to exploit the potential of the ST computers in music applications. This mainly means using the ST in MIDI systems, and much of the book is devoted to a description of MIDI in general, MIDI as it applies to the ST, and running MIDI applications on the ST. A knowledge of computing is certainly an asset, but most of the information is usable by someone with little knowledge of computing. It is assumed though, that the user has a basic knowledge of using the ST and running programs on it. The small manual provided with the ST provides instructions on using the mouse, running applications programs, etc., and this is all the preparation you should need. For those who are more technically minded and favour the do-it-yourself approach, there are chapters covering some simple hardware projects, programming the ST's sound chip, and MIDI programming (including some useful MIDI processing routines).

R. A. Penfold

Acknowledgements

Fast BASIC is a registered trademark of Computer Concepts, Gaddesden Place, Hemel Hempstead, Herts., HP2 6EX.

ST and ST BASIC are registered trademarks of Atari Corporation, Sunnyvale, CA 94086, U.S.A.

Notator is a registered trademark of C-LAB Software, Postbox 70 03 03, 2000 Hamburg 70, Germany, whose U.K. agents are Sound Technology PLC, 6 Letchworth Business Centre, Avenue One, Letchworth, Herts., SG6 2HR.

Hybrid Arts EZ-Track and EZ-Score are registered trademarks of Syndromic Music, 24–26 Avenue Mews, Muswell Hill, London, N10.

Contents

Chapter 1
THE INTERNAL SOUND GENERATOR

The ST range of computers have gradually established themselves as the standard machines for electronic music applications, ousting the Commodore 64 from this position. The inclusion of MIDI ports on all the ST computers obviously played a big part in this success, but a few other computers have been similarly equipped but failed to gain widespread acceptance amongst music users. The ST computers do not just have built-in MIDI ports in their favour, they use the powerful 68000 microprocessor, have excellent graphics capability, plenty of memory even in the basic models, and are quite affordable. They really do deliver Atari's promise of "power without the price"!

There are several versions of the ST from the original 520ST to the new "Mega" STs with their massive memory. The 520ST models have been gradually improved over the years. Originally the operating system had to be loaded from disc each time the computer was used, leaving little memory free for applications programs. Later models have the operating system built-in on ROM chips so that a large proportion of the RAM is left free (and most owners of the early machines have now had the ROM chips fitted to their machines).

The Atari 520 STFM

The 520STM has a built-in modulator so that it can be connected to a television set which will act as a colour monitor. This is a useful addition, as many people who are primarily interested in using the ST for music applications opt for the monochrome monitor. The built-in modulator allows programs that run in the low or medium resolution colour modes to be run without having to go to the expense of buying a colour monitor. However, the quality of the display provided by a colour television might not be very good, and if programs that will only run in a colour mode are to be used a great deal, there is no real substitute for a good colour monitor. Most music software seems to be usable in medium resolution colour or high resolution monochrome, but there have been a few that are monochrome only (although mostly seem to have been modified so that the current versions also run in colour). There may be a few music programs that will only run in one of the colour modes, but I have not yet encountered one. It is perhaps worth mentioning that the quality of the ST monochrome display is absolutely superb. Unless a program makes really good use of colour I have no hesitation in opting for monochrome operation.

The standard version of the 520ST at the time of writing this book is the 520STFM, which has both a the television modulator and a floppy disc drive built-in. Floppy discs are used for program and data storage, and virtually all software for the ST computers is only available on disc. The STs do not have a cassette port, and little software is available in cartridge form (I think that all STs are equipped with a cartridge port). Even with cartridge software you might still need a disc drive for data storage. An ST computer is of very limited use without a disc drive. If you have an ST without a built-in disc drive you will certainly need to buy an external drive before you can use it properly (or use it at all) in music applications. The built-in drive of the original 520STFM was single-sided type having a formatted capacity of about 360k. This is adequate for most purposes, but there are some programs that require a double-sided drive with its 720k formatted capacity. A 520STFM plus an external double-sided drive might be suitable, or it might be possible to have the computer upgraded to a double-sided built-in drive. Current 520 STFM's are fitted with a 720k drive.

The 1040STF is basically just a 520STFM with an extra 520k of memory added to the printed circuit board (all 1040 STFs have a double-sided drive). The absent "M" in its name indicates that no television modulator is included. However, some suppliers seem to offer a modulator as an optional extra, and it seems likely that it will be included as standard in the near future. If you are interested in software that requires a double-sided drive and the extra memory, a 1040STF is the obvious choice. A lot of software will operate with 520k of memory, but works much better with 1040k of RAM. For example, a lot of sequencers will work on a 520k machine, but have relatively limited note capacity. If you only wish to store short sequences a 520ST series machine should suffice, but otherwise a 1040k version will be needed.

A lot of people (myself included) have opted for a 520STFM plus an upgrade to 1040k of RAM. This is not a very difficult or expensive modification, and if you are reasonably competent at electronics construction it can be accomplished using a low cost do-it-yourself memory upgrade kit. This gives results that are much the same as would be obtained using a 1040STF, but at a significantly lower price. Whether or not this remains a worthwhile way of doing things obviously depends on the future pricing policy of these computers, but it is something worth considering.

The Mega STs look very different to the 520 and 1040 series, and have the keyboard and main electronics as separate units, rather than in a single box. Despite this they are basically just the same electronics plus larger amounts of memory (2080k or 4160k). They should eventually be fitted with the "blitter" chip as standard, but at the time of writing this there seem to be some delays here. The blitter is designed to speed up certain processes, and in particular it gives higher speed graphics with software that is designed to exploit it. For most music applications the blitter and the increased memory would not seem to be very important. However, some sampling systems that are based on the ST use the computer's memory for sample storage, and can use very large amounts of memory. A Mega ST could be very advantageous with a system of this type, but we are talking in terms of some quite expensive up-market equipment which few electronic musicians can give serious consideration.

A point to keep in mind if you are not an ST owner but are thinking of buying one for music purposes, or for any other application come to that, is that you should try to choose the software first. When you have selected the best software for your purposes, determine exactly what hardware is needed in order to run it properly, and then buy the appropriate version of the ST plus any peripherals (monitor, second disc drive, etc.) that are needed to run it. Buying a computer system and then looking for software to run on it is definitely a case of "putting the cart before the horse".

BASIC Sound Generation

Probably few musicians buy an ST because they want to use its internal sound generator. On the other hand, if you write software and you wish to have music and (or) sound effects to accompany it, you will need to make use of the sound generator. Some programs use the MIDI output port plus external instruments to provide a musical accompaniment, but this is normally as an optional extra to the standard output from the built-in sound chip.

We will start our exploration of the ST's music capabilities with details of programming the sound chip from BASIC. This is the easiest way of controlling the device, as ST BASIC has two special commands for handling sound. The "SOUND" instruction is particularly useful, and is aimed specifically at producing musical notes rather than just sounds of arbitrary pitch for sound effects applications. There is also a WAVE instruction which is concerned with the control of envelope shapes and other factors.

The SOUND instruction is followed by five numbers which respectively control the channel, volume, note, octave, and duration values. These are each described in detail below

CHANNEL: The sound generator is a three channel type, which merely means that it can produce three notes at once (three voice polyphony in other words). BASIC does not automatically assign any notes you include in a program to a spare sound channel. It is up to the programmer to handle this by assigning each note to a specific sound channel. These channels are simply numbered 1 to 3.

VOLUME: There are sixteen volume settings available, from 0 to 15. 0 represents minimum volume, through to maximum volume at a value of 15. I suppose that strictly speaking there are only fifteen volume settings, since a value of 0 switches off that channel and gives zero output. The importance of this zero volume setting will become apparent shortly.

NOTE: The note value is in the range 1 to 12, and this gives a coverage of one octave in semitone increments. This list shows the corresponding note for each number.

1	C
2	C sharp
3	D
4	D sharp
5	E
6	F
7	F sharp
8	G
9	G sharp
10	A
11	A sharp
12	B

OCTAVE: A range of eight octaves are available, using octave values of 1 to 8. Middle A (440Hz) would to be note 10 in octave 4. The pitch range covered by the sound generator under BASIC control is quite impressive, but due to the way computer sound generators work, the pitch accuracy at high frequencies is usually very much less good than the accuracy at low pitches. You may find that the pitch of some high notes is less than perfect.

DURATION: This value determines the length of a note, and is in fiftieths of a second. Some sources quote this as being in sixtieths of a second, and it probably depends on whether your ST has a European (50Hz) or American (60Hz) display. I will assume here that this value is in fiftieths of a second, and any readers who are using an American version of the ST will have to alter duration values to compensate for the slight difference in the internal timing software of their computers.

The duration value does not operate quite as you might expect. Load ST BASIC and then try typing this command into the computer: –

```
SOUND 1,12,10,4,50 RETURN
```

One might reasonably expect this to produce middle A on channel 1 at a fairly high volume level for one second. What will actually happen is that the note will be produce alright, but after one second the "ok" prompt will return and the note will continue to sound! What the note duration parameter is actually doing is providing a hold-off that prevents a new note from starting for the specified time. It does not terminate a note after the given period of time has elapsed. If you are supplying a long sequence of notes to the sound generator, this fact will not be apparent at first since a new note will be commenced as soon as the duration period on the previous one has expired. It will always be apparent at the end of the sequence when the last note continues indefinitely. To avoid this, each note or sequence of notes must be terminated by a "dummy" SOUND instruction having a volume parameter of zero. For instance, to halt activity on channel 2, the instruction: –

```
SOUND 2,0,0,0,0
```

could be used.

To switch off the demonstration SOUND example given previously, all you have to do is press the "RETURN" key. The key "click" will then take over from your SOUND instruction, and will switch off the sound generator once it has been completed.

Sequencing

Producing a simple sequence from BASIC is quite easy. Probably the most simple way of tacking the task is to use a FOR...NEXT loop to repeat a SOUND instruction as many times as you need notes in the sequence. Note values can be held in DATA statements, and READ on each loop of the program. Note durations can be programmed in the same way. Any parameter in a SOUND instruction can be in the form of a variable incidentally. This simple demonstration program at the bottom of the page produces an ascending scale of C major.

Lines 20 and 50 are the FOR...NEXT loop (called "LOOPS"), and as eight notes are included in the sequence these loop the program eight times. The note, octave, and duration values are held in variables "n", "o", and "d". These are READ at line 30 and used in the SOUND instruction at line 40. I have not bothered to make the volume programmable, but this could be achieved by using a fourth variable in the program.

The sequence is stored in the DATA statement at line 60, and this contains sets of the three SOUND values. Note that as the parameters are read in the order note, octave, and value, they must be placed in the DATA statement in the same order. Large amounts of data can be held in DATA statements as it is quite in order to use a number of these statements if there is too much data to be held in a single statement. My sources of information on ST BASIC do not seem to quote a limit for the amount of data that can be stored in DATA statements, but with most BASICs the maximum line length (usually about 255 characters) sets the limit. In practice it would be difficult to program long sequences using this method, but it should be perfectly suitable where a short "jingle" in a program is all that is required.

WHILE...WEND

This improved sequencer program, shown overleaf, makes use of the ST BASIC WHILE...WEND loop.

Line 20 sets variable n ("note") at a starting value of 1. The WHILE...WEND loop at lines 30 and 60 keeps the program looping around these lines for as long as n is greater than zero. The program operates in much the same way as the previous one, with note, octave, and duration values being READ from DATA statements at the end of the program. However, in this case the volume parameter of the SOUND instruction has also been made a variable ("v"). This enables the volume of notes to be programmed, but it also enables the sound statement at line 50 to be used to terminate the sequence. Using zero for each parameter at the end of the final DATA statement has two effects. Firstly, it ends the WHILE...WEND loop. Secondly, it sets the volume parameter at zero so that the sound generator is silenced. This method is much neater than the original version, and there is no need to specify the number of notes in the loop instructions. This makes things easier

```
10 REM SIMPLE SEQUENCER
20 FOR LOOPS = 1 TO 8
30 READ n,o,d
40 SOUND 1,12,n,o,d
50 NEXT
60 DATA 1,3,20,3,3,20,5,3,20,6,3,20,8,3,20,10,3,20,11,3,20,1,4,80
70 SOUND 1,0,0,0,0
```

```
10 REM IMPROVED SEQUENCER
20 n = 1
30 WHILE n > 0
40 READ v,n,o,d
50 SOUND 1,v,n,o,d
60 WEND
70 DATA 12,1,3,20,12,3,3,20,12,5,3,20,12,6,3,20,12,8,3,20,12,10,3,20
80 DATA 12,12,3,20,15,1,4,80,0,0,0,0
```

if you program a fairly long sequence, or if you keep changing the sequence. However, you must always remember to include the four zeros at the end of the final DATA statement.

Associated with SOUND there is the WAVE instruction. This is used to control such things as the sound chip's noise generator, channel enable/disable register, and envelope shaper. It is very different to the SOUND instruction in that ST BASIC does not provide much help with this one. It is basically just taking the values you put in the WAVE instruction and placing them in some of the sound chip's registers. You therefore need an understanding of the sound chip in order to make proper use of this instruction. Many programmers prefer to simply control the registers of the chip directly!

Anyway, the WAVE instruction is normally followed by four parameters, but it can have a fifth. Briefly, the parameters are "enable", "envelope", "shape", "period", and "wait". "Enable" determines which tone channels are enabled, and can be used to mix the output of the noise generator with one or more of the tone channels. "Envelope" is used to enable or disable the envelope shaper (which takes over from the volume value when it is enabled). "Shape" selects one of several envelope shapes. These shapes are built into the sound chip, and apart from selecting the one you require they are not under software control. The duration of each envelope (or each cycle of a repeating type) is controlled by the "period" value. The "wait" parameter is often omitted, and it does not send a value to a register of the sound chip. Its effect seems to be the same as the duration value in a SOUND instruction. In other words, it just provides a programmable delay before the WAVE instruction is terminated and the program moves on to the next instruction.

The function of each part of the WAVE instruction should become clearer if you study the following section which describes the sound chip in some detail.

Chip Description

For those who would like to delve more deeply into using the sound chip this full description of the device is provided. The importance of this information depends on how you will be using the ST. If you are only interested in running applications software the sound chip is of no real importance at all. If you are only interested in writing MIDI software it remains equally unimportant. If you are programming the ST in a language that provides high level support for the sound chip, a detailed knowledge of the chip may be no more than slightly useful. It depends which language you use, and whether you wish to fully exploit the sound chip's potential. If you wish to program sound on the ST using a language that does not provide any major support for the sound chip, a detailed knowledge of its registers is essential, as this will be the only way of accessing it (apart from using the XBIOS calls from a suitable language, which still necessitates a fairly detailed knowledge of the chip).

This description is only intended for reasonably experienced programmers, and direct control of the sound generator is not something I would recommend for beginners. The sound chip is capable of quite good results, but it needs carefully planned programming to get the best from it. Quite frankly, programming a MIDI instrument via the MIDI output port is a considerably easier way to get much better results. On the other hand, the internal sound chip is interesting for those who like dabbling with computer hardware, and who like a challenge!

The sound chip used in the ST computers is the AY-3-8910 or equivalent. This does not seem to be held in very high regard these days, but it was considered to be one of the best computer sound chips a few years ago. It seems to have lost ground in the face of competition from the "SID" chip of the Commodore 64, the four channel sampling sound system of the Commodore Amiga, and the Ensoniq synthesiser chip of the Apple IIGS. Despite these advances, the AY-3-8910 remains a reasonably competent device capable of three channel tone generation plus noise generation. It lacks sophistication in that it does not have variable waveforms and filtering, but it does provide a variety of envelope shapes. You will often see the Atari ST's sound chip referred to as a PSG, and we will use this convention. PSG simply stands for "programmable sound generator".

In order to program the PSG effectively you need to understand the way in which it generates tones. The effect of values written to its tone control registers can otherwise seem to be rather strange. Electronic sound generators can use a system which is analogous to a piano, where each note is produced by a separate string or other form of resonator. The electronic equivalent of a mechanical resonator is an oscillator, which is a circuit that produces a series of electrical pulses. When these are used to drive a loudspeaker they produce an audio tone at the frequency of oscillation.

This method has been used in the past with electronic organs, but it requires a lot of oscillators in order to give a wide range of notes. This is expensive, it takes a long time to get everything set up and tuned correctly, and frequent readjustment might be needed. Most computer sound generators, including the ST's PSG, use a totally different approach. The basic idea is to have a high frequency oscillator, and to feed this to some form of complex frequency divider having numerous outputs. In this way a single oscillator can be used to provide a wide range of output frequencies. The oscillator used in this system can be a highly accurate and stable quartz crystal controlled type, and tuning adjustments are then unnecessary.

The ST's PSG uses a variation on this technique, where the oscillator is fed to three divide by 'N' counters (Fig.1.1). Here 'N' is a value written to registers of the device, and by changing this value the output frequency of each divider can be varied over very wide limits. This gives what are effectively three adjustable tone generator circuits. This is not as good as using a complex divider circuit, which enables any desired number of notes (within reason) to be produced simultaneously. Three tones at once is sufficient for most computer sound generator applications though, and it enables much simpler circuits to be used.

first. What you are actually controlling with the pitch control registers is the period of one output cycle. The larger the pitch value, the longer the duration of each cycle.

Another consequence of this method of frequency synthesis is that very high resolution is obtained at low frequencies, but very poor resolution is produced at high frequencies. If you take any number and divide it by two, three, four, etc., you will find that the answers decrement in large steps at first, but as the divisor gets larger, reductions in the answer get smaller. Computer sound generators are often quoted as having very wide output frequency ranges. While the ranges quoted might be completely accurate, it is as well to bear in mind that where accurate output frequencies are required the usable output frequency range might be substantially more restricted at the high frequency end of the range.

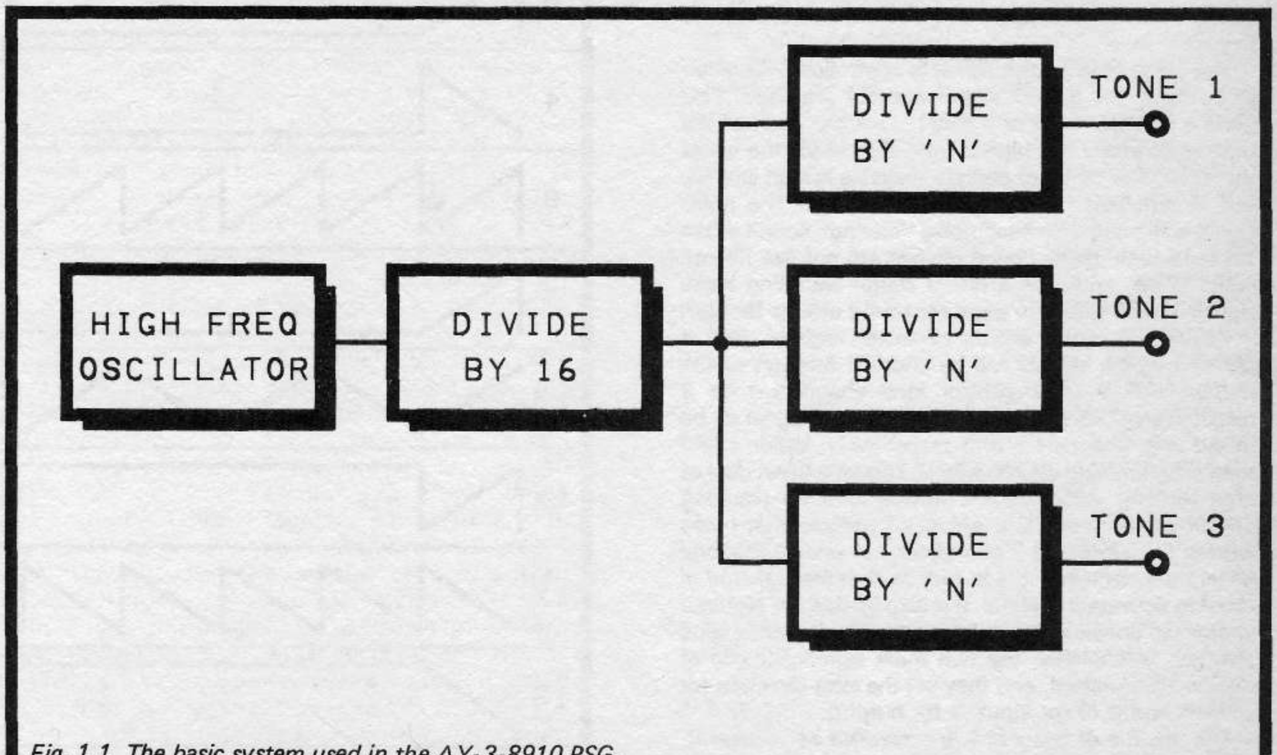


Fig. 1.1. The basic system used in the AY-3-8910 PSG.

The output frequency from one of he PSG's tone generators is equal to the input frequency divided by sixteen, and then divided by the value sent to the pitch control registers. There are two pitch control registers per tone generator, with one providing fine tuning and the other giving coarse tuning. Only four bits of the coarse tuning registers are implemented (the least significant bits), giving 12 bit overall resolution. This provides some 4096 different output frequencies. Due to the system of frequency division used to vary the output frequency, high values in the pitch control registers give a large division ratio and a low output frequency. This topsy-turvy relationship between control values and output pitch can be a bit confusing at

The Registers

The AY-3-8910 is controlled by sixteen registers, and basic details of these are provided below: –

Register Number	*Function*
0	Channel 1 pitch (fine)
1	Channel 1 pitch (coarse)
2	Channel 2 pitch (fine)
3	Channel 2 pitch (coarse)
4	Channel 3 pitch (fine)
5	Channel 3 pitch (coarse)
6	Noise pitch

7	Mixer control
8	Channel 1 volume/envelope control
9	Channel 2 volume/envelope control
10	Channel 3 volume/envelope control
11	Envelope period (fine)
12	Envelope period (coarse)
13	Envelope shape
14	Port A
15	Port B

Registers 14 and 15 are input/output ports which are used for (amongst other things) the ST's parallel port. They have no relevance to the sound generation process, and will not be considered further here.

Registers 0 to 5 are the pairs of pitch control registers for the three tone channels. Their action has already been described. Remember that only the four least significant bits of the coarse pitch control registers are used (giving a control number range of 0 to 16). The total value written to a pair of registers is equal to the coarse value multiplied by 256 and then added to the fine tune value.

The pitch of the noise signal is controlled by register 6, but only the 5 least significant bits are used. This gives a control range of 0 to 31. Like the pitch of the tone generators, the higher the value used, the lower the pitch. The range of pitches available is a bit limited, but is sufficient for simple sound effects. The noise signal is the usual "white" noise "hissing" sound when set for a high pitch. Lower pitches are not like filtered white noise, and give a rather rough sounding noise signal. This is still quite good for sound effects though.

Register 7 is the enable or mixer register, and it controls which signals will be coupled through to the output. Bits 0 to 2 control tone channels 1 to 3 respectively. Bits 3 to 5 enable the noise signal to be mixed with channels 1 to 3 respectively. When I first tried programming an AY-3-8910 I spent a great deal of time sending values to the device with no resulting output signals from it. Eventually I realised that I was writing 1s to bits 0 to 2 of register 7 to enable the tone generators, whereas it is in fact 0s that are required in order to activate a channel. It is also 0s that are required in order to enable mixing of the noise signal with a tone channel. Incidentally, the two most significant bits of this port are utilized, and they set the data direction for ports A and B (0 for input, 1 for output).

The volume of channel 1 is controlled by register 8, but only the four least significant bits are used (giving the 0 to 15 volume setting range in the ST BASIC SOUND instruction). A value of 0 switches the tone generator off, a value of 15 gives maximum volume. If bit 4 of this register is set to 1 (i.e. a decimal value of 16 is written to this register) the volume is no longer controlled by bits 0 to 3. Instead, the volume is controlled by the envelope generator. The three most significant bits of this register are left unused. Registers 9 and 10 operate in exactly the same way as register 8, but they control channels 2 and 3 respectively.

The envelope period, or period of one cycle in a repetitive type, is controlled by the 16 bit value written to registers 11 and 12. The least significant byte is written to register 11. All bits of both registers are utilized, giving full 16 bit resolution (0 to 65535 in decimal terms). The clock signal is divided by 256, and then by the number written to these two registers. This gives a very wide envelope period which can be anything from a small fraction of a second to several seconds. Unfortunately, the envelope shaper only uses sixteen volume levels (including "off") and the output signal can be usually be heard to step up and down in volume during the course of an envelope.

Register 13 is used to select the desired envelope shape. Only the four least significant bits of this register are used. The available shapes and the (decimal) numbers needed to select them are shown in Fig.1.2.

Fig.1.3 gives details of the AY-3-8910's registers in a form that should be useful for quick reference purposes when first getting to grips with the device.

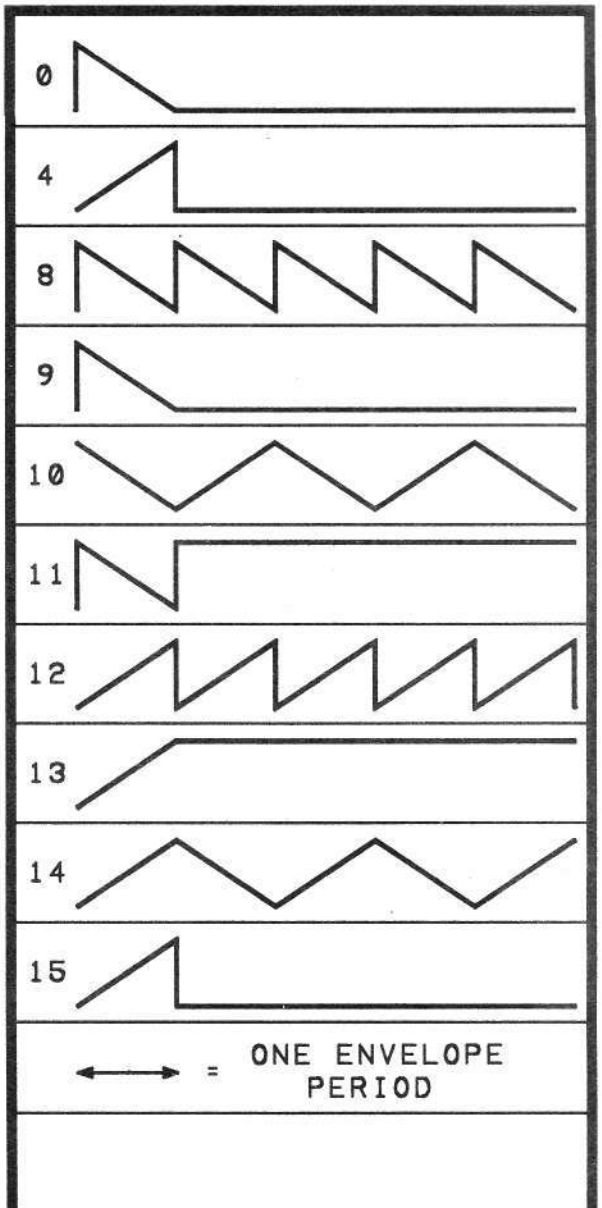


Fig. 1.2. The PSG envelope shapes.

REGISTER	FUNCTION	BIT							
		7	6	5	4	3	2	1	0
0	CHANNEL 1 PITCH	FINE TUNING							
1		UNUSED				COARSE TUNE			
2	CHANNEL 2 PITCH	FINE TUNING							
3		UNUSED				COARSE TUNE			
4	CHANNEL 3 PITCH	FINE TUNING							
5		UNUSED				COARSE TUNE			
6	NOISE PITCH	UNUSED			PITCH CONTROL				
7	ENABLE	I/O		NOISE			TONE		
		B	A	C	B	A	C	B	A
8	CHANNEL 1 VOLUME	UNUSED			ENV	VOLUME CONTROL			
9	CHANNEL 2 VOLUME	UNUSED			ENV	VOLUME CONTROL			
10	CHANNEL 3 VOLUME	UNUSED			ENV	VOLUME CONTROL			
11	ENVELOPE DURATION	FINE ADJUSTMENT							
12		COARSE ADJUSTMENT							
13	ENVELOPE SHAPE	UNUSED				ENVELOPE			
14	I/O PORT A	8 BITS OF PORT A							
15	I/O PORT B	8 BITS OF PORT B							

Fig. 1.3. Details of the PSG registers.

Accessing The Registers
The 68000 microprocessor used in the ST computers does not have a separate input/output map. Input/output devices appear in the memory map, and are accessed just like memory devices (which means using PEEK and POKE from BASIC). Although there are sixteen read registers and sixteen write registers, the device only occupies two addresses in the memory map. This is achieved by having one address to select the register you wish to access, and another address for actually reading from or writing to that register. The required register is selected by writing its register number to address &HFF8800, and then the value for that register is written to address &HFF8802.

The best way to gain an understanding of any computer peripheral chip is to try programming it, and if necessary, to learn from your mistakes. As an example to get you started, try the following ST BASIC program. It gives a sort of brief explosion type sound effect using the PSG's noise generator.

The program loops six times, and on each loop it POKEs a value first to the register select register, and then to the selected register. These values are held in variables "r" and "v", and they are READ from the DATA statement at line 70. The first pair of values write 31 to the noise pitch register, which gives minimum pitch. The next pair enable the noise and mix it into channel C, while the next two lines hand over volume

```
10 REM EXPLOSION PROGRAM
20 FOR LOOPS = 1 TO 6
30 READ r,v
40 POKE-B &HFF8800,r
50 POKE-B &HFF8802,v
60 NEXT
70 DATA 6,31,7,223,10,16,11,0,12,20,13,9
```

control to the envelope shaper. The remaining values set the envelope period and type. Using a higher noise pitch value and shorter envelope period you can obtain a gunshot type sound effect. Note that the program uses the POKE-B instruction to write to the PSG. This form of the instruction does not exist in the original version of ST BASIC, and if this is the version you are using, the program will require slight modification. Slight modification might also be required if you use some other version of BASIC.

In order to become competent at programming a sound generator chip you really need to experiment with it for a while in order to find out just what sounds are available. Inevitably, programs will not always produce quite the desired effect, and some "fine tuning" will often be required. Practice makes perfect!

Chapter 2
MIDI CONNECTIONS

While the built-in MIDI ports of the Atari ST computers are not a unique feature, they are something that few other computers have as standard. In fact some computers do not even seem to have MIDI ports as optional extras, or as so-called "third-party" add-ons. With the computing power of its 8MHz 68000 microprocessor, plus excellent graphics capability, and all at a remarkably low price, it was inevitable that the ST computers would take over from the Commodore 64 as the computer for music applications. There is probably more MIDI software available for the Atari ST computers than there is for all the other home and personal computers put together. The list of available software seems to grow daily! Many people are buying these computers specifically to use them in music applications (myself included).

Just how much you need to know about MIDI depends on the complexity of the MIDI system you will be using, and on whether or not you wish to undertake any of your own programming. If you only wish to use a fairly simple MIDI applications program with a single synthesiser setup, you may be able to get away with very little knowledge of MIDI at all. Even with a basic setup of this type a more comprehensive knowledge of the subject is likely to prove more than a little useful. As with so much of today's technology, there are often very simple solutions to problems that arise. However, without a reasonable understanding of the subject you may never find the solution. If you are intending to use multi-instrument setups a more detailed knowledge of MIDI's workings and terminology is almost certain to be required, and for MIDI programming you must understand at least the more common codes used, and the general method of coding.

MIDI can seem a bit confusing at first, but apart from one or two idiosyncracies it is really quite straightforward. It is a very versatile system though, and the almost limitless possibilities mean that there is a lot to learn if you are going to become a real MIDI expert. A great deal of thought needs to go into setting up a system that will suit your requirements, and you should avoid the temptation to rush into things. MIDI software and equipment are relatively cheap when you take into account the capabilities they provide, and what similar equipment would have cost a few years ago (if something with a similar specification could have been obtained at all). In absolute terms, mistakes are still likely to be quite expensive to rectify.

There is insufficient space available here for a complete course in the theoretical and practical aspects of MIDI. However, we will consider both aspects in reasonable detail, including full details of the all-important MIDI messages. Various ways of combining MIDI equipment will also be considered, but only in the context of systems based on the Atari ST computers. Details of the MIDI message types and codes are not provided here, but the technicalities (including a full list of MIDI code numbers) are covered in separate chapters. Similarly, MIDI applications software is only considered superficially here, but this subject is covered in some detail in a subsequent chapter.

MIDI Advantages

If, for the moment, we just consider MIDI at the most basic level, it is just a means of sending messages from one device to another. MIDI is an acronym for "musical instruments digital interface", and as yet it is only used with electronic music equipment. It could be used with virtually any piece of electronic equipment though, and it would be quite feasible to have a MIDI controlled robot for example. The point that I am trying to make here is that MIDI is a means of communication, and it is only musical in that it provides communications between pieces of electronic music equipment. Some people seem to think that a MIDI output provides a signal that only needs to be connected to a hi-fi system in order to produce music. In fact it will only produce music by way of a suitably equipped electronic musical instrument. Connecting a MIDI output to a hi-fi system is a bit like connecting the printer port to the hi-fi system and expecting it to speak the words that are sent to the port!

Apart from two or more pieces of MIDI equipment the only other requirement is a cable or cables to provide the necessary connections between the various pieces of equipment. In the past there have been difficulties in interfacing one item of equipment to another, especially in systems that used equipment from several manufacturers. A lack of true standards meant that connecting equipment together with standard leads in what seemed to be a perfectly acceptable manner resulted in unpredictable results. In many cases it resulted in total failure, and some intervening electronics would often be required before individual items of equipment would work properly together as a true system.

MIDI is a true standard, and any piece of MIDI equipment should work well with any other item of MIDI gear. The only proviso here is that the two pieces of equipment must obviously be devices that could reasonably be expected to operate together as a system. A second point to bear in mind is that MIDI is a very versatile system that can couple virtually any type of information from one piece of equipment to another, but the MIDI specification does not lay down a minimum standard for equipment. It is a standard for the basic communications hardware, and the system of coding/decoding messages that are past from one piece of gear to another. Few (if any) items of MIDI equipment send and (or) recognise all MIDI messages. What this means in practice is that you should not assume that any features of an instrument or other item of MIDI equipment are accessible via its MIDI input socket. You must carefully check the MIDI specifications in equipment manuals to find out just what can and what can not be achieved. This is particularly important with older instruments, many of which had the MIDI interface added very much as an afterthought. In the early days of MIDI there were a lot of high quality instruments with sophisticated features that had the most basic of MIDI implementations!

Another problem in the pre MIDI era was the large number of connecting cables required when building up any reasonably complex system. The basic method of connection was to have one lead to provide gate/trigger coupling, and another one to connect the CV (control voltage) sockets. The former provided note on/off switching, while the latter gives note selection. In some systems there would be a second CV connection, with this second one giving control over the voltage controlled filter (or perhaps the VCA). The point to note here is that two or three connecting leads are required, and that this only provides monophonic operation (i.e. only one note at a time can be played). Polyphonic operation can be accommodated with a suitably sophisticated instrument, or (more probably) using a bank of monophonic synthesisers, but for (say) eight note polyphonic operation some sixteen or twenty four connecting leads would be required! Even with twenty four cables used, there is still only a rather crude form of control over the instruments.

MIDI is much more convenient in that only one cable is needed in order to convey information from one device to another. The system is not limited to basic note on/off and note value information, and even if an instrument is a sixteen or thirty two note polyphonic type, one cable is all that is required! With a few modern MIDI "black boxes" and a few connecting cables you can have a neat and compact system. Not so many years ago an equivalent system would have cost a fortune and filled a decent sized room.

MIDI Wiring

The standard MIDI connector is a 5 way 180 degree DIN type. Actually the standard specification does allow for 3 way XLR type connectors to be used, but only if the manufacturer makes DIN adaptors available as an extra so that standard (DIN type) MIDI leads can be used if required. In practice XLR connectors seem to be little used, but might be found on some high quality instruments intended to be able to withstand "life on the road".

Ready-made MIDI leads seem to be available from most retailers of electronic musical instruments, as well as a number of Atari stockists. On the other hand, it is not difficult to make your own leads at low cost if you are handy with a soldering iron. The necessary connections are shown in Fig.2.1. Although 5 way connectors are used, only three pins are actually used (the other two apparently being reserved for possible future expansion of the MIDI standard).

The type of cable needed is the twin screened variety, which has two insulated inner conductors covered by an outer conductor and an overall layer of insulation. There are numerous types of twin screened lead available, but assuming you are only going to use cables about 1 to 4 metres long, any twin screened type should suffice. One of the thinner and more flexible types is probably the best choice. Incidentally, MIDI is only guaranteed to work over a distance of 15 metres, and you might need to use a high quality cable to obtain something approaching this maximum range.

Pin 4 on one plug connects to pin 4 on the other, and the two pin 5s are also connected. These two sets of connections are carried by the inner conductors. The outer conductor is used to connect the two pin 2s together. The signal is carried by the inner conductors, and the outer one is needed to shield the cable to prevent it from radiating radio frequency signals that could cause interference to any nearby radio equipment. Internally, pin 2 of MIDI ports is only connected to earth at MIDI outputs (including "THRU" types). All inputs have an opto-isolator, which is an electronic component that consists of a light emitting diode (LED) having its light output directed onto a photo-transistor. The component is housed in an opaque casing so that the photo-transistor is shielded

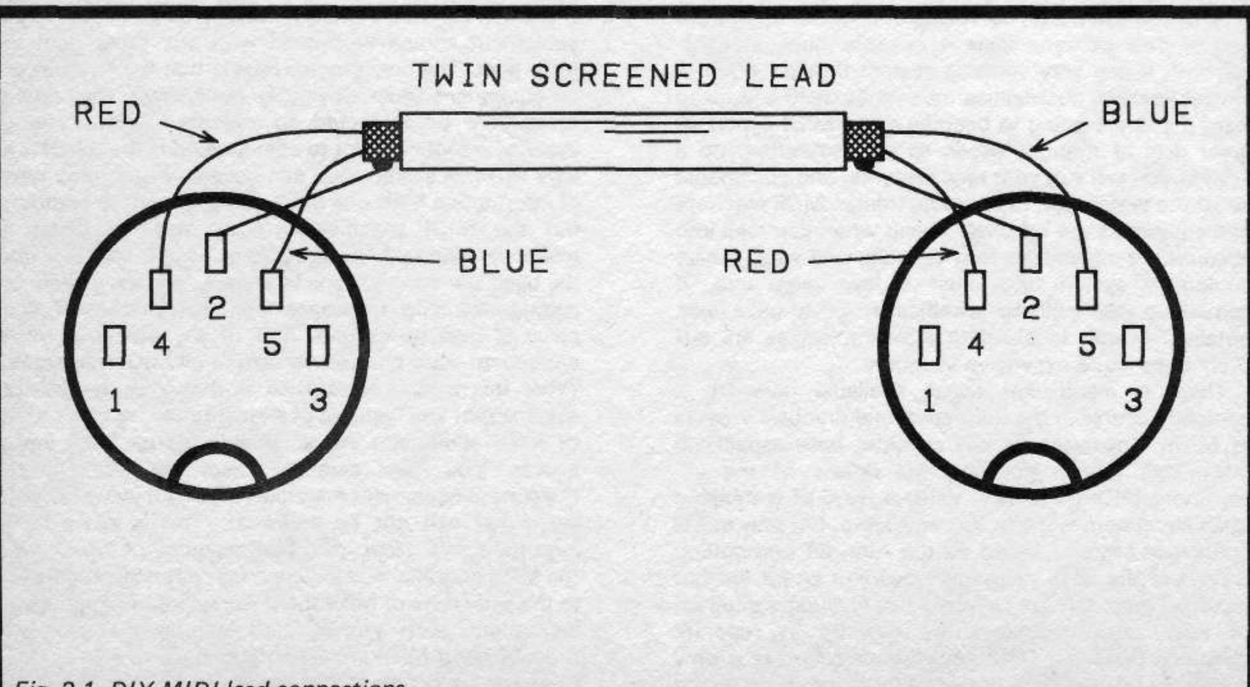


Fig. 2.1. DIY MIDI lead connections.

from any ambient light. Passing a small current through the LED results in its light output switching on the photo-transistor.

This may all seem to be quite clever but an over-complicated way of handling things. The important point is that there is no electrical connection between the various pieces of equipment in the system. At least, there is no direct connection via the MIDI ports. This is important for three reasons. One of these is that it minimises the risk of "hum" loops being produced by multiple earth connections. Probably most people involved in electronic music will be all too familiar with "hum" loops, and the background "buzzing" that they produce. The opto-isolation does not totally banish them, but it does at least remove another possible cause of these loops.

The second reason is that it avoids problems with the fairly high voltages that can exist between the earth rails of mains powered equipment. It is mainly double-insulated equipment (that does not use the mains earth connection) where this can be a problem. Even though there is normally very little power behind these voltage differences, they can still damage sensitive (and expensive) electronic components. This is not a purely academic problem, and on one occasion I did some serious damage to a computer by not having it correctly isolated from a short wave radio receiver it was controlling.

Reason number three is that computers and other digital controllers are very good at generating electrical noise. Try operating a radio set next to most home and personal computers and you are likely to find reception very difficult indeed. There is little chance of electronic musical instruments picking up this radiated signal, but with direct electrical connection to a computer there is a real risk of a certain amount of noise being inadvertently coupled through the connecting cable. The opto-isolation and non-linking of the equipment earths avoids this possibility.

ST MIDI Ports

The MIDI ports of the Atari ST computers are, unfortunately, not quite true MIDI standard ports. I suppose that for most purposes they can be regarded as fully standard in that they can be used as normal "IN" and "OUT" ports using any standard MIDI cable. In fact the "IN" socket is quite normal, but the "OUT" socket is actually a combined "OUT" and "THRU" type. The normally unused pins 1 and 3 are used to carry the "THRU" socket connections. Just why Atari chose to do things this way is not immediately obvious, and presumably was not to save the cost of a 5 way DIN socket! Perhaps there was no space for the third socket? Anyway, this arrangement is better than simply having the "THRU" facility omitted altogether (it seems to be absent from many pieces of MIDI gear, particularly keyboard instruments).

As an ST computer will normally be used as a controller, the "THRU" socket will often be unnecessary. The role of all three types of MIDI port is something that we will consider a little further on. If you should need the "THRU" facility, one solution would be to make up a little unit and connecting lead to give standard "OUT" and "THRU" sockets. A cheaper and easier solution is to use a non-standard lead to suit the combined "THRU"/"OUT" socket of the STs, and one of the larger Atari dealers may well be able to supply a suitable lead. If not, and provided you are at least moderately efficient with a soldering iron, it is not too difficult to make up a double lead to suit this port. Connection details for this lead are provided in Fig.2.2. Note that the outer conductors of both leads should connect to pin 2 of the plug at the ST end of the lead.

Basic Connections

There are almost limitless combinations of MIDI equipment that can be used, and there are usually several ways of connecting together a given set of MIDI devices. Provided you understand the basic function of the three types of MIDI port, you should not find it too difficult to wire systems together in a manner that will provide the functions you require. Here we will consider several setups that should give you the general idea of how systems can be constructed.

A basic ST plus a keyboard equipped MIDI instrument would use the arrangement shown in Fig.2.3. Just what such an arrangement would achieve in practice is entirely dependent on what software is used on the ST. Using different programs a variety of functions could be provided. The most common application for this setup would be to have the ST functioning as a real-time sequencer. This is the MIDI equivalent of a tape recorder, or perhaps a player-piano is a better analogy. We will not consider applications programs in detail here, as they are covered by a separate chapter, but with a real-time sequencer, anything played on the keyboard can be recorded by the computer and played back later. In order to record a sequence the keyboard must provide information to the computer, and this information is carried by the lead which connects from "OUT" on the keyboard instrument to "IN" on the ST computer. To play back a sequence the information must flow in the opposite direction, and this is the purpose of the cable which connects "OUT" on the ST computer to "IN" on the keyboard instrument. MIDI can provide a two way flow of information, but only via this system of cross-coupled "IN"/"OUT" sockets.

In a multi-instrument set-up the "THRU" sockets on some pieces of equipment must be brought into operation. Fig.2.4 shows the standard method of connection for a system that consists of an ST plus a keyboard instrument and two rack-mount instruments. This utilizes what is called the "chain" method of connection. This name is derived from the fact that a number of instruments are connected in a long series, with the "THRU" socket on one connecting to the "IN" socket on the next. As you might have guessed, all the "THRU" socket does is to provide a replica of any signal picked up at the "IN" socket. In this way the signal from the ST or other controller can be coupled to a number of MIDI devices.

In theory you can connect as many instruments as you like into the chain. In practice this method of connection might not give satisfactory results with really large setups. The "chain" system has a reputation for introducing cumulative delays, but there should really be no significant delay between a signal entering at the "IN" socket and being reproduced at the "THRU" socket. The problem is more likely to be one of

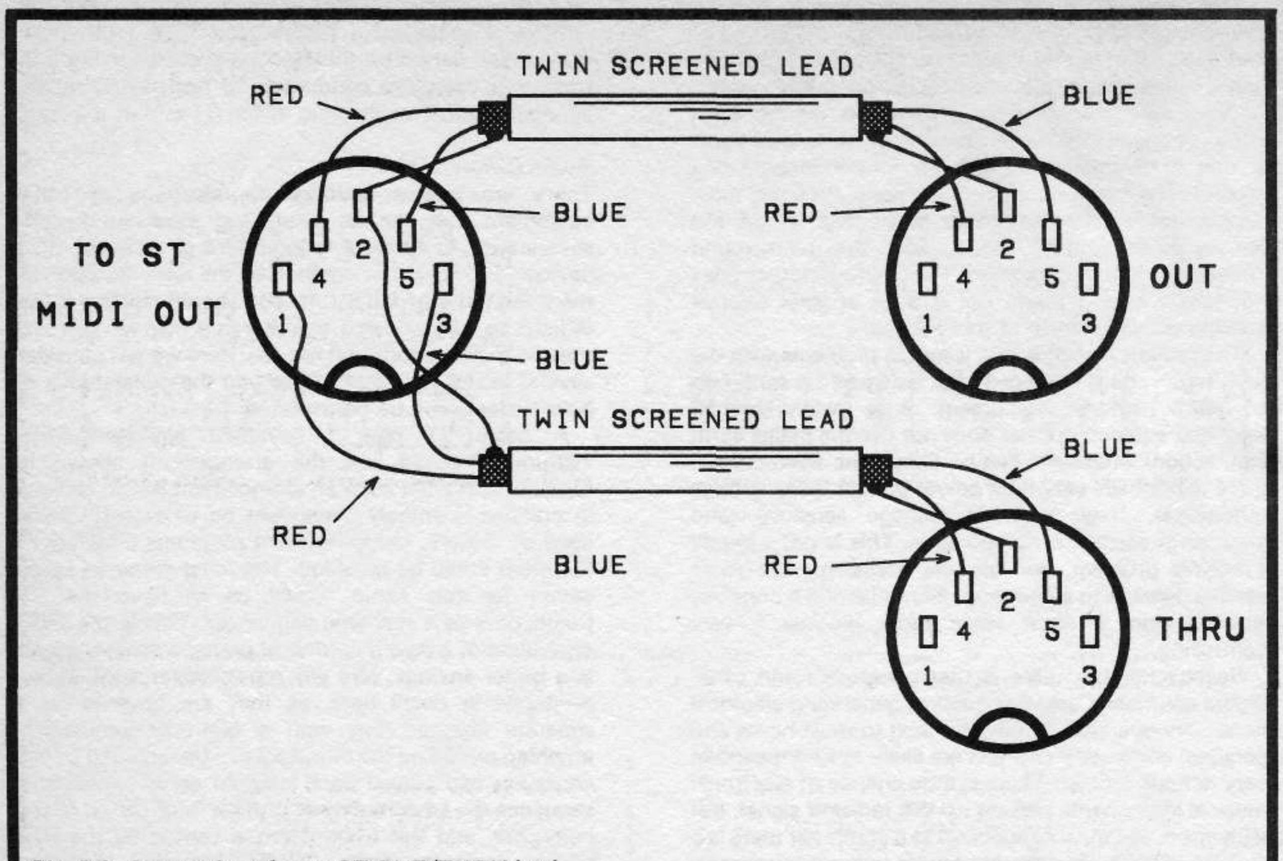


Fig. 2.2. Connections for an ST OUT/THRU lead.

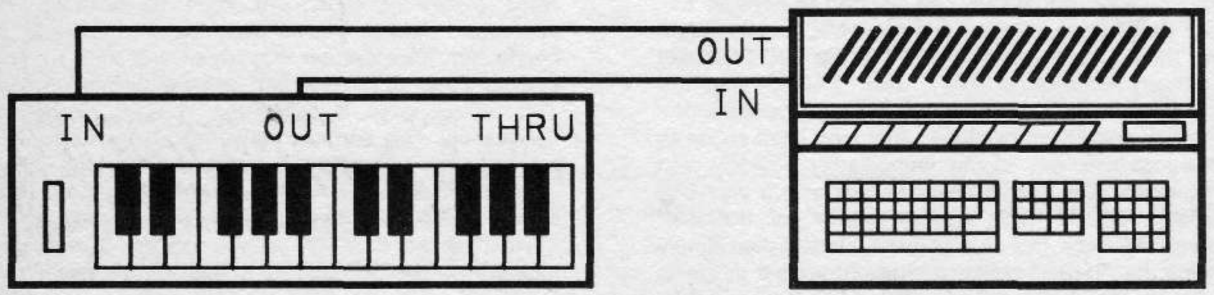


Fig. 2.3. A basic ST plus MIDI keyboard instrument setup.

