Construction project:

Universal Midi Interface

Enter the exciting world of Midi music with this low cost interface. It's easy to build, and is simply interfaced to almost any standard PC via the Centronics-type parallel printer port.

by ROB EVANS

Buy a new electronic musical instrument these days, and chances are it will be equipped with sockets labelled "MIDI". MIDI is an acronym for Musical Instrument Digital Interface, which is a hardware and software specification for the transfer of data between electronic musical instruments. The nature of this standard allows remote access to almost every control aspect of a MIDI equipped instrument, via the MIDI sockets.

A complete discussion of the MIDI standard appeared in the January issue of EA, including typical codes and applications. One of the most versatile applications mentioned is MIDI control from a personal computer (PC). This is usually achieved via an interface unit, designed to provide the correct word format and baud rate, as defined by the MIDI specification.

Most MIDI interface units are designed for a specific computer, and are accessed via the expansion bus or user port. Naturally, they will not work on other computers without some (or considerable) modification. In the event of a computer upgrade for example, your expensive MIDI interface is ready for its gold watch! The essence of this problem is not the actual design of an interface, but the lack of a common computer connection system. If a universal high speed port was available, the specialized MIDI interface would be quickly replaced by a generally available, low cost unit.

The closest thing to a universal computer port is the "Centronics"-type parallel printer socket found on virtually all machines, due to a printer being the most popular peripheral. The data and handshaking lines of this port tend to have a common format and pinout, which is ideal for conversion to the MIDI standard. The parallel data can easily be processed by a parallel to serial device (for example, a UART), which may be configured to produce the correct MIDI serial word format.

This method has the advantage of complete compatibility to most computers, and ease of programming — since the interface is addressed as if it were a printer. The disadvantage is that the printer port will only *transmit* information, preventing the interface from receiving MIDI data.

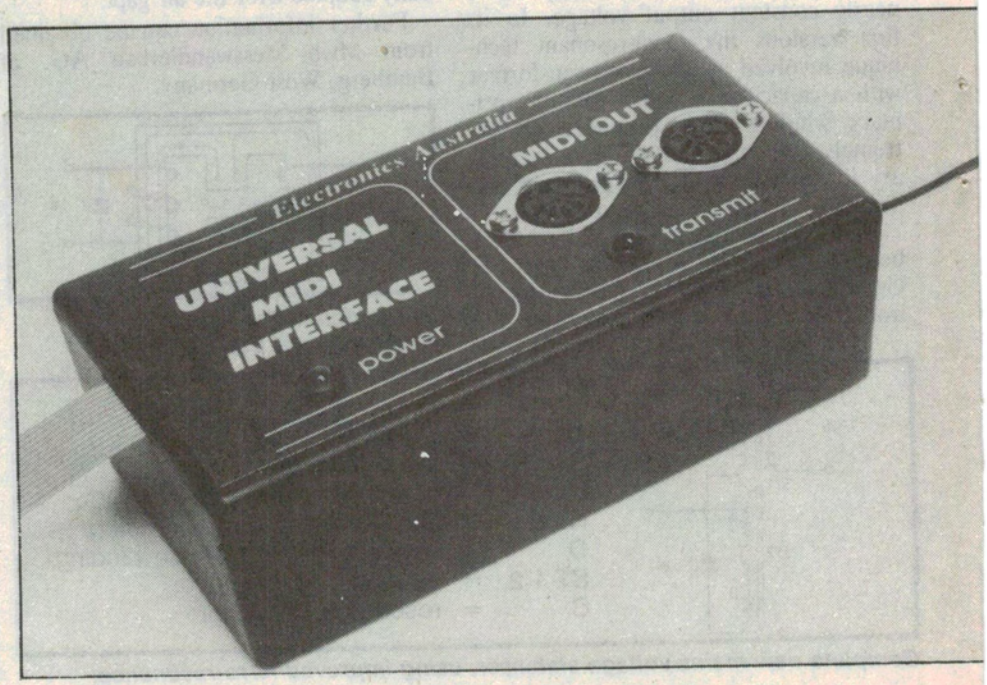
The lack of a MIDI input is not too serious, considering that most sequencing and control applications only involve transmitted data. Many hours of machine code programming are required if a computer is to cope with the speed of incoming MIDI data bytes. In fact, the idea of a universal interface is compromised if the software is difficult to write, and will not readily transfer to another computer.