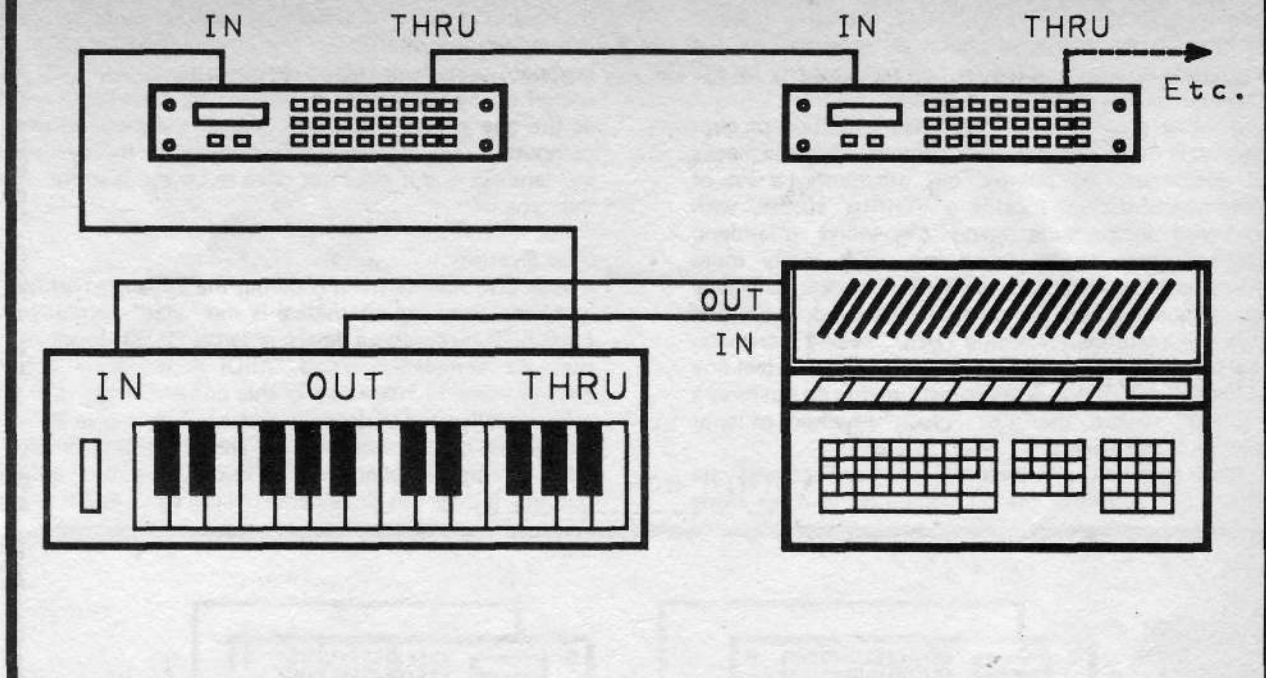


Fig. 2.4. A system utilizing "chain" connection.

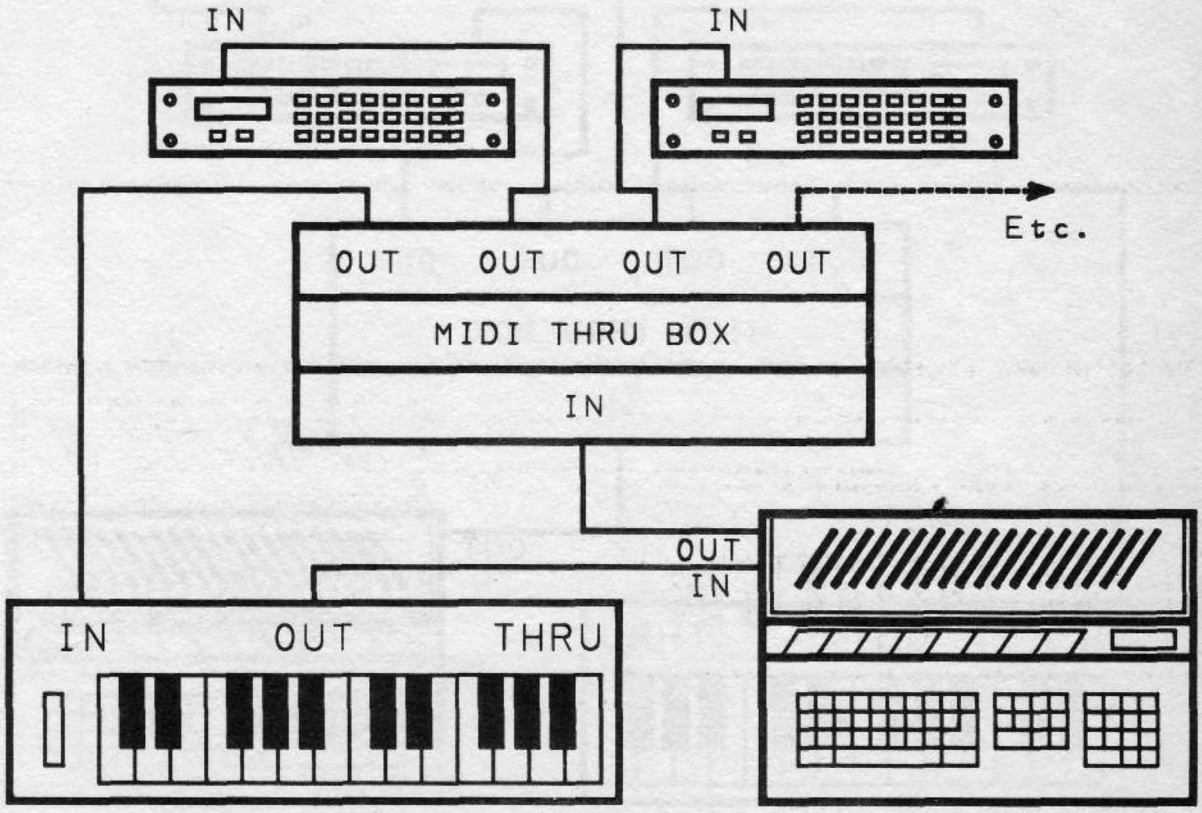


Fig. 2.5. Instrument connected using the "star" system.

the signal being degraded slightly on each coupling from a MIDI "IN" to a "MIDI" OUT. With a long series of instruments the signal could be "smeared" to the point where it can no longer be read properly by the instruments towards the end of the "chain".

Another problem with the "chain" method of connection is that it can not be implemented with all pieces of equipment. As pointed out previously, a lot of instruments do not include a "THRU" socket, with keyboard instruments being the worst offenders. Matters seem to be improving, with many more instruments now sporting a "THRU" socket, and they seem to be standard equipment for rack-mount units. If only one instrument lacks the THRU" socket there is no real problem. Simply use this instrument as the last one in the "chain". If two or more instruments do not have a "THRU" socket, then the "chain" method of connection is not usable.

Note that only one instrument is connected to the MIDI "IN" socket of the ST computer. In most cases this is all that will be needed. The keyboard will be used for recording real-time sequences, and no other device will supply information to the computer. In a multi-keyboard system, it makes sense to have the "OUT" socket of the instrument which has the best keyboard as the one which is coupled back to the input of the computer. This can be any instrument in the system incidentally, and it does not have to be the first one in the "chain".

Star System

If your instruments will not permit the "chain" method of connection, the alternative is the "star" system of Fig.2.5. This requires a device called a "THRU-box", or they are sometimes called "MIDI expanders". This second name is little used in this context these days, and is mostly used to describe add-on units to give MIDI pianos and organs more voices. The THRU box simply takes an input signal and splits it to provide a number of outputs. It does not simply channel the input signal to a

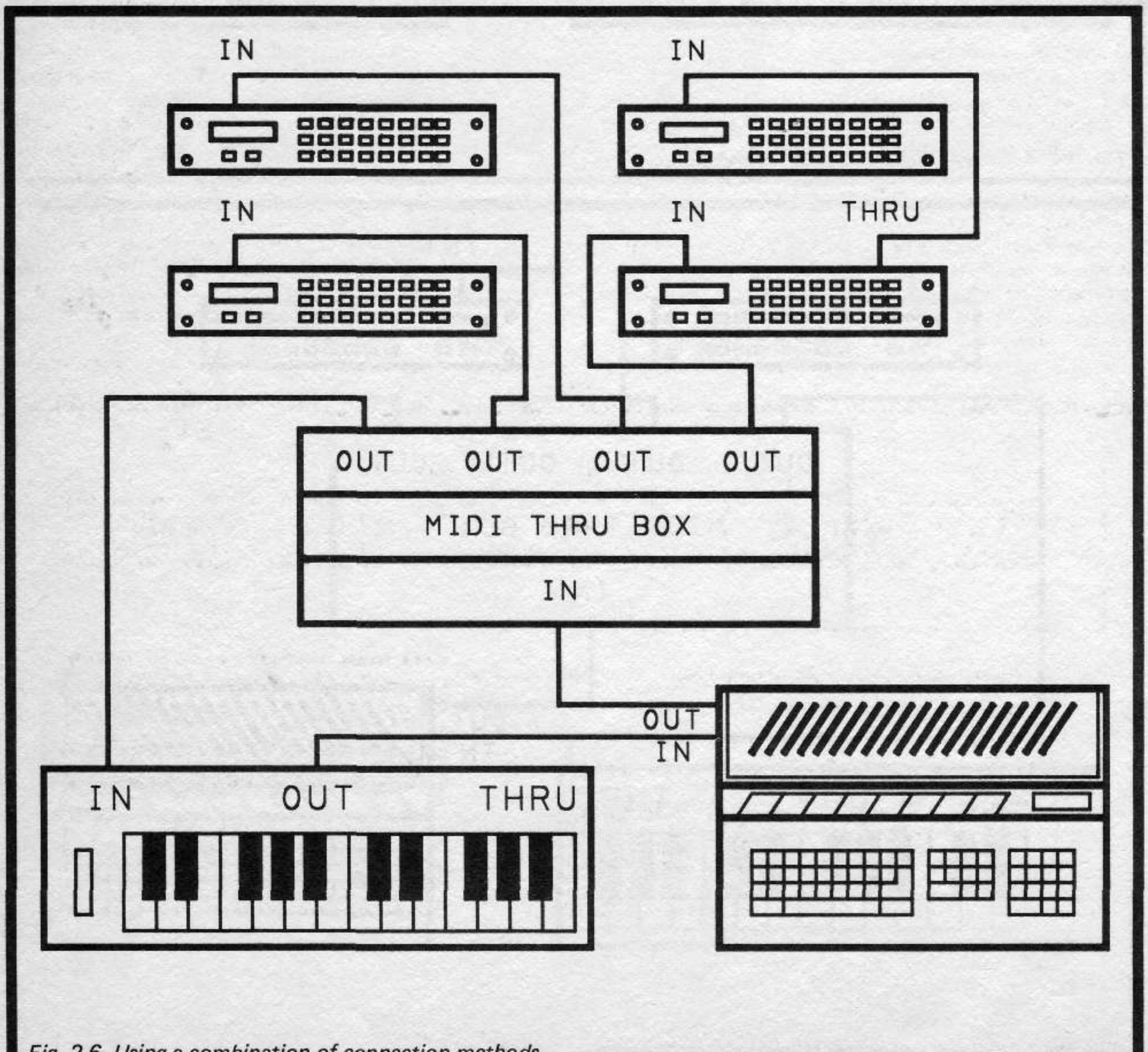


Fig. 2.6. Using a combination of connection methods.

number of output sockets, and this approach would not be acceptable. MIDI uses a drive current of about 5 milliamps, and splitting this between (say) five output sockets would only give 1 milliamp per output. This is unacceptable as it would probably not give sufficient output current for each instrument driven from the system. A THRU box Must have an amplifier or amplifiers so that each output can be driven at the correct output current.

Even if your instruments do have "THRU" sockets, you may prefer to use the "star" system. It avoids problems with delays, signal smearing, or whatever. On the other hand, it does mean the added expense of a THRU box. I think that I would be inclined to try the "chain" method first, and only bother with a THRU box if problems were experienced. Of course, you do not need to use one system or the other, and a combination of the two could be used if desired (perhaps if the THRU box has fewer outputs than you have instruments). Fig.2.6 shows an example composite system of this type.

More Complex Setups

For most users the only items of MIDI equipment they will use are musical instruments, plus perhaps a computer or other micro- controller. There are many other MIDI devices available though, including such things as audio mixers and effects units. As far as the MIDI connections are concerned, these devices are wired into the system in exactly the same way as the musical instruments. The only exceptions are devices which act as controllers, and those which are MIDI processor. By a MIDI processor I mean a device which actually processes the MIDI signal in some way, and not an audio processor that is under MIDI control.

A MIDI processor generally goes ahead of one item of equipment, since in most cases it is only the information for one device that must be altered. There are various types of MIDI processor available, and a look through some leaflets from the main equipment manufacturers will give you a good idea of the sort of thing that can be obtained. A typical example would be a MIDI filter, which is a device that can be set up to remove certain types of MIDI message from the data stream. For example, you might want an instrument in the system not to respond to external pitch bending. Some instruments can provide built-in filtering of various types, but with one that has no pitch bend disable facility an external data filter could be used. This would be used ahead of the instrument to which the filtering must be applied, but after any other instruments in the system, as in Fig.2.7. If the filtering must be applied to more than one instrument, then the filter can be moved to an earlier point in the chain. With the "star" method of connection the filtering can only be applied to one instrument (Fig.2.8) or all of them (Fig.2.9).

Filtering or other forms of MIDI processing can also be applied to the signal from a keyboard prior to it being fed into the ST computer. The necessary method of connection is shown in Fig.2.10. Filtering is the most likely form of processing to use in an arrangement of this type. In particular, real-time sequencers normally record all the MIDI data that is fed into them. This gives a very faithful reproduction of the original performance when the sequence is played back, but it can result in the sequencer running out of memory before the sequence is completed! Most sequencers can handle a few thousand notes or more, and ST based systems can often handle 50,000 notes or more. However, this assumes that only basic note on and note off messages are received.

Other types of data can take up a very large amount

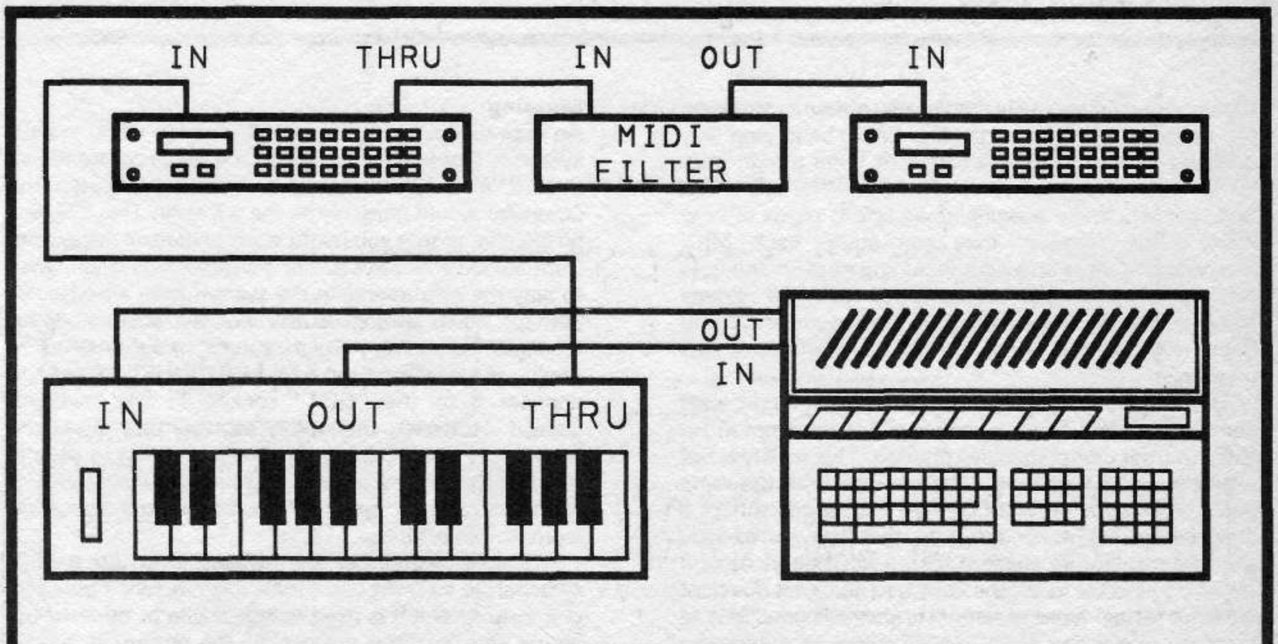


Fig. 2.7. "Chain" connection with filtering applied to one instrument.

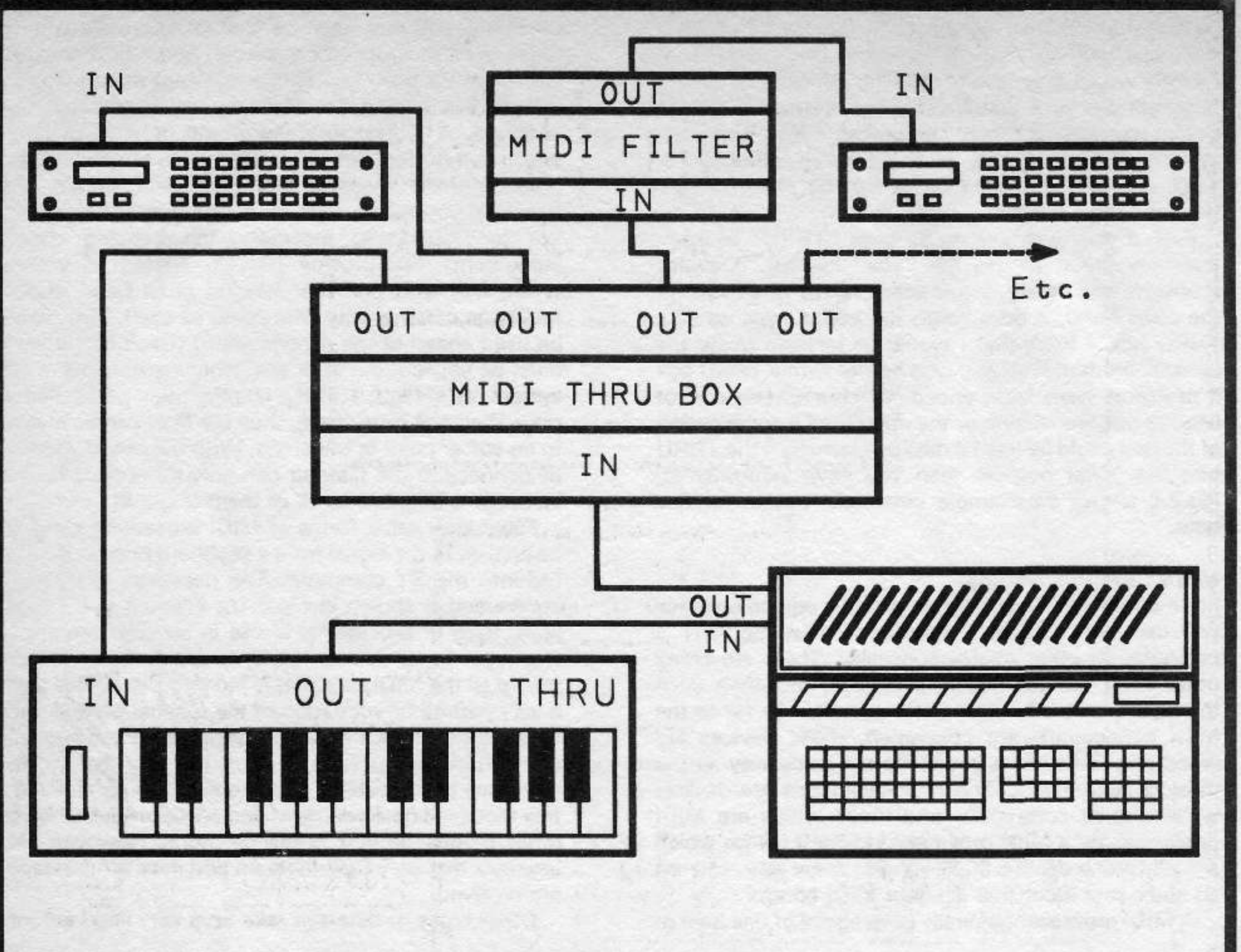


Fig. 2.8. "Star" connection with filtering applied to one instrument.

of memory, and seriously deplete the amount remaining for note storage. In particular, pitch bend and key pressure data can be transmitted at quite a high rate, filling up memory remarkably quickly. You will often find that sequencer specifications talk in terms of how many MIDI "events" they can store. Each MIDI message counts as an event, including note on and note off types. A storage capacity of 100,000 events therefore represents an absolute maximum of 50,000 notes (i.e. 50,000 note on and 50,000 note off messages).

The more sophisticated sequencers permit at least some basic MIDI filtering, and it is not uncommon for them to offer comprehensive filtering. This facility is not available on all sequencers though, and with the more basic types an external MIDI filter can be more than a little useful. Bear in mind though, that a new and better sequencer might be cheaper than a MIDI filter! Also, it might be possible to set the keyboard so that it does not transmit certain types of data. It is always a good idea to carefully check the MIDI specifications of equipment and the control settings/functions available. There are often details of some very handy features tucked away in the "fine print".

Merging

An essential point to bear in mind when using a MIDI system is that MIDI only caters for a single controller per system. In a system based on an ST computer, the controller would normally be the ST itself. This can be a bit limiting, in that you might want to control the system from some other device. For example, you might wish to play the instruments in the system from a keyboard, perhaps when setting up the required sounds, or for rehearsal purposes. It is, of course, quite possible to unplug the lead from the ST's MIDI "OUT" socket and connect it to the "OUT" socket on the keyboard instead. However, this hardly represents a quick and convenient way of doing things. Something to keep in mind is that repeated plugging-in and unplugging of leads can quickly result in the connectors becoming worn and unreliable.

A simple solution to the problem is to use a MIDI switcher. In its most basic form this has two inputs and one output, and it is used to select one or other of the inputs and couple it through to the output. It can be used in the manner shown in Fig.2.11, which might look a little confusing at first, but is really quite straightforward. With the switcher set to position "2" you have a standard arrangement where the ST

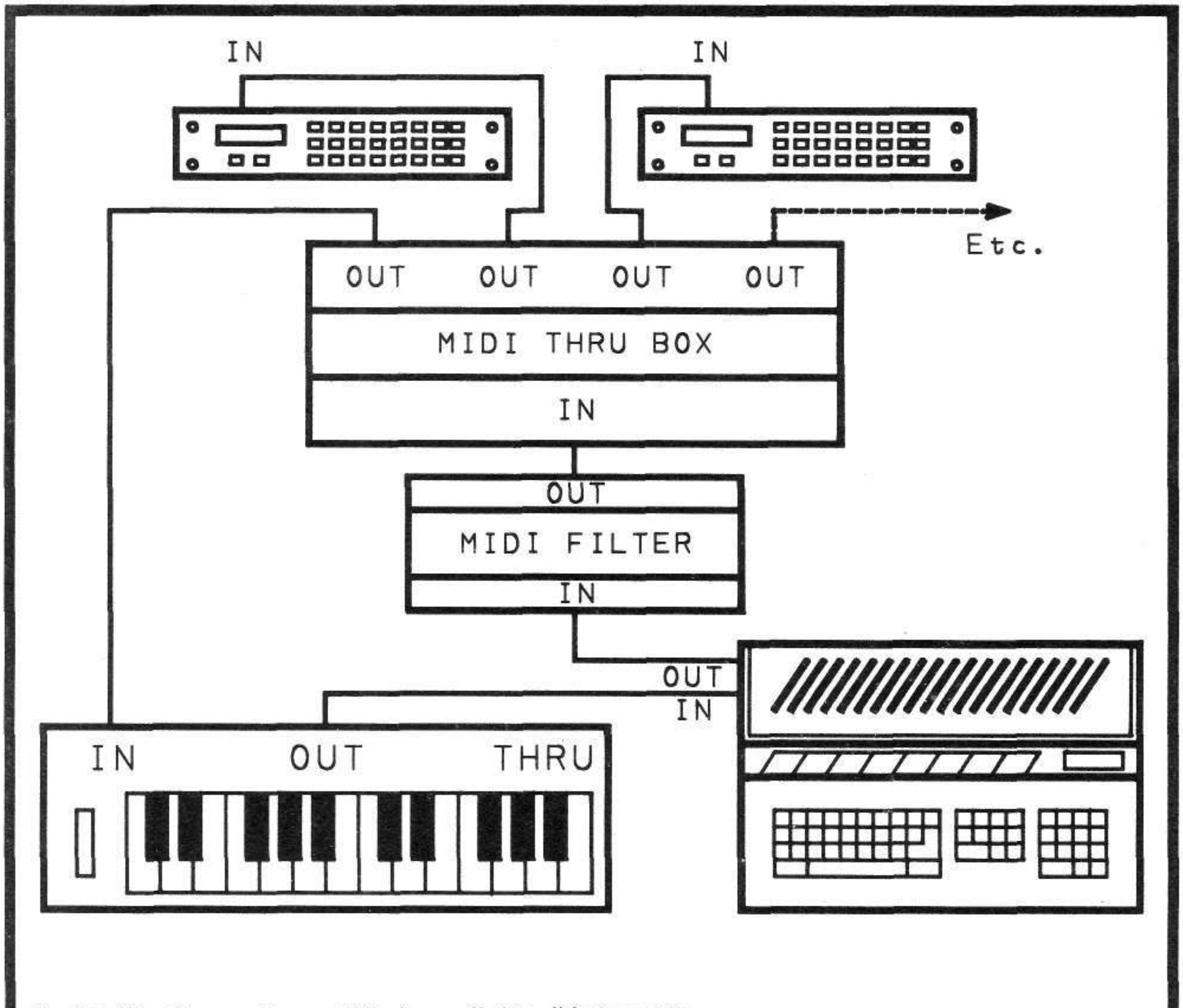


Fig. 2.9. "Star" connection and filtering applied to all instruments.

controls all the instruments in the system, and the first instrument is a keyboard type which can feed information back to the ST for real-time sequencing. Suppose you wish to control the instruments from the keyboard for a while, and that you do not need the ST. Setting the switcher to position "1" gives the desired effect. The signal from the keyboard is still coupled to the "IN" socket of the ST, but the output from the ST's "THRU" socket is now coupled through to the two rack-mount instruments. The ST takes no active role in the system, and the setup is essentially the same as if the "OUT" socket on the keyboard instrument was coupled through to the "IN" socket on the first of the rack-mount instruments. Note though, that the ST is unlikely to provide a proper coupling from its "IN" socket to its "THRU" socket unless it is left switched on.

Some MIDI switchers (in fact most of them) have a multi-way switch and several outputs. Being able to direct a signal to one of several instruments is not necessarily a particularly useful facility, especially in a computer based system. For most purposes a basic two way switcher will suffice. Going further up-market there are MIDI "patch-bays", "directors", or whatever the manufacturer decides to call them! Whatever the name, the idea is to have a unit which has a number of inputs and outputs, with the input and output of every piece of equipment in the system being connected to it. Switches on the unit are used to connect the inputs and outputs in the required manner. Often there are several programmed methods of connection available at the touch of a button. This is a sort of highly flexible THRU box. For a complex setup a unit of this type is certainly very desirable, but the ones I have encountered have been quite expensive. Probably most users would feel that the money could be spent more effectively on some other aspect of the system.

A more sophisticated form of data direction device is available in the form of a MIDI "merge" unit. As its name implies, it merges two signals together to give a single output. A very basic merge unit simply mixes the two signals together, and with simultaneous inputs will not give an acceptable output. Simply jumbling two digital signals in this way gives a totally "scrambled"

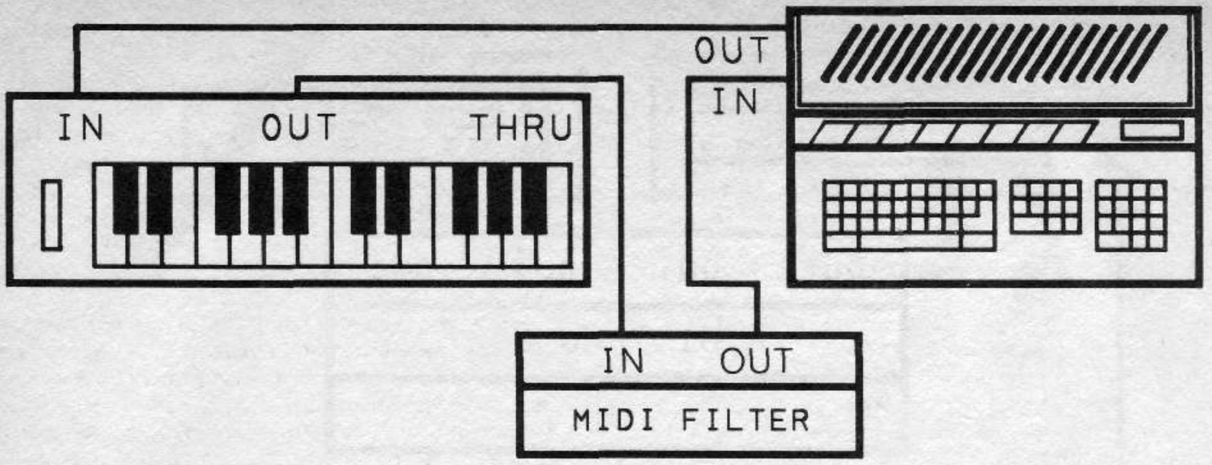


Fig. 2.10. Filtering the MIDI input to the ST.

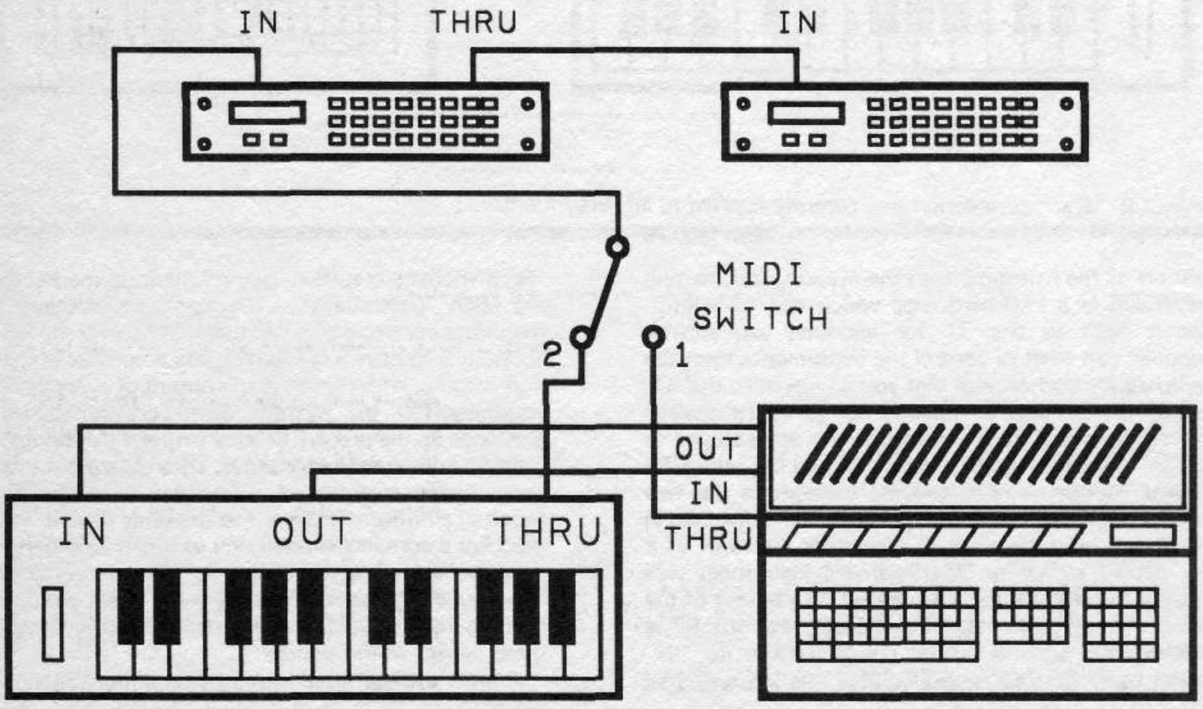


Fig. 2.11. Using a MIDI switcher.

output signal. A unit such as this might not seem to be very useful, but it gives an action that is very much like an automatic switcher. In other words, you must arrange things so that only one input or the other is active, and the unit then effectively switches this input signal through to the output. A unit of this type should work perfectly well in the system of Fig.2.11 if it is used in place of the switcher. Output signals from either the keyboard instrument or the ST should then be coupled through to the two rack-mount instruments without any need to operate a changeover control.

A sophisticated merge unit will provide proper mixing of the two input signals. If a signal is received on one input, this is coupled through to the output socket. If an input signal is received at the other input while the first signal is still being processed, then this second signal is stored in a small block of memory (called a "buffer"). When the first signal ceases, the stored signal is then coupled through to the output. If, during the course of the stored signal being transmitted, the signal on the first input should be resumed, this signal is stored in another buffer until the output is free and it can be transmitted. This can only work if the two inputs, on average, are only busy for about 50% of the time. In practice it is unlikely that there would be so much data on the two inputs that it would be impossible to transmit it all. It could happen though!

This is a very useful accessory, but is again one that many users would probably consider to be too expensive. With a proper merge unit a system of the type outlined in Fig.2.12 becomes a possibility. Here two rack-mount instruments are being played simultaneously from a MIDI keyboard and the ST computer. The ST could be providing something like a simple repetitive accompaniment for the piece played "live" from the keyboard. This method of connection includes a lead from the final "THRU" output back to the "IN" socket of the ST. This enables sequences to be recorded from the keyboard into the ST if required. Although a fairly simple method of connection, such a setup is very versatile. You can play the instruments from the keyboard, operate them from the ST, use both methods of control simultaneously, and play sequences into the ST while monitoring your playing on the instruments. All this with no switching whatever being needed when moving from one mode of operation to another.

An important point to realise when designing MIDI systems is that it is only signals received on the "IN" socket of a piece of equipment that appear at its "THRU" output. Signals generated by the device do not appear at its "THRU" socket, and signals received at its input do not appear at its "OUT" socket. On the other hand, some instruments do seem to have options

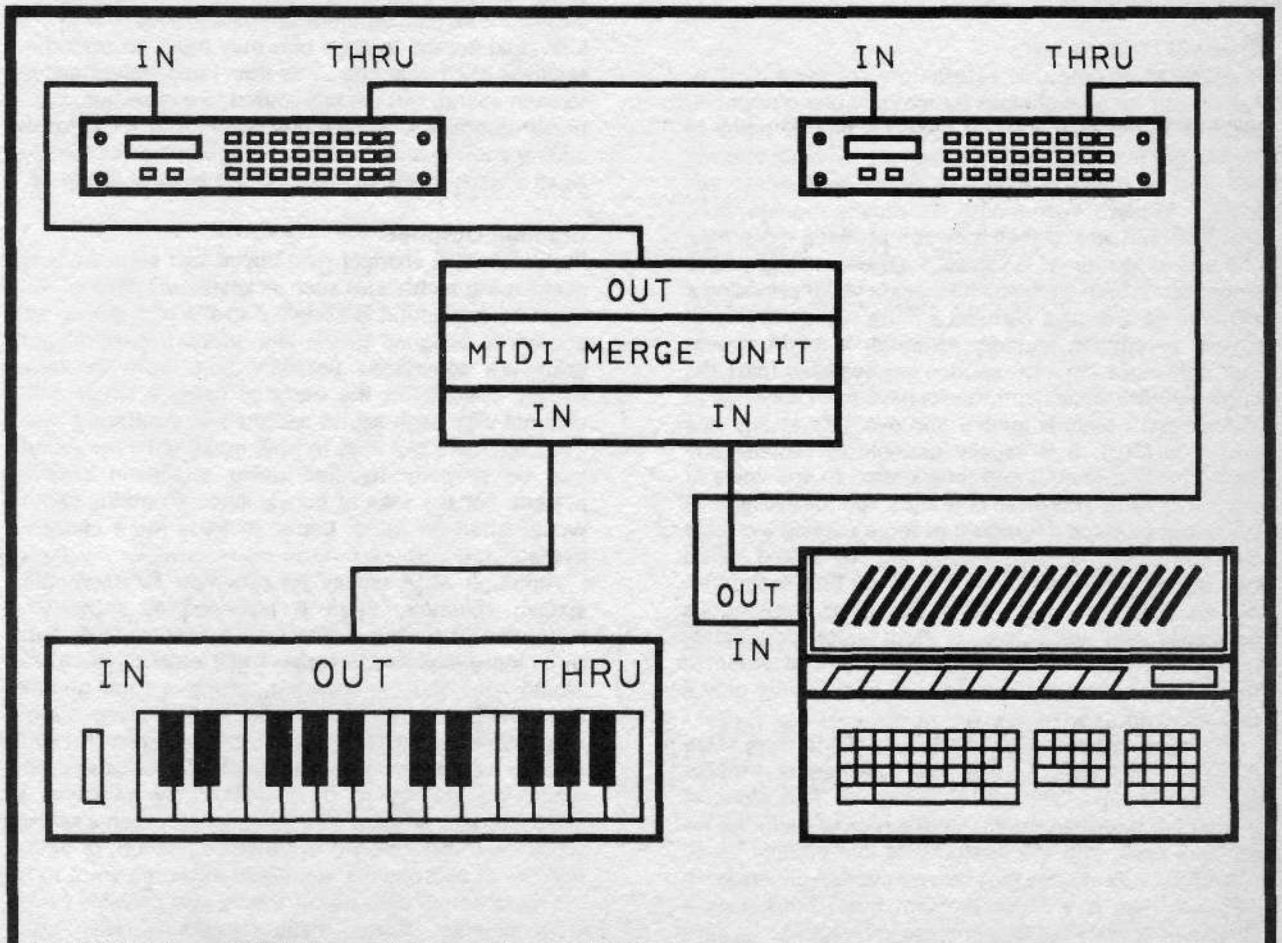


Fig. 2.12. Using a MIDI Merge unit to combine two MIDI output signals.

that allow signals to be routed to sockets where they would not normally appear. This can be very useful indeed, and could quite possibly enable a setup that would normally need a MIDI switcher or merge unit to be produced without using either of these. If you have an instrument with a facility of this type, make sure that it is only enabled when it is required. Odd things could start to happen if signals start appearing at the wrong places in a system. Be especially careful to avoid a situation where the signal originated by a device is fed back to its "IN" socket and then retransmitted through the system. In theory this would result in signals being circulated through the system indefinitely, and in practice it would almost certainly cause the system to run totally out of control.

It is also worth bearing in mind that some MIDI sequencing software for the ST permits data received on the "IN" socket to be "echoed" to the "OUT" socket. This is an increasingly common feature, and one that is well worth having. It can greatly simplify matters when setting up a real-time sequencing system.

When designing any reasonably complex MIDI system you should always draw out the entire system, showing all interconnections. Then trace the signal paths to make sure that all the MIDI messages will get through to where you want them to go. Check for any feedback loops that could result in signals being circulated indefinitely!

Channel Limitations

A single MIDI output is satisfactory for most requirements, but for an ambitious system just one output can become rather restrictive. It depends on the types of instrument in the system, and the way in which you will use it. For many purposes a single synthesiser will suffice. Modern instruments are mostly multi-timbral, which simply means that they can produce more than one sound at a time. A typical synthesiser might have eight voices, with each one being capable of providing a different sound, plus perhaps a ninth voice capable of several percussion sounds. Although it might appear that only eight different sounds are available (plus the drum sounds), most synthesisers have from about 32 to 128 different sounds loaded and available at any one time. Via MIDI, it is usually possible to allocate any sound (or "program") currently loaded to any voice of the instrument. This gives only eight sounds available at once, but perhaps a hundred or more sounds available almost instantly simply by assigning any stored sound to a voice of the instrument. Even with only monophonic operation on each voice this gives tremendous potential, and with two or four note polyphonic operation on each voice it gives the kind of potential that would have needed a bank of instruments only a few years ago.

However, suppose that you would like to have more than eight sounds plus the percussion channel available at any one time, and added a second and identical instrument into the system. On the face of it there is no problem here, and the instruments can provide up to sixteen sounds at once plus two percussion channels. In practice there is a slight problem here. MIDI uses a system of channeling that enables messages to be sent to just one device in the system, or with a suitable instrument, to one voice of an instrument in the system. There are sixteen channels, and while this may have seemed to be quite generous when MIDI was first devised (which was mostly in 1981 and 1982), modern instruments make this look a bit miserly. In our two instrument example above, with each voice and the percussion tracks assigned to different MIDI channels, some eighteen MIDI channels would be required. Using a single MIDI output this is clearly not possible.

Some instruments are not as versatile in assigning sounds to voices as are most synthesisers. In particular, sound samplers do not normally use program changes to provide a choice of dozens of different sounds, all "on-tap" and assignable to any voice as and when required. Due to the way sound samplers work, they inevitably require quite large amounts of memory per sound, and no instruments currently available have sufficient memory to keep a hundred or so sounds stored and ready to use. When using sound samplers you often need to have a different voice per sound. If you need to individually sequence each sound, then each one must have a different MIDI channel. In other words, you may be restricted to a choice of sixteen sounds per score.

Even if a sampler can have more sounds stored than it has voices, this does not necessarily improve matters. I have an Akai S700 sampler which is a six voice instrument, and can normally store up to six different samples. With a memory expansion board added it can store up to sixteen different samples at once, but to give individual access to each one they must be placed on separate MIDI channels. You then have instant access to each sound, but are still limited to a maximum of six notes at once. Unless sound layering is to be used, adding another instrument to the system is not possible as all sixteen available channels are already occupied.

Multiple Outputs

Problems with channel restrictions can sometimes be eased using techniques such as keyboard splitting. This is where one sound is assigned to the high notes, and another is assigned to the low notes. In fact multiple splits are sometimes possible, and drum machines usually operate on the basis of using a single MIDI channel with each sound assigned to a different note. Even so, there is a limit to how many different sounds can be properly handled using a sixteen channel system. For the sake of convenience if nothing else, it would often be much better to have more channels available.

Although MIDI makes no provision for more than sixteen channels, there is no need to accept the restriction of having a single MIDI output. With (say) three individually addressable MIDI outputs, each one would have sixteen channels, giving a total of forty eight channels available. It is difficult to envisage such a large number of channels ever being required, but such a setup would provide almost limitless possibilities, and would be unlikely to be overtaken by advances in electronic instruments. The limitations of such a system would be those imposed by your imagination, or by the number of instruments you could afford connect to it!

This is not all a matter of theory and possible future developments. Some manufacturers of multi-track MIDI software for the ST have realised that their software can potentially go beyond the capabilities of a single MIDI output port, and they have made available interfaces that provide extra MIDI outputs. As far as I am aware, there is no standardisation of these interfaces, and each type is only designed to work with particular programs from one manufacturer. If you set up a complex MIDI system that makes use of multiple MIDI ports, you must therefore ensure that you obtain the right interface for the program you intend to use.

The C-Lab "Export" interface is a good example of a multiple MIDI port. It is designed for use with the C-Lab "Creator" and "Notator" programs, which provide 64 track operation. The "Export" interface connects to the ST's "modem" port (the RS232C serial port) and provides three additional MIDI output designated ports "B" to "D". The ST's MIDI output acts as port "A", giving a total of four MIDI outputs and some sixty four channels. In other words, the "Creator" and "Notator" sequencers can operate on the basis of one track per MIDI channel. With some sequencers you have perhaps thirty two tracks available, but only a single MIDI output and sixteen MIDI channels. Thirty two tracks into sixteen channels will go, but only on the basis of two tracks per MIDI channel or some similar arrangement.

A setup based on the ST plus a MIDI interface box could be something along the lines of the system depicted in Fig.2.13. While this does not look much different to a standard MIDI system connected in the "star" configuration, it is substantially superior in that it has many more channels available, and can potentially provide much more complex and accurately timed sequencing. With complex systems and a single MIDI output there is a real risk of MIDI "choke". This is simply where a sequence at times requires more data to be sent over the MIDI interface than can be handled in the available time. Just what happens when MIDI "choke" occurs depends on the software in use, but it will at least cause a significant error in the timing of some notes. It could easily result in some notes being missed out altogether, or notes being left switched on, and at worst could result in the computer crashing.

A system of the type outlined in Fig.2.13 probably goes beyond most peoples' needs, but if you are going to set up a complex system there is a lot to be said in favour of a MIDI output box and supporting software.

Whether you use a single instrument or a highly sophisticated system, the ST computer should be capable of controlling it very effectively. There is plenty of MIDI software available for the ST, covering a wide range of prices and just about every conceivable MIDI application. You will be disappointed if you expect the ST plus MIDI to make you a better musician overnight. You will not be disappointed if you expect it to remove the shackles so that you are limited only by your imagination and not by other factors. With the aid of an ST computer and MIDI equipment you should be able to develop your skills much faster than you would have thought possible.

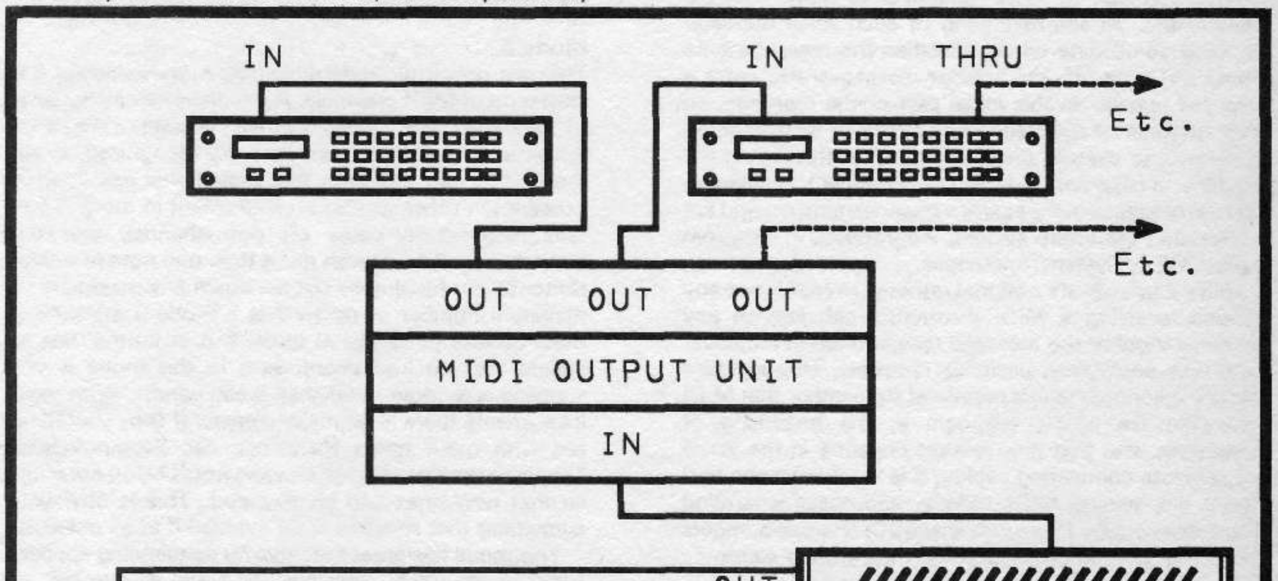


Fig. 2.13. A multiple MIDI output port gives many more channels (a total of 64 in this case).

Chapter 3
GETTING THE MESSAGE

In the previous chapter various ways of interconnecting MIDI equipment to an ST computer were considered, but we did not consider the types of message that can be sent via MIDI. This is quite an involved subject which fully merits this chapter which is solely devoted to MIDI message types. The scope of MIDI goes well beyond the basic note on, note off, and note value information of the gate/CV system. It can handle key velocity, key pressure, synchronisation of sequencers, and many other types of information. This aspect of MIDI can be a little bewildering at first, but it is well worth taking some time to study the message types and gain a firm understanding of as many of them as possible. Unless you understand a reasonable range of MIDI messages you will have little chance of fully exploiting MIDI and your ST.

Modes

Before looking at the various types of MIDI message it would be as well to examine the subjects of channels and operating modes. Unless you understand the four MIDI modes and channelling, a lot of MIDI messages will be difficult or impossible to fully understand.

The concept of MIDI channels is quite easy to understand. At the beginning of each MIDI message there is some data which specifies the message type (note on, note off, etc.). Most messages also carry a channel number in this initial part of the message, so that they can be directed to one particular instrument in a system, or even to one voice of an instrument in the system. These are the MIDI "channel" messages. Some messages do not carry a channel number, and are directed at the entire system. Appropriately, these are called MIDI "system" messages.

MIDI channels are notional rather than real, since any device receiving a MIDI instruction can act on any channel number the message may contain in whatever way the equipment designer chooses. This includes simply ignoring channel numbers! Remember that MIDI channels are simply numbers at the beginning of messages, and that they are not channels in the sense of separate connecting cables. It is for this reason that there are several MIDI modes, and these operating modes only differ in the way that MIDI channel numbers are treated. In all other respects they are the same.

Mode 1

Mode 1 is alternatively known as "omni on/poly", and was originally called just "omni" mode (and often still is). This is the most simple mode, and the one to which most instruments default at switch-on. The "omni" part of the name means that channel numbers are ignored, and that an instrument in this mode will respond to any note on and note off messages that are received regardless of the channel number they contain. Exactly how received notes are assigned internally to an instrument's voices depends on the design of the instrument, and there is no standard for this. In most cases an instrument in mode 1 will respond to notes received via the MIDI input in exactly the same way as it would respond to the same sequence played on its keyboard.

This mode is intended as a basic mode which should enable any piece of MIDI equipment to function to some degree in conjunction with virtually any other piece of MIDI gear. It lacks versatility though, and is far from ideal for most sequencing. It is fine if you are using a single instrument which has all its voices producing the same sound, but it is not much use for anything else.

Mode 2

This has the alternative name of "omni on/mono". Like mode 1, channel numbers are ignored in this mode. The "mono" part of the name indicates that only monophonic operation is provided. This mode was presumably included in order to accommodate monophonic synthesisers, but few monophonic instruments equipped with MIDI interfaces have ever been produced. This mode is not normally included on polyphonic instruments, as there is no obvious advantage in downgrading them to monophonic types (which is effectively what would happen by switching to this mode)!

Mode 3

This is a powerful mode which does acknowledge the existence of MIDI channels. It has the alternative name of "omni off/poly". The "omni off" section of the name indicates that channel numbers are recognised, while the "poly" part indicates that polyphonic operation is possible. In other words, an instrument in mode 3 will only respond to notes on one channel, and the instrument will work with more than one note at a time switched on. MIDI does not set down any maximum or minimum number of notes that a mode 3 instrument must be able to handle at once. It is up to the user to ensure that an instrument used in this mode is not supplied with more notes than it can handle. With most instruments there is no major disaster if they should be fed with more notes than they can accommodate. Usually it simply results in existing notes being cut short so that new ones can be sounded. This is obviously something that should still be avoided if at all possible.

This mode has great potential for sequencing applications, since it is possible to have a number of instruments on separate MIDI channels providing different sounds. This gives you a sort of computer controlled orchestra, and tremendous potential to develop complex pieces of music. This mode was originally called "poly" mode incidentally, and is still occasionally referred to by this name.

Mode 4

This mode is widely regarded as the most powerful one for sequencing purposes, although I suppose that this is not strictly true. Its alternative name is "omni off/mono", but it is probably still better known by its original "mono" name. As the "omni off" part of the name implies, MIDI channel numbers are recognised in this mode. The "mono" part of the name is perhaps a little misleading in that it suggests that a mode 4 instrument can only provide monophonic operation. This is not true though, and operation is monophonic only in that each voice of the instrument operates monophonically. If an instrument has sixteen voices, then in mode 4 each voice is assigned to a different MIDI channel and overall, sixteen note polyphonic operation is possible. This may not seem to be much better than the basic mode 1, but it gives tremendous scope when applied to a multi- timbral instrument. Each channel can then have a different sound, and a single instrument can provide a computer controlled orchestra.

This sort of system is actually less powerful than a mode 3 type having a number of instruments, because in mode 4 each channel only gives monophonic operation. It is popular with MIDI sequencer users because it gives extremely good results at an affordable price. A sixteen channel mode 4 instrument (or two eight channel mode 4 instruments) should cost substantially less that sixteen mode 3 instruments! An important point to realise is that you do not need to have all the instruments in a system working in the same mode. It is perfectly alright to have something along the lines of the system shown in Fig.3.1. Here the ST is controlling two mode 4 instruments on channels from 1 to 15, and an eight channel polyphonic instrument in mode 3 on channel 16. These three instruments could not be accommodated using mode 4 alone, as this would require twenty two MIDI channels, and there are only sixteen available. When sequencing using more than two instruments it is normally the case that a mixture of modes 3 and 4 offers the greatest potential, but this obviously depends on the precise facilities offered by the instruments.

Beyond Mode 4

The MIDI specification only details these four operating modes, but as MIDI and instrument technology has progressed, so have MIDI operating modes. A number of equipment manufacturers now produce instruments which have operating modes that do not adhere strictly to the MIDI specification, and which provide greater versatility. These are given various names such as "multi" mode, and "special" mode, and as far as I am aware there is no standardisation of these modes. In order to find out whether or not any of your instruments support any extra modes, and if so, what facilities they provide, you must carefully study the manuals.

What is probably the most basic enhanced mode is a version of mode 4 where the channels occupied by an instrument do not need to be contiguous. In other words, instead of having (say) a six voice instrument which can occupy channels 1 to 6, 2 to 7, 3 to 8, etc., each voice could be assignable to any desired MIDI channel. A common and very useful mode is one that is very much like mode 4, but with each voice not being restricted to monophonic operation. The exact way in which this works varies from one instrument to another, but as an example a sixteen note polyphonic instrument could be designed to provide two note polyphony on eight channels, four note polyphony on four channels, or eight note polyphony on two channels. With something like a thirty two note eight voice polyphonic instrument you might be able to have the instrument on eight MIDI channels with four note polyphonic operation on each channel. Some manufacturers seem to be working hard on these "single instrument ensembles" which are primarily aimed at MIDI sequencing applications. This sort of mode is generally called "multi" mode, but this is an unofficial title.

Some instruments are very flexible in their allocation

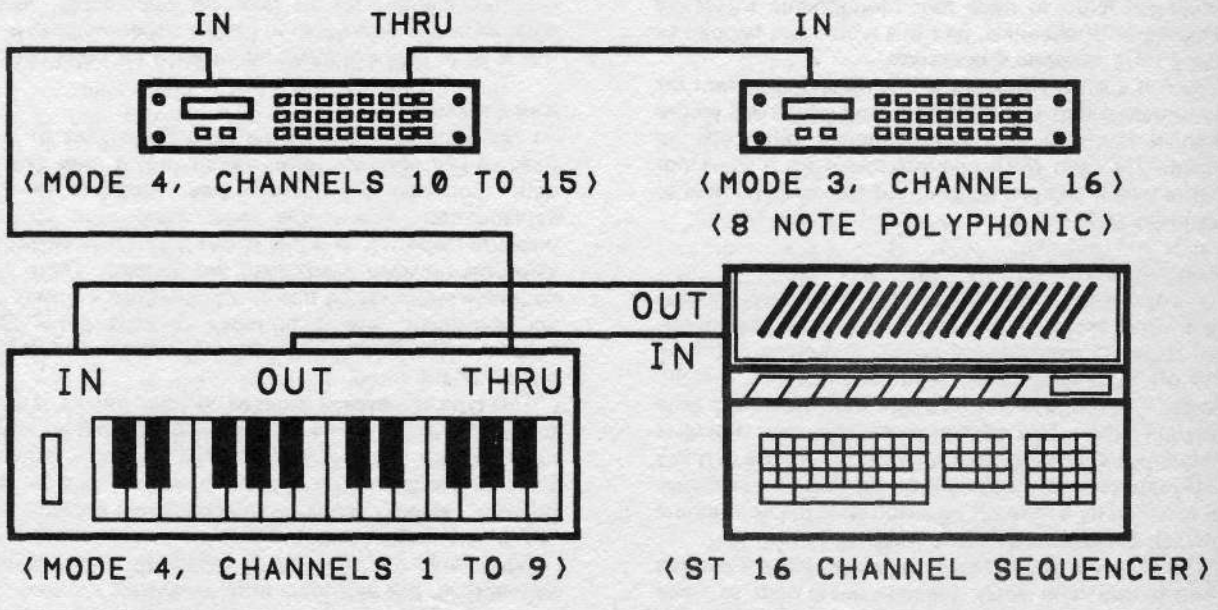

IN
THRU
IN
(MODE 4, CHANNELS 10 TO 15)
(MODE 3, CHANNEL 16)
(8 NOTE POLYPHONIC)
OUT
IN
IN
OUT
THRU
(MODE 4, CHANNELS 1 TO 9)
(ST 16 CHANNEL SEQUENCER)

Fig. 3.1. A MIDI system can use more than one mode, as in this Mode 3/4 example.

of notes, voices, and MIDI channels. With (say) a six voice six note polyphonic instrument, it could be designed so that up to six notes at once could be provided by each voice. This is not to say that 36 notes (six at once on each of six channels) can be provided. Usually with this type of mode you are still restricted to a maximum of six notes at once (or whatever), but those six notes can be on any voice or combination of voices. In other words, you could go straight from six notes on one voice to one note on each of the six voices without any mode change.

These extra modes are mainly aimed at those who use MIDI for sequencing, and are the type of feature that should prove very useful to someone with an ST based system. Although they might seem to compromise compatibility with other MIDI equipment, this is not usually the case. To a sequencer it does not matter whether it is driving eight separate polyphonic synthesisers in mode 3, or one instrument that is using a special MIDI mode to give the same effect. To the user there can be a massive difference, in that the single instrument ensemble approach can be vastly cheaper, but the end results are not necessarily inferior. Incidentally, when an instrument can provide polyphonic operation on several MIDI channels, each voice is sometimes referred to as a "virtual" instrument.

Transmission Modes

So far we have only considered reception modes, and not transmission modes. Really, modes are a standard for handling received data, and talking about transmission modes is perhaps not totally valid. With a sending device you do not normally set an operating mode as such. If you have a sequencer giving monophonic operation on four channels, then I suppose that it could be accurately described as a mode 4 device. You would not set the device to mode 4 though, you would set it up to have four monophonic tracks on separate MIDI channels, and this would just happen to give a form of mode 4 operation.

Even if a MIDI controller is driving an instrument (or instruments) that are in mode 1, and which will ignore channel numbers, a channel number must still be included in each MIDI channel message. It does not matter which channel is used, but the convention is to use channel 1.

Note On/Off

For basic sequencing it is only necessary for the controlling device to be able to switch notes on and off, and to select the required notes. Switching notes on and off is handled using separate messages, not by having a single note on message that includes a note duration value. This second method is only usable in step-time sequencing, which renders it useless in many MIDI applications. Each note on message must always be followed by a note off message after the appropriate interval, or notes will be left "droning".

The note on and note off messages have the same basic format with each message being sent as three separate pieces of information. Most MIDI messages require more than one block of information, but as a three block message can be sent in less than one thousandth of a second there is not normally any problem with the data stream becoming overloaded. The first block of data contains the note on or note off code number, plus the MIDI channel number. The second block carries the note number, and MIDI supports a note range of 0 to 127. Each increment by one represents an increase in pitch by one semitone. A range of 128 semitones is a compass of well over ten octaves, which should be more than adequate. This is over three octaves more than the range covered by most pianos. In fact few MIDI equipped instruments actually accommodate the full note range. It is useful to bear in mind that many can handle a wider compass via their MIDI interface than they can using their keyboard. Incidentally, middle C is at a note value of 60.

The third block in the message carries the velocity value for the note, which with most instruments controls the volume of the note. Most instruments are velocity sensitive these days, but there are still some that are not, and many early MIDI instruments did not implement this feature. This data must always be present though, so as to maintain full compatibility between items of MIDI equipment. A non-touch sensitive instrument simply ignores any velocity information it receives, and transmits an intermediate "dummy" value in any note on messages it transmits.

Note off messages only differ from the note on type in that the message code in the first block is different. It might seem at first that not all the information provided in note off messages is actually needed, but remember that there could be sixteen polyphonic instruments connected to a MIDI output. A note off message must make it clear which note on and which channel of which instrument must be terminated. There is an alternative method of switching off notes, which is to use a note on type having a velocity value of 0. I am not quite sure why this alternative method was deemed necessary, but some instruments do seem to use it (the SCI synthesisers in my original MIDI system certainly seemed to use this method for all note off operations). MIDI equipment should be able to handle either method, and this is all of only academic importance for most users.

Key Pressure

At one time very few instruments responded to any form of key pressure (after-touch), but it now seems quite common for overall key pressure to be implemented. This is the most basic type of key pressure response, and it is a sort of average pressure value for however many keys are pressed. There are various ways in which this information can be used by an instrument, but it normally controls either the volume or the filtering after the initial attack and decay phases of the signal.

This type of message requires only two blocks of data to be sent, and the first of these is the overall pressure code number together with the MIDI channel number for the message. The second block of data is the pressure value, which is from 0 (minimum pressure) to 127 (maximum pressure).

Polyphonic key pressure is much the same as the overall type, but individual MIDI messages are sent for each note. This is much more sophisticated, and offers excellent control, but it is relatively difficult to implement. It is a feature that, as yet, is far from common. The polyphonic key pressure message takes the same basic form as the overall type, but a third block of data is needed to identify the note to which the pressure value applies. This block is placed between the code number and pressure value blocks.

An important point to keep in mind when using instruments that implement either form of key pressure is that holding down keys can result in a lot of MIDI data being generated. This applies more to the polyphonic type than overall key pressure. With polyphonic key pressure, if you are holding down around five keys at a time, five sets of key pressure data will be transmitted. Key pressure is not something that is sent once per note, and there can be several sets of pressure data for each note. The implications of this for real-time sequencing are clear – you could easily end up with most of the available memory being taken up with pressure data rather than note on/off messages! This might not matter with short sequences if a lot of free memory is available. Otherwise, it might be necessary to disable the instrument's aftertouch, or to use some method of filtering to remove the pressure data.

Key pressure is not something that is restricted to use with keyboard instruments. A lot of MIDI rack-mount modules will respond to aftertouch, and there is no reason that a step-time sequencer should not be designed to implement key pressure. However, if you have equipment that can handle this type of thing, the memory problem described above must be kept in mind.

Controls

MIDI includes a general purpose control message, which can be used to control master volume, filter resonance, or anything a manufacturer cares to implement. The only MIDI control number which the MIDI specification allocates to a specific function is control number 1, which is the modulation wheel. It seems to be the convention for control 4 to be a foot pedal, control 7 to be the main volume, and for control number 64 to be the sustain pedal. However, these are only conventions, and not all equipment necessarily conforms to them.

Controller messages are standard three block types, with the first block carrying the controller message code and the channel number. The next chunk of data is the number of the control, and the last one is its new value (from 0 to 127). Control numbers from 0 to 63 are used for continuous controls (i.e. adjustable types like volume and tone controls) whereas control numbers from 64 to 95 are used for switches and only give a simple on/off action. For these switch type controls, only values of 0 (off) and 127 (on) are valid, and other values will be ignored.

The continuous controls are complicated slightly by a system of pairing, which has control numbers 0 to 31 paired with controls 32 to 63 respectively. The idea is to have the values sent to a pair of controls merged to give one large number. This permits much more accurate settings than are possible using one control in isolation. Whereas one control has 128 different settings from 0 to 127, a pair of controls gives 16384 settings from 0 to 16383. In practice the degree of control provided by pairs of control values is usually higher than is needed. Also, where a control is being continuously varied, such fine resolution requires vast amounts of data to be sent. It is unlikely that MIDI could send data fast enough to fully utilize such high resolution. In practice it seems to be quite common for the "fine tuning" controller not to be implemented. It is then only control numbers from 0 to 31 that are used, while those from 32 to 63 are ignored. As a point of interest, in several instruments I have used not even the full 0 to 127 range has actually been used, with some controls only having 64 or 32 different settings.

Mode Change

Control numbers from 96 to 127 are either not assigned to any purpose, or are used for things such as mode changing. The functions of the control numbers that have been assigned a task are listed below:-

Control Number	*Function*
121	Reset all controllers
122	Local control on/off
123	All notes off
124	Omni mode off
125	Omni mode on
126	Mono mode on (poly mode off)
127	Poly mode on (mono mode off)

These are all switches where the value sent to them is either 0 (off) or 127 (on) or a dummy data number of 0 is used. The only exception is control number 126. Here the value sent specifies the number of voices to be used in mono mode (a value of 0 sets all available voices to mono mode).

Local control on/off is where the normal (built-in) method of controlling the instrument can be disabled. This usually means switching off the keyboard. One reason for doing this is merely to prevent accidental operation of an instrument while it is being sequenced from a computer. It also enables a keyboard instrument to effectively be used as a separate keyboard and sound generation module. The salient point here is that the keyboard will still transmit on the MIDI "OUT" socket, and the instrument will respond to data received on the MIDI "IN" socket. You can even feed the MIDI output through some form of processor (which could be the ST running a suitable program) and then feed the processed signal back into the instrument, as shown in Fig.3.2.

The all notes off message is not intended as the normal way of switching off notes. It seems to be intended more as a means of switching off any "droning" notes in the event of some form of malfunction. Incidentally, changes in MIDI mode also switch off any notes that are switched on at the time.

MIDI does not have specific messages to select mode 1, mode 2, etc., but instead the right combination of omni on/off, mono, and poly have to be selected. This should all be quite straightforward if you consider modes in terms of their current names rather than their numbers (e.g. omni on/poly instead of mode 1).

Program Change

The program change message uses two blocks of data. The first of these is the appropriate message code and channel number, while the second is the new program for that channel. In this context "program" generally

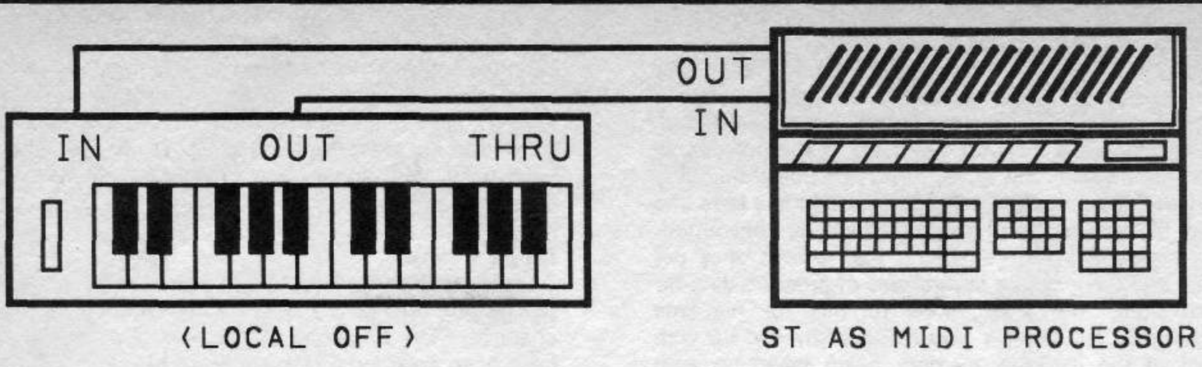


Fig. 3.2. The ST can be used as a versatile MIDI processor.

means a set of control settings for a synthesiser, so we are really talking in terms of a change in sound for the voice of an instrument. It is usual for synthesisers to have around 32 to 128 programs stored in memory, with any of these being assignable to any voice of the instrument. The ability to change sounds mid-sequence via MIDI is more than a little useful. If you have a synthesiser with eight voices and capable of multi-timbral operation in mode 4, on the face of it you only have eight different sounds available. If the sequencer and the synthesiser both support program changes, then you could have as many as 128 different sounds available. You would still be limited to no more than eight notes at once, but would have a vast range of sounds available for use in each piece of music. Not quite as good as having a bank of synthesisers, but nearly!

There is potential for a lot of confusion with program changes as there is no standard method of numbering. One manufacturer might use numbers from 0 to 127, while another might use numbers from 1 to 128. Some manufacturers have a totally different approach. For instance, my Casio CZ1 synthesiser has programs selected by two banks of push-buttons which are labelled "1" to "8" and "A" to "H". This gives sixty four programs from "A-0" to "H-8". The manual for an instrument should make it quite clear if there is a discrepancy between the numbers of programs, and the values used in program change messages to select them. Often there is a chart showing program change values and which program each one selects.

It is important to realise that there is no standardisation of program numbers and sounds. It is up to the user to ensure that a program change message will actually produce the desired sound from an instrument. It is also worth noting that few instruments have the full range of 128 programs available. Most seem to offer 64 or 100 programs, and will ignore any out of range program change messages.

It is worth noting that program change messages are not only recognised by synthesisers and other instruments. Devices such as MIDI controlled mixers and effects units often make use of them as well. The usual way in which this works is that sets of control adjustments are assigned to program numbers so that they can be called up as and when required using the appropriate program change instructions. program changes are an important part of much MIDI sequencing.

Pitch Bend

Pitch bending could be accomplished using an ordinary MIDI controller, but it has been assigned its own MIDI message type. This message consists of three blocks, with the first one containing the pitch bend message code and the channel number. The other two blocks of data contain the pitch bend value, with the two numbers having to be combined into one large pitch bend value at the receiving device. The MIDI specification does not lay down rules stipulating exactly how much given changes in pitch bend value actually affect the pitch of an instrument. Pitch bend information recorded from one instrument might not produce exactly the same degree of bend if it is played back to a different instrument.

System Messages

System messages consist of one or more blocks of data, and the first block always contains the system message code. As no channel numbers are used for these instructions, the vacant area left by the unused channel numbers can be used to define the precise nature of the message. System messages have a variety of functions, but they are mainly concerned with timing, and the synchronisation of sequencers. This almost invariably means keeping the built-in sequencer of a drum machine properly synchronised with the main sequencer which controls the rest of the system. There is another important category of system message in the form of "system exclusive" types, which seem to be playing an increasing prominent role in the world of MIDI.

Song Position Pointer

A MIDI sequencer which is capable of using MIDI synchronisation signals can keep track of the number of beats that have elapsed since the start of a sequence (or "song"). The maximum number of beats that can be handled is 16384, and each beat is equal to 1/16th note. The idea of this is to enable a sequencer to randomly access any part of a song. Perhaps more accurately, the idea is to enable two sequencers operating in tandem to be set to exactly the same point in a song, and any desired point in a song. Without this random access feature they could only be kept in synchronisation by always starting them both from the beginning of a sequence, or by using a lot of trial and error.

The song position pointer uses two blocks of data, but these two numbers are combined to give one large number of 0 to 16383. Note that some sequencers take a significant time to adjust to a song pointer instruction, and that they must be given time to respond to one of these messages before they are restarted. Also note that this message only moves the sequencers to a certain point in a song, it does not set the sequencers in motion.

Song Select/Tune Request

Although these messages have names that suggest a similar function, they are actually quite different. The song select message is used to select the desired sequence from a sequencer that can store more than one song, and which supports this feature. As with virtually every type of MIDI message, do not assume that your equipment actually implements this feature. Always check the MIDI specifications very carefully to find out what features are supported and which are ignored. This message uses one data block, and this is the song number. Song numbers are from 0 to 127 in terms of the actual number sent in a song select message. However, this is another example of the identification numbers used by manufacturers not necessarily being the same as the actual values used in the MIDI message, and not necessarily being the same from one manufacturer to another.

In the tune request message it is "tune" in the sense of tuning an instrument. This is for use with instruments which have an automatic tuning facility. The message only consists of the message code with no data being sent. No timing information is sent with this message, and all it is really doing is telling instruments to tune themselves against their internal tuning references. Presumably any instruments that implement this feature would then accurately tune themselves to the usual pitch of A = 440Hz, and would all be accurately in tune with each other. This is not a feature that seems to be much used these days, and few instruments seem to support this MIDI message.

System Exclusive

Most MIDI messages are universal, and can be implemented by any equipment manufacturer. This is an important aspect of MIDI, as one of the prime reasons for its introduction was, as far as possible, to eliminate incompatibility between devices from different manufacturers. On the other hand, MIDI needed to be flexible enough to permit future expansion and developments. Manufacturers needed to be able to do their "own thing", and implement any novel ideas that they might develop. Without this built-in flexibility it was unlikely that MIDI would have been adopted by all the main electronic music equipment producers.

Much of MIDI's flexibility lies in the system exclusive message. This consists of the message code followed by the manufacturers identification code. The idea of this code is that it enables equipment to filter out and ignore system exclusive messages that do not have the correct manufacturers identification number. This is an important feature, because the data that follows the identification code can be anything the equipment manufacturer desires. The data here will either be meaningless to the wrong piece of equipment, or worse still it could have totally the wrong effect and render a piece of equipment temporarily useless. The system is only usable with this method of filtering included. There is no fixed number of data blocks in a system exclusive message, and there can be as much data here as the application requires. The end of a system exclusive message is marked by a special (single block) MIDI message.

System exclusive messages are used for such things as program dumps, sample dumps, or any non-standard feature that an equipment producer wishes to implement via MIDI. A number of recent instruments seem to use system exclusive messages to provide MIDI control of their sound generator circuits, rather than making these adjustable via standard MIDI controller messages. This is perhaps an unfortunate trend, as units (or a computer like the ST plus some software) that are intended for general MIDI programming via controller messages are not usable with these system exclusive oriented instruments. They can only be programmed by way of a matching programmer unit, or custom software.

On the other hand, manufacturers who use system exclusive messages are supposed to publish details of the method of coding used, and to allow anyone to freely use this coding. Once system exclusive details have been published, no changes should be made to the specification (except perhaps, to extend it rather than modify any existing details). It is quite in order for a software company to produce programming software that accesses instruments via system exclusive messages, and a number of programs of this type are available for the ST computers.

There have been moves towards standardising some system exclusive messages. As far as I know, the only standard system exclusive message at present is the MIDI sample dump standard. This uses the sample dump standard identification code where the manufacturers code number would normally be, and it then has quite a complex method of sending samples. This complexity is inevitable, as this standard is designed to accommodate a wide range of instruments at different levels of sophistication. It is also designed to leave sufficient "headroom" for future developments. It has facilities for error checking, and has two way communications so that a receiving device can temporarily halt the flow of data if it is becoming overloaded. However, it is only fair to point out that not all samplers use the sample dump standard at the moment, and it might never be adopted as the only sample dump standard.

Incidentally, this type of system exclusive message is sometimes known by the rather contradictory name of "system exclusive common" message.

System Real-Time

The system real-time messages are the ones which provide synchronisation between two sequencers, and are analogous to the clock pulses used to synchronise drum machines in the pre-MIDI era (and probably still much used today). The MIDI system is substantially different to the old system though. In particular, the clock signal is not just a regular series of electronic pulses. It is a regular series of MIDI clock messages, and it is sent continuously, not just while a sequence is in progress. Because of this a number of other messages are needed in order to make the system workable. It should be pointed out here that in the original MIDI specification synchronisation was handled in a slightly different manner. This original method is now completely obsolete though, and is certainly not to be found on any currently produced equipment.

With a continuous clock signal it is obviously necessary to have stop and start messages so that sequences can be started and halted as required. Actually there are two types of start message, "start" and "continue". They differ in that a "start" message results in the sequence starting from the beginning, whereas continue causes it to start from wherever it left off (i.e. the current position of the song pointer). If a song pointer instruction is used to move to the middle of a sequence, it is "continue" and not "start" that should be used to restart the sequence.

The system real-time messages include a "reset" instruction, which simply takes the equipment back to its initial state (i.e. the state in which it would be if you were to switch it off and then turn it on again). This is not implemented on all instruments, and would not be worthwhile with many disc based instruments such as samplers, which can not produce any sound in their switch-on state (they must first have data loaded from disc).

Another little used facility is active sensing. The idea here is that the MIDI controller sends out an active sensing message at reasonably frequent intervals (not more than 0.3 seconds between each one), and the controlled devices then check that they are receiving these messages at suitable intervals. If a gap of more than 0.3 seconds should elapse without an active sensing message being received, all notes are terminated. This is quite a good idea as it avoids having an instrument stuck with notes activated or in some other "hung-up" state if a connecting cable becomes damaged, or something of this sort should occur.

This facility seems to be little used in practice though. It has the disadvantage of increasing the amount of MIDI data that is transmitted, which increases the risk of MIDI "choke". This would not seem to be a major problem though, as it only needs three to four messages per second to be transmitted. Perhaps equipment manufacturers feel that the processing power of their instruments and controllers could be put to better use in other MIDI departments, or perhaps they feel that this is an unnecessary complication that few people will want to bother with. Anyway, you are unlikely to encounter any equipment that actually implements this feature.

All these system real-time messages are single block types, and none of them contain any data. Being forms of system message, they do not carry a MIDI channel number either.

Finally

This is a full list of the standard MIDI messages, and if you are intending to use MIDI anything more than occasionally you really need to have at least a basic knowledge of what messages are used, and what features can be controlled via MIDI. It is well worth spending some time studying the basic details of all the messages. It is unlikely that you will ever use them all, but you really need to know what functions are available to you before you can use MIDI effectively. Do not assume that all of these messages will be implemented by your MIDI equipment. You must study the specification sheets for your equipment to determine which messages are actually implemented. MIDI specification sheets normally include a chart which lists MIDI message types, and indicates those that the equipment transmits and those which it recognises. There are often substantial differences between the types of data that are transmitted and recognised. If your instruments make use of system exclusive messages there should be details of these in the manuals, or this information should be available from the equipment manufacturers.

Even if you are only intending to use the ST with commercially produced applications software, and do not intend to take the do- it-yourself approach, a basic understanding of the MIDI messages is still very important. The same is true of MIDI as implemented on your particular equipment. Without this knowledge you could waste a great deal of time trying to implement a feature which your equipment does not support, or miss out on some useful feature which it can implement. Grasping the fundamentals of MIDI will take a certain amount of study, but it should prove to be well worth the effort.

Chapter 4
MIDI TECHNICALITIES

The previous two chapters have provided most of the important information MIDI users will need to know. In this chapter we will consider MIDI in a more technical manner, giving the kind of detailed information that is needed by MIDI programmers and hardware designers. For users who will only use ready-made equipment and software this is only of academic importance. You may well prefer to skip this chapter if you are not interested in the technicalities, but a quick read through this chapter might give you a better understanding of just what happens each time you generate some MIDI data. Another point worth making is that MIDI software does not need to be complex in order to be useful. Some very simple utilities can prove to be invaluable on occasions, and they are easily written by someone who has a reasonable grasp of MIDI codes and is moderately proficient at programming in any language relevant to the ST computers. A knowledge of the technical side of MIDI could be more useful than you might think, especially for an ST owner.

The Interface

MIDI is a form of serial interface, and it is very similar to the standard RS232C and RS423 computer serial interfaces. Computer interfaces send data in the form of binary numbers, with a "high" voltage (actually only about 3 to 5 volts in most circuits) representing a 1, and a low voltage (0 volts to about 2 volts) representing a 0. If you are not familiar with the binary numbering system, using numbers that consist of only 1s and 0s can seem to be decidedly useless! In fact any value that can be represented in ordinary decimal form can be represented by a binary number. Except for very small values it requires a lot more digits for the binary version though.

The binary system is very simple in concept, and is basically the same as the decimal system. However, its base number is 2 rather than 10. Whereas in a decimal number the columns of figures represent (working from right to left) units, tens, hundreds, thousands, etc., in a binary number they represent units, 2s, 4s, 8s, 16s, 32s, 64s, 128s, etc. A few years ago the majority of micro computers were 8 bit types (such as the Atari 400/800 series), and this meant they dealt in blocks of binary data 8 digits long. A "bit" is just a binary digit, and is a contraction of these two words. An 8 bit chunk of data is normally called a "byte". 8 bit bytes enable numbers from 0 (00000000 in binary) to 255 (11111111 in binary) to be accommodated. 8 bit computers can handle larger numbers, but only by using two or more bytes together, and processing each byte of a large number separately.

The ST computers have a 68000 microprocessor, which is a 16 bit type. In fact many of its internal registers are 32 bit types. This gives much greater processing power, but in many ways it is irrelevant for MIDI use. MIDI only deals with 8 bit bytes of data. In fact MIDI data (as opposed to instruction bytes) only use 7 bits of data, and are in the range 0 to 127). Some MIDI data requires 14 bit numbers, but these are sent over the MIDI link as two 7 bit numbers. MIDI can only handle data of more than 8 bits in this way.

Some interfaces send data in parallel form, which simply means that there is a separate lead to carry each bit of data, or eight signal leads plus a ninth one for the earth connection for an 8 bit system. The ST's printer port is an example of a serial interface. MIDI is a form of serial interface, and the signal is sent down a single wire plus an earth or return lead. The signal must be literally sent bit-by-bit and not one byte at a time. This tends to be much slower than the byte-by-byte transfers of a parallel interface, but it is more practical in that the need for expensive multi-way cables is avoided. Also, parallel interfaces tend to have rather limited maximum operating ranges (just two metres for a Centronics style parallel printer interface for example).

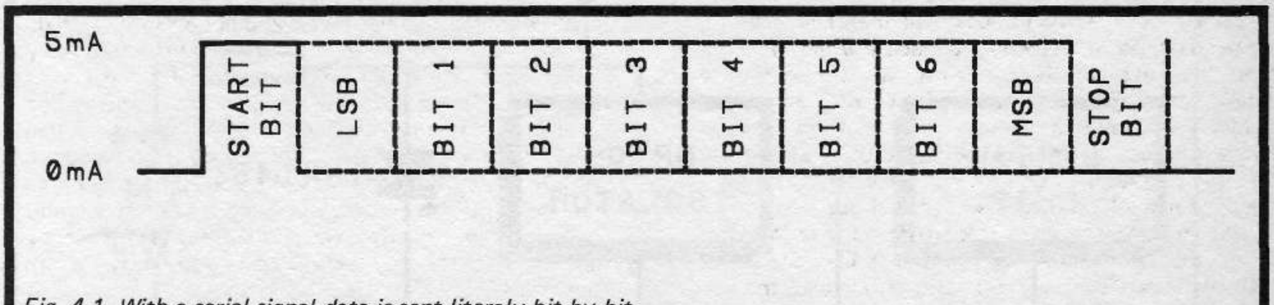


Fig. 4.1. With a serial signal data is sent literaly bit-by-bit.

The standard method of operation for a serial interface is as shown in Fig.4.1. First a "start" bit is sent, and this indicates to the receiving device that a byte of data is about to commence. The receiving equipment then samples the state of the input line at regular intervals. The transmitting device outputs the bits of the byte that are being transmitted, one by one, also at regular intervals. It starts with the least significant bit (the "l.s.b." or units digit) and works in sequence through to the most significant bit ("m.s.b." or the 128s bit in this case). There is then a "stop" bit, which is really does no more than ensure that there is a reasonable gap between one byte and the next. It does not carry any data and it is not needed for synchronisation purposes.

The RS232C serial system has various word formats, with from 5 to 8 data bits, one or two stop bits, and sometimes a form of error detection known as parity checking is implemented. This system of error detection involves the transmission of extra bits on some bytes. Fortunately, MIDI is properly standardised, and it only uses a word format of one start bit, eight data bits, one stop bit, and no parity. This word format can be accommodated by all the serial interface chips I have encountered, and MIDI hardware does not require any non-standard components.

A lot of difficulties are experienced by users of RS232C equipment due to problems with the handshake lines. These enable a receiving device to instruct the sending equipment to temporarily halt the flow of data in the event that data is received at a higher rate than it can be processed. There is no risk of any similar problems with MIDI interfacing as handshaking is not used. At least, handshaking of the hardware variety is not used. Some equipment uses system exclusive messages where a two-way dialogue takes place so that the flow of data can be regulated, and any errors can be corrected. This system can work very well, and the lack of hardware handshaking is not a major drawback.

Some serial system are "synchronous", which means that they use an extra connecting cable to carry some form of synchronisation signal. MIDI is a form of

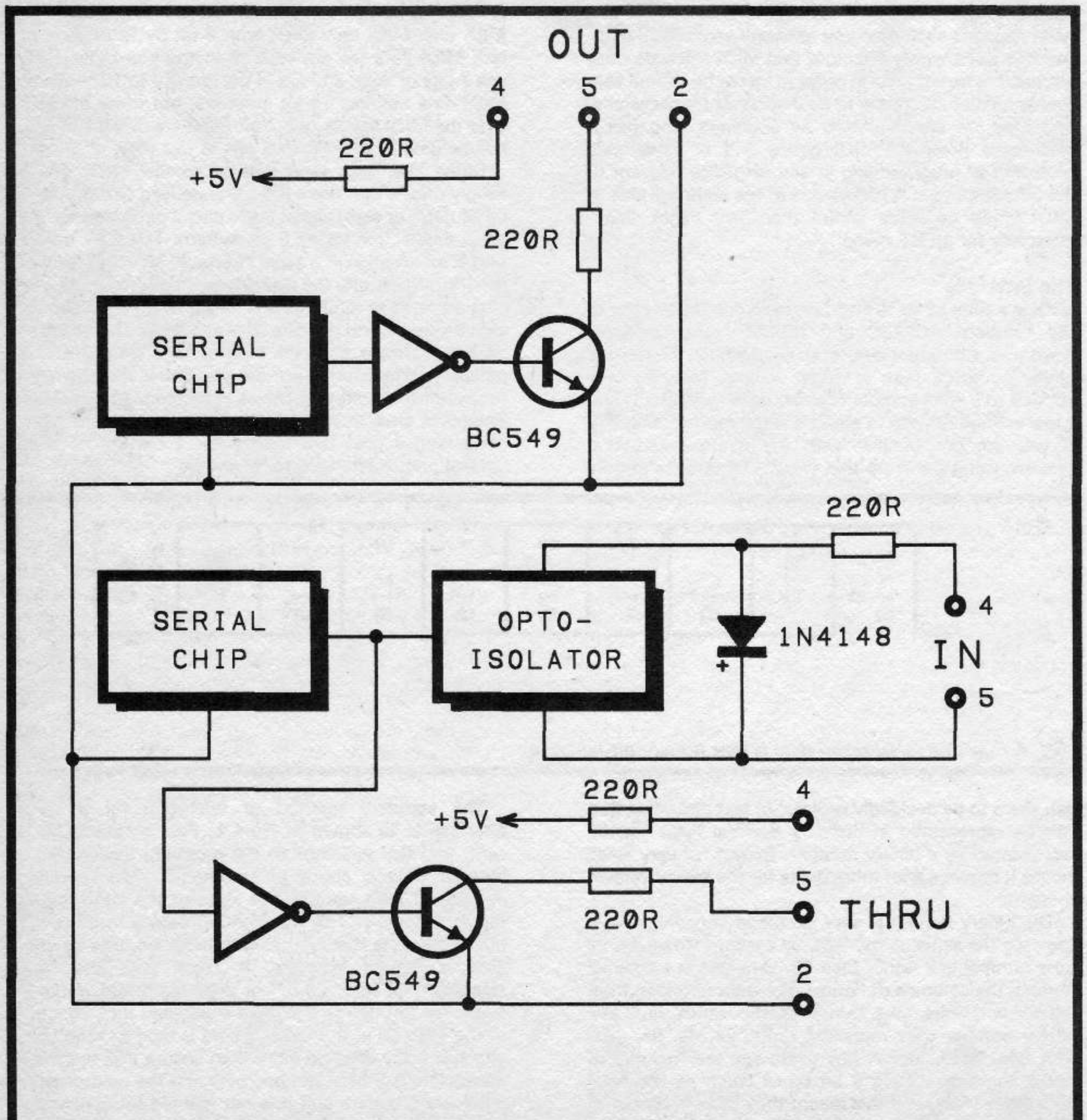


Fig. 4.2. Basic hardware for a MIDI interface.

"asynchronous" serial interface, which means that the timing signals are sent on the same line as the data. In fact the only synchronisation signal is the start bit at the beginning of each byte. This indicates the commencement of a byte of data, and that the voltage on the connecting lead must be tested at regular intervals thereafter until a full byte of data has been received. It does not ensure that the transmitting and sending devices are properly synchronised while each byte of data is sent. This is achieved by sending/receiving data at a standard rate, with (usually) quartz crystal controlled oscillators (as used in quartz watches) to ensure excellent accuracy at both ends of the link.

The standard MIDI "baud" rate is 31250 baud, or 31.25 kilobaud if you prefer. This simply means that data is transmitted at a rate of 31250 bits per second (assuming a continuous flow of data). This is not a standard RS232C baud rate, and might seem to be a unusual choice. Originally the baud rate was 19200 baud, which is the highest standard baud rate for RS232C interfaces. However, this was deemed to be too slow, and in the final MIDI specification it was increased to 31250 baud. This is convenient from the hardware point of view, as it is well within the capabilities of most serial interface chips. Also, 31250 multiplied by 32 equals 1000000, and this fact enables the baud rate of MIDI interfaces to be controlled using "off the shelf" crystals intended for communications applications and microprocessor circuits.

The Hardware

RS232C and RS423 interfaces use different voltages to represent logic 0 and logic 1 levels, but MIDI is different in that it uses a 5 milliamp current loop. In other words, the current is switched on to indicate one logic level, and switched off to represent the other logic state. This is done due to the use of opto-isolators at each input, which keep items of equipment in the system electrically isolated from one another (as explained in chapter 2).

A MIDI link is therefore something along the lines shown in Fig.4.2. This is mostly straightforward, but note that the opto-isolator can not be a "bog-standard" device such as the TIL111. It needs to be a high speed type, and I have found types having a photo-transistor driving a common emitter switch (such as the 6N139) give the most reliable results. Alternatively, it is possible to get away with using a relatively slow and insensitive device such as the TIL111. It must be connected in the same configuration used in the 6N139 etc., but with an external switching transistor. This seems to give good results, and is usually much cheaper than using a high quality opto-isolator such as the 6N138 or 6N139.

MIDI Codes

All MIDI instructions have a header byte that consists of two 4 bit sections (or "nibbles" as they are sometimes called). The most significant nibble indicates the nature of the instruction (note on, note off, or whatever). The least significant nibble is the channel number in most messages, but no channel number is required for any form of system message. With system messages the most significant nibble is the system message code, and the least significant nibble defines the precise type of system message (MIDI clock, reset, etc.). In terms of the total decimal value in a header byte, it is just a matter of taking the values of the two nibbles and adding them together. For instance, an instruction nibble of 128 and a channel value of 12 would be sent as a byte having a total value of 140. With MIDI it is often easier to work with hexadecimal numbers, as each nibble represents one digit of a hexadecimal number.

The most significant bit of header bytes is always set to 1, but this bit of data bytes is always 0. It is for this reason that MIDI data bytes only cover a 0 to 127 range, and not the full 0 to 255 span afforded by 8 bit operation. The point of arranging things this way is that it enables receiving equipment to sort out MIDI messages from amongst MIDI data. Although this might appear to be unnecessary with one MIDI message being fully transmitted before the next one is commenced, things do not always happen in this way. It is obviously necessary for MIDI clock messages to be sent at strictly regular intervals, without them being delayed too long while a message in progress is completed. The MIDI specification therefore allows for clock messages to be mixed into other messages. Complete bytes must always be sent, and a byte must not be aborted so that a clock message can be sent. It is still possible to have something like a note on message and the note value sent, followed by a MIDI clock message, and then the velocity data byte of the note on message! As the most significant bit of the clock message will be set to 1, the receiving equipment can recognise it as such and will not mistake it for the velocity data byte.

Note On/Off

The note on nibble is 1001 in binary, which is equivalent to 144 in decimal. From here onwards, values will be provided in binary, followed by the decimal equivalent shown in brackets. The least significant nibble is the channel number, which is from 0000 (0) to 1111 (15). As MIDI channels are normally numbered from 1 to 16, this means that the value used in a MIDI channel message to select the desired channel is actually one less than the MIDI channel number. In other words a value of 0 selects channel 1, a value of 1 selects channel 2, and so on. The note on message is followed by two data bytes, which are the note number and the velocity value.

Note off messages have 1000 (128) as the most significant nibble, and the channel number as the least significant nibble. The header byte is followed by two data bytes, which are again the note number and velocity value. A note on message having a velocity value of 0 can be used as an alternative form of note off message.

Key Pressure

Overall key pressure (sometimes called "channel" pressure) has the instruction nibble 1101 (208) and is followed by a single data byte. Polyphonic key pressure has 1010 (160) as the instruction nibble, and is followed by two data bytes. These are the note value first, and the pressure value second. For both types of message the least significant nibble of the header byte contains the channel number.

Control Change Etc.

The control change header byte has 1011 (176) as the most significant nibble in the header byte, while the least significant nibble is the channel number value. The header is followed by two data bytes, which are the control number followed by its new value. Controls from 0 to 31 are paired with controls from 32 to 63 (respectively), and these operate as high resolution continuous controls. Each pair of seven bit numbers are combined to give a single 14 bit value. The lower numbered controller always provides the most significant bits, with the higher numbered control providing the seven least significant bits. In terms of decimal numbers, the range available is from 0 to 16383. Note that it is quite acceptable to only change one or other of the controls in a pair, and a change to one does not necessitate a change to the other.

Not all equipment actually uses the high resolution capability of the MIDI continuous controls, and most equipment only uses a resolution of seven bits or less. For 7 bit resolution it is the most significant nibble (lower control number) that is utilised, and the least significant one that is ignored. For less than seven bit resolution the least significant bit or bits are left at zero, while the most significant bits are utilized.

Control numbers from 64 to 95 are used for switch type controls. Only control values of 0 (off) and 127 (on) are valid with these, and other control values will be ignored. Control numbers from 96 to 120 are, as yet, unassigned. These are available for future expansion, and may be assigned specific functions in the future.

The remaining control numbers (121 to 127) are used for mode changes and similar functions. These have a value of 0 for the control value byte, apart from controls 122 (local on/off) and 126 (mono on). Local control is a standard on/off switch type control, and is 127 to activate the keyboard (or whatever), and 0 to switch it off. When mono mode is switched on, the control value selects the number of voices to be set to mono mode (a value of 0 sets all the instrument's voices to mono mode). The MIDI specification only calls for mono mode channels to be contiguous, but some instruments have special modes which allow them to be assigned to any desired channels.

Pitch Wheel

The pitch wheel header byte has 1110 (224) as its most significant nibble, and the channel number value as the least significant nibble. Two data bytes are used, and the two seven bit values these contain are combined to give a 14 bit pitch wheel value. The least significant byte is the one sent first. A value of 10000000000000 (8192) represents zero pitch change. If less than the full 14 bit resolution is implemented some of the least significant bits are ignored by a receiving device, and always set at zero by a transmitting device.

Program Change

The program change code nibble is 1100 (192). The least significant nibble of the header byte is the channel number value. The header is followed by a single data byte, which is the number of the new program for that channel. The value in the data byte is from 0 to 127, but some manufacturers number programs differently. Where this is the case, equipment manuals often have a conversion chart to make things easier.

Table 1 provides a summary of the channel messages for quick reference purposes. The channel mode messages require some further amplification, and this is provided in Table 2.

System Messages

These all have 1111 (240) as the most significant nibble in the header byte. No channel numbers are used, as these messages are sent to the whole system. This leaves the least significant nibble free to indicate the type of system message. Table 3 gives a full list of these messages, but note that some of the sixteen available

Table 1

Header	*Function Data*	
1000 (128)	Note Off	Note Value/Velocity Value
1001 (144)	Note On	Note Value/Velocity Value
1010 (160)	Poly Key Pressure	Note Value/Pressure Value
1011 (176)	Control Change	Control Number/Value
1100 (192)	Program Change	New Program Number
1101 (208)	Overall Pressure Pressure Value	
1110 (224)	Pitch Wheel	l.s.b./m.s.b.

Table 2

Control No.	*Function*	*Data*
121	Reset All Controls	0
122	Local Control	0 = off, 127 = on
123	All Notes Off	0
124	Omni Mode Off	0
125	Omni Mode On	0
126	Mono Mode On	Number of Channels (0 = All Channels Set to Mono Mode)
127	Poly Mode On	0

Table 3

Nibble Code	*Function*	*Data*
0000 (0)	System Exclusive	ID/As Required
0001 (1)	Undefined	
0010 (2)	Song Position Pointer	l.s.b./m.s.b.
0011 (3)	Song Select	Song Number
0100 (4)	Undefined	
0101 (5)	Undefined	
0110 (6)	Tune Request	None
0111 (7)	End System Exclusive	None
1000 (8)	Clock Signal	None
1001 (9)	Undefined	
1010 (10)	Start	None
1011 (11)	Continue	None
1100 (12)	Stop	None
1101 (13)	Undefined	
1110 (14)	Active Sensing	None
1111 (15)	System Reset	None

codes are as yet undefined. Many of them do not require data bytes, and are just single byte messages.

The values shown in brackets are the decimal equivalents for the binary nibbles. These must be boosted by 240 to give the total decimal value for each header byte (e.g. the value sent for a clock signal is 240 + 8 = 248). The system exclusive message is followed by a data byte which gives the manufacturer's identification code, and then as many data bytes as required follow on from this. The "end system exclusive" message marks the end of a system exclusive message. Table 4 provides a list of manufacturer's identification numbers. The sample dump standard is a "system exclusive common" message, which can be used by any MIDI equipment producer.

This is only a brief description of the MIDI codes, but more background information on many of them is provided in chapter 3, which should be consulted if any points here are unclear.

Table 4

Manufacturer	*Number (decimal)*
SC1	1
Big Briar	2
Octave	3
Moog	4
Passport Designs	5
Lexicon	6
Ensoniq	15
Oberheim	16
Bon Tempi	32
SIEL	33
Kawai	64
Roland	65
Korg	66
Yamaha	67
Casio	68
Sample Dump Standard	126

Chapter 5

APPLICATIONS PROGRAMS

The ST has what is almost certainly the biggest range of available music oriented software. There are other computers which have a substantial music software base, but their music software lists mostly seem to be static or shrinking. By contrast, the range of music programs for the ST computers seems to be continually growing. If a piece of software for a certain music application is not available for the STs, then it is probably not available at all!

There are dozens of ST music programs available, but inevitably some programs cover much the same ground as one or more others. There are only a few different categories of music software, but programs within each category can vary considerably in terms of price, facilities, and sophistication. In general you get what you pay for, but some titles do seem to offer much better value than others. Also, there is little point in buying an expensive program that has advanced facilities you do not need, when a much more simple program at less than half the price would be just as good for your purposes. When selecting software you need to carefully study all the possibilities, obtain demonstrations of likely candidates if you can, and give a lot of thought to which one best suits your needs. While there may seem to be no point in selecting a program that goes well beyond your current needs, it would be as well to leave some room for future expansion. This applies particularly to sequencing software, where you may quickly progress to the point where you are producing pieces of far greater complexity than you might have expected before starting any sequencing work.

Sequencing

Of the multifarious types of music software that are available, I suppose that the various sequencing packages probably form the largest category. There are two main types of sequencer; the "real-time" and "step-time" varieties. With a real-time sequencer, in its most basic form, anything you play on the keyboard is stored in the computer's memory, together with timing information. This information can then be played back into the instrument. It is a sort of space-age version of the old player- piano idea. As with player-pianos, just how accurate (or otherwise) the played back music happens to be when compared with the original performance depends on the sophistication of the system. Most sequencers, unless told otherwise, record all the MIDI data that they receive. If you are using an instrument that fully implements velocity, polyphonic aftertouch, etc., then the sequence played by the computer should be a very accurate reproduction of the original performance. This assumes that the sequencer operates with a fairly high degree of timing resolution, which is not always the case. If low timing resolution is all that the sequencer can offer, the played back sequences might sound rather "mechanical" and lacking in artistry.

A basic step-time sequencer enables notes to be programmed by entering data into the computer via the computer's keyboard or mouse. This tends to be a relatively slow and tedious way of entering sequences, but it does have its advantages. The most obvious one is that you do not need to be particularly talented at playing a keyboard, or any instrument for that matter. Virtually anyone can program any sequence, regardless of how complex it might be. The computer will always play it note-perfectly. How quickly step-time sequences can be entered into the computer is largely dependent on the sophistication of the program. Entering data only in numeric form can be very slow, confusing, and error-prone. Some form of graphics display can make it very much easier and quicker to enter sequences.

Why ST Sequencing?

Obviously the ST (or some other computer) is not the only way of entering the world of MIDI sequencing. Many of the more expensive electronic instruments have built-in sequencers, as do some in the low to middle price range. These include step-time and real-time sequencers of various levels of sophistication, as well as some dual role types which permit both types of sequencing. Also, some neat stand-alone sequencer units are available, and these offer quite respectable levels of performance and ranges of facilities.

The only real advantage of built-in or stand-alone units are that they are much more portable than a system based on an ST. This could be of great importance, but synthesisers with five octave keyboards and many other popular instruments of today are quite sizeable and heavy objects. An ST plus monitor will not necessarily make that much difference to the overall bulk of the system. Probably the main drawback of the ST for "on the road" work is that the monitor is relatively vulnerable to damage, and would need to be treated with due respect.

It is the monitor that gives much of the ST's advantage when compared to built-in and stand-alone units. The latter generally have simple liquid crystal displays (l.c.d.) or even just simple light emitting diode (l.e.d.) types. The displays on electronic music equipment have improved greatly over recent years, and some do now sport l.c.d. types that are capable of producing a few lines of text or simple graphics. However, even the low resolution colour display of the ST can show much more information than most of these, and the medium and high resolution displays are even better. The larger display of a 12 or 14 inch monitor is also much easier to see, which can be crucial if you will be looking at the display for long periods of time. Being able to display a reasonable amount of data at a time is important for much sequencing work. With only small amounts of data on view you can easily keep losing track of where you are, making entering notes or editing sequences very difficult.