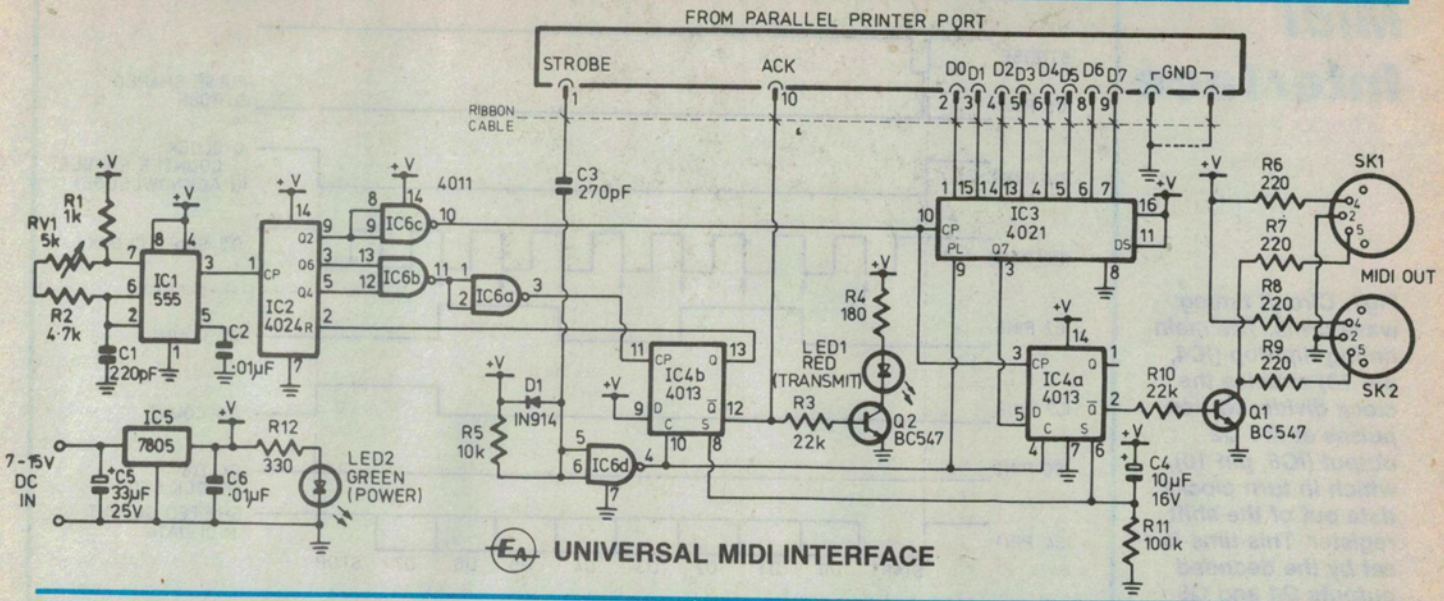
The design

The initial prototype for the EA MIDI Interface was put together in a matter of hours, then connected to the lab PC and a synthesizer. With a few lines of BASIC programming, the EA corridors reverberated with some very dubious music! Clearly, the unit was easy to install and responded to very simple programming.

This initial design was based on a UART handling the parallel to serial conversion, and raised a few questions regarding cost and availability of these specialised 40-pin chips. The last thing a universal interface needs is a chip that is not universally available! Also, only half the chip was being used (effectively a UAT rather than a UART).

After some research it appeared that a "discrete" design using common





The circuit is based around a standard shift register (IC3), while the remaining logic controls the timing.

CMOS chips was not only lower in cost and smaller, it was far more interesting than an LSI "black box".

The final result is a interface for anyone who wants to talk MIDI via their computer. Perhaps it should be named "The People's Interface" or maybe the "Volksmidi"!

The circuit

The circuit of the Universal MIDI Interface can be divided into two main sections, the parallel to serial conversion and the timing circuits. When a Strobe pulse is received from the printer port, the timing circuitry allows the parallel data to be shifted to the serial output for the correct number of clock cycles. The Acknowledge line is then pulsed to indicate that the interface is ready to process another parallel byte.

The master clock for the timing process is based around a simple 555 timer (IC1) running in astable mode at 250kHz. According to the MIDI specification, the serial data must be clocked out with an accuracy of +/- 1%. The 555 clock has this level of stability in our prototype, which avoids the expense of a crystal locked oscillator. In practice, we found that a clock error of more than 3% was required before the MIDI data was rejected by the receiving instrument.

A 7-stage binary counter (IC2) divides the clock by 16, to produce the 31.25kHz MIDI clock at its Q2 output. This is inverted by IC6c to provide the appropriate edge (see Fig.1) to clock the shift register formed by IC3 and IC4a.

The clock divider (IC2) is normally held in a reset state by the set condition of IC4b, a D-type flipflop. IC4b is then cleared by a narrow pulse version of the Strobe pulse from IC6d, thereby enabling IC2 and clock pulses to the shift register.