Another advantage of using the ST is that even the more simple sequencer software for this computer seems to be packed with features in comparison to most built-in and stand-alone units. It is difficult to find anything comparable to the up-market ST sequencer packages, apart from some software packages for one or two other 16 bit computers.

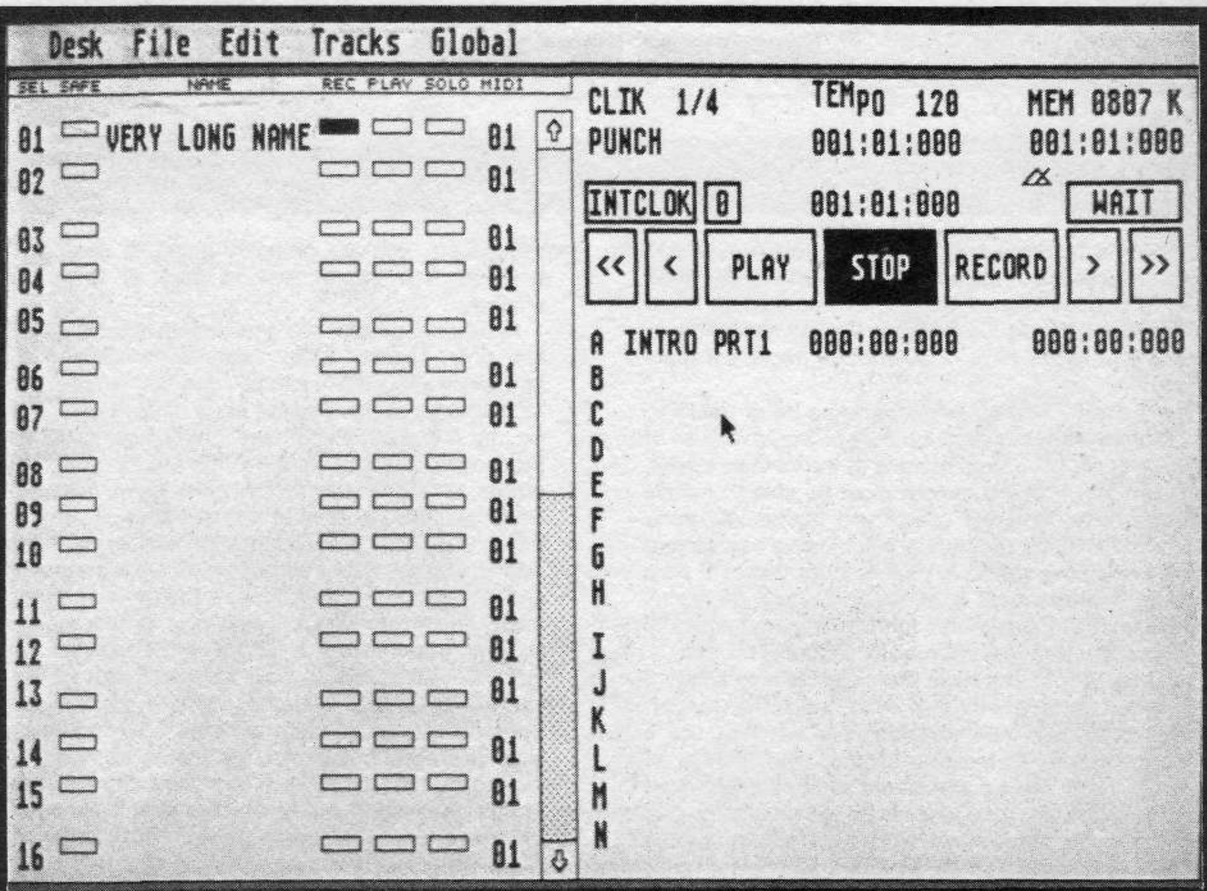
In the value for money stakes a built-in sequencer is likely to be the cheapest solution. It is also likely to provide the least features, but if a built-in type is adequate for your purposes there is clearly no point in paying out money for an ST based system or a stand-alone unit. An ST based system could well prove to be the most expensive solution, but it is also likely to be the one that provides the most facilities. If you need a powerful sequencer there is probably no better solution than an ST plus one of the more sophisticated sequencer programs. A point to keep in mind when comparing costs is to remember that a sophisticated sequencer system based on an ST computer is not restricted to sequencing applications. With some additional software it can be used for games, word processing, computer art, and a hundred and one other things that computers are used for every day. These alternative applications include other music types, including such things as voice filers/editors and visual sample editors. A stand-alone sequencer is just that, and nothing more.

Real-Time Sequencing

As explained previously, a real-time sequencer merely records MIDI data received on the MIDI input, and plays it back on demand, from the MIDI output. All the ST sequencer programs I have encountered, including very low cost types, seem to go well beyond this most basic level of operation. The most obvious addition is some form of speed control, so that the tempo during playback can be varied. You can then play a difficult piece slowly, and then play it back at the correct speed. Unlike trying the same thing with tape recording, changes in tempo when using MIDI do not produce any changes in pitch.

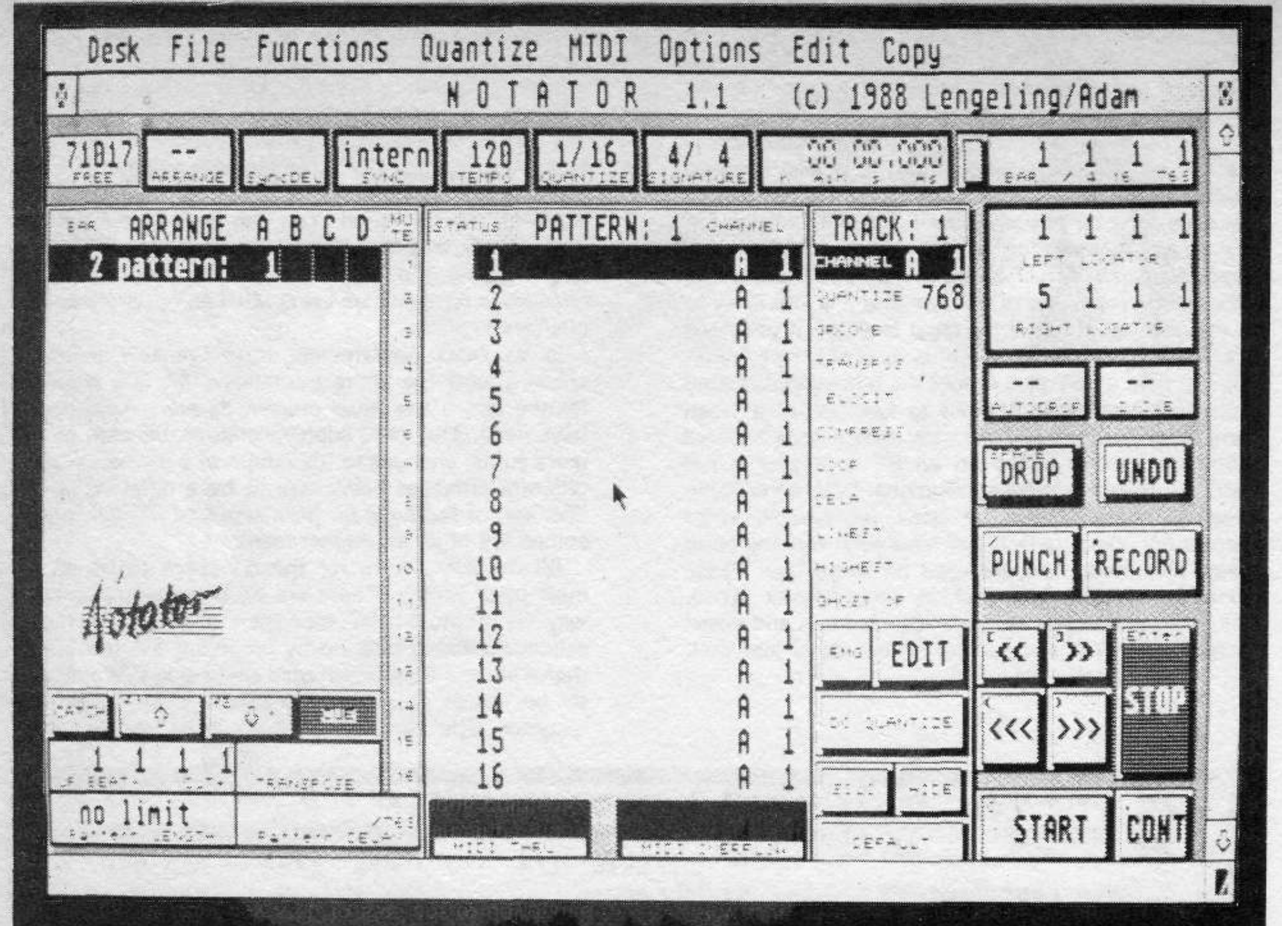
In its most fundamental form, variable playback speed affects the entire sequence. This is a standard feature which has been present in every sequencer I have used. The more sophisticated sequencers permit more subtle changes to the tempo of a sequence, with different passages being able to have different tempi. This sort of facility is far from standard though, and is something of an up-market feature.

All the sequencers for the ST seem to be of the multi-track variety. These are used in much the same way as a multi-track tape recorder, with complex sequences being built up by recording the first track, then a second track is recorded along-side that one, and so on. Most real-time sequencers even have tape recorder style controls, such as play, record, stop,


Desk File Edit Tracks Global
SEL SAFE NAME REC PLAY SOLO MIDI
INTCLOK
WAIT
<<
<
PLAY
STOP
RECORD
>
>>

```
01   VERY LONG NAME   01      CLIK 1/4      TEMPO 128        MEM 0807 K
02                    01      PUNCH         001:01:000       001:01:000
03                    01      INTCLOK 0     001:01:000       WAIT
04                    01
05                    01      A INTRO PRT1  000:00:000       000:00:000
06                    01      B
07                    01      C
08                    01      D
09                    01      E
10                    01      F
11                    01      G
12                    01      H
13                    01      I
14                    01      J
15                    01      K
16                    01      L
                              M
                              N
```

The "Shareware" 32 track sequencer start-up screen. It has the usual tape deck style controls (top right).



The "Notator" start-up screen. It too has the usual tape deck style controls (bottom right hand corner).

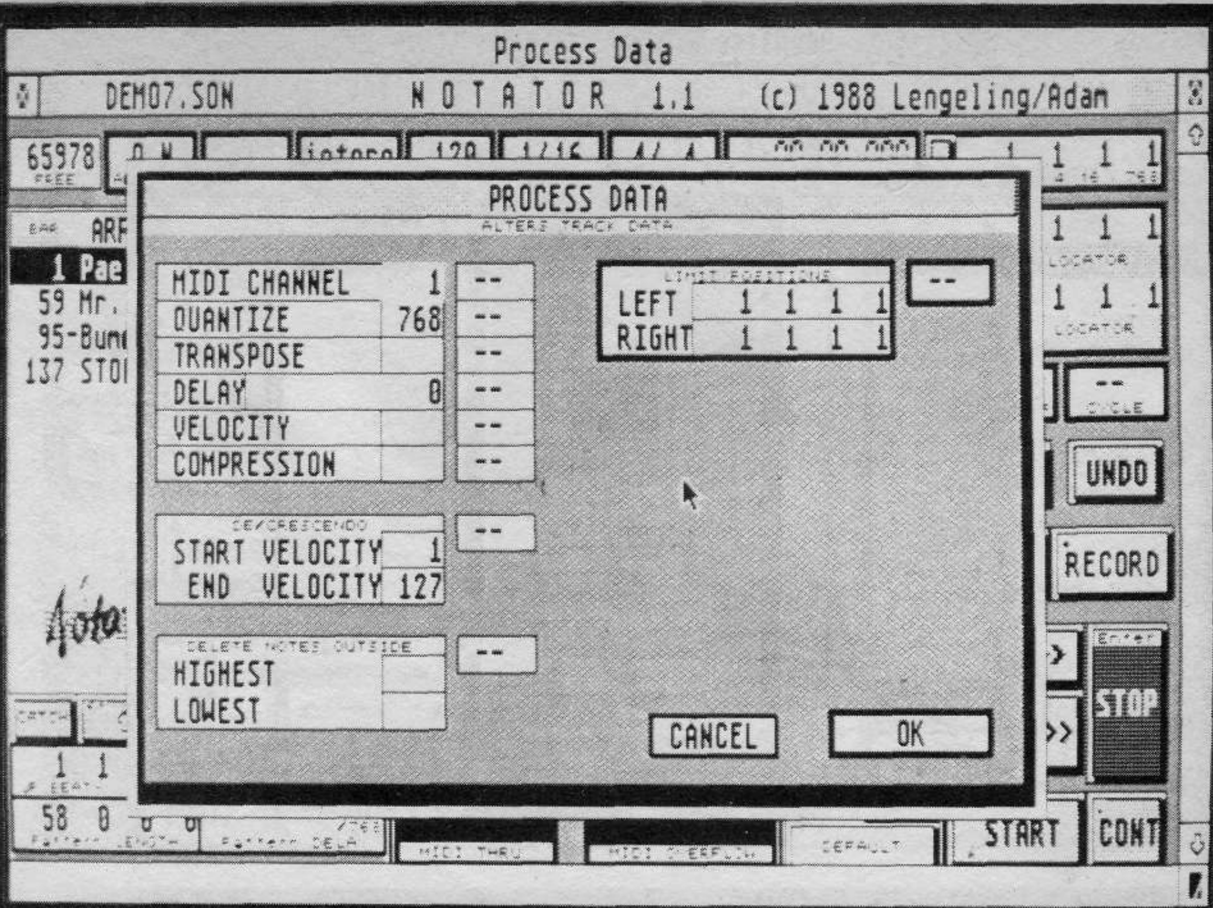
rewind, and fast-forward. This is not just a gimmick, and it helps to make them easy to use, especially when you are first getting to grips with sequencing. Most sequencers offer at least eight tracks, and sixteen, twenty four, thirty two, or even more tracks are quite ordinary.

There can be advantages in having a lot of tracks to play around with, but vast numbers of tracks may be of no value at all. For a large number of tracks to be useful, both you and your equipment must be able to handle them properly. Whether or not you can handle thirty two or more tracks properly is a subjective assessment and is something you have to decide for yourself. As a point of interest, most large scale classical orchestral works (such as Beethoven's Fifth Symphony) could be accommodated by a sixteen track sequencer.

Whether or not you have the equipment to justify a large number of tracks is a more objective matter. Unlike multi-track tape recording you can not use a single instrument to build-up an unlimited number of tracks. To play back a completed sequence you must have an instrument, or a voice of an instrument, to play each track. For polyphonic tracks the instrument must be capable of providing an adequate number of notes simultaneously. With a single synthesiser of reasonable sophistication you would probably be able to handle eight monophonic tracks. If you wish to have (say) twenty tracks, with some of these playing polyphonic sequencers, you are probably going to need at least three or four instruments to play all those tracks properly.

Another important point to keep in mind is that there are only sixteen MIDI channels available. Some up-market sequencers can be used with special add-on MIDI interfaces that provide extra MIDI "OUT" sockets for the ST, and enable (say) thirty two tracks to be sequenced on the basis of one per MIDI channel. By no means all sequencers which have more than sixteen tracks support a feature of this type though. With more tracks than MIDI channels, tracks must share channels. With one track only using notes above a certain pitch, and another only using notes below that threshold pitch, you might genuinely be able to use more than sixteen tracks with a single MIDI output. Some instruments, including most samplers, can be set for "keyboard split" operation, where each voice is assigned to a certain range of notes. This feature can be used to match tracks of a sequencer to voices of an instrument (which can each produce a different sound). In fact this system can be used to effectively put three or more instruments/tracks on each MIDI channel, but the more voices you use per MIDI channel, the more restricted the compass of each one must become.

Some sequencers have (say) sixteen tracks, but enable tracks to be merged. Thus, if you run out of tracks, you may be able to merge some together to clear


Process Data

DEMO7.SON NOTATOR 1.1 (c) 1988 Lengeling/Adam

PROCESS DATA

MIDI CHANNEL 1

QUANTIZE 768

TRANSPOSE

DELAY 0

VELOCITY

COMPRESSION

START VELOCITY 1

END VELOCITY 127

HIGHEST

LOWEST

LEFT 1 1 1 1

RIGHT 1 1 1 1

One of "Notator's" pop-up control screens. Using an add-on interface up to 64 tracks on separate MIDI channels can be handled.

tracks for further recording. If only a single MIDI output and sixteen channels are available, this would seem to be as good as having twenty four or thirty two tracks. Two tracks merged into one and operating on a single channel is no different to two separate tracks sending on the same channel. Either way the sixteen MIDI channels will be the factor that limits the possibilities of the system. A possible advantage of having the extra channels is that it could be easier this way if you wish to make drastic alterations to a piece. Once you have merged two tracks it is not usually possible to un- merge them again! There is actually a way around this, which is to save the sequence to disc prior to making any drastic (and non-reversible) alterations. If your editing does go completely wrong, you can then return to the original version and try again.

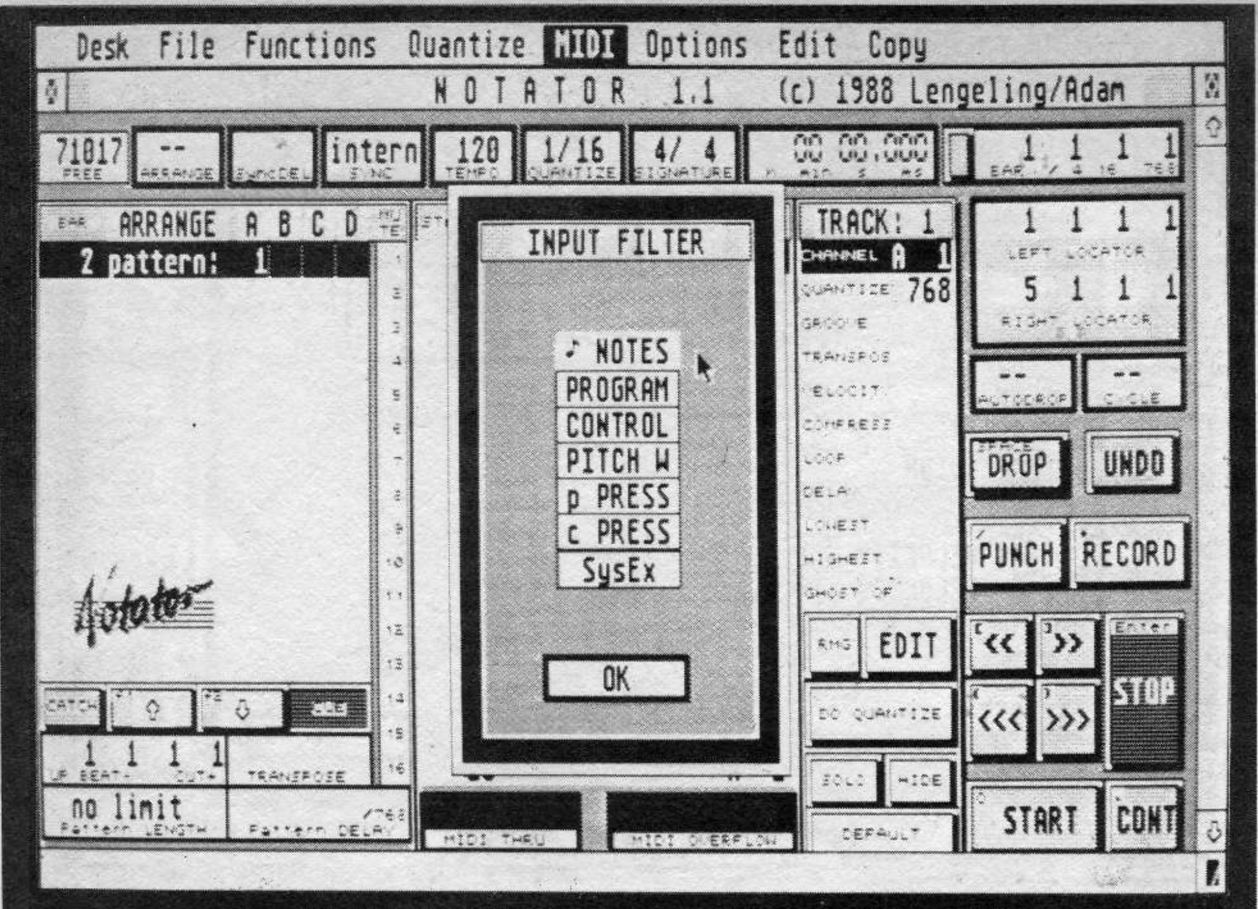
Another way of squeezing more on to each track is to have a facility that enables a fresh track to be recorded onto and merged with an existing track. This is a slightly risky way of doing things in that if a serious mistake is made and you want to remove the newly added material, there may be no means of deleting it without also deleting the original material on that track. Recording new material onto a separate track and then merging it with the existing track once you are satisfied with it is generally a much safer method.

A useful feature that seems to be gaining popularity is one which enables a short sequence to be repeated throughout a score. The idea behind this is to have a drum machine controlled directly from the main sequencer, rather than being controlled by its internal sequencer and synchronised to the main sequencer. This avoids getting the wrong drum sequence and is generally a more satisfactory way of doing things.

Filtering

Most sequencers have some form of user selectable filtering of MIDI data. Filtering can operate during record, playback, or both. Something like aftertouch filtering would normally be used during record as a means of conserving memory. With this type of data filtering there may be no point in filtering it once it has been recorded. The exception would be when undertaking multi- track sequencing to build up a complex piece of music. Each track, as it is recorded, might be totally free from MIDI "choke". However, when all the tracks are replayed together it could be a different story. There could be quite severe problems with MIDI choke, and filtering out non-essential data at playback could totally eliminate the problem. Some up-market sequencers have quite sophisticated control routines that will automatically remove the least important data if MIDI "choke" starts to occur. This avoids problems such as notes left droning.

A facility that is not, strictly speaking, a form of



One of the "Notator's" MIDI set-up screens. This one controls input filtering (there is a similar facility for output filtering).

filtering, but is often included in the filter facility of sequencers, is the MIDI clock on/off switching. This either enables MIDI clock messages (plus start, stop, continue, etc.) to be transmitted, or turned off. Unless you are using a drum machine or other device that requires these signals it is best to switch off the MIDI clock messages. This helps to take the load off the computer and reduces the risk of MIDI "choke".

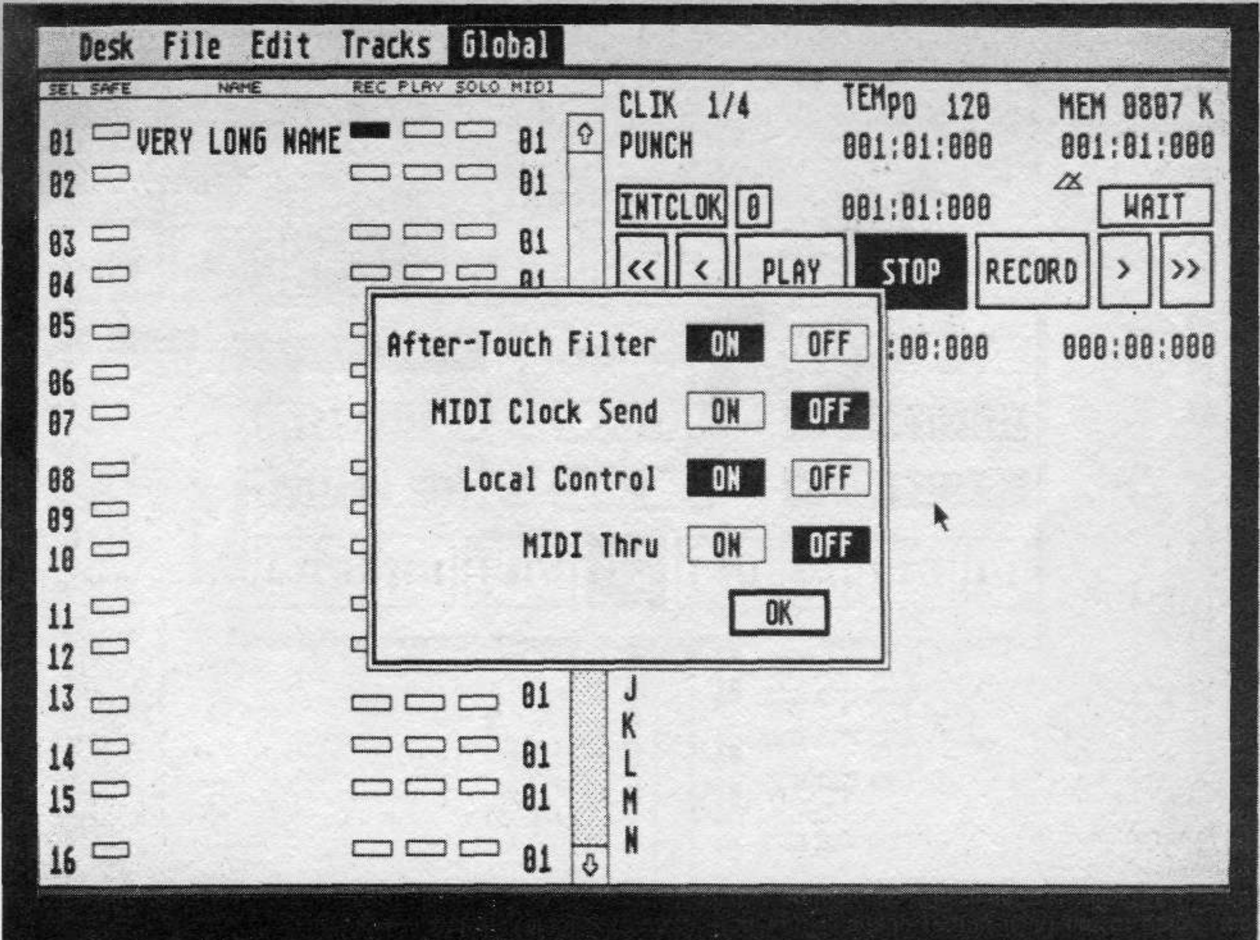
Other features often found in the MIDI filter or MIDI control part of sequencer programs are local on/off and MIDI THRU controls. Local on and local off simply send the corresponding MIDI message. This is useful because it represents a quick and easy way of controlling this facility. Also, some instruments do not seem to have any built-in control sequence that permits local on/off switching. You can then only access this feature via your sequencer. MIDI THRU has the effect of transmitting on the MIDI output any data received on the MIDI input. This enables a MIDI keyboard connected to the MIDI input of the ST to perform a dual role. It can be used for playing sequences into the sequencer, or it can be used to play an instrument connected to the MIDI output of the ST, or both at once. This facility is often used in conjunction with local off capability if the MIDI keyboard is actually the keyboard of a sequencer or other instrument. There is otherwise a slight risk of the instrument responding to both the notes received via its keyboard and those received via the ST. However, most instruments will ignore the any notes via their MIDI input if they are already playing the same notes due to keyboard information. Things can go wrong though, especially if you start using "special" MIDI modes.

Quantisation

This is a term that seems to keep cropping up in different contexts, and with slightly different meanings. In a sequencer context it normally refers to some form of note duration correction facility. In other words, if you play with rather ragged timing, the sequencer's quantisation facility can tidy things up for you.

Ideally the sequencer should record sequences with high resolution timing so that it can give an accurate rendering of your playing. Remember that most music is played with considerably less than mathematically perfect timing, not because the player is not very good, but as a means of giving expression and feeling to the music. A heavily quantised piece may be perfectly timed from the mathematical stand-point, but it may also be totally lacking in expression.

Again, ideally, quantisation should be implemented during playback, and not on the data stored in the computer's memory. The important difference here is that quantisation during playback can be changed or removed altogether if you change your mind. It is taking



```
Desk  File  Edit  Tracks  Global
SEL SAFE       NAME        REC PLAY SOLO MIDI
01  VERY LONG NAME                       01      CLIK 1/4      TEMPO 120      MEM 0807 K
02                                       01      PUNCH         001:01:000     001:01:000
03                                       01      INTCLOK  0    001:01:000     WAIT
04                                       01      <<  <  PLAY  STOP  RECORD  >  >>
05                                                             :00:000        000:00:000
06
07
08
09
10
11
12
13                                       01      J
14                                       01      K
15                                       01      L
16                                       01      M
                                                 N

After-Touch Filter   [✓ ON]  OFF
MIDI Clock Send      ON  [✓ OFF]
Local Control        [✓ ON]  OFF
MIDI Thru            ON  [✓ OFF]
                     OK
```

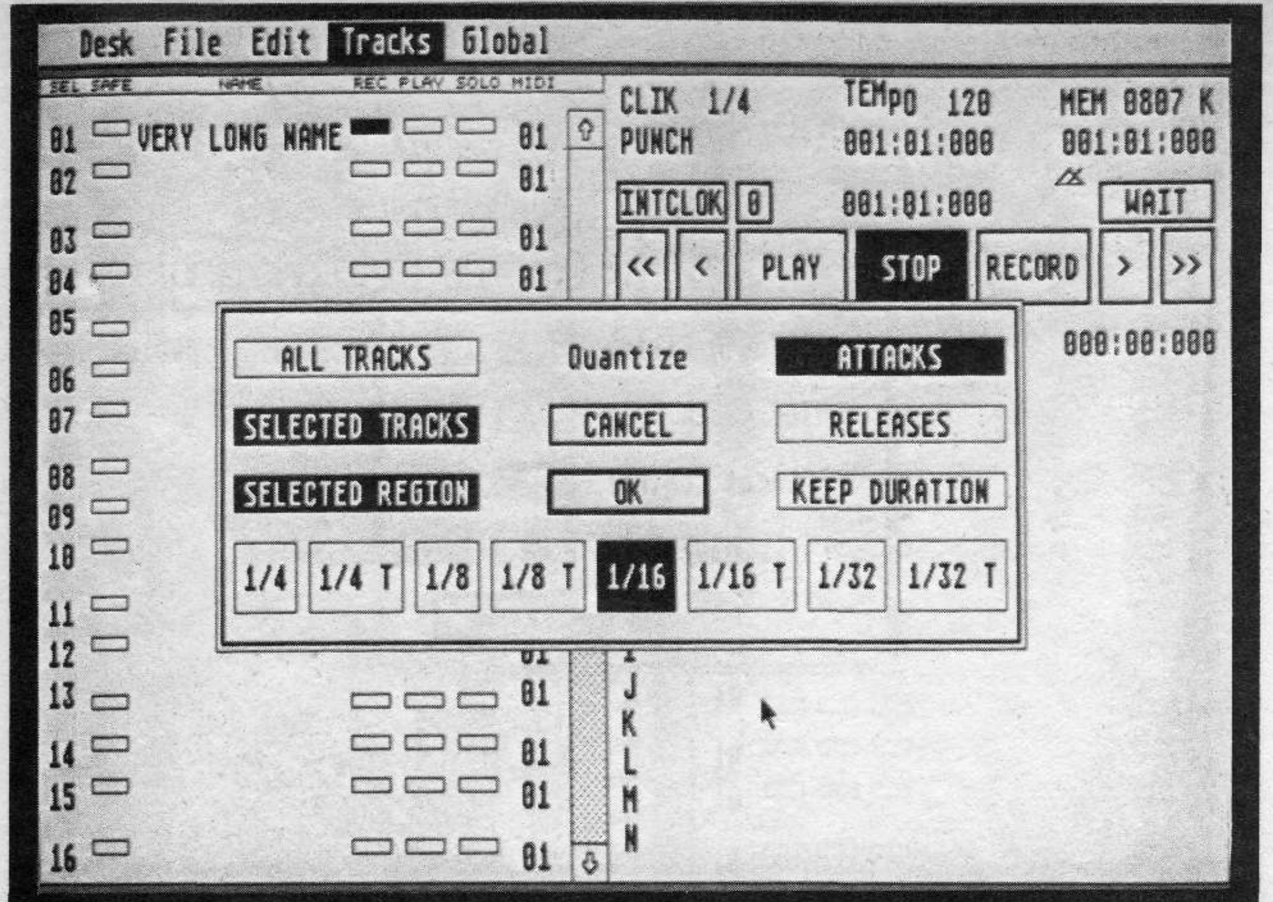
One of the set-up screens of the 32 track sequencer. This includes MIDI clock on/off switching.

the data stored in memory and processing it before it is sent out on the MIDI port. To remove the quantisation you merely instruct the sequencer to stop this pre-processing of the data. If the quantisation is implemented by adjusting the timing values stored in memory, there is unlikely to be any way of reversing the process. As explained previously, with any drastic and non-reversible process it is always a good idea to save the data to disc prior to making the change. If it does not give the effect you expected, you are then in a position to go back one step to the original version stored on disc, instead of having to start right from the beginning again. Some sequencers automatically keep a copy of your data (usually in memory rather than on disc) so that you can back-track one stage if desired. This is called "non-destructive editing", and seems to be implemented in few sequencer packages. Even where this feature does exist it might not work with all types of editing.

Many of the quantisation facilities offered by sequencer packages seem to be far from ideal. In some cases this feature has been put forward as a great asset, but closer examination of the specifications would suggest that the quantisation is unavoidable due to the low timing resolution of the system! Most sequencers these days are more sophisticated and do not have this enforced quantisation. This is just as well, since a sequencer which can not record sequences with reasonably accurate timing accuracy is of dubious value. Most sequencer packages for the ST seem to record with quite high timing resolution, and apply the quantisation to the data stored in memory. In other words, the quantisation is mostly of the non-reversible type. The degree of quantisation is normally selected on the basis of correcting timing to the nearest eighth note, sixteenth note, etc.

More sophisticated forms of quantisation are now emerging, and these take a variety of forms. They are all aimed at tidying up playing without introducing a monotonous "mechanical" sound to the final product. The success of this type of quantisation is something your have to decide for yourself, but some quite impressive results can certainly be obtained. Some sequencers used to provide a "randomise" feature where the lengths of notes were randomly shortened or lengthened slightly in order to give a less mechanical "feel" to the music. This facility seems to have fallen from favour and has been largely replaced by more sophisticated techniques.

A simple but useful feature to be found on most sequencers is a metronome. This provides the standard metronome "click" sound via the ST's internal sound generator, and in some cases a so-called MIDI "click" is also sent. There may even be an on-screen graphical representation of a metronome, complete with swinging pendulum! Apart from the obvious one, another use



The quantise control screen of the 32 track "Shareware" sequencer. It has a respectable range of features for a low cost package.

of the metronome is to provide a sort of count-down facility so that all tracks can be started at precisely the same point. In other words, you start the sequencer by operating the on-screen "record" control, wait until the metronome has "ticked-off" (say) two bars, and then commence playing at the start of the third. Up-market sequencers usually provide an adjustable lead in period, but this is by no means always present on the less expensive sequencer programs. Another way of handling things is a facility that automatically starts recording as soon as something is played on the MIDI keyboard.

Editing

Probably the main attraction of MIDI multi-track recording over conventional multi-track tape recording is the editing possibilities it opens up. Obviously some degree of editing is possible with tape recording, and a skilled sound engineer can punch out unwanted notes, piece together several imperfect takes to make one good one, and so on. The advantage of MIDI is that this type of thing becomes very much quicker and easier. What would otherwise require a highly skilled sound engineer and a great deal of work can often be accomplished at the touch of a few buttons by virtually anyone. Also, MIDI editing capabilities go well beyond those offered by tape recording systems.

If you have ever wondered why some sequencer programs cost so much more than others which seem to have a similar basic specification, the answer probably lies in their editing facilities. With the less expensive sequencers the editing facilities may be bordering on the non-existent. By contrast, with the most expensive of ST sequencer packages you have access to all or virtually all the data stored in memory, and can make any change within reason. We are not just talking in terms of being able to remove an incorrect note. With a sophisticated sequencer you can take out the wrong note and put in the right note!

The more basic sequencer programs only permit editing on a track basis, rather than at individual MIDI event level. For example, you would typically be able to quantise a track, merge tracks, delete a track, name a track, and time shift a track. This last feature means being able to move a track backwards or forwards in time, so that any lack of synchronisation between tracks can be corrected. Transposition is another common feature, and as one would expect, this can be used to shift the notes in a track up or down in pitch by a certain amount. The amount of shift can generally be anything from one semitone to several octaves.

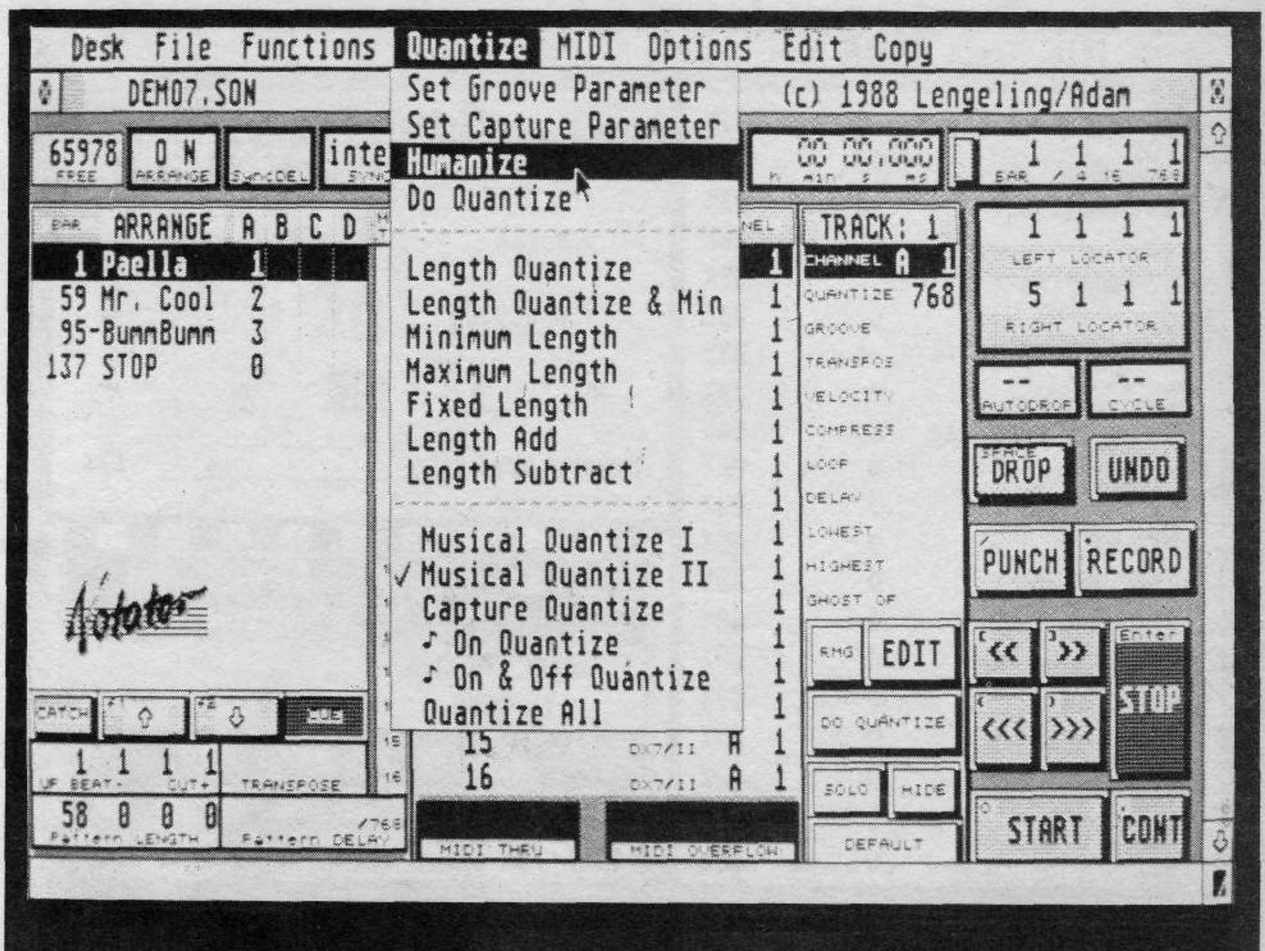
Another common feature is the ability to exercise some control over how the velocity information is handled. This basically means reducing the dynamic range (or disabling velocity sensitivity altogether), or boosting it. Not all instruments handle velocity information in precisely the same way, and a track when played back might not be quite as you would like relative to the others. This velocity control feature might enable you to improve matters As many recent instruments have a similar feature built-in, it is not necessarily an important facility to have in a sequencer. You may well be able to adjust your instruments to react to velocity information in precisely the desired manner. This obviously depends on which instruments you are using.

Event Editing

Even with just some of these basic editing facilities a multi-track MIDI sequencer is an extremely powerful piece of equipment. On the other hand, it lacks versatility and can be a bit frustrating at times. One minor error can leave no alternative but to re-record a whole track. Most basic sequencers permit part of a track which contains errors to be recorded-over with new (and, hopefully correct) material. Using this method it can be quite tricky to make quite minor corrections, although it is something that can be mastered with practice.

Providing a program with the ability to alter data at MIDI event level might seem to be only a minor improvement, but it really does offer tremendous versatility and convenience. Unfortunately, programs with this ability do seem to be quite expensive. The problem is not so much giving a program the ability to alter data, but making this facility easy to use. In order to present masses of complex data in a form that makes it easy to interpret and alter it is necessary to have some complex programming and some clever or graphics displays. With a program that has complex editing facilities, the editing part of the program is likely to be much larger than the main sequencing part!

The way in which data is presented for editing purposes varies tremendously. At a most basic level a text-only display can be provided, with something like different headings for different categories of data, or even just a list of data in hexadecimal form. It is then a matter of scrolling through the data to find the section you wish to edit, and to then make the changes using the mouse and (or) keyboard. Sequencers have a "tape counter" or accurate clock to help you keep track of where you are in a sequence, and this should help to direct you to the area of data you wish to edit. Apart from editing purposes, a sequencer that can display received data is very useful for diagnostic purposes when something goes wrong with the system, or something appears to be amiss. A quick study of the data spewed out by your instruments can also be very educational for someone starting to use MIDI equipment.



"Notator" provides plenty of quantise options.

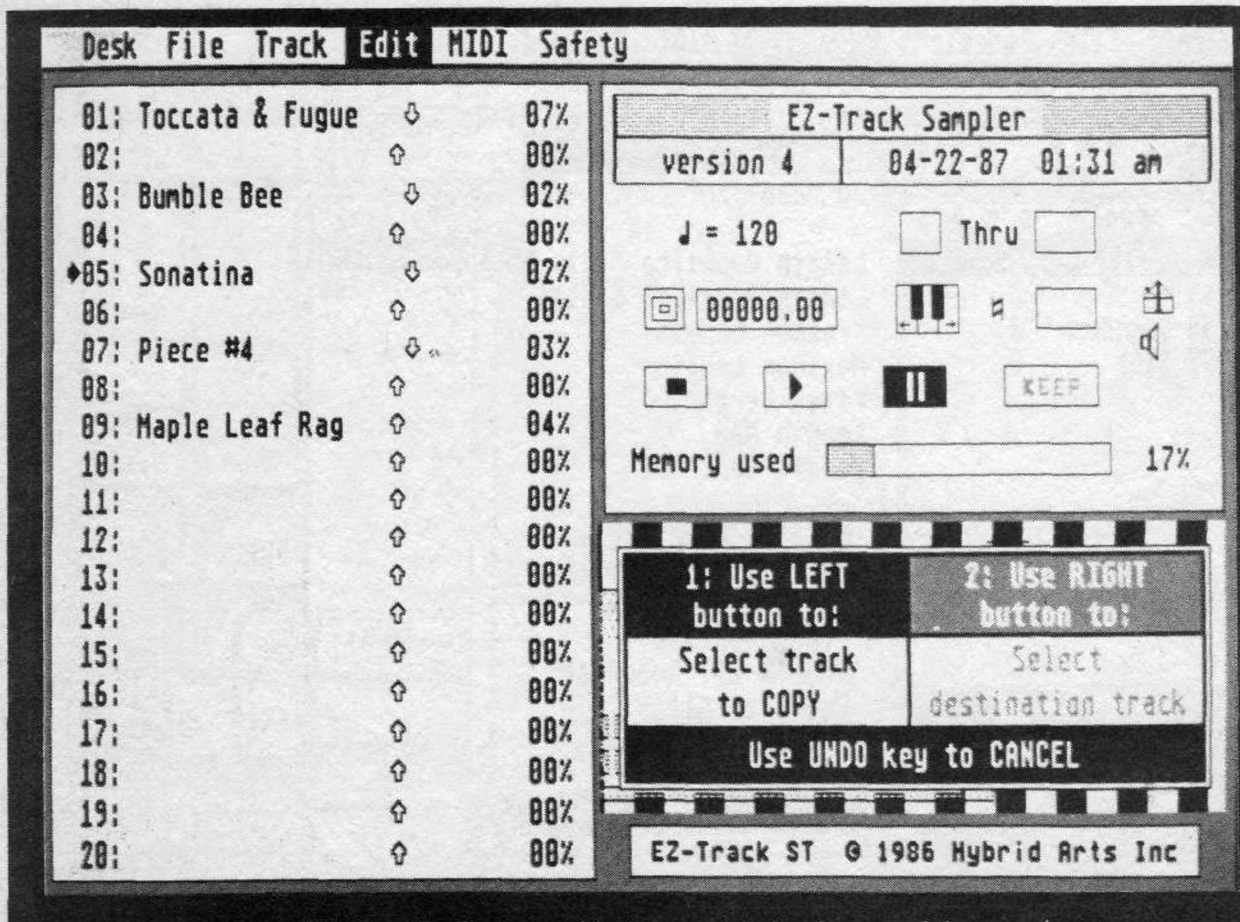
Editing of the type described above is called "event editing", because you are presented with full details of all the MIDI events that have been recorded. This includes such things as pitch bend and aftertouch data, and is not confined to straightforward note on and note off events. Up-market sequencer programs for the ST often offer some form of graphical editing environment, or *"grid editor" as these are generally called.* A grid editor would typically show notes on something resembling a multi-story stave, but notes would be represented by horizontal bars between lines, rather than in conventional notation form. The longer the bar, the longer the note duration. The vertical positioning of the bars would therefore indicate their relative positions. Editing could be accomplished by "dragging" notes in order to change their duration, position on the "stave", etc. This type of display and method of control can make editing notes very much easier than a simple event editor, or even one of the more sophisticated event editors. It requires no knowledge of MIDI whatever. The drawback of a system of this type is that it usually gives only pretty basic editing facilities. Note values and durations are easily altered, but other data may not be accessible.

Notation Programs

Taking the grid edit idea a stage further, some programs can display sequences in standard notation form. This is obviously a great boon to those who have been brought up on traditional music notation, and are used to doing things this way. Programs that have this feature cover a wide range of prices, and some are much more sophisticated than others. These days even the cheaper programs seem to offer a reasonable range of facilities. The display is not usually just a basic stave and notes type – *even the cheaper programs usually support* multiple staves, various key signatures, clefs, time signatures, and notation marks. The difference between the cheaper and more expensive packages tends to be just how many staves can be accommodated, and how many of the more unusual aspects of notation are supported (how many of the "C" clefs can be used for example).

Really there are several substantially different types of notation program. Some are not primarily notation programs, but are up-market real-time sequencers which offer a notation display as one of the editing options. There will often be compromises with this type of package, and you might for instance, be restricted to displaying one stave at a time.

Some notation or "scorewriter" programs are only intended for producing printed out scores, and have no MIDI sequencing capability at all. These can either be the music equivalent of word processors, or they can be designed to take data files from sequencers and convert these into standard notation scores (with the aid of



```
01: Toccata & Fugue   ⇩   07%
02:                   ⇧   00%
03: Bumble Bee        ⇩   02%
04:                   ⇧   00%
♦05: Sonatina         ⇩   02%
06:                   ⇧   00%
07: Piece #4          ⇩   03%
08:                   ⇧   00%
09: Maple Leaf Rag    ⇧   04%
10:                   ⇧   00%
11:                   ⇧   00%
12:                   ⇧   00%
13:                   ⇧   00%
14:                   ⇧   00%
15:                   ⇧   00%
16:                   ⇧   00%
17:                   ⇧   00%
18:                   ⇧   00%
19:                   ⇧   00%
20:                   ⇧   00%
```

```
EZ-Track Sampler
version 4      04-22-87  01:31 am
♩ = 120
00000.00
Memory used                17%

1: Use LEFT         2: Use RIGHT
   button to:          button to:
Select track        Select
  to COPY           destination track
     Use UNDO key to CANCEL

EZ-Track ST  © 1986 Hybrid Arts Inc
```

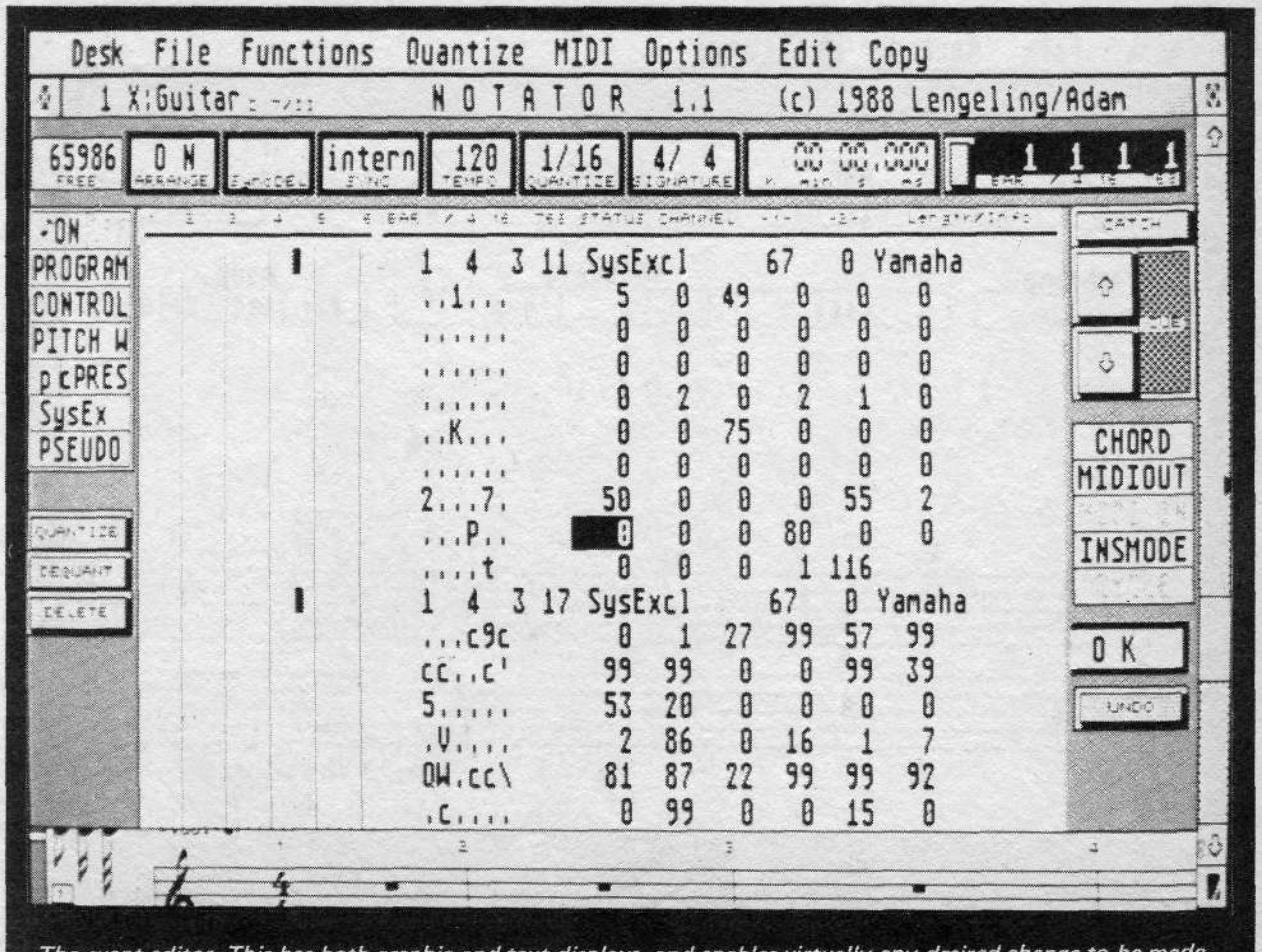
An "EZ-Track" edit window (the bottom right hand corner of the screen). Note the visual metronome (above and to the right).

some editing by the user). Both types can obviously be very useful, but seem rather limited in scope when compared to some other forms of sorewriter program.

We have not considered the subject of step-time sequencing in any detail yet, and the choice of step-time sequencer programs for the ST computers seems to be relatively limited. Probably there is insufficient demand for step-time sequencers for the software houses to expend much effort in this direction. If you require a good step-time sequencer there are two basic options. The first is to buy one of the better real-time sequencers which has really good editing facilities at event level. Although these facilities may be primarily intended to permit alterations to recorded data, most will allow data to be added "from scratch". In other words, you can program any MIDI data and timing information, and build up complex sequences if desired.

The second approach is to use a scorewriter program that does include MIDI sequencing capability. I suppose that a notation program that can play the scores via MIDI might seem to be an out-and-out step-time sequencer. However, the programs of this type that I have encountered (both for the ST and other computers) usually include real-time sequencing. In other words, you can play a sequence into the unit via a MIDI keyboard, and as you play the notes they pop-up onto the stave. This is quite impressive to watch, but the programs I have tried have had what is only very limited real-time sequencing capability. They are often easily overloaded by playing too many notes too quickly. Programs of this type normally include a print-out facility so that sheet music can be produced. An ordinary 9 or 24 pin dot matrix printer can produce quite good results, but make sure that the printer you intend to use is fully compatible with the notation program you select.

For real-time sequencing, a notation program is probably the best type of sequencer to use. If you are not familiar with standard music notation, and you wish to undertake a lot of real- time sequencing, then it will probably be worthwhile taking some music lessons! The standard system of music notation has evolved over a long period of time to the point where it enables very complex pieces of music to be written down in a very compact form. Dynamics, variations in tempo, etc., are all accommodated by this system. With a good notation program if, for example, you mark a point in the score "ppp", the music will be played very quietly (assuming you are using touch sensitive instruments). Notation programs generally offer much greater versatility than that afforded by grid editing, and they are much quicker and more convenient than programming masses of data using an event editor. Programs of this type seem to be gaining in popularity, and there is much more of this type of software available now than was the case a few years ago.



```
Desk  File  Functions  Quantize  MIDI  Options  Edit  Copy
1 X:Guitar          N O T A T O R   1.1     (c) 1988 Lengeling/Adam

65986   0 N      intern   120    1/16      4/ 4      00 00,000     1  1  1  1
FREE    ARRANGE  SYNC     TEMPO  QUANTIZE  SIGNATURE

                 1  4  3 11 SysExcl       67    0 Yamaha
                 ..1...       5   0  49   0   0   0
                 ......       0   0   0   0   0   0
                 ......       0   0   0   0   0   0
                 ......       0   2   0   2   1   0
                 ..K...       0   0  75   0   0   0
                 ......       0   0   0   0   0   0
                 2...7.      50   0   0   0  55   2
                 ...P..       0   0   0  80   0   0
                 ....t        0   0   0   1 116
                 1  4  3 17 SysExcl       67    0 Yamaha
                 ...c9c       0   1  27  99  57  99
                 cc..c'      99  99   0   0  99  39
                 5.....      53  20   0   0   0   0
                 .V....       2  86   0  16   1   7
                 OW.cc\      81  87  22  99  99  92
                 .C....       0  99   0   0  15   0
```

The event editor. This has both graphic and text displays, and enables virtually any desired change to be made.

All In One Approach

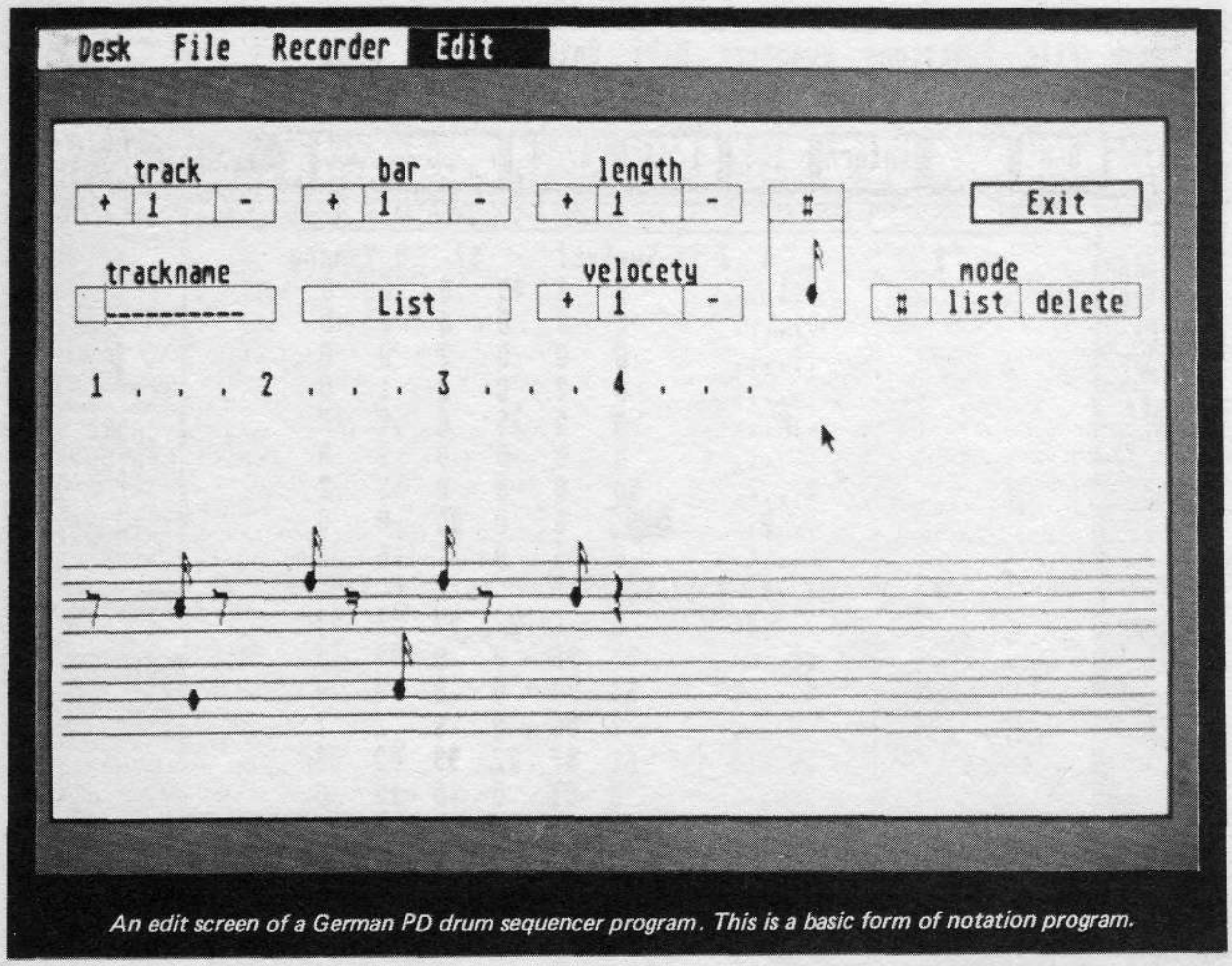
There has been a general trend over recent years towards programs that perform a whole range of related tasks rather than just one (the so-called "integrated" approach). This is something which can easily be applied to MIDI sequencing software. Instead of having separate real-time, step-time, notation, and score printing programs, why not have one program to do the whole lot? This offers much greater convenience than having separate programs, even if these programs have compatible file formats so that data from one can be loaded into another.

A few integrated sequencing programs for the ST are now beginning to appear, and offer tremendous versatility. Software of this type is highly desirable, but as yet it is also quite expensive. In fact a package of this type could easily cost more than a 520STFM plus monochrome monitor. On the other hand, on a number of facilities to the pound basis this type of software can offer much better value than many of the cheaper sequencers. If you need all the facilities provided by an integrated package, or even a large percentage of them, a program of this type is likely to be the best choice.

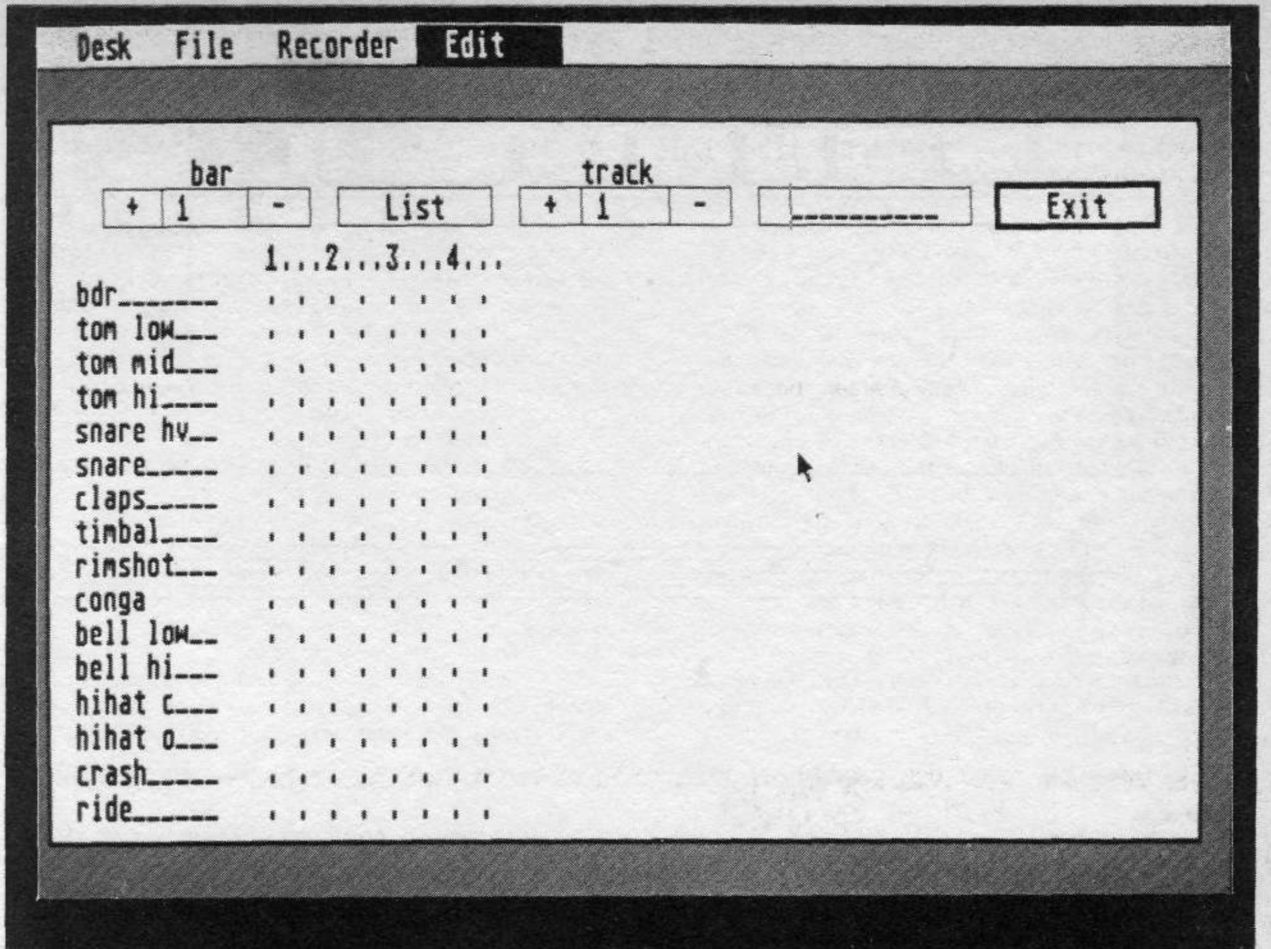
The exact facilities offered will clearly vary somewhat from one package to another. Normally a system of this type will basically be an up-market real-time sequencer having a comprehensive range of facilities, including a good event editor plus (possibly) some form of grid edit facility. Data from the real-time sequencer section should be usable with the notation part of the program. Also, and importantly, the notation part of the program should ideally be able to export data into the real-time sequencer. This interaction of the two sections of the program brings a number of important advantages.

The most obvious one is that it permits no-compromise real-time sequencing to be mixed with step-time sequencing via either the notation section of the program or the event editor. For the real-time sequencer there is the convenience of using the notation part of the program for sequencing; and then using the event editor for "fine tuning" purposes. If you program a passage to be very quiet, but you do not feel that the program makes it quiet enough, by using the event editor you can probably correct this and have the finished sequence exactly as you want it. If the reproduced music sounds a bit mechanical, you may be able to manipulate note durations slightly to add some expression to it. A combination of MIDI and an ST computer provides the musician with tremendous creative potential, but you need some top quality software in order to stretch the creative possibilities to the limit.

A multi-purpose program should work well as a scorewriter for producing printed out scores. A full range of notation marks (pedal, cresc/dim, etc.) should be available, and a useful range of printers should be



An edit screen of a German PD drum sequencer program. This is a basic form of notation program.



The grid-edit style screen of the German PD drum sequencer. This gives a form of step-time sequencing.

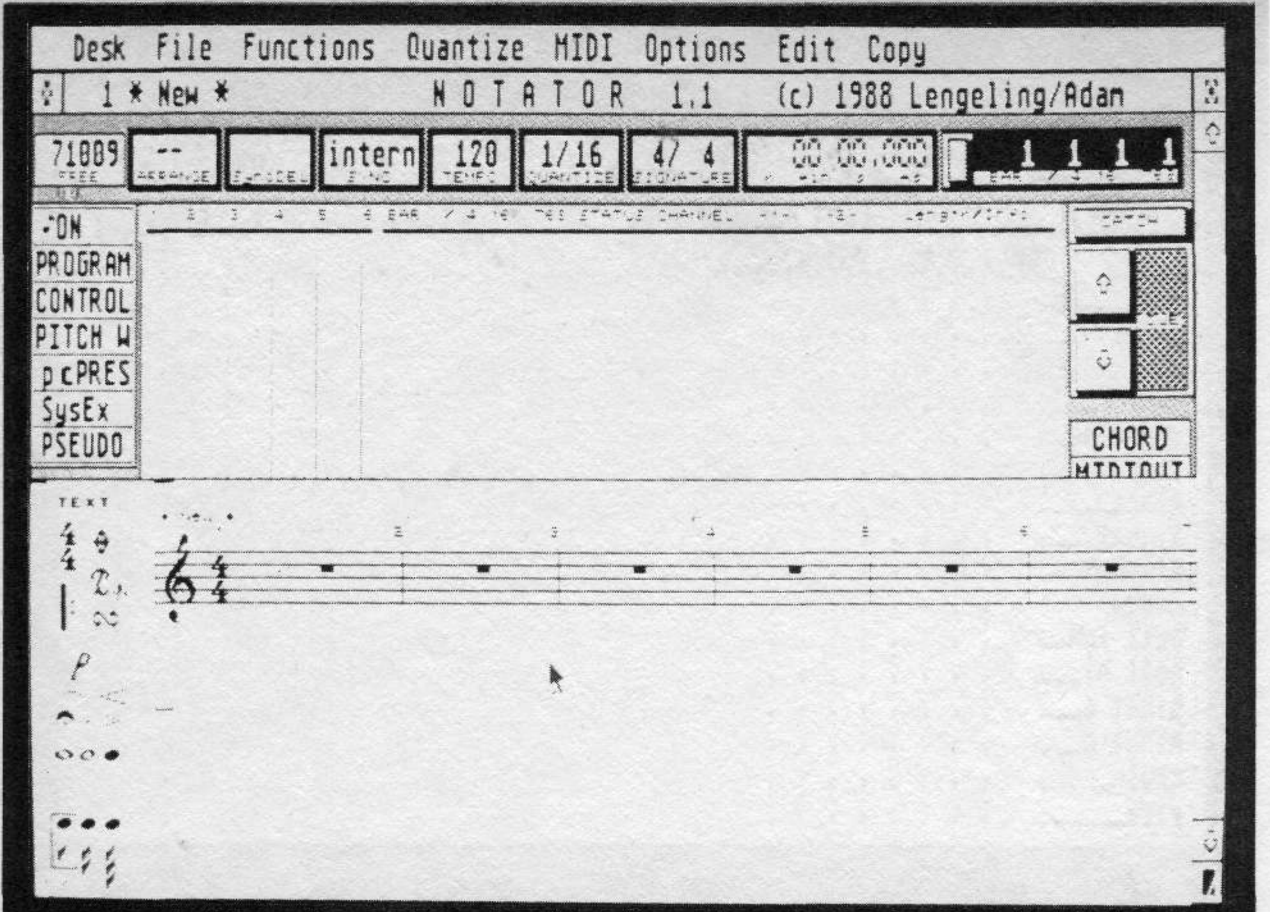
supported. There may well be some useful extra features, such as the ability to record system exclusive messages to disc, and to play them back over the MIDI output. This is useful for such things as storing program dumps, or sample dumps. There is a potential snag with this type of facility in that some system exclusive messages are two way types. An instrument may only send data to the computer if the computer responds with the correct acknowledgement messages. This requires software designed specifically for the make of instrument concerned. Some programs support the sample dump standard, but few (if any) seem to go beyond this.

The more sophisticated sequencer and integrated packages often have a large range of minor features available, but these vary considerably from one program to another. Some are potentially quite useful, but others seem to have no immediately obvious use! As with just about everything concerned with MIDI, it is a good idea to carefully read through software specifications to familiarise yourself with the available functions.

In Practice

If we go through the basic sequence of events when recording a track in real-time, and assume that everything is set up and is ready to go, the first task is normally selecting the required track. This is generally just a matter of "clicking" the ST's mouse on the appropriate box or other on-screen icon. Usually you can name tracks, and the name normally used is one that describes the sound and instrument to be played on that track (e.g. "DX7 flute"). Remember that a MIDI sequencer only stores MIDI data and replays it. It is up to you to make sure that each track is sequencing the right sound, and meaningful names for tracks (not just "sound 1", "sound 2", etc.) will make it much easier to get everything right when you return to a piece after a period of time.

With the aid of pull-down menus and on-screen control panels you then set up the sequencer with the desired parameters. For example, you might wish to switch in the aftertouch filtering, activate the metronome facility, and activate the MIDI THRU facility. You will normally have to set the MIDI channel number as well. While you may have to set the channel number on the MIDI keyboard, with notes being stored in the ST's memory with whatever channel number they are received with, most sequencers are much more versatile than this. It may be possible to set the sequencer to record only those notes that are on a



Into the "Notator" score edit function. This in fact can be used as the sole method of note entry, and can print out up to 32 stave scores!

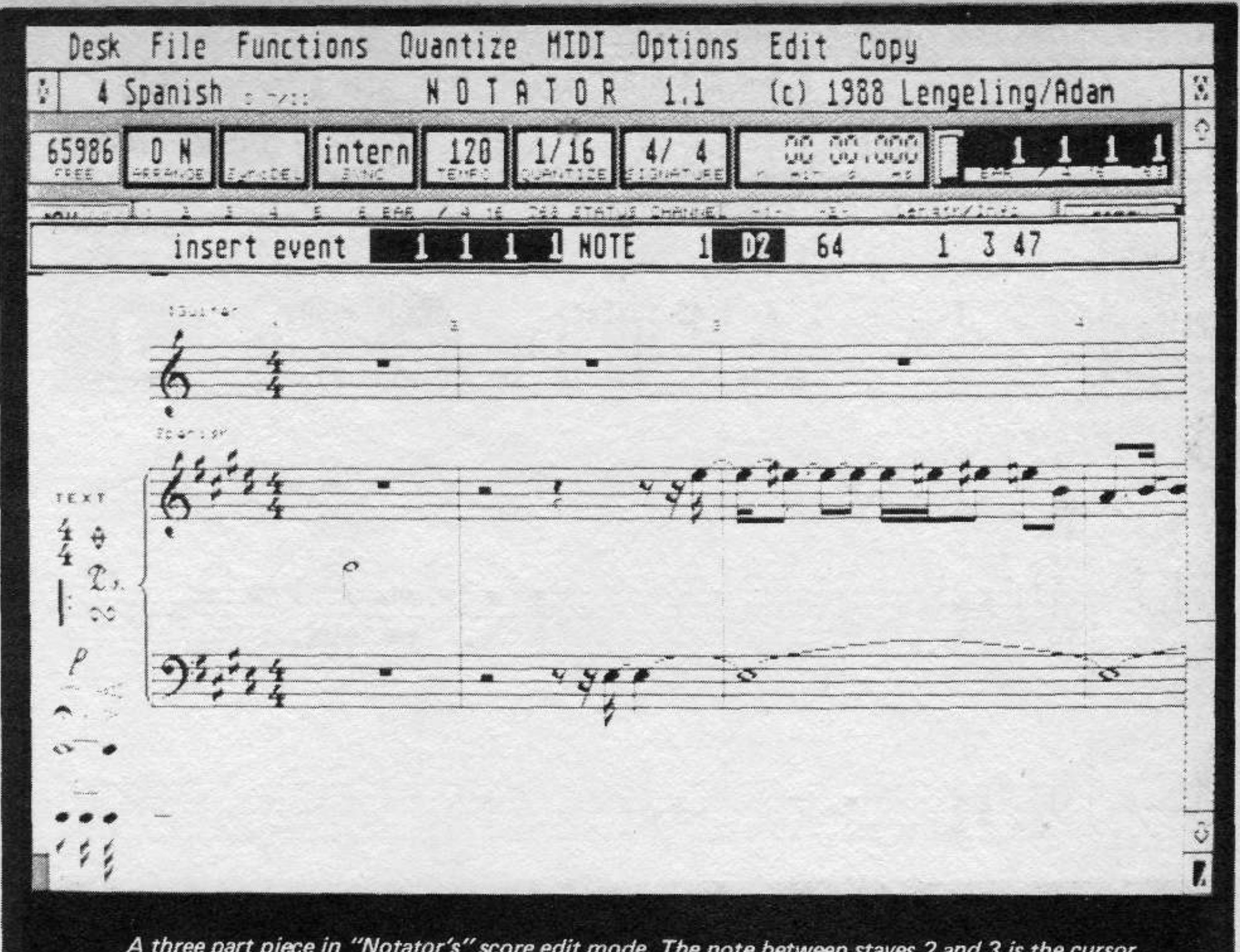
certain MIDI channel, but this is often not possible, or necessary.

More usually the sequencer will record notes on any MIDI channel and place them into the track being recorded. If notes are on several channels they are effectively merged onto the same channel, which is whatever channel you select for that track using the sequencer's control panel. The MIDI channel is something that can be changed at will even after a track has been recorded. It is generally much easier to have control of the system centralised at the sequencer, but this feature is more than just useful if the MIDI keyboard is one of those that can only operate on channel 1. The ability to record several tracks at once is something that you are only likely to find on one or two of the more sophisticated sequencers, but most users prefer to work on the basis of one track at a time anyway.

With everything set up and ready to go, the next step is to actually record the first track. Hopefully, the sequencer has some form of built-in "countdown" facility to lead you in, but if not you can simply leave a couple of empty bars a the beginning of every track to act as your "countdown". Operate "RECORD", and after the "countdown" has finished start playing your masterpiece. When the track is finished you use what will normally be tape recorder style controls to "STOP" recording, and to "FAST REWIND" (or "<<") back to the beginning of the sequence. The "PLAY" control should then reproduce the track, provided you have a suitable instrument connected to the MIDI output and set to the correct MIDI channel. If nothing has been recorded it is probably because the track you are using has not been set to the "record" mode. There should be some form of control to set each track to "record", "play", or "off".

Recording the next channel is much the same, with the track being named, everything being set up correctly and ready to go, and "RECORD" then being operated. Remember to set the second track to the "record" mode, and the first track to the "play" mode so that you do not record over it. It is very easy to get absent minded and end up with the wrong control settings, but most sequencers will not let you do anything really silly. You do not have to set track one to the play mode, and could simply switch it off. However, with track one in the "play" mode you can hear it playing via an instrument connected to the ST's MIDI output port, so that it is easy to get the second track properly synchronised with the first one. As you gradually build up more tracks you might find it beneficial to only have certain tracks playing as you add each new one.

Having built up your masterpiece you can edit it by changing the tempo, using the "punch in/out" facility to record over any less than perfect sections. When editing sections of a track the "tape counter" is often


Desk File Functions Quantize MIDI Options Edit Copy
4 Spanish
NOTATOR 1.1 (c) 1988 Lengeling/Adam
65986
0 N
intern
120
1/16
4/ 4
1 1 1 1
insert event 1 1 1 1 NOTE 1 D2 64 1 3 47
TEXT

A three part piece in "Notator's" score edit mode. The note between staves 2 and 3 is the cursor.

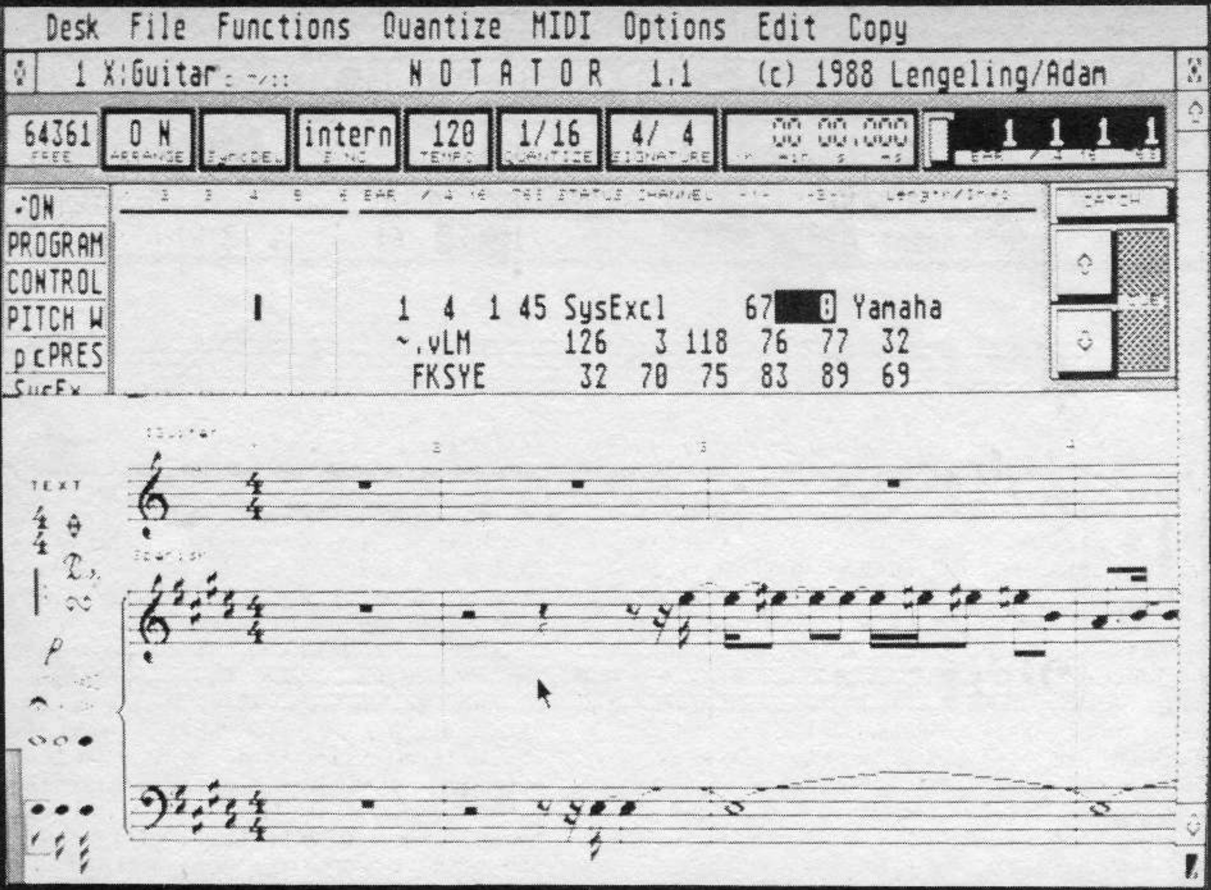
important in finding the right section. This type of editing is much easier if the sequencer provides some assistance, such as the ability to play back the track, and then mark the offending section by "clicking" the mouse. You can then record over the marked section, taking several attempts if necessary, without running the risk of obliterating a larger piece of the track. For beginners it is easy to make matters worse rather than better, and it is a good idea to store the sequence safely on disc before undertaking any editing. If your sequencer has one, the event editor is generally the easiest and safest means of making any minor corrections. It will probably permit fine adjustment of any aspect of a track that is not quite as you would wish.

Other Program Types

MIDI software for the ST does not just mean sequencers, and there are several other types of program available. Probably the most popular of these are the patch generation and librarian programs, particularly those for the Casio CZ and Yamaha FM synthesisers. The most simple of these programs are merely for storing patches on discs via the ST's disc drive. Few synthesisers have built-in disc drives, and storage of sounds beyond the capability of any internal memory has usually been by way of cassette tapes or memory cartridges. Cassette tapes are slow, inconvenient, and often not very reliable. In fact few synthesisers are actually equipped with a cassette port these days. Cartridges are a much better way of handling the problem, but tend to be quite expensive.

Dumping and retrieving patches via MIDI and an ST, provided you already own an ST computer and the librarian program is not too expensive, and is a more practical solution. It is fast, reliable, and inexpensive in that a large number of patches can be stored on a single 3.5 inch single sided disc. With a well written program it is also easy to select the desired sounds and load them into the synthesiser. Some programs offer advanced facilities such as being able to sort out sounds of a particular type (bass sounds for example).

Going one stage beyond basic librarian facilities there are the "editor" packages. Adjusting the sound generator circuits of modern synthesisers is quite a long and difficult task. In fact many users seem to settle for the factory preset sounds, or sounds bought on cartridge (or whatever). With the aid of MIDI and a computer such as the ST it is possible to make adjustment of the sound generator settings very much easier. One way of handling things is to have a program which has on-screen "controls" such as sliders and switches which can be manipulated using the mouse. These controls are related to various MIDI controllers, and can be used to program any MIDI instrument which makes use of MIDI controllers for its sound generator circuits. Operating any control results in the corresponding MIDI


Desk File Functions Quantize MIDI Options Edit Copy
1 X:Guitar
N O T A T O R 1.1 (c) 1988 Lengeling/Adam
64361
0 N
intern
120
1/16
4/ 4
00 00.000
1 1 1 1
PROGRAM
CONTROL
PITCH W
TEXT

```
1  4  1 45 SysExcl      67 ▮▮ 0 Yamaha
~.vLM          126   3 118  76  77  32
FKSYE           32  70  75  83  89  69
```

"Notator" has fully integrated functions. Data entered via the score editor can be "fine-tuned" in the event editor. The boundary between the two can be "dragged" up or down to the desired position.

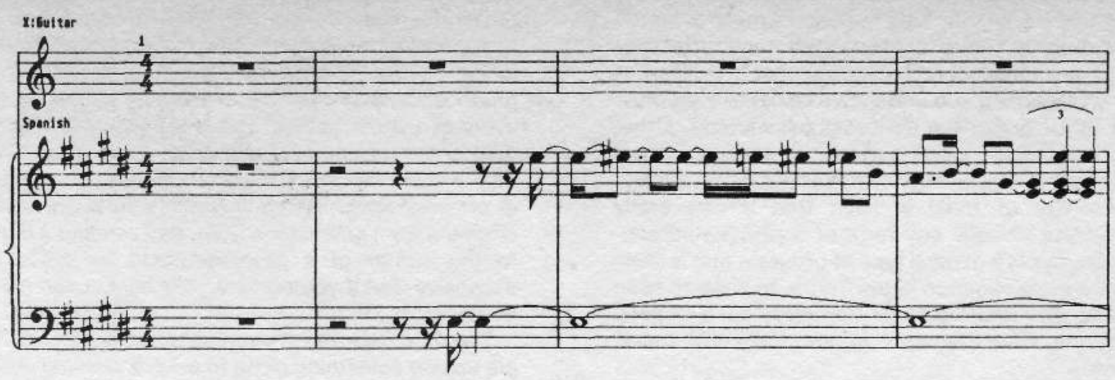
controller being set to the appropriate value. With instant access to a large number of controls it is very easy to trim the sound generator circuits to produce precisely the required sounds. It takes things back to "the good old days" (about ten years or so ago) when all synthesisers had rows of front panel knobs and switches and were easy to adjust.

This method is not applicable to all instruments, as not all give access to the sound generator circuits via MIDI. Also, many of those that do now seem to provide access via system exclusive messages rather than by way of standard MIDI controller messages. This is perhaps not a very helpful development, as it means that general purpose MIDI controller software (or MIDI controller hardware units) can not be used to much effect with instruments of this type. You may be able to adjust a few basic parameters such as master volume and modulation amount, but nothing more than this.

Controlling software for a number of the more popular instruments is now available for the ST computers, and this seems to be the fastest growing area of the ST MIDI software market. There could soon be more software of this type than sequencing software. Although these programs have the same goal, their methods of operation are very different. Some have simple on-screen controls, while others are somewhat more complex. For example, to adjust the envelope shape of a sound you might manipulate a graphical representation of the envelope shape rather than using ADSR style controls.

Getting the desired sound from some modern synthesisers can be quite difficult and time consuming. Adjusting one parameter can have an affect on others, making it difficult to predict just what will happen when a given control is adjusted. In particular, FM synthesis has often been criticised on this count. There are some quite novel and advanced aids to getting the right sound in some editor programs. One idea is to have the computer generate random variations that are close to any given starting sound. If you have a sound that is close to the one you want, a system of this type can help you to home-in on precisely the required sound. Another idea is to have conventional ADSR controls for the VCA and VCF so that sounds are easily set up and fine tuned, just like adjusting a conventional analogue synthesiser. This may not seem to be a particularly brilliant feature, but remember that FM synthesisers do not have conventional VCA and VCF envelope shapers. The computer is having to do some heavy calculations in order to turn the control settings into synthesiser parameters that will have the desired effect.

For the more technically minded there are waveform displays, and even 3-D Fourier analysis displays. A basic Fourier analysis display shows the frequency components present in a signal, and for a simple repetitive waveform this is the fundamental signal plus harmonics



Sample print-out from "Notator" using a 24 pin dot matrix printer. (N.B. Illustration shown slightly reduced for reproduction.)

(multiples of the fundamental frequency). A display of this type is usually a 2 dimensional type which has frequency on the X axis and relative strength shown on the Y axis. A basic display of this type has severe limitations in that the waveform of most signals changes considerably during the course of the signal. With most natural sounds the harmonics decay much faster than the fundamental signal. A 3 dimensional Fourier display has an additional (Z) axis, which effectively goes into the page away from the viewer. The display is shown as viewed from slightly above and to one side, so that a "mountain range" type display is obtained. For the experienced user this shows exactly how a sound will vary over its entire envelope period.

Editor and librarian software is a valuable asset, and most users seem to consider purchases of this type of software as money well spent. It is only fair to point out though, that this kind of software is not available for all MIDI equipped synthesisers. If you have one of the more popular synthesisers, then there will probably be several editor/librarian programs to choose from. If you have one of the less popular ones, you will be lucky if there is even one editor program available for it. If there is an editor program available for one of the rarer instruments, then it is almost certain to be a program for the ST.

Visual Editors

Editors are available for some sound samplers, but here the requirements are usually very different to those for synthesisers. The storage of data is not usually a problem, as samplers almost invariably have a built-in disc drive. Filters and envelope shapers are generally quite straightforward and easy to adjust. The problem is more one of obtaining good loop points. There is insufficient space available here to give a complete course in the art of sound sampling and looping. However, much sound sampling is done on the basis of recording a relatively short sound, and then repeating the end section of the sound over and over again in order to give an output that can be sustained indefinitely. Standard synthesiser voltage controlled filters and attenuators can then be used to process the signal to give the required decay characteristic, etc.

This sort of thing is fine in theory, but tends to be very tricky to implement properly in practice. The human ear is not easily fooled, and it is difficult to find a section of the signal that can be repeated without at least slight "clicks" or other audible glitches being produced. Some of the more recent sound samplers have built-in l.c.d. displays that enable the stored waveforms to be examined so that the best loop points can be found reasonably easily. Others have provision for a monitor to be connected so that this can be used to provide a graphics display for the same purpose. A number of sound samplers are devoid of any facility of this type though, and unless an oscilloscope is available, finding the optimum loop points is a matter of trial and error. I know from my own experience that finding good loop points by this means can be very time consuming indeed, and is not always successful.

A visual editor package takes sound samples via the MIDI output of the sampler and feeds them into the computer's memory. They can then be used to provide an on-screen waveform display which will make it easy to find the optimum loop points. Sometimes the program is simply used as an aid to selecting the loop points, with the controls of the sampler then being adjusted to the appropriate settings, but often all the setting up and adjustments can be carried out from the computer via MIDI.

If you are into producing your own sound samples and you have a sampler that lacks any form of built-in waveform display, a visual editor can certainly save a great deal of time. With awkward samples you may still be unable to obtain really good results, but at least you will realise that a "silent" loop is unobtainable fairly early in the proceedings, instead of wasting hours trying to achieve the impossible. One point worth making though, is that most of these programs are quite expensive. You may well consider that a full set of factory discs or a small oscilloscope represents a more cost effective solution!

This covers the main categories of MIDI software for the ST, but no doubt other types of software will emerge in due course. There are developments in the field of so-called "intelligent" music programs. This usually means what is basically a conventional sequencer, but with some novel features to provide automatic harmonising and accompaniment. The same sort of facility is possible using a computer as a real-time MIDI processor, but this is more difficult as the computer has to come up with almost instant solutions. This sort of thing seems likely to become more common in the future as more sophisticated (and effective) programs are developed. I suppose that this type of feature is something you either love or hate, and I must confess that I prefer the do-it-yourself method. Other types of program are starting to appear, such as control programs for complex MIDI controllable audio mixers. The versatility of MIDI is such that it can easily accommodate virtually any form of sophisticated control. Again, this is a general type of program that is likely to become more common in the future. In order to keep up to date with developments you really need to keep studying the advertisements in the electronic music magazines.

PD Software

A book such as this can give you a good idea of what types of MIDI software are available for the ST, and the types of function they can provide. You can only fully appreciate the creative possibilities provided by the ST and MIDI software by trying out some programs, and if you own an ST computer (or have access to one) this need not be expensive. Buying a selection of commercial MIDI software certainly would be, but there is a much cheaper alternative in the form of "PD" software. "PD" stands for "public domain", and software of this type is available from computer user groups and commercial organisations. So-called public domain software seems to cover three distinct types of software in reality.

The first type is the true public domain software. I have seen various explanations as to what constitutes true public domain software, and most of these seem to suggest that the originator of the program retains the copyright, but allows anyone to use the program in any way they wish. This includes modifying the program and including all or part of it in your own programs. The only real restriction is that you can not sell the program for profit (but you can give it away). I would have to point out that this does not agree with my understanding of the term "public domain", which can be applied to books, poems, music scores, or anything in which copyright can be held. My understanding of the term is that it applies to material in which no one holds the copyright. This can be due to the copyright having lapsed due to the passage of time, something being of a very general nature so that no one could reasonably claim copyright on it (such as a list of ASCII codes), or the author having given up the copyright.

Whichever interpretation is correct, as far as the user of public domain software is concerned, it is free. In practice there are usually some costs involved, but these are simply the cost of the disc onto which the software is copied, postage charges, and so on. There is no charge for the software itself. There seems to be relatively little software in the true public domain category, and most of the software of this type is pretty basic (which is not really surprising).

The second category is "shareware" software. This is commercial software, but anyone can copy it, try it out, and use it. However, if you find a shareware program useful and intend to go on using it, you are asked to send a donation to the author. You may get something like a printed manual for your money, or the payment may bring no advantage other than peace of mind. The registration fees are normally quite low compared to the cost of similar ordinary commercial programs. You have to set against this the fact that you are usually very much on your own with shareware programs in the event of any difficulties. This is especially the case with programs that originate overseas. A call to the author on the other side of the Atlantic could prove costly even at off-peak times! Much shareware software seems to originate from outside the U.K., and sending a donation to the author of a program could be difficult and expensive (but if you contact your bank it can probably be done).

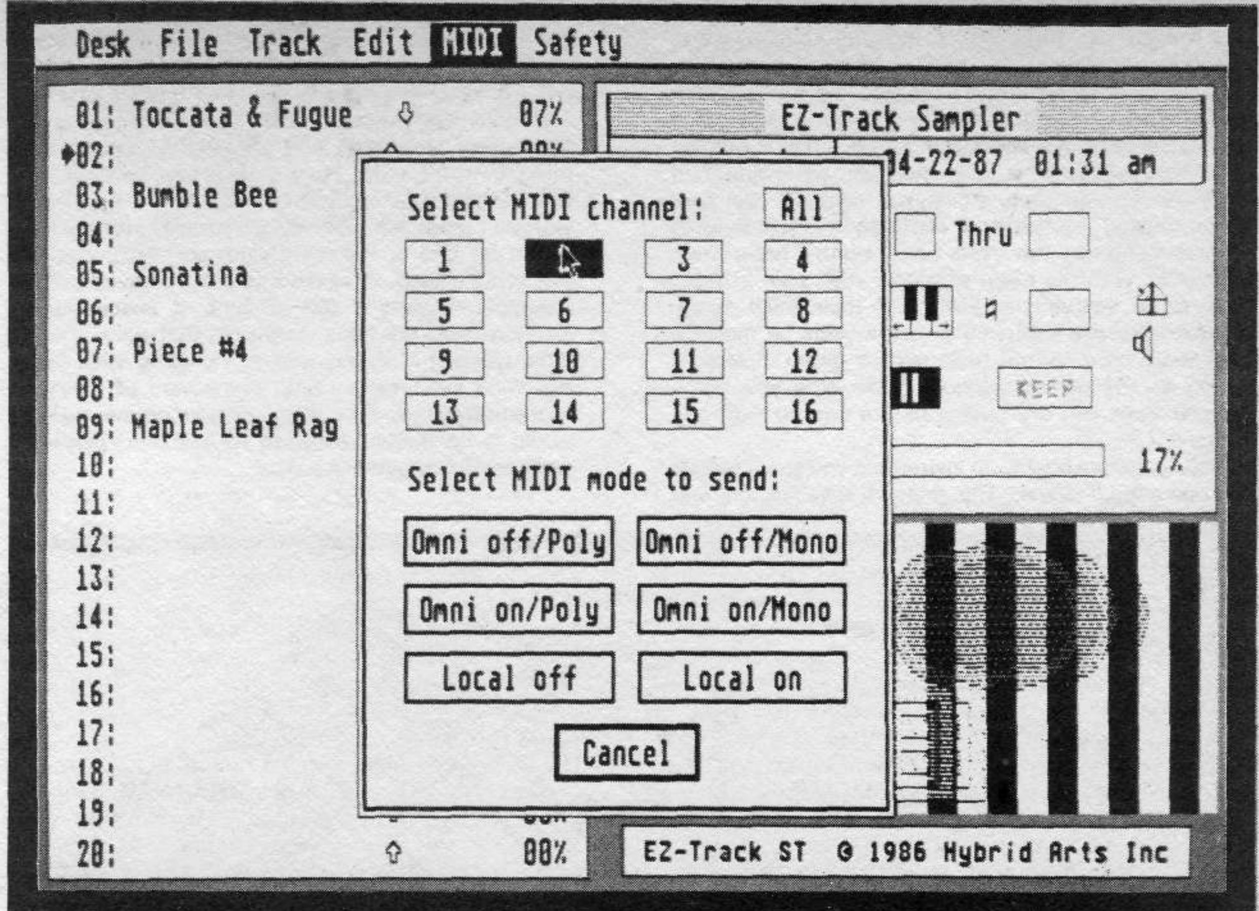
The third category is demonstration programs. These are usually something close to the full working commercial programs, but with one or two important features disabled. This generally means that no data can be saved to disc, but there may be one or two other facilities absent. These are excellent as they enable you to try out and fully evaluate software without having to spend (and possibly waste) large sums of money. Note though, that some are running demonstrations. A demonstration program of this type simply goes through an automatic sequence which shows what the program can do. You have no control over the program, and have less opportunity to evaluate it.

Some Programs

The number of public domain programs is constantly growing. You will need to get hold of one of the public domain software catalogues for the ST in order to find out exactly what is available at present. You should find some simple patch generators for Casio CZ and some Yamaha DX synthesisers, and voice libraries for these instruments. There is an interesting 32 track real-time sequencer available (this is shareware and not a public domain program), which some claim to be better than the cheaper commercial sequencers while others are less enthusiastic. It will only cost you a few pounds to buy the disc and decide for yourself, but when I tried it out it seemed to compare quite well with the low cost commercial alternatives.

In common with much public domain and shareware software, the documentation supplied on the disc is extremely sparse. The controls are not quite as obvious in operation as the author would have you believe, but with a little experimentation you should be able to get everything up and running properly. This program certainly represents a good and inexpensive way of trying your hand at real-time MIDI sequencing. If you rapidly progress to more sophisticated software, or decide that MIDI sequencing is not for you, at least you will not have wasted a lot of money on a basic sequencer program.

There is a demonstration version of the Hybrid Arts "EZ Track" sequencer program, which is another basic real-time MIDI sequencer. This is a fully working version of this 16 track program except that saving of data to disc is not implemented. There is no documentation at all included on the disc I obtained, but with some experimentation it is not too difficult to work out how to use the program. Like the 32 track sequencer mentioned above, it uses a standard GEM environment. There is also a demonstration version of the Hybrid Arts "EZ Score Plus" program which you might be able to



A screen from the demonstration version of "EZ-Tracks". Demo discs provide inexpensive means of giving in-depth trials to commercial software.

obtain. This is a basic but interesting notation and step-time sequencer program, which also has some real-time sequencing capability. It is again a largely working version of the program, but you will have to buy the real thing if you want to use it in earnest. In both cases you have a chance to try out some quite powerful MIDI software at very little cost. Remember that once you have finished trying out the programs you can wipe the discs clean and use them as blank discs, recouping some of your outlay!

You are unlikely to find examples of the more unusual types of MIDI program in the public domain software libraries, but there is no harm in studying a few lists to see if you can find something of interest. The quality of public domain and shareware software covers a wide span. Some programs do not work at all, or do not do very much if they do work. Other programs are quite professional, being both easy to use and fully operational. You are paying nothing for the software, and have to accept that some programs will be worth what you paid for them. You do not have a lot to lose by trying out a few programs, but you could gain a lot of valuable experience from them, and might pick up a real bargain.

MIDI Instruments

The requirements for instruments in a MIDI studio are not that much different to the requirements for "live" performances, but they are slightly different. Obviously the sounds of the instruments are all-important regardless of the exact way in which they will be used. Equally obviously, for MIDI work the instruments must be equipped with MIDI sockets. Synthesisers, samplers, and any reasonably up-market electronic music equipment seems to be fitted with MIDI sockets these days. MIDI is also to be found on a lot of electronic pianos and organs, as well as some other instruments such as the more sophisticated portable keyboards. It is best to check this point rather than jumping to conclusions, especially if you are buying secondhand equipment. Anything pre 1982 will not be equipped with MIDI sockets, and MIDI did not immediately "take off" after its inception. Accordingly, few instruments immediately after this period are MIDI equipped.

The fact that an instrument has MIDI sockets does not mean that it is ideal for sequencing purposes. Many of the instruments currently available are multi-timbral and can implement mode 4 and (or) some form of "special" or "multi" mode. This enables them to act as (typically) eight virtual instruments, giving tremendous potential and flexibility. Two instruments of this type effectively give you a programmable orchestra. Some instruments can only implement mode 1 and mode 3, and these are far from ideal for much sequencing work. Using instruments of this type, in order to get several sounds simultaneously you will need to have several instruments, and with mode 1 you can not individually sequence the instruments from a single MIDI output. Such an approach is one most of us would find prohibitively expensive. Again you need to take care when buying secondhand equipment. Typical MIDI specifications of a few years ago are rather basic when compared with the norm of today. Also bear in mind that some instruments have had their MIDI specifications steadily improved over the years. A model a few years old may not have such a good implementation as the current version. Make sure you know exactly what you are buying before parting with any money!

Ideally, before buying an instrument you should study its operating manual. The general specification and details of the MIDI implementation should make it quite clear just what can and what can not be achieved. The MIDI specification normally includes charts which show what MIDI messages are recognised, which ones are transmitted, the modes that can be used, and so on. Does the instrument have mode 4 or multi-mode? Does it respond to velocity, pitch bend, and aftertouch information? Does it have any useful system exclusive capabilities or other useful MIDI extras? The more complete the MIDI implementation, the more useful the instrument is likely to be in a MIDI studio environment.

If you are only interested in step-time sequencing you do not require a keyboard instrument. However, most sequencer users rely on either real-time sequencing, or a mixture of real-time and step-time sequencing. Either way you will require at least one instrument having a good quality keyboard, or a separate MIDI keyboard. The better synthesisers and samplers have good quality five octave keyboards that are velocity sensitive and often transmit some form of aftertouch information. One of these is perfectly adequate for most purposes. If you want more than five octaves you will probably have to opt for one of the more expensive MIDI equipped electronic pianos. A seven and a bit octave keyboard inevitably requires a fair amount of space though. Non-keyboard players should note that there are now other types of MIDI instrument, including MIDI guitars and wind instruments. With the advent of these and sophisticated step-time sequencing programs, the MIDI studio is no longer restricted to keyboard players of reasonable competence.

The "Roland D110" multitimbral synthesiser module. This is ideal for MIDI sequencing.

For a multi-instrument system there is a lot in favour of using some rack-mount types. The most obvious advantage to these is that they avoid the cost of several keyboards if you only need one for recording purposes. Another big advantage for the spare bedroom MIDI studio is that they are much more compact than keyboard instruments. A stack of three or four rack-mount instruments will fit onto quite a small (but strong) table, desk, or even a reasonably deep shelf. It is also worth noting that some rack-mount units have better MIDI specifications and occasionally other features which are not present on any keyboard alternatives.

Chapter 6
ST ADD-ONS

The electronics do-it-yourself addict can produce some simple, inexpensive, but useful music add-ons for the ST. The units described in this chapter are all reasonably simple, and in most cases are extremely straightforward to build. With one exception (the gate/CV interface) they should be well within the capabilities of those with relatively little experience of electronics construction. In fact some of them are so simple that they should be suitable for complete beginners to electronics construction. Stripboard layouts and wiring diagrams are provided for all the circuits apart from the gate/CV interface, and no difficult construction techniques are involved. In most cases they can be built for a fraction of the cost of any comparable ready-made units. A substantial amount of technical information is provided for those who require it, but you can ignore the technicalities and simply build up the projects if that is the limit of your interest.

The gate/CV interface can not be recommended for complete beginners at electronics construction, and not just because it is a bit more complicated than the other projects. There can be difficulties in using a unit of this type, and you need to be sure that the instrument you intend to control via this device is fully compatible with it prior to commencing construction! Remember that there was no true standard method of interfacing synthesisers in the pre MIDI era.

THRU Box

The virtues of the THRU box were discussed in a previous chapter, but to briefly recapitulate, in order to use the "star" system of MIDI connection a number of MIDI outputs are required. The "star" system is not possible with unaided ST computers as these have just a single MIDI output port. A THRU box has a MIDI input plus two or more THRU outputs, so that a number of units can be driven without any need for "chaining" them together. This avoids problems with so-called MIDI delays, and should improve reliability. If your instruments lack THRU sockets there may be no alternative to using a THRU box. If only one instrument lacks this facility, then this one can be used as the last one in the "chain". However, if two or more instruments do not have a THRU socket, the "chain" method of connection is unusable.

A THRU box does not have to have an opto-isolator at its input. The MIDI instruments or other items of equipment driven from the unit should all have isolated inputs, and this should prevent the THRU box from producing any interconnections between them, or between these units and the driving device. Despite this, I still prefer the use of an opto-isolator at the input of any MIDI equipment, including something as basic as a THRU box. To be strictly within the MIDI standard the opto-isolation must always be present at any MIDI input. Of greater importance, the circuit driving the input is something of an unknown quantity. Using an opto-isolator at the input of the unit is a good way of ensuring that the unit will operate properly with any MIDI output that provides a proper 5 milliamp current loop signal.

The two types of opto-isolator most commonly used in MIDI applications are the CNY17 and 6N139, or similar devices to these two types. Pin-out details and internal circuits for these two components are shown in Fig.6.1. The CNY17 is a standard device of the infra-red l.e.d. and photo-transistor variety. In a MIDI application this would normally be used in a circuit of the type shown in Fig.6.2. R1 provides current limiting which helps to set the l.e.d. current at approximately the correct level. It also gives the input side of the opto-isolator a degree of protection if the equipment is

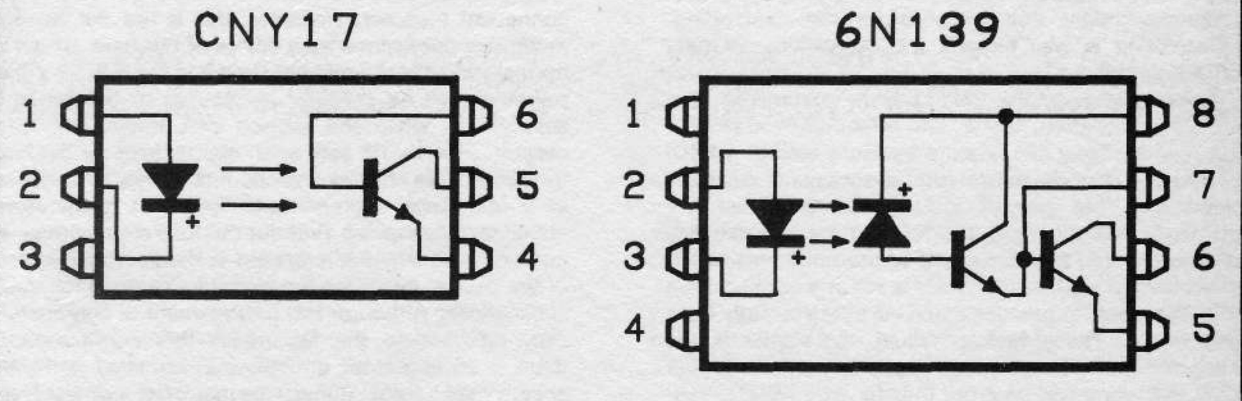


Fig. 6.1. Internal circuits and pinout details for the CNY17 and 6N139.

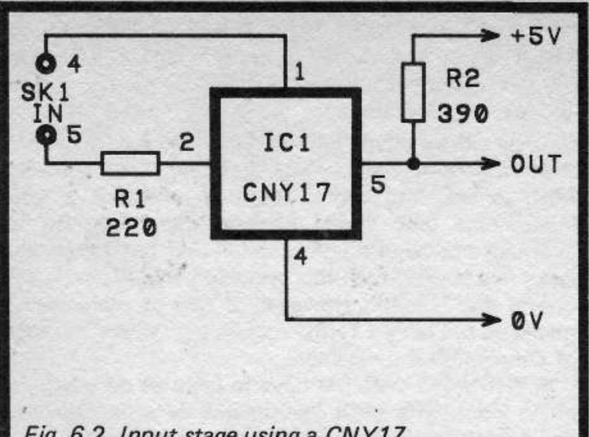


Fig. 6.2. Input stage using a CNY17.

wrongly connected or a malfunction occurs. On the output side of the circuit, R2 is the collector load resistor for the photo-transistor. Normally this transistor is switched off, and R2 takes the output to logic 1. When the l.e.d. is activated, its light output results in the transistor switching on. A heavy leakage current flows, but the effect is much the same as if the device was switched on via a base current. It pulls the output voltage down to little more than the 0 volt supply potential, or logic 0 in other words.

Although fine in theory, I have experienced problems with this type of input stage. Most opto-isolators, including the popular TIL111 and near equivalents, are totally unsuitable for this application. They fail on two counts, and one of these is simply that they lack efficiency. The 5 milliamp input signal produces such a low output current that a good logic compatible output signal can not be produced. The other problem is simply one of operating speed. Opto-isolators are inherently slow devices, and an ordinary type can not handle high frequencies. The maximum fundamental frequency in a MIDI signal (obtained when sending alternate 1s and 0s) is 15625Hz (i.e. half the baud rate). This is at the upper end of the audio range, and switching at this frequency might not seem to be too demanding. However, it is well beyond the capabilities of most opto-isolators.

Devices such as the CNY17 have guaranteed high efficiencies of about 100%, and offer improved switching speeds. They can operate perfectly well in a MIDI application, but there can still be occasional reliability problems. This centres around the fact that the efficiency of one opto-isolator can be substantially different to that of another one of the same type. Also, although the MIDI loop current is set at a nominal level of 5 milliamps, in practice it can vary substantially from this figure. These factors would not matter if the opto-isolator had a switching speed that was far higher than that demanded for the transfer of a MIDI signal, but this is not the case. A fast opto-isolator such as the CNY17 is only just fast enough for this application. The value of the load resistor therefore needs to be right for the efficiency of the device used and the exact input current. In practice some fine "tuning" of the load resistor value may be needed in order to get good results.

Much better results can be obtained using a more complex opto-isolator such as the 6N139, although it is only fair to point out that the cost of devices of this type is relatively high. However, they are probably worth the extra cost. The l.e.d. has its output directed at a photo-diode, and this offers a very fast switching speed when compared to a photo-transistor. On the other hand, it offers an extremely low level of efficiency (typically only a small fraction of 1%). The diode is connected so that its leakage current provides a base current to a switching transistor. This transistor in turn provides an amplified current to the base of a second transistor. This gives a combination of very high efficiency and high operating speed. In some opto-isolators a similar arrangement is used, but the photo-diode is omitted and the first transistor operates as a photo type. This arrangement seems to offer a similar level of performance.

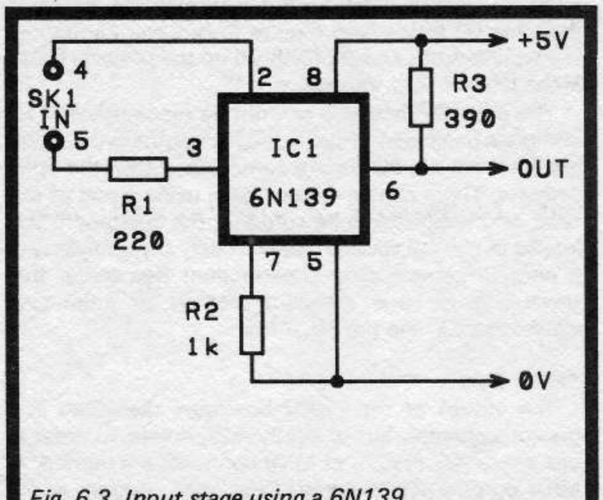


Fig. 6.3. Input stage using a 6N139.

It is important to realise that this type of opto-isolator is not a Darlington Pair device. It would be a Darlington type if the collectors of the two transistors were to be connected together. However, this is not the correct method of connection for a device of this type, which is normally used in the manner shown in Fig.6.3. The first transistor has its collector connected to the positive supply rail, while the second one drives the load resistor. Resistor R2 acts as an emitter load for the first transistor. This ensures that the first transistor operates at a reasonably high current, and that it therefore operates quite rapidly. Without this load resistor there is normally a substantial reduction in the operating speed of the device, rendering the circuit inadequate for MIDI applications. Although this arrangement is only marginally different to the Darlington Pair configuration, there is an important difference in terms of performance. While a Darlington device may offer a similar level of sensitivity, its switching speed would be very much slower. In fact Darlington types are extremely slow in operation, and are totally unsuitable for an application of this type.

The Circuit

The circuit diagram for the MIDI THRU box appears in

Fig.6.4. One way of obtaining the desired action would be to have an opto-isolator circuit followed by several inverters and open collector output stages, with each inverter/output stage driving a THRU socket. I found that the 6N139 was capable of providing a much higher output current than the 5 milliamps required for each MIDI output. This permits this much more simple arrangement to be used. The second transistor in the 6N139 is used as an open collector output stage which drives four MIDI THRU sockets. This requires IC1 to provide an output current of only about 20 milliamps, and it is perfectly capable of doing this. In fact there should be no difficulty in adding two or three more THRU outputs to the unit if required. It is just a matter of adding extra sockets and pairs of current limiting resistors to the circuit. A single resistor of about 470R in value would suffice for the current limiting, but it seems to be quite normal to use this twin resistor arrangement. This method presumably gives better protection against excessive current flows in the event of an output being incorrectly connected or a malfunction occurring.

The circuit is powered from a 6 volt battery such as four HP7 size cells in a plastic battery holder. These holders have standard PP3 style press-stud type connectors incidentally. No on/off switch is shown in the circuit, and it is probably not worthwhile adding one. Under quiescent conditions only minute leakage currents flow, and these are typically well under one microamp. This is not sufficient to significantly run down the battery even over a period of several months. The current consumption when the unit is working depends on the number of outputs that are actually in use, and the density of the data stream. Under worse case conditions with four outputs in use the average current consumption can be no more than about 10 milliamps. Under normal operating conditions the average current consumption will be far lower than this (probably only about 1 milliamp).

Construction

Details of the component board are provided in Fig.6.5. This layout is based on a 0.1 inch pitch board which has 30 holes by 17 copper strips. The board is not sold in this size, but it is easy to cut a larger piece down to the right size using a hacksaw. Cut along rows of holes, and then use a file to smooth any rough edges that result. Next drill the three mounting holes, which should be about 3.2 millimetres in diameter (to take 6BA or M3 mounting bolts). Then make the eleven breaks in the copper strips at the points indicated in the diagram. A special strip cutter is available, but a hand-held twist drill of about 5 millimetres in diameter will do the job

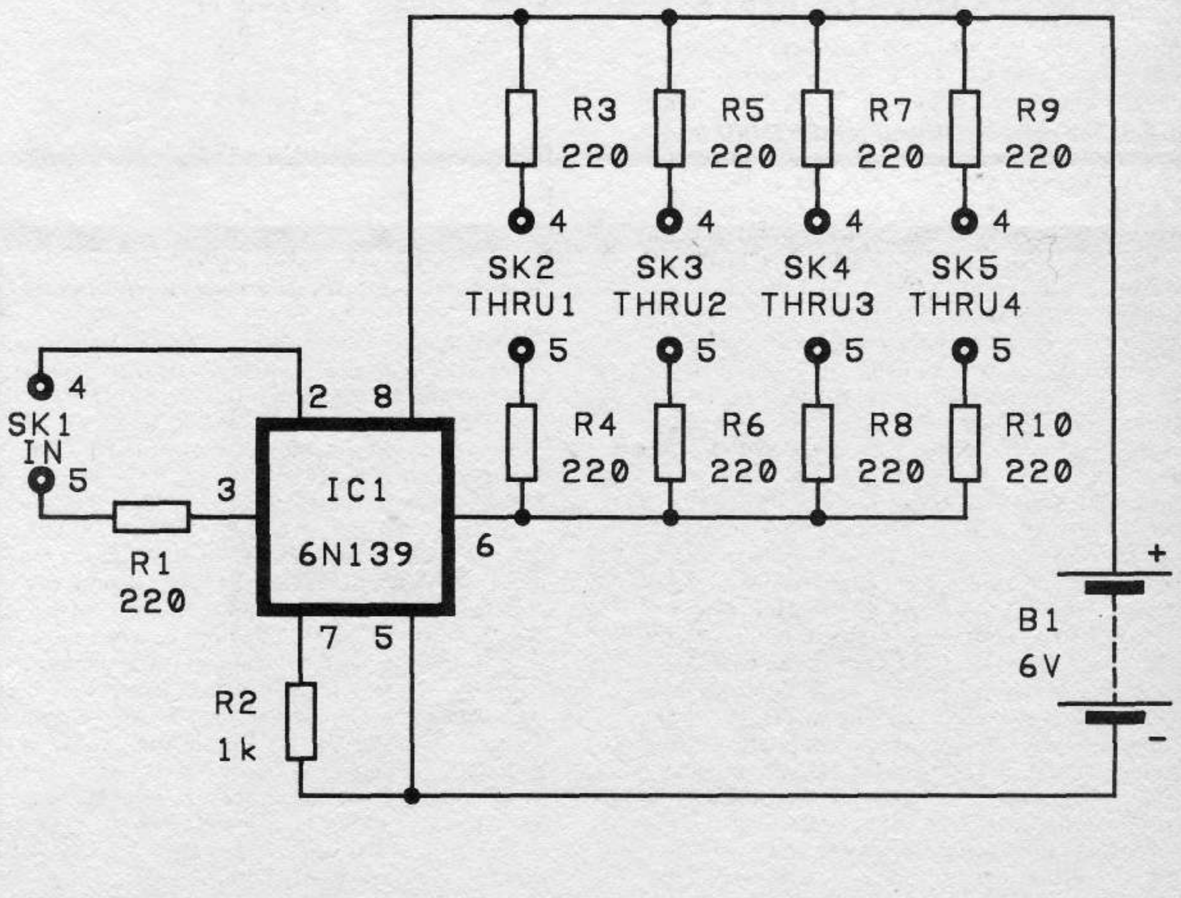


Fig. 6.4. The circuit diagram of the MIDI THRU Box.

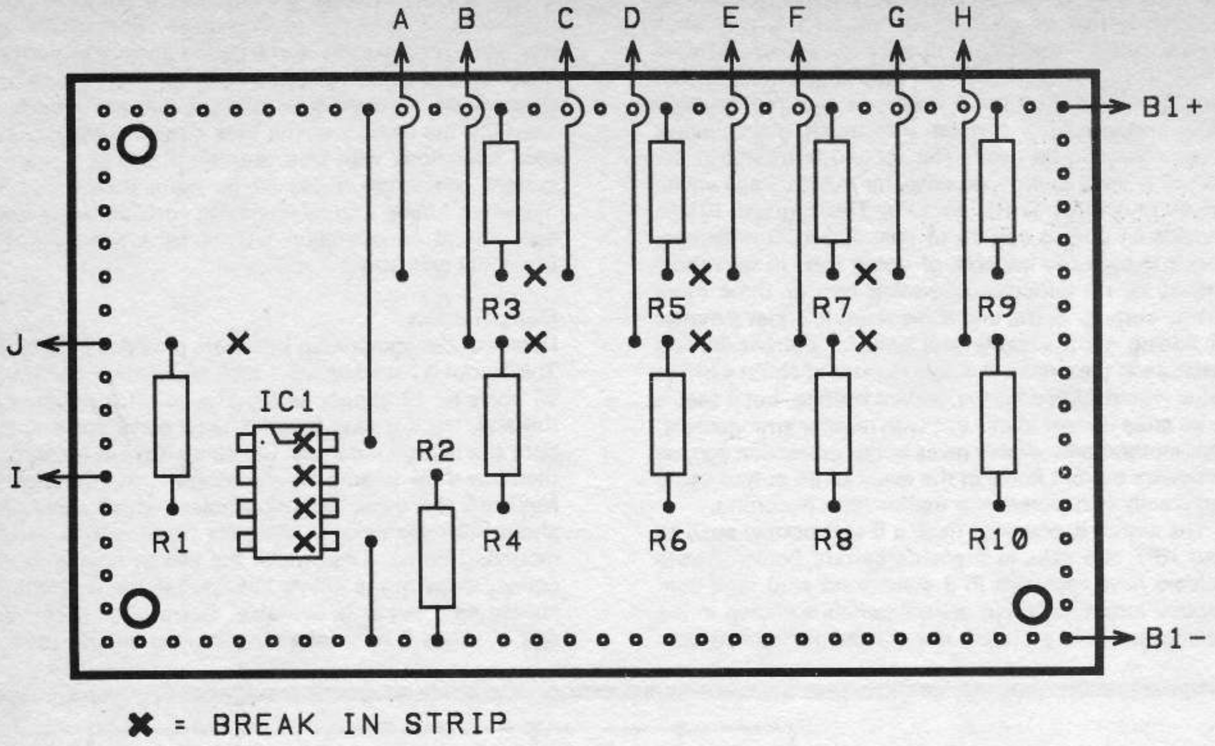
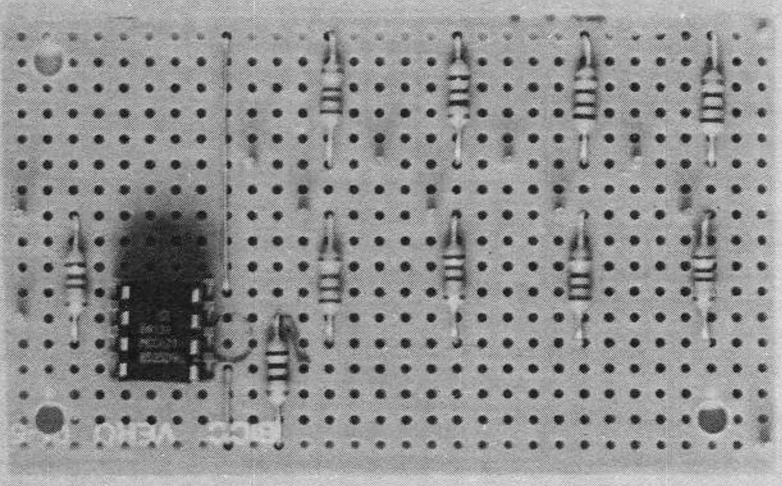


Fig. 6.5. The stripboard layout for the THRU Box.



The MIDI thru box stripboard.

quite well. Make sure you cut the strips properly, but be careful not to bore deeply into the board, which could seriously weaken it!

Fitting the components in place should not be difficult as the layout is reasonably well spread out. Start with the resistors, then add the link wires and IC1. Fit and connect the components one at a time. Preform the leads of the resistors so that they can be dropped into place on the board, doing some reforming if you do not get it quite right first time. The leadout wires are trimmed on the underside of the board using a small pair of wire clippers, and then the soldered joins are made. If you are new to electronics construction it would be as well to try soldering a few pieces of wire to an odd scrap of board before trying any construction in earnest. Any small electric soldering iron (of about 15 to 25 watts in rating) should be suitable, but it must be fitted with a miniature bit of around 1 to 3 millimetres in diameter. Keep the bit well tinned with solder (i.e. make sure the tip of the bit is kept covered with a coating of fresh, shiny solder).

Soldering is not difficult using modern solders and components, but it inevitably takes a certain amount of experience to become proficient at it. The correct solder to use is a multi-core flux type having a 60% tin and 40% lead content in the alloy. Avoid types which do not contain flux, or which are made from a 40% tin and 60% lead alloy. For small electronic work of this type 22 s.w.g. solder is usually easier to use than the thicker 18 s.w.g. type. The main point to keep in mind when making soldered joints is that the bit of the iron should be applied to the joint first, and then the solder should be fed into the joint. The solder should then flow over the end of the lead and the copper track to leave a neat mountain shaped joint. A rounded globular shape is usually indicative of a "dry" joint. If this happens it is best to remove the solder, clean the end of the leadout wire by scraping it with the blade of a penknife, and then try again.

The main enemy when using stripboard is the solder bridges that are easily produced between two adjacent copper strips. These are most likely to occur where there are a lot of joints close together, which mainly means where integrated circuits are fitted to a board. Check the finished board very carefully for these solder bridges, paying particular attention to the areas of the board where there are a lot of connections.

The link wires can be made from 20 to 24 s.w.g. tinned copper wire, and pieces of leadout wire trimmed from the resistors may well suffice here. IC1 is not a static sensitive device, but it is not a particularly cheap component either. I would therefore recommend that it is fitted on the board via an 8 pin DIL integrated circuit holder. At this stage 1 millimetre diameter printed circuit pins are fitted to the boards at the points where the connections to the off-board components will be made. Single-sided pins will do, and these are pushed into the holes from the copper side of the board. Solder them in place, and then generously tin the tops of the pins with solder.

Virtually any small metal or plastic case will accommodate the circuit board, but be careful to choose one that will also take the battery and leave sufficient space for the five sockets. The sockets are all of the standard MIDI (5 way 180 degree DIN) variety, and must be of the "chassis" mounting type (not printed circuit mounting sockets). They each require a main mounting hole of about 15 millimetres in diameter, plus two smaller holes of about 3.2 millimetres in diameter for the short M3 or 6BA mounting bolts. The positions of the smaller holes can be located, after the main mounting holes have been drilled, by using a socket as a sort of template. The component panel is bolted to the base panel of the case using M3 or 6BA bolts plus fixing nuts. It is advisable to use spacers over the mounting bolts in order to keep the component panel clear of the base panel. If the case is a metal type it is essential to use spacers as the connections on the underside of the board will otherwise be short circuited through the case. Even when using a plastic case it is still a good idea to use spacers, as without them there is a strong risk of the board becoming seriously distorted as the mounting nuts are tightened, and it could easily become badly cracked.

The wiring to the sockets is detailed in Fig.6.6, which operates in conjunction with Fig.6.5. Point "A" in one diagram connects to point "A" in the other, point "B" in one diagram connects to point "B" in the other, and so on. Use ordinary multi-strand p.v.c. insulated

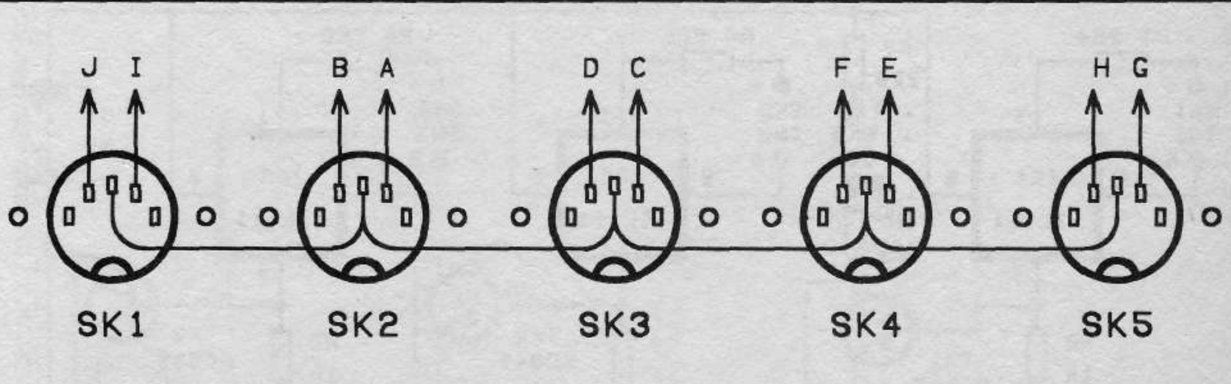


Fig. 6.6. The THRU Box wiring.

connecting wire for these interconnections (not the single core type). Making the connections should not be difficult provided the ends of the leads and the tags of the sockets are well tinned with solder first. Note that pin 2s of the sockets (the middle pins) are all wired together, as shown in Fig.6.6.

In use the unit is used in the standard "star" configuration. Its "IN" socket (SK1) is connected to the "OUT" socket of the ST, and its "THRU" sockets are then connected to the MIDI devices that are to be driven by the ST. Standard MIDI cables are used for all the interconnections. Of course, the unit is not only suitable for use with ST computers, and it should work perfectly well with any MIDI control unit.

Components (Fig.6.4)

Resistors (all 0.25 watt 5% carbon)

R1, R3 to R10	220 (9 off)
R2	1k

Semiconductor

IC1	6N139

Miscellaneous

B1	6 volt (i.e. 6 x HP7 cells in plastic holder)
SK1,2,3,4,5	5 way 180 degree DIN, chassis mounting (5 off)
	Case
	0.1 inch pitch stripboard 30 holes by 17 strips
	Battery connector
	Wire, solder, etc.

MIDI Mixer

The merits of MIDI merge and switch units were discussed in an earlier chapter. Just how useful a unit of this type will be depends on the precise setup you have in mind, and it might also depend on the software you are using with the ST. If the programme you are using has a "THRU" facility, it might be possible to arrange a system that is convenient to use but does not require a merge or switching unit. This is something that was covered previously, and it will not be covered again here. If you do wish to build up a system that requires a MIDI switcher, then this unit should fit the bill.

Strictly speaking it is not a MIDI merge unit or a MIDI switcher. It falls half way between these two types of equipment, and is what I suppose could be termed a MIDI "mixer". Its effect is analogous to an audio mixer, and it combines three MIDI input signals into a single output signal. However, whereas a mixed audio output may be perfectly usable, a mixed MIDI signal is not. Any receiving device would be unable to sort out one message from another, and the mixed signal would be totally unreadable. The idea of this unit is to give a sort of automatic switching action. Feeding an input signal to one of the three input sockets results in that signal appearing at the output socket. If you feed a signal to each input socket in turn, then each signal will appear, in turn, at the output of the unit.

The effect is much the same as if the unit automatically switched itself to direct each input signal through to the output. This is convenient in use as it enables various units to be used to control the system without the need for any manual switching. A certain amount of care needs to be exercised when using a unit of this type though. A true MIDI switching unit will only couple one input signal through to the output. If two or three input signals are present, only one will find its way through to the output, and there should be no malfunctions in the units receiving the output signal. With this mixer unit, two or more signals received simultaneously will almost certainly result in a "scrambled" output signal, and its effect on the controlled units would then be unpredictable.

The Circuit

The full circuit diagram of the MIDI mixer unit appears in Fig.6.7. It consists basically of three opto-isolator circuits with their open collector outputs wired together. Switching on any one of the output transistors therefore produces a 5 milliamp loop current to flow, and gives the required mixing action. R7 and R8 are the output current limiting resistors, and are common to all three output stages.

It would be possible to base the circuit on three

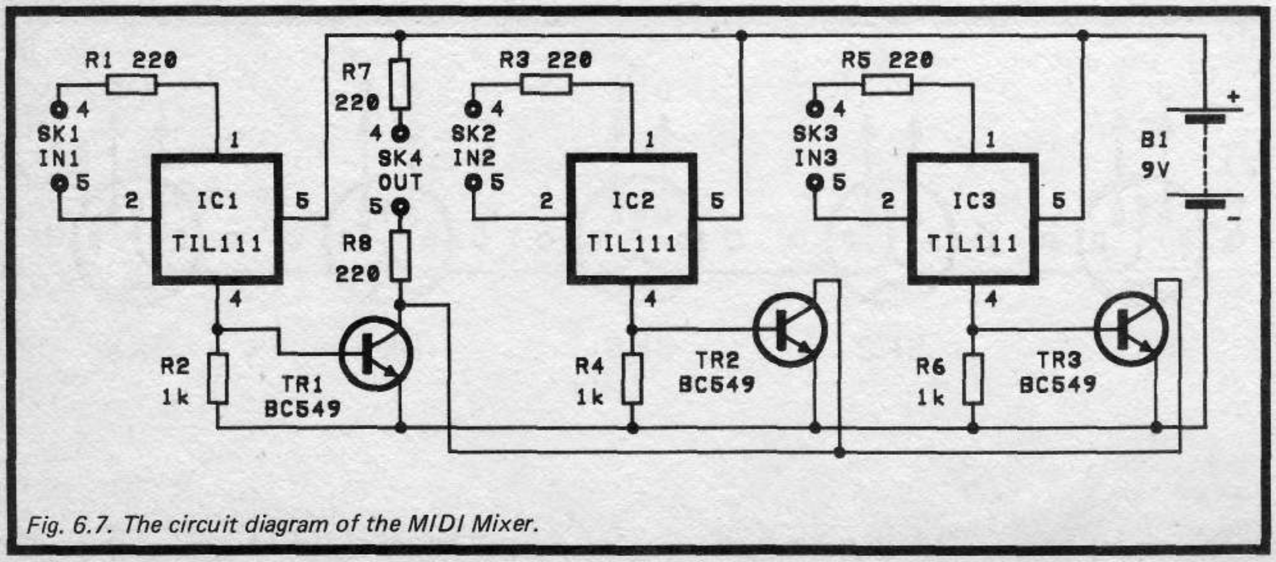


Fig. 6.7. The circuit diagram of the MIDI Mixer.

6N139 (or similar) opto-isolators, but this would be a rather expensive solution to the problem. The "budget" solution used here is to base the circuit on three inexpensive opto-isolators such as the TIL111 (4N27 and MC72P opto-isolators seemed to work just as well). As explained previously, inexpensive opto-isolators such as these have neither the speed or the efficiency for MIDI applications, but both deficiencies can be counteracted by using an external switching transistor. The transistor in each opto-isolator is used as an emitter follower stage which directly drives a discrete switching transistor. For instance, TR1 is the switching transistor for IC1. An emitter load resistor for each internal transistor ensures they operate at a realistic currents, and this aids fast switching. The fact that they are operating in the emitter follower mode also aids rapid switching. The external transistors provide a large amount of current gain which compensates for the lack of efficiency in the opto-isolators.

In terms of performance this arrangement does not equal that of a 6N139, which seems well able to handle baud rates in excess of ten times the MIDI baud rate! However, performance seems to be perfectly adequate for this application where only one output must be driven. I tried a number of TIL111 and similar devices in the circuit, and none failed to work properly. It is just possible that a very low efficiency device would fail to operate in this circuit, but even if you buy an extra opto-isolator to allow for the possibility of one not functioning reliably in the unit, the cost of the semiconductors for this project will almost certainly still be little more than that of a single 6N139.

Like the previous circuit, the current consumption under quiescent conditions is so low as to be of no significance. An on/off switch is therefore unnecessary. The current consumption when the unit is fed with an input signal depends on the density of the data stream, but is not more than about 2.5 milliamps, and is typically well under 1 milliamp. A medium capacity 6 volt battery (such as 4 HP7 size cells in a plastic holder) should provide many hundreds of hours of operation.

Construction

Details of the component layout for the 0.1 inch matrix stripboard are shown in Fig.6.8, while the wiring to the input and output sockets is illustrated in Fig.6.9. The board measures 38 holes by 17 copper strips.

Detailed constructional notes will not be provided for this project, as they would largely be repeating the constructional notes for the THRU box described previously. One point of difference is that the opto-isolators used in this project are much cheaper than those used in the MIDI THRU box, and you might not consider it worthwhile using sockets for them. You might find it difficult to obtain suitable sockets for them anyway, as their 6 pin DIL encapsulation is an unusual type. If you do decide to use sockets (and I always use them for any DIL device), 6 pin DIL types are stocked by one or two component retailers, or it is not too difficult to trim down an 8 pin DIL type to the required size. Alternatively, you could fit 8 pin DIL sockets on to the board and just ignore the two terminals of each one that are not required.

The unit is wired into the system using standard MIDI leads, and to test it you merely need to feed the inputs from three MIDI control units (the ST, MIDI instruments, or whatever) and connect the output to a MIDI instrument. Operating any one of the control units should result in the instrument responding to its signal in the normal way. Of course, if you only require two inputs, it is quite in order to leave one of the input sockets unconnected (or you can omit one of the input sockets and the associated opto-isolator circuit).

Components (Fig.6.7)

Resistors (all 0.25 watt 5% carbon)

R1	220	R7	220
R2	1k	R8	220
R3	220		
R4	1k		
R5	220		
R6	1k		

Semiconductors

IC1,2,3	TIL111 (3 off)
TR1,2,3	BC549 (3 off)

Miscellaneous

B1	6V (e.g. 4 x HP7 in plastic holder)
SK1,2,3,4	5 way 180 degree DIN, chassis mounting (4 off)
	0.1 inch matrix stripboard 38 holes by 17 strips
	6 pin DIL i.c. holder (3 off – see text)
	Case
	Battery connector
	Wire, solder, etc.

MIDI Switcher

For some systems a simple MIDI switcher might be preferable to the mixer unit described above. Although a true switching unit has the disadvantage of requiring manual adjustment to select the desired input, it does have some possible advantages. One of these is simply that it does not require a battery as it is purely a passive device. It consists of no more than a few switches and sockets. Another, is that only one input at a time is connected through to the output. Inadvertently supplying the unit with more than two input signals at once will not produce corrupted data at the output. A third advantage is that the unit is bidirectional. In other words, it can be used to select one of several input sources and connect it through to an instrument or series of instruments), as in Fig.6.10, or to feed a single source to one of several destinations, as in Fig.6.11. I do no know if this second method of use has any widespread practical application, but it is there if you should need it!

Fig.6.12 shows the basic circuit for a four in/one out or four out/one in MIDI switcher. This consists of just the five input/output sockets plus a 4 way 2 pole rotary switch. The switch connects the required input through to the output, and the only point to watch is that no crossed wires are produced. The wiring diagram of Fig.6.13 should help with the avoidance of any errors of this type. It is assumed in the wiring diagram that the switch is a standard 6 way 2 pole rotary type with the end-stop set for four way operation.

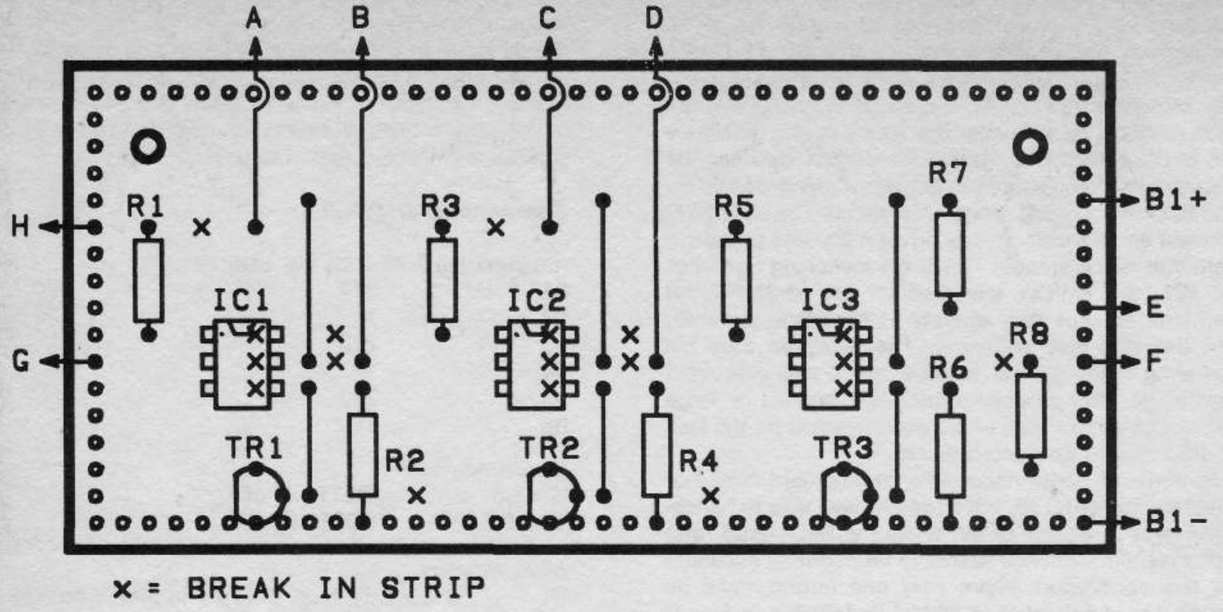
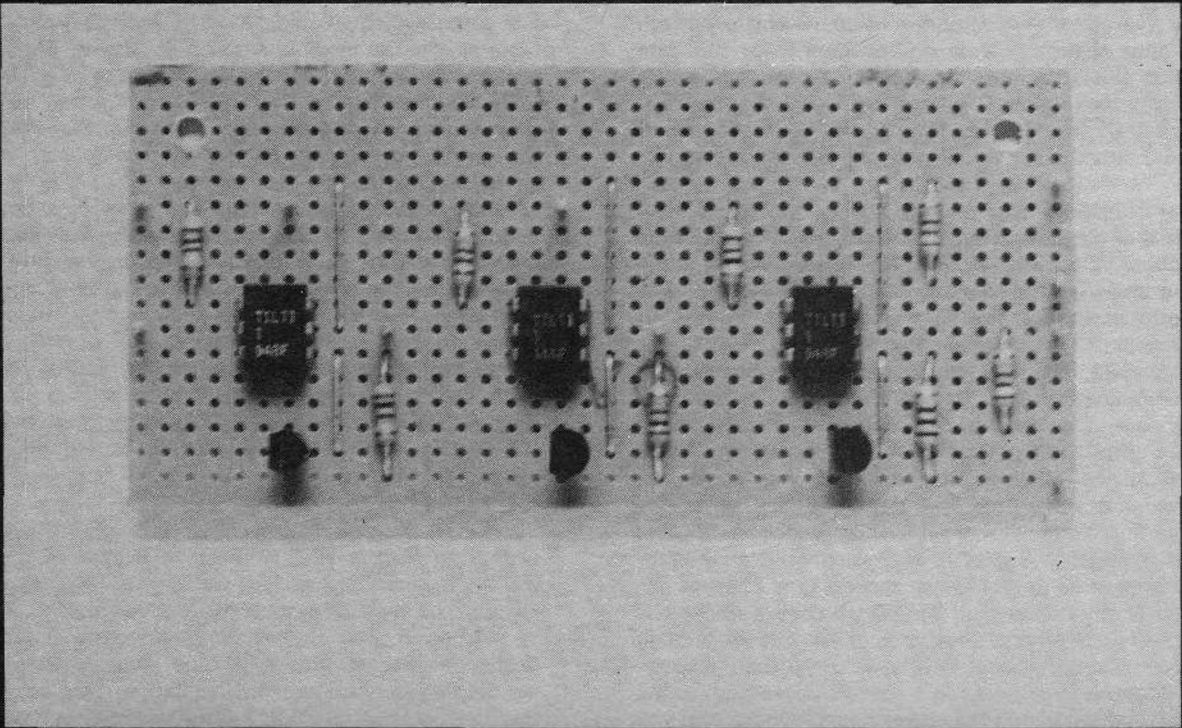


Fig. 6.8. The stripboard layout for the MIDI Mixer.



The MIDI mixer stripboard.

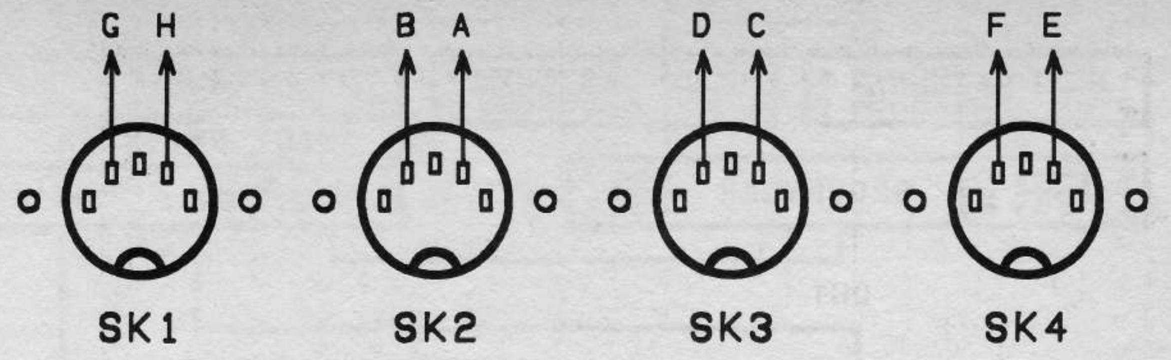


Fig. 6.9. The MIDI Mixer wiring.

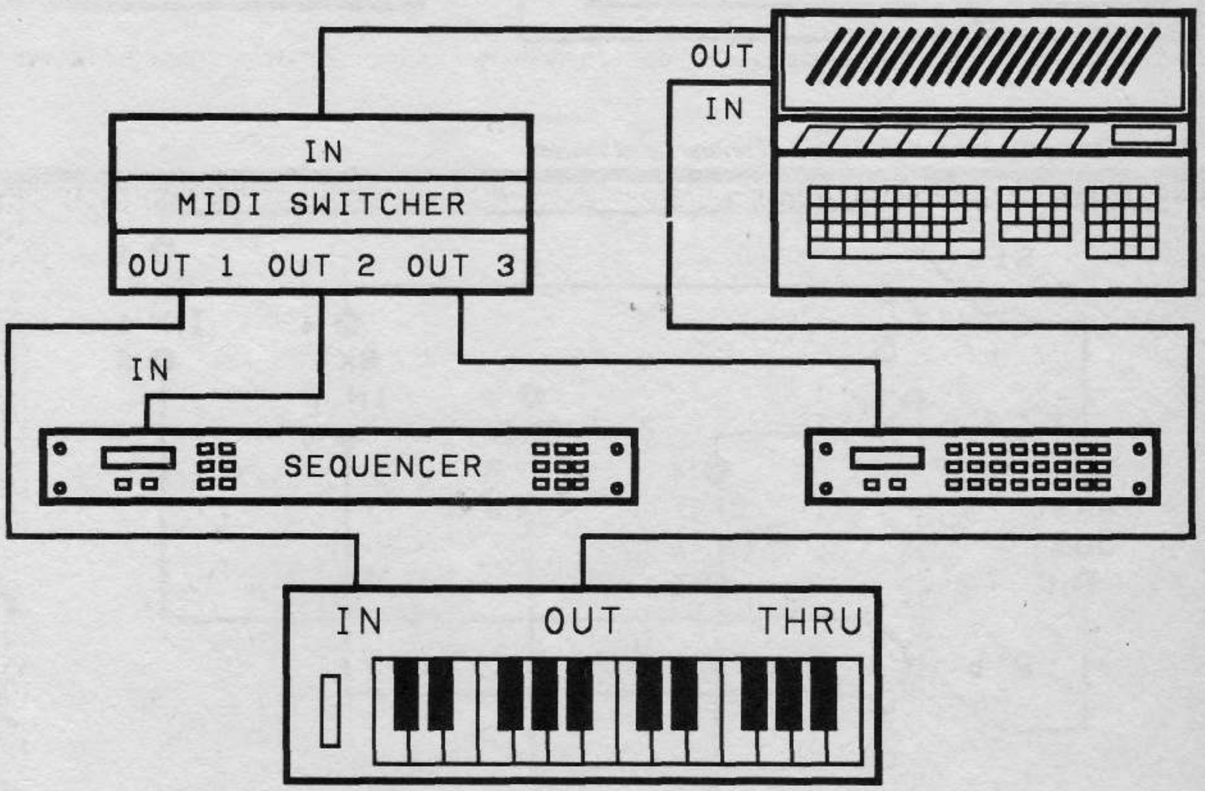


Fig. 6.10. Feeding a MIDI signal to one of several destinations.

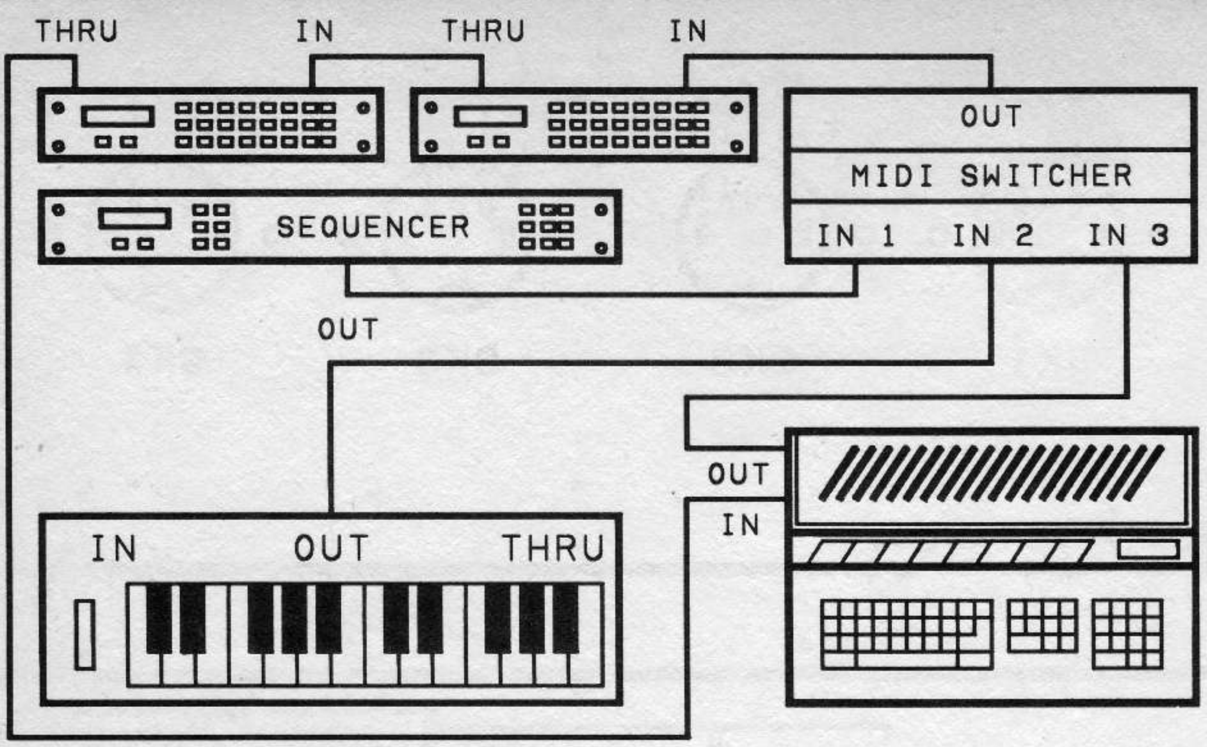


Fig. 6.11. Using a switcher to select one of several signal sources.

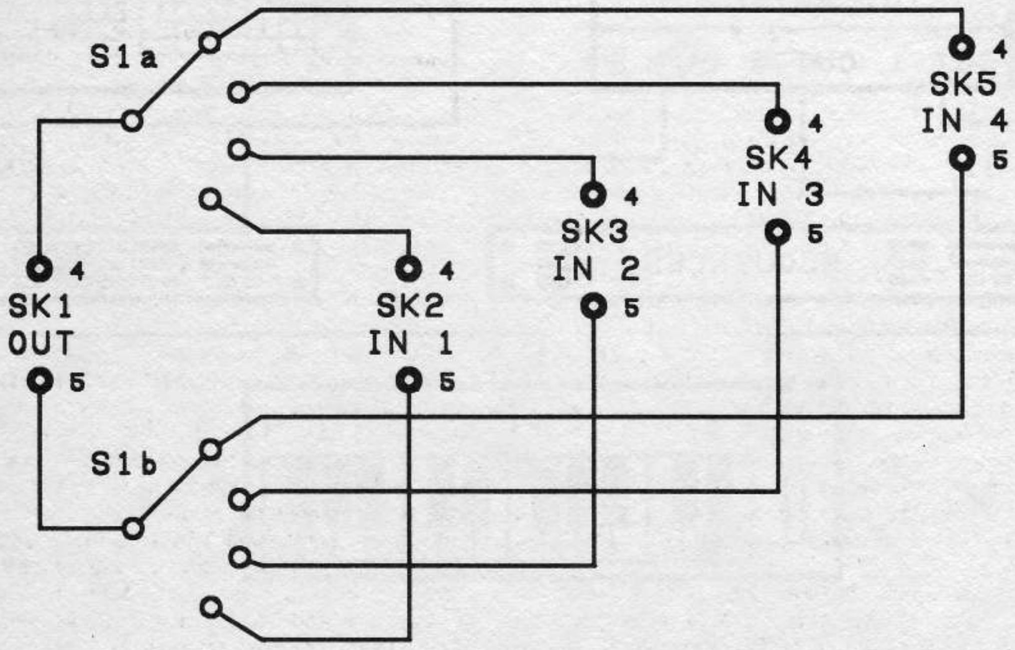


Fig. 6.12. The circuit diagram of the MIDI 4-1 switcher.

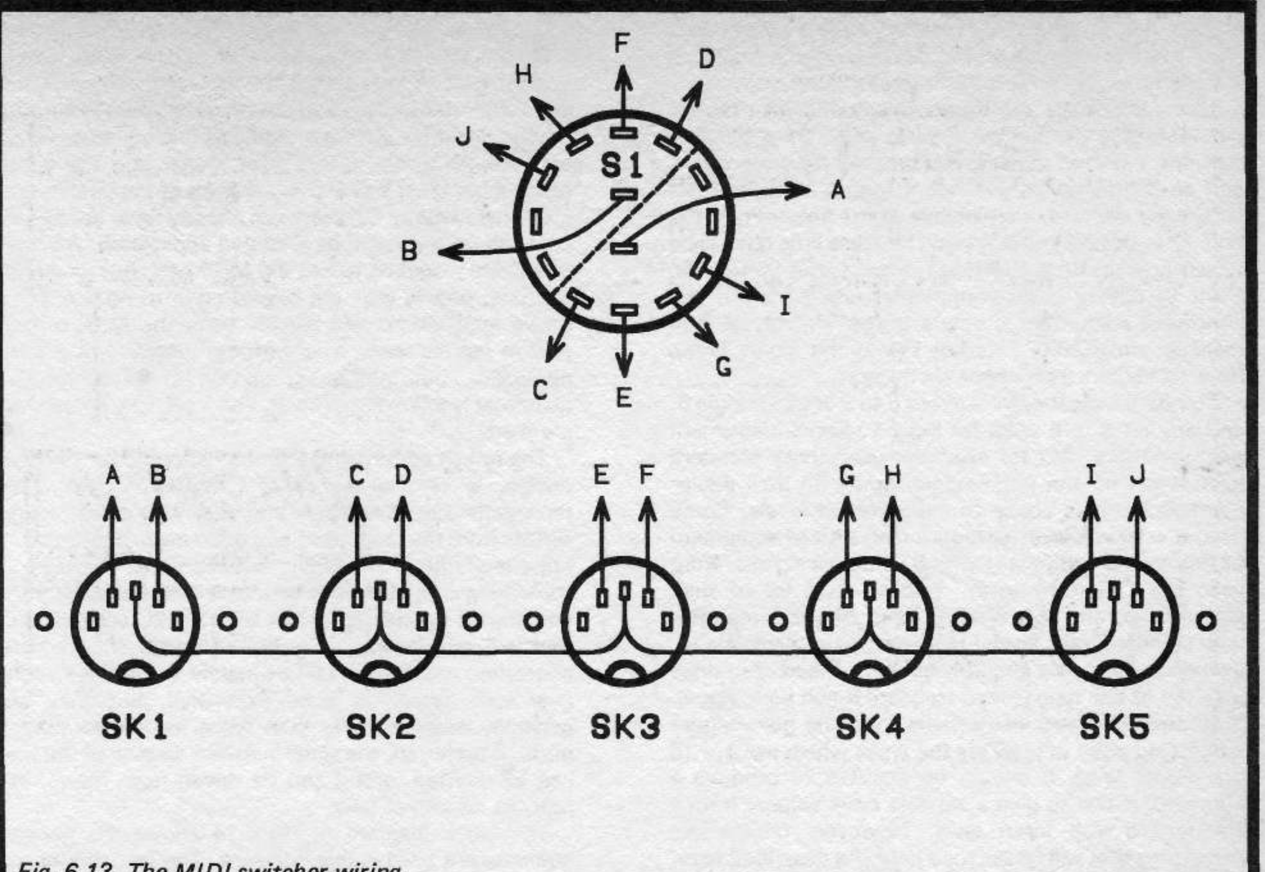


Fig. 6.13. The MIDI switcher wiring.

CV/Gate Controller
Much of this book has been concerned with MIDI, which is not surprising when one considers its dominant role in modern electronic music systems. There are still a lot of pre-MIDI instruments in existence though, and many of these are suitable for computer control. These are the synthesisers that use the gate and CV (control voltage) method of control. If you try using an ST computer with this type of instrument, you need to bear in mind that you are likely to run into the problems of non-standardisation that MIDI was designed to overcome. The gate/CV interface described here is only suitable for synthesisers that have a logarithmic control voltage characteristic. In other words, instruments of the "standard" 1 volt per octave type. With these, if middle C is obtained at a control voltage of 4 volts, the C an octave higher would be obtained at 5 volts, and the one an octave above that would be obtained at 6 volts. The Cs one and two octaves below middle C would be produced by control potentials of 3 and 2 volts respectively.

One could reasonably consider this to be a linear characteristic, with the pitch of the synthesiser increasing by one volt per octave, or 83.33 millivolts (0.08333 volts) per semitone if you prefer. The control voltage to musical interval relationship is indeed a linear type, but the control voltage to frequency relationship is not. Each 1 volt increment in the control voltage produces a doubling of frequency, which is a very non-linear characteristic. This is not something of purely academic importance, and there have been some synthesisers produced that have a linear control voltage/pitch characteristic. In other words, with middle C produced by an input of 4 volts, raising the pitch by one and two octaves would require control voltages of 8 and 16 volts respectively. The Cs one and two octaves below middle C would require control voltages of 2 and 1 volt respectively. The one volt per octave characteristic lends itself well to computer control, but a linear characteristic does not. This interface is totally unsuitable for use with synthesisers that have linear control characteristics.

There can also be compatibility problems with the gate or trigger input. Most synthesisers have a proper gate input, and the gate pulse has much the same effect as holding down a key of the keyboard for the same duration. A trigger input is somewhat simpler, and is usually present on synthesisers that have a basic attack/decay style envelope shaper. Here the duration of the pulse has no effect on the sound produced, and it is just a matter of producing a pulse that is not below a certain (and usually very short) duration. The degree of control provided via the trigger input is not inferior to that provided by the keyboard, since how hard a key is struck and how long it is pressed has no effect on the sound either! It is only fair to point out that there is some confusion over the terms "trigger" and "gate", and you might need to consult the manual for an instrument to determine if whether it has a true gate input or a simple trigger type. If an instrument has an

ADSR (attack-decay-sustain-release) type envelope shaper, then it will almost certainly have a proper gate input. If it has a simple attack/decay envelope shaper, it will probably just have a basic trigger input.

There is usually no major incompatibility problem between gate and trigger inputs, apart from the fact that one provides more limited control than the other, and an instrument which has a trigger input can not accurately mimic one which has a gate input. From the computer control point of view, it makes little difference which type is being controlled, except that there is no point in having a programmable gate period if the controlled instrument is only a trigger input type. The main incompatibility problem lies in the signal levels used by various instrument producers.

Standard logic levels are about 0 to 2 volts for logic 0, and around 3 to 5 volts for logic 1. Some instrument manufacturers (SCI for example) used these standard logic levels for the gate/trigger inputs on their equipment, but others chose to use different levels. Some used a +15 volt level for logic 1, but a lot of equipment of this type is compatible with 5 volt logic signals. Korg used the "short to earth" system on a lot of their instruments, and this type of input is not compatible with standard logic levels. However, as explained later, an extremely simple circuit is all that is needed to drive an input of this type from a standard 5 volt logic signal. The most awkward instruments from the gate/trigger interfacing point of view are the types which use a −15 volt signal level. It should be possible to produce a converter circuit to give a suitable drive voltage from a standard 5 volt input level. However, this is not something that will be pursued here. As described here, the gate/CV interface is only suitable for use with instruments that have a 1 volt per octave (logarithmic) control voltage characteristic, and a 5 volt logic compatible gate or trigger input. Instruments which have a "short to ground" type gate input should work properly with this interface provided the simple add-on circuit is used at the gate output of the unit.

Which Port?

There are several ports to choose from on the ST when it comes to selecting which one to use for a simple add-on unit of this type. The obvious choice is the parallel printer port which can provide latching outputs that are ideal for driving a digital to analogue converter and providing the gate/trigger signal. On the other hand, a printer is a very popular form of peripheral, and presumably many ST users will already have the printer port tied up for use in its intended application. Another possibility would be to use the MIDI port, but presumably most people who are interested in using the ST in music applications will already have the MIDI output port in use as such. The cartridge port offers a third possibility, but interfacing on to the buses of the computer is relatively difficult, and a bit risky if mistakes are made.

The fourth option, and the one I adopted for the final design, is to use the serial ("modem") port. This represents a relatively safe and easy way of extracting signals from the computer, and most users will probably not already have something connected to this port. If something should already be in use with the serial port, you should be able to obtain an RS232C selector unit that will enable you to switch between two devices connected to this port. Unfortunately, a suitable switch-over unit might be quite expensive, but they are generally less expensive than types for parallel printer ports. If preferred, the serial interface section of the unit can be omitted, and it can be driven from the printer port, as described later.

The block diagram of Fig.6.14 shows the general arrangement used in this interface. The first task of the interface is to convert the incoming serial data back into parallel form. There are a number of special integrated circuits that are designed specifically for serial interfacing, but many of these are only suitable for use with microprocessor based circuits as they are designed for software control. For the present application a UART (universal asynchronous receiver/transmitter) is the only form of serial interface chip that is suitable. These

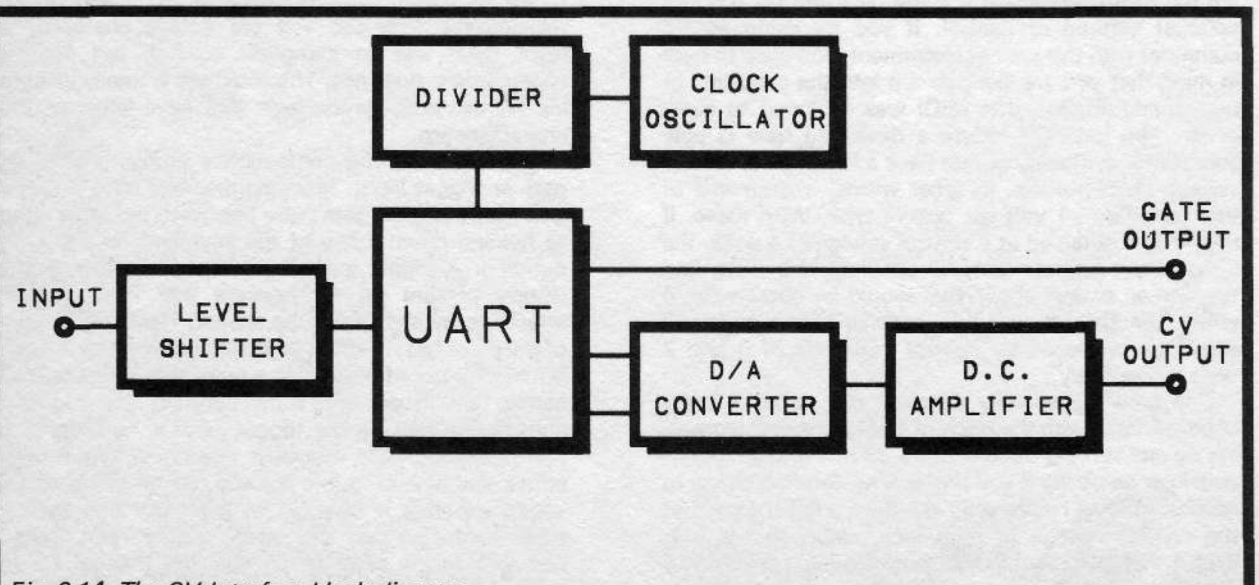


Fig. 6.14. The CV Interface block diagram.

are controlled via input terminals which can be driven from the data bus of a computer, or simply wired to the appropriate supply rail. They are therefore suitable for both software and hardware control. UARTs also have tristate outputs which can drive the data bus of a microprocessor circuit, or be used to drive indicator l.e.d.s (or just left unused) in non-micro based designs.

This unit is based on an industry standard UART, and the serial input signal is fed to this via a level shifter stage. The standard RS232C serial output signal of the ST computer's "modem" port is at nominal levels of plus and minus 12 volts, but the UART requires an input signal at ordinary 5 volt logic levels. The level shifter provides the necessary changes in voltage levels, and it also provides an inversion of the signal. This inversion is needed in order to give a signal of the right polarity for the UART.

The baud rate at which the interface operates is controlled by a crystal controlled clock oscillator and a frequency divider circuit. The oscillator operates at 2.4576MHz, and the UART requires a clock signal at sixteen times the required baud rate. The ST will operate at baud rates of up to 19200 baud, and it is advisable to use the maximum baud rate in order to minimise the time taken for each message to be sent to the interface. A little mathematics will show that a divide by eight action through the divider stage gives the required rate of 19200 baud. The divider can provide other division rates, and it can also accommodate baud rates of 9600, 4800, 2400, 1200. If you would prefer to use a lower baud rate for some reason, there should be no difficulty in doing so.

One of the UART's eight latching outputs (the most significant output) is used to provide the gate output signal. The other seven outputs drive a linear digital to analogue converter. This converter is actually an eight bit type, and normally gives an output equal to 10 millivolts (0.01 volts) per least significant bit. In this case the least significant input is simply connected to the 0 volt supply rail, and the other seven inputs are driven from the seven least significant outputs of the UART. This effectively downgrades the converter to a seven bit type having a resolution of 20 millivolts.

It gives 128 different output voltages, and can therefore provide a range of notes which is comparable to that provided by the MIDI system. Obviously not all instruments can handle such a wide compass, but most analogue synthesisers seem to cover a very wide pitch range via their CV inputs. In fact many seem to cover a far wider compass via their CV input than can be covered using the keyboard (even allowing for any octave up/down switch that might be included).

With increments of only 20 millivolts at the output of the converter, some amplification is needed in order to boost this to the required 83.33 millivolts per semitone of the 1 volt per octave CV system. This requires a simple d.c. amplifier circuit which has adjustable voltage gain, so that the output voltage range can be adjusted to precisely the correct one.

The Circuit

Fig.6.15 shows the full circuit diagram for the gate/CV interface. The clock oscillator uses TR1 in a popular configuration, and this circuit has an output of sufficient amplitude to drive the next stage without the need for any buffering. This next stage is a CMOS 4024BE seven stage binary divider. Here we are using the third output in order to obtain a divide by eight action, and stages four to seven of the device are left unused. However, you can bring more stages into play if you require a lower baud rate. This table shows the standard baud rates that are obtainable, and the pin of IC1 that should be used in order to obtain each one.

Baud Rate	*Pin Of IC1*
1200	3
2400	4
4800	5
9600	6
19200	9

IC2 is the UART, and this is the 6402 industry standard device. R3 and C4 provide a positive reset pulse at switch-on, and the control inputs of the device are connected in such a way as to give a word format of one start bit, eight data bits, one stop bit, and no parity. A word format which has eight data bits is essential, as we are using all eight outputs of the UART. This word format is one which is supported by the "modem" port of ST computers incidentally. The most significant output of the UART directly drives the gate output socket (SK1), and there should be no need for any buffering here.

The other seven outputs drive the digital to analogue converter chip, which is IC3. This is a Ferranti ZN426E, which is relatively inexpensive and easy to use, but has quite a high standard of performance. It has a built-in high quality 2.55 volt reference source, and this sets the full scale output voltage at the same figure. R4 and C5 are the discrete load resistor and decoupling capacitor for the converter, and these are the only discrete components that IC3 requires.

IC4 is the d.c. amplifier, and this is almost a standard operational amplifier non-inverting type. It differs from the standard form of this configuration only in that IC4 is operating without a negative supply rail. This is possible because the CA3140E used in the IC4 position is a device that can operate with its output at voltages right down to the 0 volt supply potential, even in the absence of a negative supply. Most other devices (including the standard uA741C type) are unable to do this, and will not operate in this circuit unless they are provided with dual supply rails. VR2 is used to adjust the voltage gain of the circuit to precisely the required level, and VR1 is an offset null control. In theory the circuit should work perfectly well at low output voltages, but in practice operational amplifiers tend to be plagued by small offset voltages that cause errors in the output level. These are often unimportant, but would give unusable results in a critical application of this type. VR1 is used to trim these offsets to an insignificant level.

The level shifter/inverter stage is based on TR2, and this is just a simple common emitter switching stage. The main circuit requires a reasonably stable 5 volt supply, but IC4 requires a higher voltage if it is to provide a reasonably wide output voltage range. With the suggested method of powering the circuit, a 9 volt battery acts as the power source, but the main circuit is powered via 5 volt monolithic voltage regulator IC5. IC4

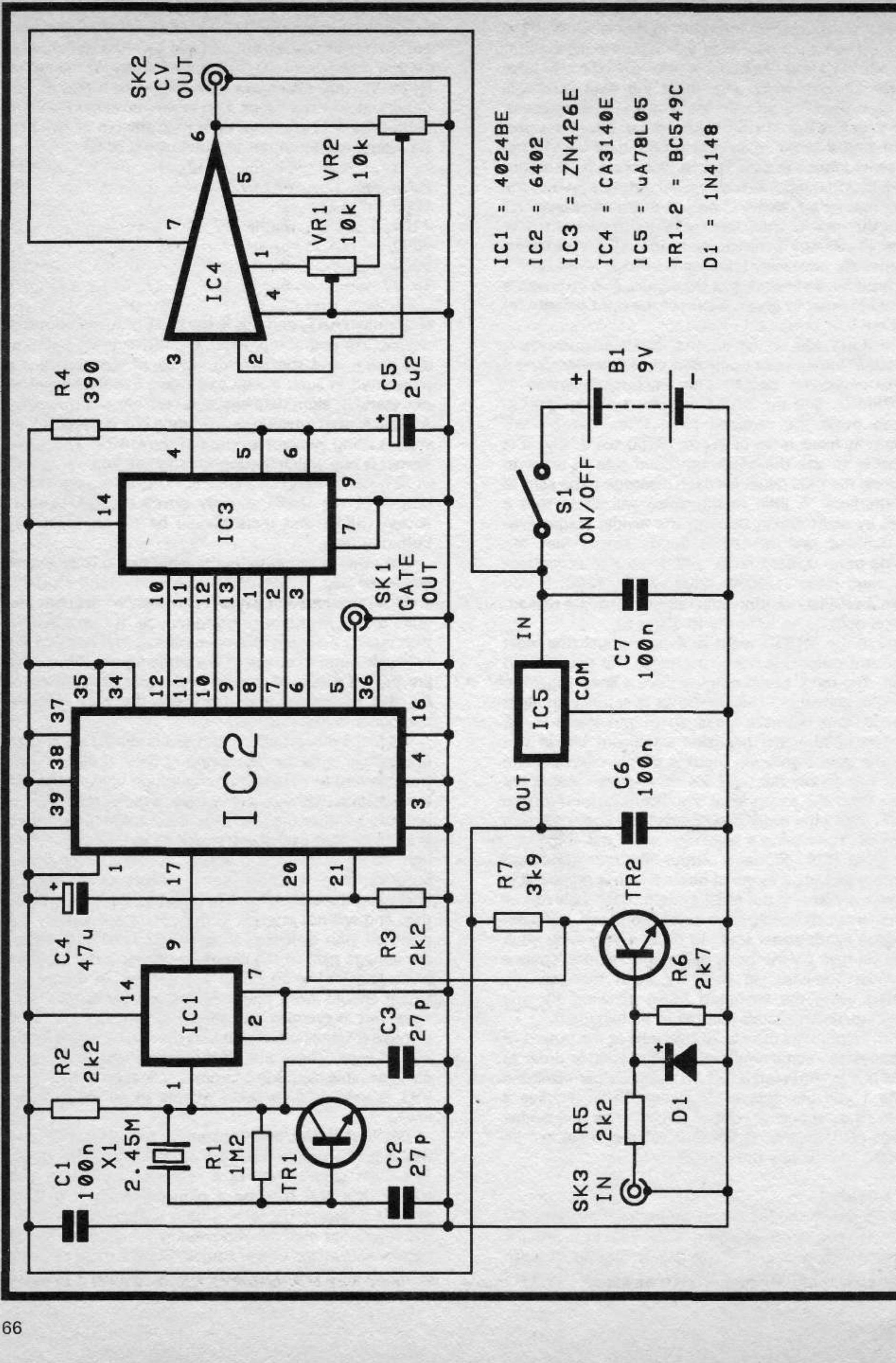


Fig. 6.15. The full circuit diagram of the CV Interface.

is powered direct from the 9 volt battery. You should note that the maximum output voltage of IC4 is about 2 volts less than its supply voltage, or about 7 volts in this case. This is an over-simplification in that the battery voltage will actually start at about 9.5 volts, and will drop to around 8 volts as it nears exhaustion. About 6 volts therefore has to be regarded as the maximum output level that can be reliably achieved by the circuit. This gives a range of 6 octaves, and should be sufficient for most purposes. However, if a higher maximum output voltage is required, it is merely necessary to power the unit from two 9 volt batteries connected in series (giving an 18 volt supply). The unit should then give the full coverage of over ten octaves.

Although many serial devices have very high current consumptions, this is not the case for the 6402 which is a low power CMOS device. The total current consumption of the circuit is only about 8 to 9 milliamps (very little of which is consumed by IC2), and any 9 volt battery should be able to supply this economically.

Parallel Port

If you would prefer to drive the unit from the printer port, this is not difficult to do and the circuit can be greatly simplified. The following components can be omitted: –

IC1, IC2, TR1, D1, C1, C2, C3, C4, R1, R2, R3, R5, R6, R7, X1, and SK3

IC3 is then driven from the parallel port in the manner shown in the "skeleton" circuit of Fig.6.16. The seven least significant lines of the printer port drive IC3, while the most significant line is used to provide the gate signal. In other words, the parallel interface chip of the ST is used in exactly the same way as the parallel output of IC2. Fig.6.17 provides details of the connections to the printer port of the ST. Note that you need a 25 way D *plug* to make the connections to the printer port, but a 25 way D *socket* to make the connections to the "modem" port.

If you use this method of driving the interface it is probably best to write data direct to the parallel interface chip rather than to use the normal parallel printer port commands. This is simply because the gate/CV interface does not require any handshaking lines to be implemented, since it can keep up with the data flow from the computer. Writing data to the printer port in the usual fashion could cause problems as the computer's operating system will be looking for responses from the handshake lines in the normal way, but the gate/CV interface has no outputs to drive the handshake lines. This seems to result in the computer hanging up indefinitely. Using direct POKEs (or whatever) to the parallel interface chip overcomes this problem, and makes programming of the interface quite straightforward.

The parallel printer port's data lines are provided by port B of the sound chip (refer to chapter 1 for details of

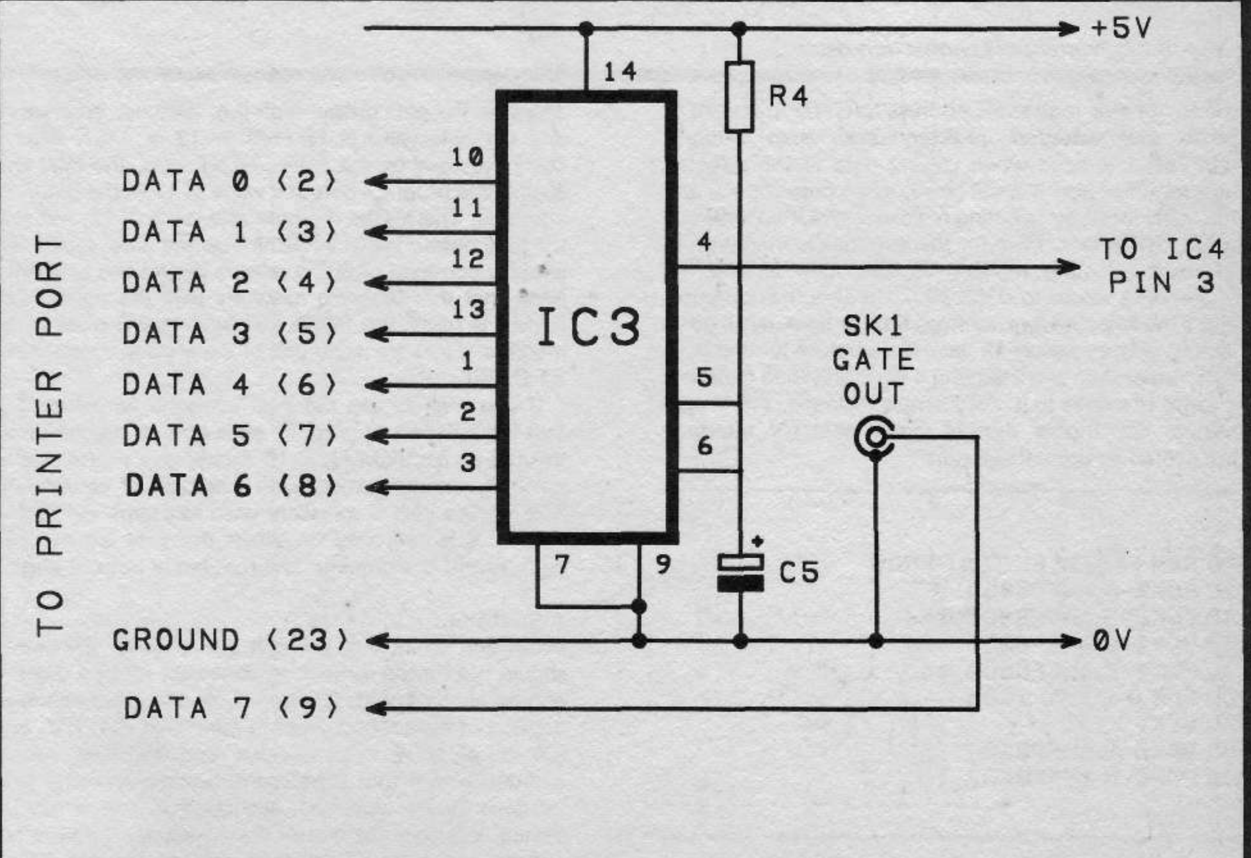


Fig. 6.16. Driving the interface from the parallel port.

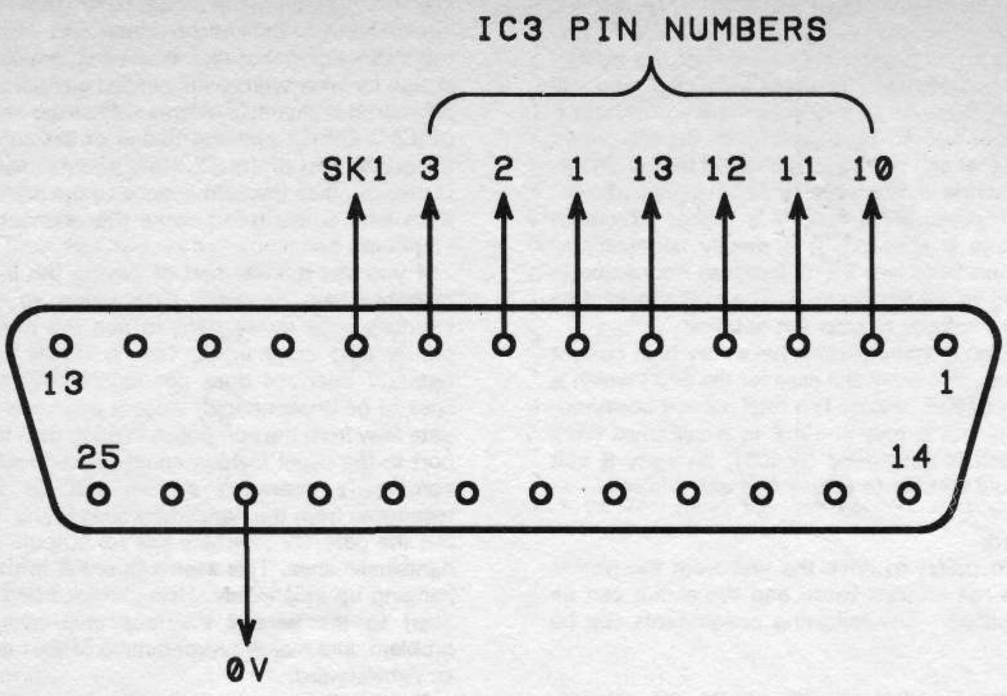


Fig. 6.17. Printer port connection details.

the sound chip registers). Address &HFF8800 is used to select the required register, and then address &HFF8802 is used when writing data to the selected register. First port B must be set as an output port, and this is achieved by selecting register 7 and then writing a value of 128 to it. Data for the interface is then written to it by selecting register 15 and then writing the appropriate values to &HFF8802. In case the computer alters the selected register from time to time, it is a good idea to select register 15 each time before writing data to it, rather than just selecting it once and then POKEing a series of values to it. As a simple example, this simple routine will trigger a note via a gate/CV interface connected to the parallel port.

```
10 REM PARALLEL TEST PROG
20 POKE-B &HFF8800,7
30 POKE-B &HFF8802,128
40 POKE-B &HFF8800,15
50 POKE-B &HFF8802,140
60 FOR D = 1 TO 1000
70 NEXT
80 POKE-B &HFF8800,15
90 POKE-B &HFF8802,12
```

This merely sets port B as an output port, and then writes an initial value of 140 to the gate/CV interface. This sets the gate output high (i.e. switches on a note), and the note value is 12 (128 + 12 = 140). After a delay provided by the FOR...NEXT loop, the final two lines of the program output a value of 12 to the gate/CV interface. This leaves the note at a value of 12, but sets the gate output low (i.e. terminates the note, which will actually continue until the release period has expired). Note that this program assumes that the current ST BASIC is used; the POKE instructions will need to be modified if you are using one of the previous versions of ST BASIC.

If you wish to use the interface with an instrument that has a "short to ground" style gate or trigger input, the add-on circuit of Fig.6.18 should give a gate output pulse that is compatible with this type of equipment. This is just a VMOS transistor used as a common source switch. It is switched on when the gate signal goes high, giving the required low resistance path to earth.

Adjustment

Whichever version of the unit you choose, the circuit should not be too difficult to construct using a custom printed circuit board, stripboard, or any other standard method of production. Bear in mind that IC1, IC2, and IC4 are all MOS input devices, and that they consequently require that standard anti-static handling precautions to be observed. Although IC3 is a bipolar device, it is also not one of the cheapest of integrated circuits, and I would also recommend that this device should be fitted in a DIL i.c. holder. Any 2.4576MHz crystal should suffice for X1, but a type in a small

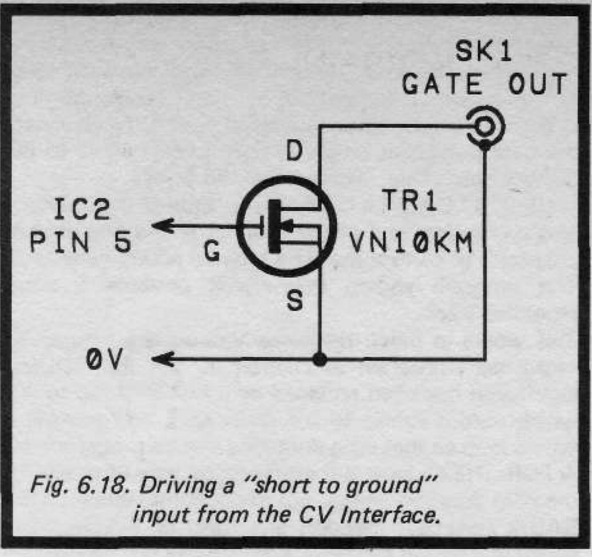


Fig. 6.18. Driving a "short to ground" input from the CV Interface.

wire-ended (HC-18/U) style case is probably the most convenient from the constructional point of view.

Assuming the interface is driven from the serial port (Fig.6.19), and that you will use it with a programme written in ST BASIC, data is written to the device using the OUT instruction. The "modem" port is device 1, and data is therefore sent to the interface using the instruction: –

```
OUT 1,x
```

where "x" is the value that is to be written to the interface. Note that the serial interface must be set up for the correct baud rate and word format before any data is written to the interface. As far as I can gather, there are no ST BASIC commands which give control of the baud rate and word format. This does not matter, as all you have to do is use the GEM desktop to set the required parameters as the serial port defaults. According to the "Atari ST Owner's Manual" there is a "Set RS232 Config." option obtained via the "Desk" menu bar. On booting my 520STFM with the "Language Disc" in the drive, this option proved to be absent. However, there is a "Control Panel" option, and the RS232 setup menu can be obtained as a sub-menu of this. The correct parameters to select are listed below: –

Parameter	*Setting*
Baud Rate	19200
Parity	None
Duplex	Full
Bits/Char	8
Strip Bit	Off
Xon/Xoff	Off
Rts/Cts	Off

As an initial test of the interface, try repeatedly writing values of 0 and 128 to it (i.e. OUT 1,0 and OUT 1,128). If all is well, this should set the gate output low and high respectively. If the gate output does not respond, switch off at once and thoroughly recheck everything. Assuming all goes well, the two preset resistors must be set up correctly before the interface is ready for use. This is just a matter of first sending a fairly high note value to the unit, and adjusting VR2 for the correct note from the synthesiser. Then send a low note value to the interface, and adjust VR1 for the correct note from the synthesiser. Repeat this procedure a few times until correct tracking between the two notes is obtained. Most synthesisers have the ability to switch between the keyboard and external control, and this can be useful for comparison purposes when making these

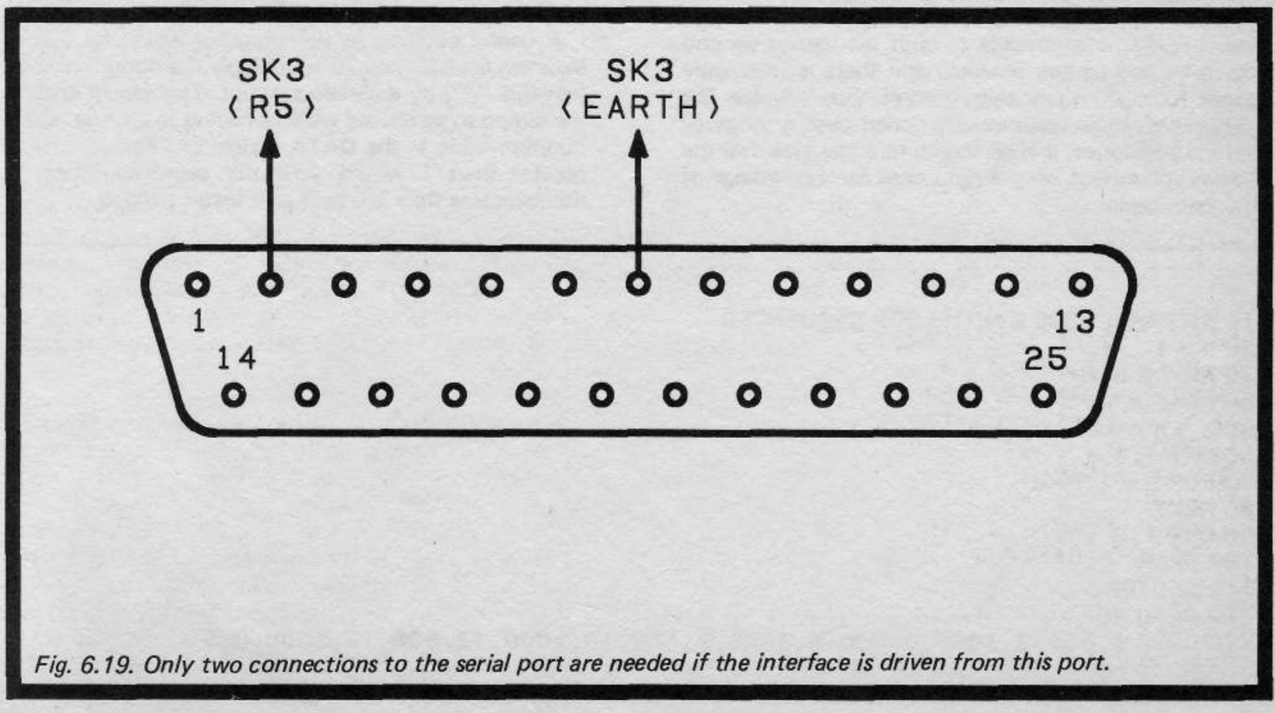


Fig. 6.19. Only two connections to the serial port are needed if the interface is driven from this port.

adjustments. Note that with many instruments external control is only possible once this feature has been enabled via the correct control settings, and that the unit will therefore have no effect until the correct settings have been selected. Note also, that VR1 and VR2 should ideally be good quality multi-turn "trimpots". Ordinary miniature preset potentiometers could be very difficult to adjust properly, and might have inadequate resolution.

Programming

As the unit is driven from the serial port there should be little difficulty in using it with any high level language (which will presumably provide support for the serial interface). If the unit is connected to the parallel port things might be a little less straightforward, but most languages give easy access to memory and input/output devices (including BASICs and C), and obviously there is no difficulty in this respect when using assembler.

It would be possible to develop quite sophisticated sequencer software for the unit if desired, but I suspect that these days a system such as this would rarely be used for anything more complex than something like a repetitive bass line. The complexity of sequences is restricted by the fact that the system is strictly monophonic, although sequences of many thousands of notes in length could probably be programmed if desired. Here we will assume that the unit is only to be used for short repetitive tracks. However, programming something of this type is not too difficult, and any reasonably competent BASIC programmer should have no difficulty in developing the suggested routines if something more sophisticated is required.

In order to produce a note from the synthesiser the program must output a value to the interface that is equal to the required note value plus 128. The extra 128 is, of course, needed in order to set the gate output high and trigger the note. With most synthesisers a note value of 1 is equal to a C somewhere below middle C, but it is normally possible to shift the tuning up and down by one or two octaves, and there is frequently some form of pitch shift control that enables the instrument to be continuously varied over a range of several semitones. It is up to you to make sure that the values you output are the right ones for the settings of the instrument.

At some time before the start of the next note the gate pulse must be terminated. You can not simply leave the gate output permanently high, because a low to high transition is needed at the start of each new note in order to trigger a new envelope period. To terminate the gate pulse you simply output a value equal to the current note value, without the 128 added.

Using ST BASIC it is possible to control the interface and synthesiser in a similar manner to the one used in chapter 1 to control the ST's internal sound generator. The program shown below will produce a short repetitive track.

This works in much the same way as the "improved sequencer" program in chapter 1, but the SOUND instruction has been replaced by a few lines (50 to 90) which output values to the serial port, and provide a timing loop so that note durations can be programmed. A FOR...NEXT loop is a rather crude way of providing the note duration timing, but my documentation on ST BASIC does not mention any form of built-in timer function. Getting the times right initially will take a little trial and error, but looking on the plus side, this method of timing seems to give (roughly) millisecond resolution so that times can be set very precisely.

The program is looped indefinitely by the GOTO instruction at line 120, but this can be omitted if a single pass sequencer is required. The RESTORE instruction at line 110 is not then needed. The purpose of this instruction is to reset the data pointer so that the data is read from the beginning again on each run through the sequence. Without this instruction the program would crash with an "out of data" error message. There are only two parameters controlled in the DATA statement at line 130, and these are the note value and duration. They must appear in the DATA statement in pairs, with the note value given first. Assuming that a note value of 1 is a C, the example values play an ascending scale of C major with alternate short and long notes. Values of 0 and 0 must be used at the end of the sequence as these are needed to end the WHILE...WEND loop.

A useful addition to the program would be a line between lines 60 and 70 to multiply the duration value (variable "d") by a certain amount. This would enable the tempo to be altered without having to change every duration value in the DATA statement. Tempo values greater than 1 would slow the sequence down - numbers less than 1 would give faster tempos.

```
10 REM ANALOGUE SYNTH LOOP SEQUENCER
20 n = 1
30 WHILE n  0
40 READ n , d
50 a = n + 128
60 OUT 1 , a
70 FOR b = 1 TO d
80 NEXT
90 OUT 1 , n
100 WEND
110 RESTORE
120 GOTO 20
130 DATA 1 , 500 , 3 , 1000 , 5 , 500 , 6 , 1000 , 8 , 500 , 10 , 1000 , 12 , 500 , 13 , 1000 , 0 , 0
```

Components (Fig.6.15)

Resistors (all 0.25 watt 5% carbon)

R1	1M2
R2	2k2
R3	2k2
R4	390
R5	2k2
R6	2k7
R7	3k9

Potentiometers

VR1	10k multi-turn trimpot
VR2	10k multi-turn trimpot

Capacitors

C1	100n ceramic
C2	27p ceramic plate
C3	27p ceramic plate
C4	47u 10V radial elect
C5	2u2 63v radial elect
C6	100n ceramic
C7	100n ceramic

Semiconductors

IC1	4024BE
IC2	6402
IC3	ZN426E
IC4	CA3140E
IC5	uA78L05
TR1	BC549C
TR2	BC549C
D1	1N4148

Miscellaneous

X1	2.4576MHz crystal
SK1,2,3	Standard jack sockets (3 off)
B1	9 volt (PP3, PP7, etc.)
	Case
	Circuit board
	Battery connector
	8 pin DIL holder
	14 pin DIL holder (2 off)
	40 pin DIL holder
	Wire, solder, etc.

Chapter 7
MIDI PROCESSING

For the do-it-yourself programmer the ST and MIDI applications provide a lot of scope. However, in my opinion at any rate, there is not a great deal of point in developing complex sequencer programs. Even if you are a reasonably expert programmer it is difficult to compete with the up-market MIDI software which is of a very high standard indeed. It is the type of thing that takes a small team of expert programmers months or even years to perfect! Even the more simple sequencer programs involve a great deal of highly skilled programming time. Given the reasonably low cost of some sequencing software, and the fact that there are very cheap programs available from shareware/public domain software sources, it is probably only worthwhile producing your own sequencer program if this is something that particularly interests you.

Of course, if all you want is a program that will provide a short and simple repetitive track for backing purposes, then even a moderately sophisticated real-time or step-time sequencer could reasonably be considered as providing substantial "over-kill". For a simple monophonic sequence something as basic as the sequencer program provided at the end of the previous chapter should suffice. Of course, it must be modified to output data to the MIDI output port rather than the serial interface, and it must supply valid MIDI data instead of gate/CV data. A suitably modified version of the program is provided below: –

```
10 REM MIDI BASIC SEQUENCER
20 n = 1
30 WHILE n > 0
40 READ n,d
50 OUT 3,144
60 OUT 3,n
70 OUT 3,64
80 FOR b = 1 TO d
90 NEXT
100 OUT 3,128
110 OUT 3,n
120 OUT 3,0
130 WEND
140 RESTORE
150 GOTO 20
160 DATA 60,500,62,1000,64,500,65,1000,67,500,69,1000,71,500,72,1000,0,0
```

This program operates in much the same way as the original, but in order to trigger or terminate a note it must output a standard MIDI three byte group. It will be assumed in this chapter that you are familiar with MIDI messages and the actual code numbers used. You should refer to chapters 3 and 4 if you are unclear about either of these aspects of MIDI. The values in the DATA statement are pairs of MIDI note values and duration values (approximately in milliseconds). Values of 0 and 0 are required to terminate the sequence and cause the program to loop back to the beginning again. A "dummy" value of 64 is used in the velocity byte (the third value in the note-on set), but this could be made programmable as well if desired. It would just be a matter of adding a third parameter into the READ instruction, using this parameter as the velocity byte, and placing sets of three values in the DATA statement.

In my experience of MIDI keyboard instruments there is no problem if you drive them from the MIDI input while at the same time playing them via the keyboard. The only point to bear in mind is that many instruments only offer six or eight note polyphony, and when using them in this way there is a definite danger of the instruments running out of voices. There should be no problem with sixteen or thirty two note polyphonic instruments though.

If you are interested in a simple real-time sequencer routine, this subject is covered later in this chapter.

MIDI Processing

MIDI processing is an area in which the ST plus a simple do-it-yourself program can give excellent results. MIDI processors have moderated in price over recent years, but a couple of these units could still prove to be quite expensive, and will only provide two processing functions. Multi-function are something of a rarity, and are far from cheap. The ST plus a simple program offers a cost-free approach if you already own an ST. This combination can provide any type of MIDI processing (within reason), and using a range of programs a variety of processes can be accommodated. Provided the program will run fast enough, it should even be possible to provide several types of MIDI process at once. Although there may be no commercial stand-alone unit that provides the type of processing you require, it may well be possible to obtain the desired action using the ST plus a some simple software.

The main limitation of using the ST as a MIDI processor is that it can not run other MIDI applications (such as sequencer software) at the same time. The ST is not a multi-tasking computer, and as it only has a single set of MIDI input and output sockets, it would make no difference if it was. Despite this limitation, being able to use the ST for MIDI processing is a very worthwhile asset, especially if your interest is mainly in "live" performance rather than MIDI sequencing and recording applications.

Before discussing some examples of MIDI processing using the ST, it would be as well to explain exactly what is meant by MIDI signal processing. A point which has to be made from the outset is that we are talking here about processing the MIDI data stream, and we are not talking about MIDI controlled processors which doctor the audio output signal of an instrument. Processing the MIDI data may not seem to have great potential for improving a system, and it is a subject which seems to have been largely neglected in the past, but it is something that can be put to very good use in many systems. It is not something that is only applicable to complex systems, and it can be used effectively with something as basic as a single MIDI keyboard instrument.

Channelising

MIDI signal processing is used for one of two reasons: to make up for some inadequacy in the system, or to add an advanced feature to the setup. Most MIDI processors provide processing of the former type rather than the latter. Probably the most common type of MIDI processing is so-called "channelising". One reason for doing this is to provide split keyboard operation. This feature seems to be increasingly common on modern MIDI keyboard instruments, but has been something of a rarity in the past. The idea is to have the keyboard split into zones, with each zone operating on a different MIDI channel (as in the example of Fig.7.1). You can then have each section of the keyboard controlling a separate instrument. With an instrument that has mode 4 or (preferably) some form of "multi" mode, much the same effect can be obtained, with each zone controlling a different voice of the instrument.

This is a feature that can often be added to a keyboard that does not have a such a function built-in. The basic idea is to take the MIDI output from the keyboard, and then alter the channel number in messages that contain certain note values. The processed output signal is then sent from the MIDI output and used to drive an instrument or instruments. A unit that provides this form of channelising is sometimes called a "note separator". One way in which this type of processing can be used is shown in Fig.7.2. For the sake of this example we will assume that the output of the keyboard is on channel 1, and that we want it to control its own sound generator circuits on channel 1, plus the drum machine on channel 2. However, the bottom four notes of the keyboard must control four voices of the drum machine, while the other notes must control the keyboard's internal sound generator circuits. This sacrifices four keys of the keyboard, but in return provides the player with a four piece percussion set which can be played from the keyboard.

What might seem to be the obvious method of interconnection for such a setup is as shown in Fig.7.3, but this will not give the desired effect. Anything played on the keyboard (including the lowest four notes) will operate the instrument's internal sound generator circuits in the normal way. What we require is for the lowest four keys to have no effect on the internal sound generator, and to only play the drum machine. This can be achieved by the original arrangement of Fig.7.2 provided the drum machine is set for mode 3 operation on channel 2, the keyboard's sound generator circuits are set for mode 3 operation on channel 1, and the keyboard is set to "local off". All the ST then has to do is to pass notes above a certain threshold level without altering them, and to shift notes at or below that level up one channel. These shifted notes will then be played by the drum machine and not by the keyboard's internal sound generator circuits.

The ST can be used as a channeliser in more sophisticated setups, such as the one shown in Fig.7.4. Here the keyboard could be simply that (i.e. no built-in sound generator circuits) with one rack-mount unit being controlled on channel 1 via the bottom half of the keyboard, and the other being controlled on channel 2

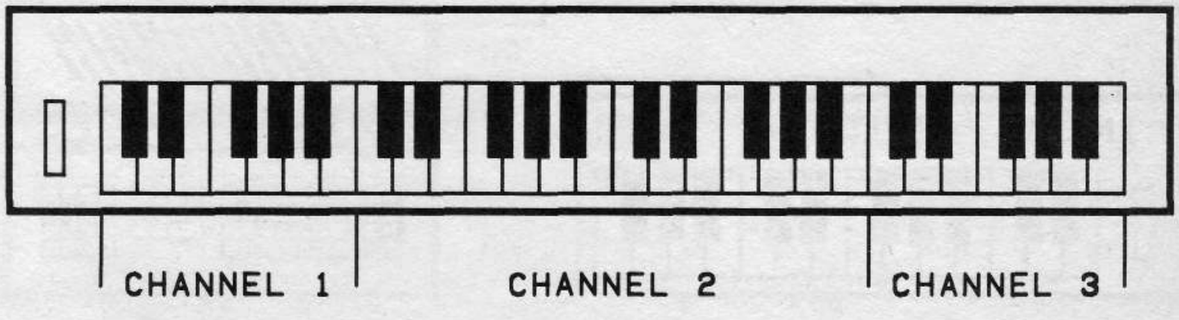


Fig. 7.1. These days much use is made of keyboard "splits", as in this example.

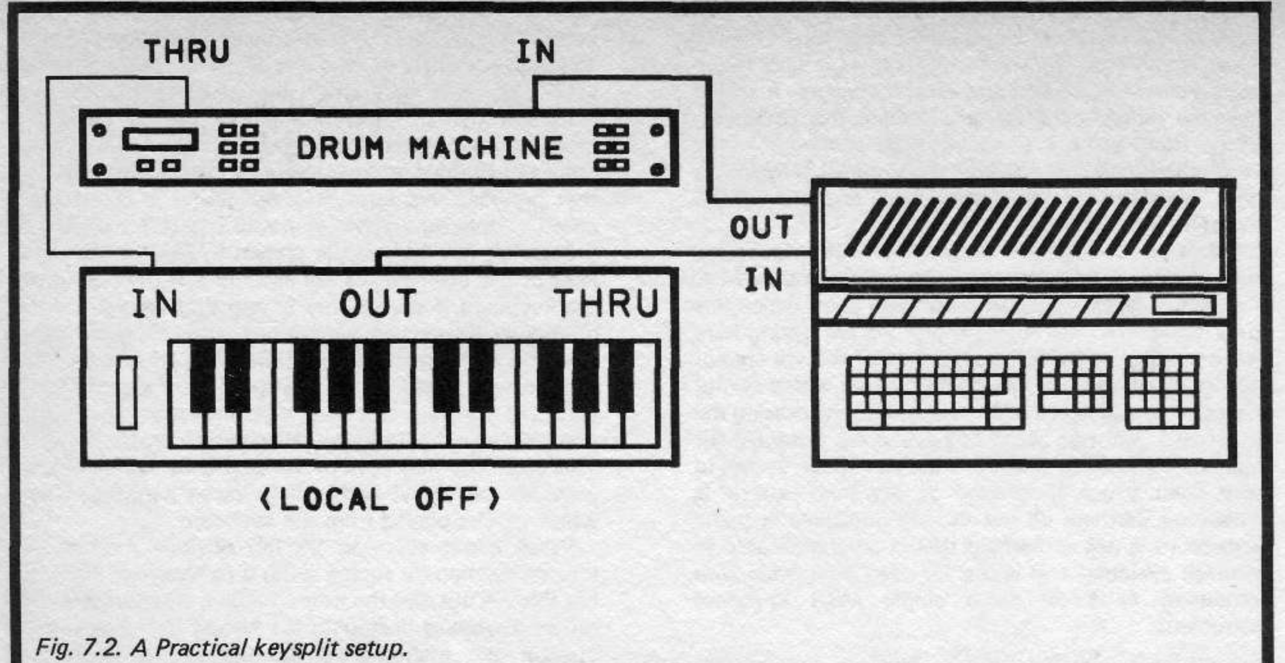


Fig. 7.2. A Practical keysplit setup.

by way of the top half of the keyboard. Alternatively, the system could be quite effective with the keyboard having an internal sound generator covering its full compass, and the rack-mount units in the same key-split operation and being used to "thicken" the sound of the keyboard instrument. Fig.7.5 shows a more sophisticated way of using the system. Here the keyboard instrument is used in the "local off" mode so that it effectively operates as a separate MIDI keyboard and sound module. The ST is used to process the output from the keyboard and feed the signal back to the sound generator circuits. The THRU output of the instrument then couples the processed signal on to the two rack-mount units. This system could be set up for a three-way split if desired.

It is only fair to point out that some MIDI channelisers provide a split of the type outlined in Fig.7.6, but the ST can not be used in this way. Here the signal is split into three separate output signals, with one for the low notes, one for the middle notes, and the third one for the high notes. All three outputs operate on the same MIDI channel (usually channel 1). Some MIDI instruments, particularly a lot of early examples, are only capable of operation on channel 1 and (or) operating with 'omni on'. In either case, they are unlikely to be suitable for use in the systems described

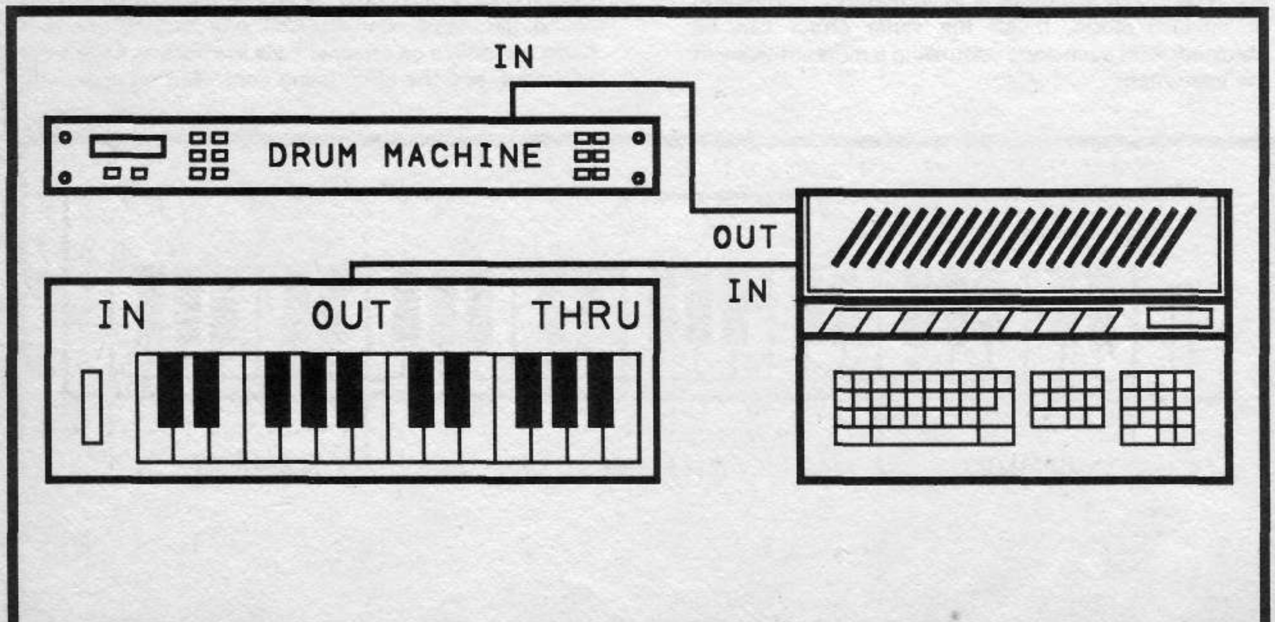


Fig. 7.3. This setup looks plausible, but will not give the desired action.

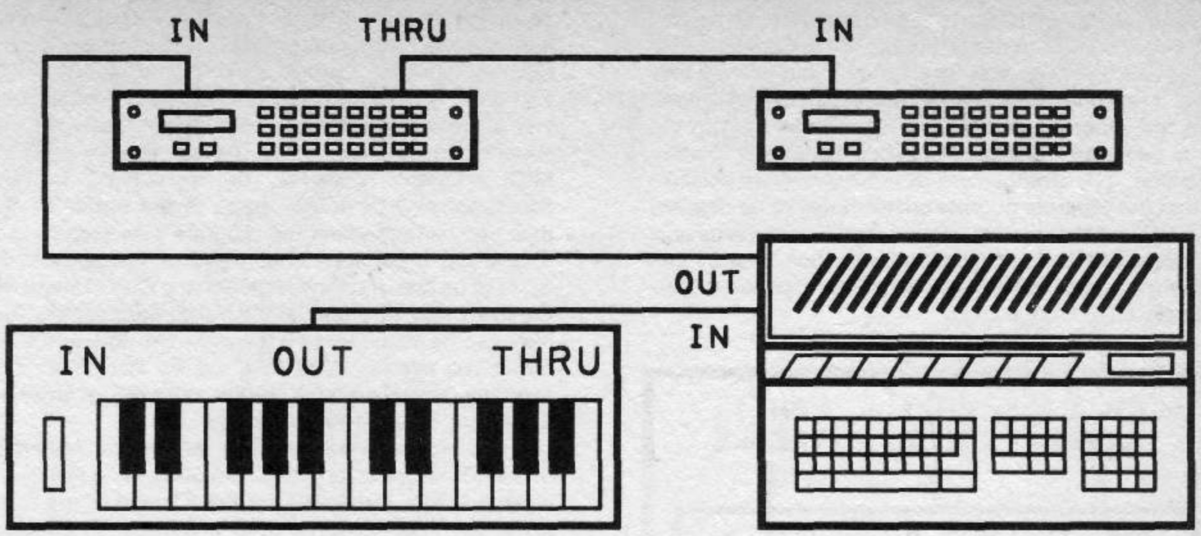


Fig. 7.4. A more complex keysplit system.

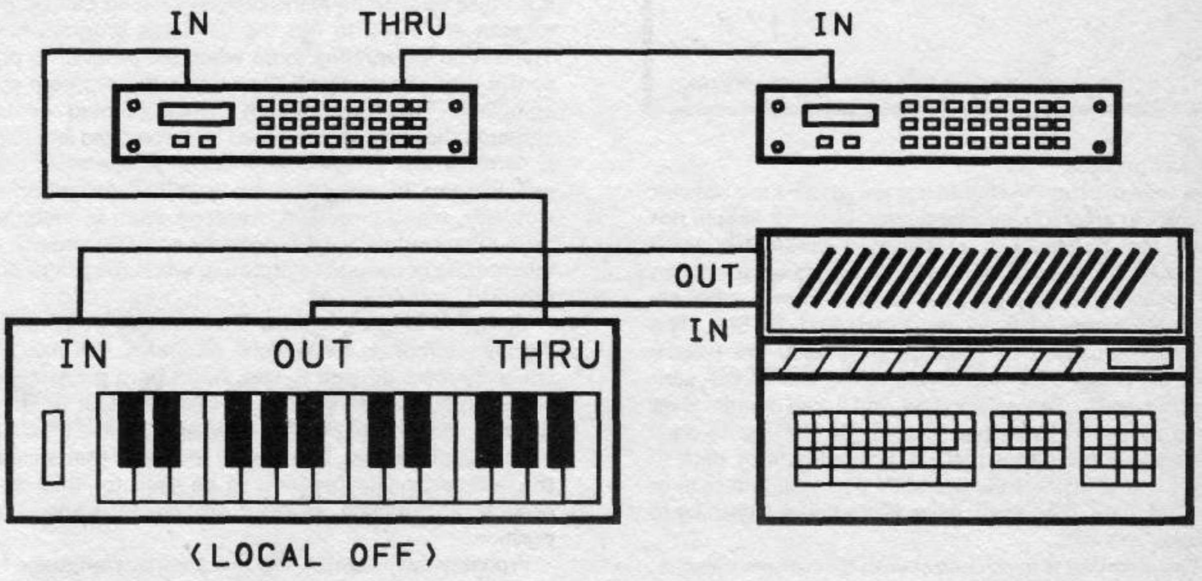


Fig. 7.5. A system capable of a three way keyboard split.

earlier as they can not pick out information on just one channel, or be switched to a suitable channel. In fact you might be able to use an instrument which is capable of mode 3 operation only on channel 1, but this will only work if just one instrument in the system is of this type. This instrument could be used on channel 1, with the other instrument or instruments being switched to other channels. The arrangement of Fig.7.6 is more versatile in that the separate outputs enable notes to be directed to individual instruments, even if those instruments only support channel 1 or "omni on" operation. The ST can not provide this type of processing as it possesses only a single MIDI OUT socket.

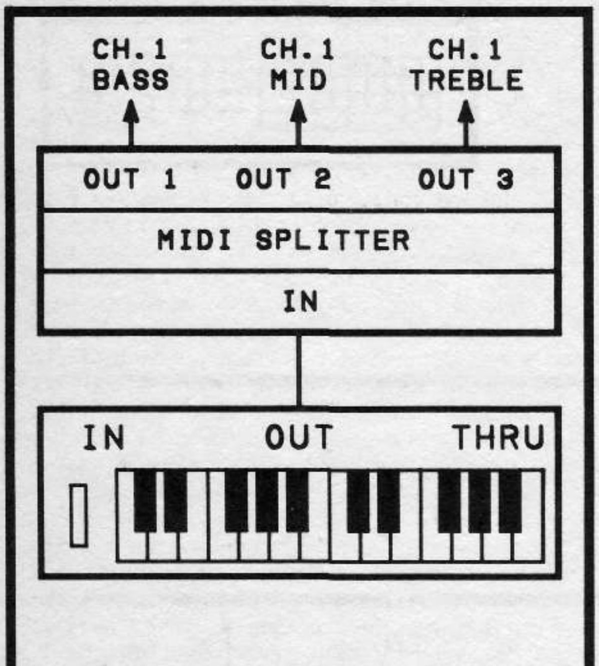


Fig. 7.6.
The ST cannot provide this type of channelising.

ST Languages

This type of program should not prove to be too difficult to write in any high level language. In fact it should not pose too much of a problem in assembler. With computers that have add-on MIDI interfaces there can be problems in reading the interface from a relatively slow language such as an interpreted BASIC. The problem is caused by MIDI sending what are usually two or three byte messages, with the bytes sent "end-to-end". Remember that MIDI can handle over three thousand bytes per second. By the time a slow language detects that there is a fresh byte of data to read, and actually reads the MIDI port, the first byte or two of data may well have been overwritten by a subsequent byte.

The situation is much better with ST for two reasons. One of these is simply that as the MIDI port is built-in, a high level language for the machine should include support for the MIDI port. In other words, the language should include routines that make it easy to read the MIDI port, so that you are not having to directly read and control the hardware. What this often means in practice is that the MIDI port is monitored by the operating system, and that when bytes are received they are read from the port and stored in an area of memory (called a "buffer"). When you read the MIDI port from a language that provides this facility, what you are actually doing is reading any received bytes currently in the buffer, not reading directly from the MIDI interface hardware. Bytes received in rapid succession will be safely stored in the buffer so that they can be read when the program gets around to it. This is not a perfect solution as it is still possible for bytes to be lost. If data is received at a high average rate the bytes may be stored in the buffer more rapidly than they can be processed by the program. Eventually the buffer will overflow and data will be lost. Even if an overflow does not occur, there is still a risk of small but significant timing errors occurring.

These problems are eased by the second advantage of the ST, which is its substantial processing power. As a result of this, languages for the ST tend to run much faster than equivalents on other machines. ST BASIC is not particularly fast by ST standards, but the latest version still manages to run at a rate which is substantially faster than most 8 bit BASICs.

Other BASICs are available for the ST, and these are mostly much faster than ST BASIC. Some of these are interpreted BASICs, and some are compiled types (and some are available in both versions). An interpreted language is one which stores the program as a series of command words, data etc. When the program is run, each command is translated into machine code instructions which are then run by the microprocessor. This is a slow way of doing things, as the interpreting process often takes very much longer than running the interpreted instruction! In fact it often takes one hundred to one thousand times longer. A compiled language converts the program to machine code, and then stores it on disc as a stand-alone program which can be run without any need to run the language program first. There is no interpreting to do when the program is run, as this was effectively all done when the program was compiled. This gives much faster running speed, although the program produced by a compiled language is normally less compact and slower in operation than one written in assembler by a skilled programmer. Actually, many compiled programs seem to include a "run-time module" which provides a certain amount of interpreting or pseudo-interpreting when the program is run.

Compiled languages have the advantage of producing fast, compact, stand-alone programs, but they are generally more difficult to use. Apart from the fact that compiled languages tend to be more difficult to learn, there is the drawback that they are more difficult to debug. A good, fast, interpreted language that enables the interpreter and program to be used together as a pseudo stand-alone program is an attractive proposition.

Probably the nearest thing to a standard language for the ST for do-it-yourself programers is Computer Concepts "Fast BASIC", which is a high speed interpreted BASIC. It is an excellent implementation of BASIC which, apart from being very fast in operation, also gives excellent support to the ST's special features and firmware, and has a built-in assembler. It also has features that are of interest to those who like a structured approach to programming. Unusually for an interpreted language, with the aid of a low cost add-on program it can produce stand-alone programs that can be run without having to load Fast BASIC, then load the program, and finally run it. Stand-alone programs produced using Fast BASIC are not compiled programs incidentally, they are a combination of the program file and the Fast BASIC run-time module (i.e. the interpreter part of the Fast BASIC program). There are no copyright problems with the run time module, and stand-alone Fast BASIC programs can be sold or given away, complete with this interpreter part of the program.

In this chapter we will only consider programs written in the latest version of ST BASIC and programs written in Fast BASIC. These two languages have been chosen simply because ST BASIC is supplied free with STs, and Fast BASIC seems to be the most popular alternative to ST BASIC (in the U.K. at any rate). Obviously there are many other programming languages available for the ST, and several of these are quite popular (GFA BASIC and various 'flavours' of C for example). There simply is not enough space available here to cover even a small selection of the available languages. However, the principles outlined in this chapter should be easily applied to other languages, and none of the programs are particularly complex. We will mainly be concerned with Fast BASIC, as its fast operating speed makes it more suitable than ST BASIC for a demanding application such as MIDI processing.

Note Separator Program

The only slightly awkward aspect of producing a channeliser program is that the byte which contains the note values is not the one that contains the channel value. Furthermore, the byte which contains the channel value precedes the one which carries the note value. The logical way to handle the problem is to read in bytes from the MIDI port, and to simply pass them straight through to the MIDI output port unless they are the first byte in a note on or note off message. If one of these messages is detected then the program must store this value. The next two bytes (the note and velocity values) are then read in and stored. The note value byte can then be checked, and if it is outside certain limits the note on/off byte can be boosted or decremented in order to give the desired shift in channel number. Remember that MIDI channels are normally numbered from 1 to 16, but that the actual values used to select them are from 0 to 15.

This example program is written in ST BASIC and its effect is to increment the channel number by 1 for any note that is above middle C (MIDI note 60). Note that like the other ST BASIC programs in this book, it has only been tested with the new version of ST BASIC and it might need slight modification to run under the original version.

```
5 REM ST BASIC CHANNELISER
10 A = INP(3) AND 255
20 IF A > 127 AND A < 161 THEN GOTO 1000
30 OUT 3,A
40 GOTO 10
1000 B = INP(3) + 256
1010 C = INP(3) + 256
1020 IF B > 60 THEN A = A + 1
1030 OUT 3,A
1040 OUT 3,B
1050 OUT 3,C
1060 GOTO 10
```

ST BASIC and Fast BASIC both provide support for the MIDI ports via the INP and OUT functions, which are respectively used for reading input devices, and writing to output devices. In both cases the MIDI port is device number 3. Both of these versions of BASIC seem to produce unusual results when reading the MIDI port. ST BASIC seems to provide negative numbers, whereas Fast BASIC provides numbers from 65280 to 65535 (not the correct 0 to 255 for standard 8 bit bytes). What seems to be happening is that 16 bit values are being read, and that the eight most significant bits are always set to 1. Fast BASIC interprets the read values as straightforward binary values, whereas ST BASIC is presumably interpreting them as some form of signed binary and is consequently providing negative values. As far as I can ascertain, the top eight bits of returned values provide no useful information. Presumably the MIDI port is read via a BIOS routine which provides 16 bit values, and neither implementation of BASIC strips the unnecessary bits.

In some applications it seems to be quite in order to ignore the fact that the INP(3) function is not returning 8 bit bytes. This is not the case here, as the program will test received values and provide one action or another, depending on what range of values received bytes fall within. We must therefore convert received values to normal 8 bit values before testing them. There is more than one way of achieving the conversion, but the most reliable method for either type of BASIC is to bitwise AND each value returned by the INP(3) function with 255. It is the purpose of line 10 to read in values and make the necessary conversion.

Bitwise ANDing is a simple process that compares two binary numbers on a bit by bit basis. A 1 is placed in the answer if both the corresponding bits of the two numbers are 1. If a bit is at 0 in either or both of the numbers, then a 0 is placed in that bit of the answer. In this case we are ANDing values read from the port with 255 (11111111 in binary), and the eight least significant bits of the answer will be the same as those provided by the values from the INP(3) function. The eight most significant bits in the "masking" number (255) are all zeros. It makes no difference what logic states these are at in the values returned by the INP(3) function - they must always be 0 in the answer since there can never be a 1 in both bits of the two ANDed values. This method is much safer than simply adding or taking away a fixed amount from received values in order to bring then into the correct range. With the bitwise AND method the correct result will always be obtained, even if one of the most significant bits should be set to 0 for some reason. This is not the case if values are simply boosted or reduced by a certain amount.

Note that the INP() function always waits for new data to be received if none is already present when an INP() instruction is reached in the program. Therefore, there is no need to construct a loop to continually test for data. You might like to add a "timeout" facility to halt the program if no data is received for a certain period of time. I found such a facility to be less than helpful though, and omitted it from the final program.

At line 20 a check is made to determine whether the received byte is a note on/off type, or some other form of MIDI message. Note on/off messages conveniently have a decimal value in the range 128 to 160, and this line simply checks to see if the received value is within this range. If it is, the program branches to a sort of pseudo subroutine at line 1000. If not, it goes to line 40 where the byte of data is transmitted on the MIDI output port.

The pseudo subroutine starting at line 1000 reads in the next two bytes of data which should be the note and velocity values. There is a potential problem here in that MIDI clock messages can be mixed into other messages. None of my equipment actually seems to do this though, and in many applications it would be possible to simply avoid having the clock signal transmitted anyway. Accordingly, this potential problem is not catered for by this program, but it would probably not be too difficult to add a suitable "catch" to the program to collect any MIDI clock messages and send them straight on to the output. In the interest of having the program run as fast as possible, it is probably best not to bother with anything of this type unless you really need to.

When all three bytes have been read in, line 1020 checks whether the note value (variable "B") is greater than 60. If it is, the header byte (which is in variable "A" and contains the channel number) is incremented by 1. I have set the split point at a note value of 60, or middle C in other words. However, it would obviously be quite easy to set the split point at any desired point simply by altering the appropriate value in the program. Also, by having several lines of this type with different split point values, multiple splits could easily be produced.

The next three lines output the three bytes in the correct order, and then the program loops back to line 10 to wait for the next byte. The reason for using this pseudo subroutine rather than the real thing is that the program must start from the beginning once again when it exits the routine. With a real subroutine it would take up where it left off, or move on to line 40 in other words. This would result in byte "A" being outputted a second time, which would obviously cause a malfunction of the system.

Fast BASIC Version

ST BASIC is reasonably fast by BASIC standards, but it is barely fast enough for MIDI processing applications. The split-keyboard program works well enough if it is only receiving note on/off messages plus perhaps a moderate amount of additional data such as aftertouch messages. With large amounts of data to process, perhaps due to pitch bending or a great deal of aftertouch information being produced by the keyboard driving the unit, things start to go wrong. I did not find that the program had any tendency to crash, but with large amounts of data to process there were quite noticeable timing errors. These could be large enough to totally change the rhythm of a piece!

In order to be certain of avoiding any significant delays when processing MIDI data it would probably be necessary to resort to a good compiled language. However, Fast BASIC lives up to its name and works fast enough for most MIDI processing applications. While I did find it possible to overload a Fast BASIC program with MIDI data, this was only by using a combination of controls that would not generally be used in normal use. I will not guarantee that the Fast BASIC version of the split keyboard program is not overloadable, but I will claim that it will perform quite adequately in most situations. The Fast BASIC version of the program is provided below: –

```
REM FAST BASIC SPLIT PROG
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT
REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCRAISE ELSE PROCNORM
UNTIL FALSE

DEF PROCRAISE
B = (INP(3) AND 255)
C = INP(3)
IF B > 60 THEN A = A + 1
OUT 3,A,B,C
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

In Fast BASIC line numbers are optional. They are retained in order to give compatibility with earlier BASICs, as are the GOTO and GOSUB instructions. This program has been written without line numbers, and using the PROCedures instead of GOTOs and GOSUBs.

The FOR...NEXT loop at the beginning of the program reads in any MIDI data left in the buffer, and dumps it into the variable "DUMMY". If any data should be received by the ST after switch-on but before running this program, this routine should clear it away and prevent it from being outputted (which could otherwise cause droning notes, or other problems). It might be useful to add a routine of this type at the beginning of the ST BASIC version if that is the version you intend to use.

The main program is the four lines of the REPEAT...UNTIL loop, which loops indefinitely in this case. To break out of the program simply press ESCAPE and then operate any key of the MIDI keyboard (this method can be used with all the Fast BASIC programs described in this book). The REPEAT...UNTIL loop reads in any bytes received by the MIDI input, and it then tests to determine whether or not they are in the range 128 to 160. If they are in the relevant range the program is directed to PROCRAISE, which reads in a further two bytes of data. It then boosts the channel number by one if the note value is greater than 60. If received bytes are not note on/off messages, the program is directed to PROCNORM where bytes are sent to the MIDI output without undergoing any further manipulation.

This basic routine can easily be expanded to provide greater versatility. For example, the version shown below provides two split points which are decided by the user when the program is run.

This operates in much the same way as the original program, but the two split points are requested at the beginning of the program and placed in variables "SPLIT1" and "SPLIT2". If only a single split point is required, simply make the second split point at a note value that is higher than the highest note of your keyboard. A value of 128 will do, and is beyond the range of any MIDI keyboard. Even though this is not a valid MIDI note value, this will not upset the operation of the program. PROCRAISE has been modified so that it raises the channel number in any note on/off message which has a value greater than "SPLIT1". It then provides a further increment in the channel number if the note value is higher than "SPLIT2". Assuming the keyboard is operating on MIDI channel 1, low notes remain on channel 1, middle notes are placed onto channel 2, and high notes are sent on channel 3.

With note separation techniques it is important to realise that only note messages are channel shifted. Messages such as pitch bend and channel aftertouch are left unaffected, as there is no way of ascertaining with any certainty which note they are aimed at. The choice is basically one of shifting all non-note on/off channel messages or leaving them unaltered. There are actually other alternatives, such as sending these other messages on the normal and shifted channels, or getting the computer to make an intelligent guess as to which channel or channels a message should be sent on. However, bear in mind that any extra processing could take up a lot of extra time, and could cause MIDI clogging. It is best to only worry about this type of thing if all the equipment involved supports aftertouch etc., and it is important to you that the "slave" instrument should respond to it.

The situation is different with polyphonic aftertouch where a note value is included in one of the data bytes.

```
REM FAST BASIC DOUBLE SPLIT PROG
INPUT  "ENTER FIRST SPLIT POINT "  SPLIT1
INPUT  "ENTER SECOND SPLIT POINT "  SPLIT2
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT
REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCRAISE ELSE PROCNORM
UNTIL FALSE

DEF PROCRAISE
B = (INP(3) AND 255)
C = INP(3)
IF B > SPLIT1 THEN A = A + 1
IF B > SPLIT2 THEN A = A + 1
OUT 3,A,B,C
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

These messages could therefore be processed in much the same way as note on/off messages, although it is obviously only worthwhile doing so if you have one of the few keyboards that support polyphonic aftertouch. I suspect that most keyboards that do have this feature also have comprehensive split facilities, and that this is all of only academic importance.

Channel Shifting

Note separation is not the only form of channelising, an alternative method is one where a straightforward shift from one channel to another is provided. This type of processing is mainly used with instruments that are restricted to operation on MIDI channel 1. Few (if any) current instruments are restricted in this way, but many early MIDI equipped instruments are confined in this manner. If you wish to control two instruments of this type on separate channels, this is possible if a channeliser is used ahead of one instrument and used to take messages on (say) channel 2 down to channel 1. This effectively puts two channel 1 instruments on channels 1 and 2, but if you are using the chain method of connection you must remember to use the processor and the "shifted" instrument at the end of the chain. Remember that the channeliser will affect any device placed after it in the "chain".

Another possibility with this type of channel shifting is to layer sounds. In other words, you might be able to play one "instrument" from the keyboard, and play another "instrument" using what is actually another channel and timbre of the same instrument. This would work best with an instrument which has a multi-mode that permits polyphonic operation on two MIDI channels. With the two virtual instruments on channels 1 and 2, and the keyboard transmitting on channel 1, the ST could be used to raise signals from the MIDI output onto channel 2, and feed them to the MIDI input. Here (with a suitable instrument) they would play the same notes but with a different sound. A point to keep in mind with this type of processing is that each note you play is actually two notes as far as the instrument is concerned. An eight note polyphonic instrument would therefore be reduced to four note polyphonic operation.

A third method of channelising is straightforward channel filtering. This is used with instruments that only have "omni on" and no "omni off" modes. Consequently they always respond to notes on any MIDI channel, and are of limited value in many MIDI setups. Again, this is a shortcoming which it pretty well extinct in modern instruments, but which was not uncommon in the early days of MIDI. A channel filter simply passes messages that are on a particular channel, but ignores any that are on other channels. Added ahead of an "omni on" instrument, this effectively sets it to mode 3 operation (assuming it is a polyphonic type) on the channel selected using the channeliser.

Obviously a number of different types of channel shifting and filtering are possible, some of which have been outlined above. Here we will consider two shift/filter programs which cover a variety of possible applications, and illustrate the basic principles involved. This first program, shown below, takes notes on any channel and outputs them on a user-specified channel.

The channel number is entered at the INPUT instruction, and it is placed in variable "C". It is then decremented by one. Remember that MIDI channels are from 1 to 16 but we must deal in the true channel values of 0 to 15 when undertaking MIDI processing. In this case it is not merely note messages that we wish to alter – all channel messages should be shifted. The channel messages are sorted out from other messages using the

```
REM FAST BASIC CHANNEL SHIFTER
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT
INPUT  "ENTER OUTPUT CHANNEL "  C
C = C - 1

REPEAT
A = ( INP(3) AND 255)
IF A > 127 AND A < 240 THEN PROCCHAN ELSE PROCNORM
UNTIL FALSE

DEF PROCCHAN
A = (A AND 240)
A = A + C
OUT 3 , A
ENDPROC

DEF PROCNORM
OUT 3 , A
ENDPROC
```

same basic system that was adopted for note message selection in the previous programs. Message header bytes are always greater than 127, and the system messages are at values of 240 and above. Channel message header bytes are therefore greater than 127 and less than 240, making it easy to detect them. In this application we do not need to read in three byte groups as it is only the header byte that is required. In PROCCHAN these bytes are ANDed with 240 to set the least significant nibble to 0000. Variable C is then added to the answer so that the new channel number is substituted for the old one. The modified byte is then sent to the MIDI output port.

This second channel shifter program, shown below, is similar to the first one, but it enables both input and output channels to be selected. It therefore provides both channel shifting and filtering, and is suitable for most channel shifting/filtering applications.

Filtering

Some instruments now have some basic MIDI filtering built-in. This can take the form of MIDI messages that can be disabled so that they are not transmitted, or making the instrument "deaf" to certain types of received message. The type of thing we are talking about here are messages such as program changes, where you might want a slave instrument to have the same sound throughout a piece, while the main instrument is switched through several programs. Another common use of filtering is to remove aftertouch messages or other data that could rapidly eat up the memory of a sequencer. MIDI filtering is something that can easily be handled by an ST acting as a signal processor if your equipment does not have suitable built-in filtering. The routine shown overleaf, removes channel aftertouch messages.

```
REM FAST BASIC CHANNEL SHIFTER/FILTER
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT
INPUT  "ENTER INPUT CHANNEL "  IC
INPUT  "ENTER OUTPUT CHANNEL "  OC
IC = IC - 1:OC = OC - 1

REPEAT
A = (INP(3) AND 255)
B = (A AND 15)
IF A > 127 AND A < 240 AND B = IC THEN PROCCHAN ELSE PROCNORM
UNTIL FALSE

DEF PROCCHAN
A = (A AND 240)
A = A + OC
OUT 3,A
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

The output channel is set in much the same way as on the earlier channel shifter program, but in this version the output channel number is placed in variable "OC". The test in the REPEAT...UNTIL loop detects any channel message header bytes, as in the previous program. However, in this case we also need to sort out messages that have the right channel number (entered as variable IC). Variable "B" masks off the channel number, and an additional test in the program compares this with "IC". If all the test conditions are met the program goes to PROCCHAN where the output channel number is altered. Otherwise it branches to PROCNORM where the unprocessed input byte is sent to the MIDI output port.

Received bytes of data are read in, and then ANDed with 240 to strip off the channel number. If a value of 208 (the channel aftertouch header nibble value) is detected, the program branches to PROCDUMP. Here the next byte, which is the aftertouch data byte, is placed in variable "B". However, nothing is done with this variable, or with variable "A" which contains the header byte. Aftertouch messages do not, therefore, get passed through to the output. Other messages are handled by PROCOUT, and are passed through to the output.

This routine could easily be changed to suit other message types. Simply change the 208 value to the appropriate one for the header nibble of the message type you wish to suppress. Also, PROCDUMP must read in the right number of data bytes for the message

```
REM FAST BASIC MIDI FILTER PROG
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT

REPEAT
A = (INP(3) AND 255)
IF (A AND 240) = 208 THEN PROCDUMP ELSE PROCOUT
UNTIL FALSE

DEF PROCDUMP
B = INP(3)
ENDPROC

DEF PROCOUT
OUT 3,A
ENDPROC
```

that is being removed. This would be two bytes for polyphonic aftertouch messages for example. For something like MIDI clock messages there are no data bytes, and there would then be no need to have PROCDUMP at all. The main program would simply have to branch to PROCOUT if the received message was a wanted type, or carry straight on with the loop if it was the type that the program had to filter out.

Perfect Harmony

A harmoniser used to be a device that processed the audio output signal of an electric or electronic instrument. The general idea was to have an oscillator which locked on to the input signal, but which was set higher in pitch by some musical interval (often a fifth). This can give so-called "thicker" sounds, and is particularly useful for enriching one finger accompaniments. MIDI offers an alternative method of processing the MIDI data to shift notes up or down in pitch by the required amount. One way of using a MIDI harmoniser would be to take the MIDI output of one instrument and to then apply the processed signal to the input of another. In this situation the MIDI harmoniser might be unnecessary, as some instruments have a "transpose" facility that could give much the same effect. Another way of using a MIDI harmoniser is to process the MIDI output of an instrument and then feed the signal back into the same instrument. With all the instruments I have used, an arrangement of this type works well, with the required two notes being produced per key-press.

Of course, you are not limited to two note harmonies, and two or more extra notes can be produced if desired. Remember though, that you must have sufficient voices available for all the notes. With three notes being produced per key-press and an eight voice polyphonic instrument, the system will only work properly if no more than two keys are operated at a time.

Shown below is a basic single note Fast BASIC harmoniser program.

```
REM FAST BASIC HARMONISER PROG
INPUT "ENTER NOTE SHIFT (IN SEMITONES) " INC
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN INP(3) = DUMMY
NEXT

REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCSHIFT ELSE PROCNORM
UNTIL FALSE

DEF PROCSHIFT
B = INP(3) AND 255)
C = INP(3)
B = B + INC
OUT 3,A,B,C
ENDPROC

DEF PRONORM
OUT 3,A
ENDPROC
```

This works in a similar manner to the previous programs. Non-note on/off messages are passed straight through to the output via by PROCNORM, whereas the note messages are processed by PROCSHIFT. This reads in complete three byte messages, increments the note value (variable "B"), and then outputs the three bytes to the MIDI port. The required shift in pitch is entered at the beginning of the program and negative values are used if a downwards shift in pitch is required.

Below is a version of the program that provides two output notes with user adjustable pitch offsets.

```
REM FAST BASIC DUAL HARMONISER PROG
INPUT "ENTER TWO NOTE SHIFTS (IN SEMITONES) " INC1,INC2
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT

REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCSHIFT ELSE PROCNORM
UNTIL FALSE

DEF PROCSHIFT
B = (INP(3) AND 255)
C = INP(3)
D = B + INC1
E = B + INC2
OUT 3,A,D,C
OUT 3,A,E,C
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

This is fundamentally the same as the earlier program, but two shifts are entered at the INPUT line. When entering these a comma is used as the separator. PROCSHIFT adds the two offset values to note value "B" and places the modified note values in variables "D" and "E". These values are then placed into three byte note on/off messages which are sent to the MIDI output port.

It is possible that the program could generate invalid note values (i.e. less than 0 or more than 127). It would be easy to trap and correct this with a few program lines such as "IF D > 127 THEN D = D - 12", so that values are brought within the legal range by transposing them up or down an octave. In practice it will probably not be worthwhile doing this. Most MIDI keyboards get nowhere near the MIDI minimum and maximum note values, and out of range values are unlikely to be produced provided you use a reasonable pitch offset. A more likely problem is that of notes being produced that are not within the compass of the instrument. Again, it is probably not worthwhile putting in lines to trap and correct this. Most instruments (probably all) include their own routines to do this, making any similar facility in the program superfluous. Unless you are using a really fast language it is advisable to keep the amount of processing to a minimum so that the program can execute at a suitably fast tempo.

Velocity Control

Although quite rare at one time, most MIDI keyboards are now touch sensitive and generate velocity values. A lot of keyboards have been criticised for having poor touch sensitivity, with it being impossible to put much expression into the music played on them. This can be due to notes tending to be quite loud even if the keys are pressed quite softly, or (more usually in my experience) playing "fffff" seeming to give something less than maximum volume from an instrument.

This problem is not necessarily the fault of the keyboard, and it could be the way in which the sound generator circuits are interpreting received velocity values. With some instruments the user has considerable control over the degree of touch sensitivity, but this is not a universal feature by any means. It is possible to introduce this feature to most MIDI instruments by setting them to the "local off" mode, and then connecting a velocity processor between its MIDI "OUT" and "IN" sockets. Quite complex processing could be used if necessary, but for most purposes multiplying the velocity values by a certain amount is quite sufficient. Multiplying values by more than one should help if the instrument seems unable to reach full volume, while a multiplier of less than one should be beneficial if the problem is of the "all or nothing" variety. A simple velocity processor program is shown overleaf.

```
REM FAST BASIC VELOCITY CONTROL PROGRAM
INPUT  "ENTER THE VELOCITY MULTIPLIER "  MULT
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT

REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCMULT ELSE PROCNORM
UNTIL FALSE

DEF PROCMULT
B = INP(3)
C% = (INP(3) AND 255)
C% = C% * MULT
IF C > 127 THEN C% = 127
OUT 3,A,B,C%
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

This sorts out note on/off messages in the same way as some of the previous programs. However, in this case the processing is done on the third byte in the message which is placed in variable C%. An integer variable is used because we must only supply in- range integers (0 to 255) to the MIDI port. Using an integer variable is an easy way of discarding any figures after the decimal point. The velocity value is multiplied by the user supplied variable "MULT", and then the three bytes of the message are transmitted. A line is included to reduce C% to 127 if it goes over this maximum permissible figure. Multiplier values of between 0.5 and 2 are usually sufficient to provide the desired effect. Of course, a processor of this type can only work with equipment that is touch sensitive. If the problem is due to something in the system lacking touch sensitivity, there is nothing a processor of this type can do to improve things.

All Change

Often when playing "live" it is necessary to introduce program changes. I mean by this, "program" in the sense of a set of parameters to give a certain sound from an instrument, and a MIDI program change message therefore gives a change in sound. In fact MIDI program changes can go beyond this in scope, and there are mixers, effects units, etc. which can be switched from one set of parameters to another via program change messages. Most synthesisers can produce program changes from the front panel controls without too much difficulty, but it is often much easier if they can be generated using a foot-pedal as the controller. Unfortunately, few instruments seem to provide such a facility.

This is something which would seem to be quite easy to implement via the ST. Simply connect a foot-switch to one of its input ports and get it to provide a suitable program change message each time the switch is operated. In practice the only complication is that joystick port 1 (the one not used for the mouse) is the obvious port to use, but it is not easily read from many ST programming languages, including ST BASIC and Fast BASIC. The problem seems to revolve around the fact that (unlike most computer joystick ports) the ST's game/mouse ports do not connect to interface devices in the main unit. Instead, they connect to the keyboard circuit. Although the ST (in its non "Mega" form) may appear to be an all-in-one unit and not one which has a separate keyboard, this is only true physically. From the electronic point of view the keyboard is a (more or less) self contained unit with its own dedicated processor, and a serial data link to provide two-way communications with the main computer circuit. The keyboard, including the joystick ports, is therefore read via a rather indirect route.

Fast BASIC does have a facility to read the mouse port (port 0) using (logically) the "MOUSE" function. The disadvantage in using this is that you would have to keep swopping over the mouse and the foot-switch, or have an adaptor that would enable them both to be connected to the port at the same time. A little experimentation showed that reading the right mouse button also read one of the trigger inputs on the other port. Consequently, a foot switch connected to port 1 in the manner shown in Fig.7.7 can be read by the MOUSE function. The connections to port 1 are made by way of a 9 way D socket incidentally. A suitable program for this setup is shown overleaf.

```
REM PROG CHANGE PEDAL PROG
PRG = 0
REPEAT
MOUSE X%,Y%,B%,K%
IF (B% AND 2) = 2 THEN PROCCHANGE
UNTIL FALSE

DEF PROCCHANGE
PRG = PRG + 1
OUT 3,192
OUT 3,PRG
TIME = 0
WHILE TIME < 400
WEND
MOUSE X%,Y%,B%,K%
ENDPROC
```

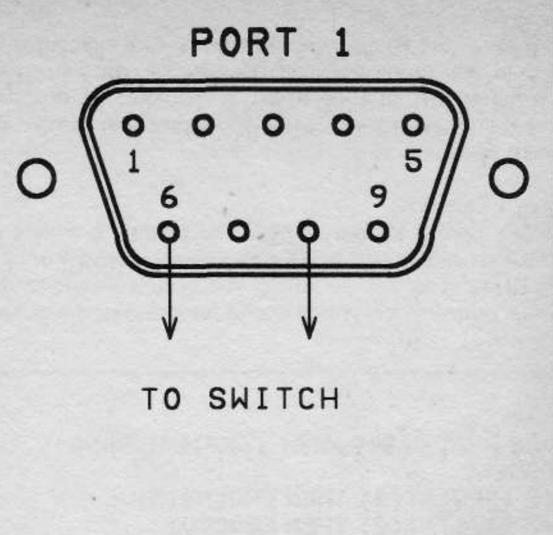


Fig. 7.7. Connecting a foot-switch to Port 1.

It is assumed that the initial program will be program 0, and that on each operation of the foot-switch the program number must be incremented by one. The program number is held in variable "PRG" and is set at an initial value of 0. This value is never transmitted, as it is assumed that the instrument will be set to program 0 via its controls, but obviously a simply routine to transmit this initial value could easily be added at the beginning of the program if desired. The MOUSE function reads several parameters into four integer variables, and in this case it is bit 1 of the value in B% that is of interest. Port 1 is repeatedly read by a REPEAT...UNTIL loop until this bit is set to 1 (which occurs when the foot-switch is activated). The program then branches to PROCCHANGE which increments "PRG" and outputs it to the MIDI output after first sending the program change header byte. The rest of PROCCHANGE provides a delay and a dummy read of port 1 before returning to the main body of the program. These are required in order to avoid multiple reads of the foot-switch, and consequent multiple jumps in the program number.

One slight problem with this setup is that very brief operations of the switch can pass unnoticed by the program. This is not really a serious drawback in practice since the foot-switch would not normally be closed for such a brief period anyway.

A second version of the program, shown below enables the user to select the output channel for the program change messages. The original version sends the messages on channel 1, and is adequate for most purposes, but this version can be used if operation on various channels will be required. This program includes a reset facility that cycles the program back to 0 if an attempt is made to raise it beyond the maximum valid value of 127. If your instruments can not handle the full 0 to 127 range, the appropriate lower figure should be

```
REM IMPROVED PROGRAM CHANGE PEDAL PROG
INPUT  "ENTER CHANNEL NUMBER (1 TO 16) "  CH
CH = CH - 1
PRG = 0
REPEAT
MOUSE X%,Y%,B%,K%
IF (B% AND 2) = 2 THEN PROCCHANGE
UNTIL FALSE

DEF PROCCHANGE
PRG = PRG + 1
IF PRG = 128 THEN PRG = 0
OUT 3,192 + CH
OUT 3,PRG
TIME = 0
WHILE TIME < 400
WEND
MOUSE X%,Y%,B%,K%
ENDPROC
```

used in the program (many instruments have a range of 0 to 63 or 0 to 99).

This is just one way of tackling program changes. It would be quite possible to have a program that permitted the user to select a sequence of program numbers, rather than just having a simple increment on each operation of the foot-switch.

MIDI Terminal

The program shown immediately below, is a sort of MIDI terminal, and it displays values received on the MIDI input as well as letting you output values to the MIDI output port. This is useful for checking purposes, and it can give access to features of an instrument that are not accessible via its controls. For example, some instruments can only be set to "local off" (and back to "local on" again) by way of MIDI messages.

The program loops indefinitely checking to see if there is any data to be read from the MIDI input or the keyboard. If some MIDI input is detected, the program branches to PROCPRINT. Here it reads the MIDI input port and prints the returned value on the screen. If data from the keyboard is detected, the program branches to PROCOUT. Here an INPUT instruction reads the value from the keyboard (which should be terminated with a "RETURN") and then sends it to the MIDI output. You must enter the messages one byte at a time, and in decimal form. For instance, values of 176, 122, and 0 would select "local off", and values of 176, 122, and 127 would switch back to "local on" again. If you will need to send certain MIDI messages time and time again, it would be a good idea to produce a program that will automatically send these messages, with each message perhaps being initiated by pressing a certain key. Fast BASIC provides easy access to GEM menus, and these offer another method of initiating the required messages.

The speed at which data can be read from the port is restricted by the speed at which the ST prints characters on the screen. If large amounts of data must be rapidly read into the computer, it would be better to store the values in a block of memory so that they can be given a detailed inspection afterwards via a suitable routine.

```
REM FAST BASIC MIDI TERMINAL PROG
REPEAT
IF INPSTAT(3) THEN PROCPRINT
IF INPSTAT(2) THEN PROCOUT
UNTIL FALSE

DEF PROCPRINT
A = (INP(3) AND 255)
PRINT A
ENDPROC

DEF  PROCOUT
INPUT B
OUT 3,B
ENDPROC
```

```
REM FAST BASIC MIDI DELAY PROG
INPUT   "ENTER NOTE SHIFT (IN SEMITONES)" INC
INPUT   "ENTER DELAY IN 200THS SEC  "   DEL
FOR FLUSH = 1 TO 200
IF INPSTAT(3) THEN DUMMY = INP(3)
NEXT

REPEAT
A = (INP(3) AND 255)
IF A > 127 AND A < 161 THEN PROCRAISE ELSE PROCNORM
UNTIL FALSE

DEF PROCRAISE
TIME = 0
B = (INP(3) AND 255)
C = INP(3)
OUT 3,A,B,C:REM THIS IS OPTIONAL
D = B + INC
REPEAT
UNTIL TIME > DEL
OUT 3,A,D,C
ENDPROC

DEF PROCNORM
OUT 3,A
ENDPROC
```

Effects

These routines should prove useful in their own right, but should also form a good basis for your own MIDI processor programs. With the aid of a computer there are few types of processing that are not possible, including quite advanced functions. As an example, one type of input message can be converted to any desired output message or messages. At a basic level the system could convert program change messages to a controller message, or a different program change type. At a more advanced level a program change message (or any other type) could be changed to a whole series of messages that could set up any MIDI controllable features on subsequent devices in the system. This is an area of MIDI which has so far gone largely "untapped", but MIDI processing is certainly well worth investigating, and provides numerous creative possibilities.

As an example of one of the more interesting effects that can be obtained using MIDI processing, try the delay program shown at the bottom of page 86. It is similar to the harmoniser program, but the altered note is not sent to the MIDI output until a user specified delay has elapsed. Keep this delay quite short (figures of around 5 to 12 work well) and use a sensible pitch offset (say a fifth or an octave). It works best with sounds that have a fairly spiky envelope shape (guitar and harp type sounds for example). The optional line is not needed if you are feeding the processed signal back into an instrument, as it will already have played the unprocessed note (unless the instrument is used in the "local off" mode of course).

Real-Time

The final program, shown overleaf, is a simple real-time sequencer. It filters out MIDI note on and note off messages (nothing else is recorded) and it places the groups of three bytes in an array. It also records times using the Fast BASIC built-in timer (the variable "TIME"). This gives a resolution of 1/200th second, which gives a very accurate reproduction of the original playing during "playback". The variable "TIME" is reset to zero each time a message is stored, and the stored times are therefore the intervals from one message to the next. This avoids having the very large values that could result if "TIME" was only set to zero at the start of the program, and the recorded times were the intervals as measured from the beginning of the sequence. "T%" controls the maximum number of messages that can be stored, and the suggested value gives a maximum of 500 notes (two messages are required per note). This could probably be made much larger if desired.

This program seems to work quite well, and seems to be able to handle as many notes as a fairly agile set of fingers can play on the keyboard. Aftertouch and pitch bend data will be filtered out and will not eat up memory, but it is possible that these could upset the timing of the circuit slightly. In order to record this type of data properly a faster language than Fast BASIC might be required, and expanded memory could also be useful on a 520ST.

There is plenty of scope for experimentation with this program. As it stands, data is recorded with whatever MIDI channel number it is received with. The stored values could be processed to change the channel number, or this could be done at playback. However, avoid too much processing at playback or the timing accuracy of the reproduced sequence could be adversely affected. The note durations are held in memory, and by manipulating these values the tempo of the sequence could be altered. By rounding the times up and (or) down it is possible to introduce quantisation. You could even add a facility to enable note values to be edited. What would probably be the most useful addition would be a routine to enable sequences to be saved on disc and reloaded again.

Hardware

For the technically minded, the MIDI interface of the ST computers is provided by a 6850 ACIA (asynchronous communications interface adaptor). This has its data register (read and write) at address &HFFFC06, and the status/control register at address &HFFFC04. Bit 0 of the status register is the "Receive Data Register Full" bit, and bit 1 is the "Transmit Data Register Empty" flag. In practice it should not be necessary to control the device directly, and it should be possible (and much easier) to read/write MIDI data via the operating system. In fact direct control of the 6850 ACIA would probably only be possible if the operating system could be persuaded not to control it. Writing a value of 21 to the control register keeps the right baud rate and word format, but disables interrupts. This seems to keep the 6850 away from the attentions of the operating system, if you should need to use direct control for some reason.

```
REM FAST BASIC REAL-TIME SEQUENCER
T% = 1000
DIM notestore| (T%,2),timestore%(T%)
F%=FALSE
N%=0

REPEAT
CLS
IF N%=T% THEN PRINT  "Note Store Is Full"
PRINT  "Press R To Record"
PRINT  "Press P To Playback"
K$=GET$
IF K$= "R"  THEN PROCrecord
IF K$= "P"  THEN PROCplay
UNTIL FALSE

DEF PROCrecord
PROCflush
CLS
PRINT  "Press Any Key To End"
N%=0
REPEAT
F%=INPSTAT(2)
IF INPSTAT(3) THEN PROCstore
UNTIL F%
ENDPROC

DEF PROCstore
A=(INP(3) AND 255)
IF A>127 AND A<161 THEN
notestore|(N%,0)=A
notestore|(N%,1)=INP(3)
notestore|(N%,2)=INP(3)
PROCtime
N%=N%+1
IF N% = T% THEN F%=TRUE
ENDIF
ENDPROC

DEF PROCtime
IF N%=0 THEN
timestore%(N%)=0
ELSE
timestore%(N%)=TIME
ENDIF
TIME=0
ENDPROC

DEF PROCplay
CLS
PRINT  "Playing..."
FOR P% = 0 TO N%
TIME = 0
REPEAT:UNTIL TIME>timestore%(P%)
OUT 3,notestore|(P%,0),notestore|(P%,1),notestore|(P%,2)
NEXT P%
ENDPROC

DEF PROCflush
FOR flush = 1 TO 200
IF INPSTAT(3) THEN dummy = INP(3)
NEXT flush
ENDPROC
```

Index