A logical AND is then performed by IC6b on the Q4 and Q6 outputs of the clock divider IC2; this decodes the 10th clock pulse. The result is inverted by

```
00010                                REM. ALL NOTES OFF * *
00020 LPRINT CHR$(144);
00030 FOR A=36 TO 96
00040 LPRINT CHR$(A);CHR$(0);
00050 NEXT A
00060 INPUT A1$
00100                                REM. NOTE TEST * *
00110 LPRINT CHR$(144);             :REM. SEND "NOTE ON" STATUS
00120 FOR X=36 TO 96                :REM. SET NOTE RANGE
00130 LPRINT CHR$(X);CHR$(64);      :REM. SEND NOTE, VEL 64
00140 FOR Y=0 TO 200: NEXT Y        :REM. NOTE ON TIME
00150 LPRINT CHR$(X);CHR$(0);       :REM. SEND NOTE, VEL 0 (OFF)
00160 NEXT X                        :REM. NEXT NOTE
00170 INPUT A1$
00190                                REM. PROGRAM CHANGE DEMO * *
00200 FOR X=0 TO 31                 :REM. SET RANGE
00210 RESTORE
00220 LPRINT CHR$(192);CHR$(X);:REM. SEND PROGRAM CHANGE
00230 READ Z                        :REM. READ NOTE DATA
00240 IF Z=-1 THEN NEXT X
00250 IF Z=-2 THEN GOTO 270
00260 LPRINT CHR$(Z);:GOTO 230      :REM. SEND NOTE DATA
00270 FOR B=0 TO 100:NEXT B         :REM. DELAY LOOP
00280 IF X=31 THEN EN
00290 GOTO 230
00300 DATA -2,144,48,64,-2,48,0,-2,60,64,-2,60,0,-1
```

These simple BASIC programs will help in getting the system up and running, and provide a starting point for more serious programming.

Midi Interface

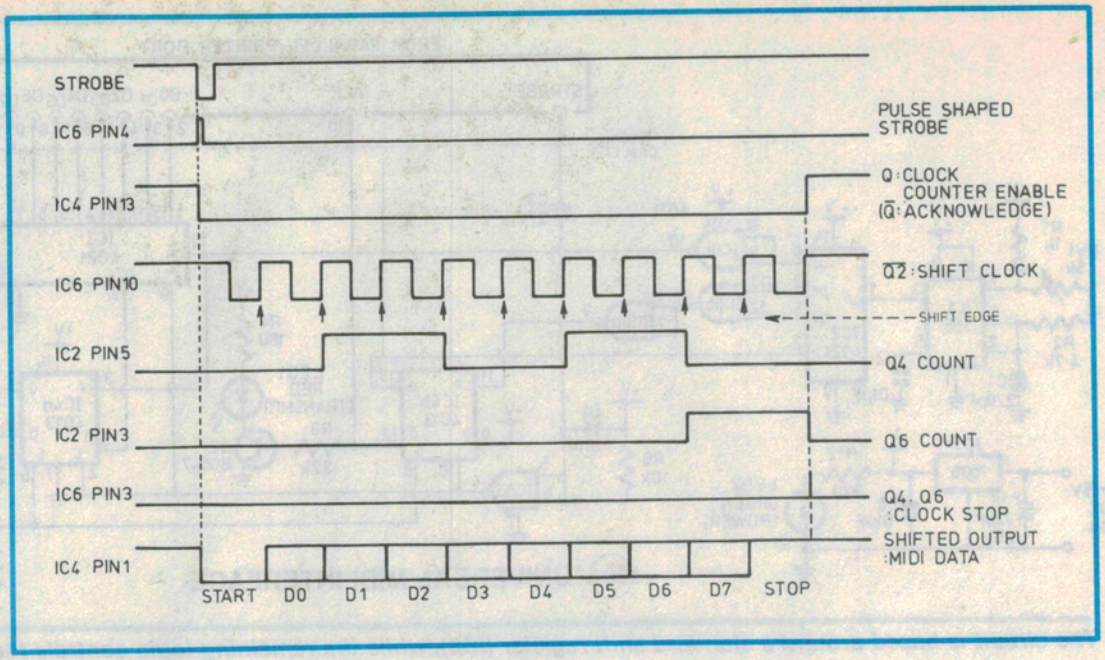


Fig1. Circuit timing waveforms: The main timing flip-flop (IC4, pin 13) enables the clock divider for ten pulses at the Q2 output (IC6, pin 10), which in turn clocks data out of the shift register. This time is set by the decoded outputs Q4 and Q6 (IC6, pin3).

IC6a and applied to the clock input of the timing flipflop IC4b. The rising edge of this pulse clocks a hardwired HIGH at the D input of IC4b to its Q output. This high level resets the clock divider IC2, halting clock pulses to the shift register; thus ending the transmission.

The above explanation is most easily followed by referring to Fig.1, which shows the timing waveforms. In summary, the timing is controlled by the action of the flipflop IC4b, which is reset by the Strobe pulse, and finally set after the appropriate number of clock cycles as decoded by IC2.

Internally, the main shift register IC3 is simply a string of D-type flipflops with assorted logic, allowing each output to be set or cleared in response to the parallel inputs. These inputs will "jam" each shift register stage to a logic level matching each bit of the data word (D0 to D7). This action is enabled by a high level pulse on the Parallel Load input (PL), which in our case is derived from the Strobe signal.

When each rising edge of the clock pulse is detected, the loaded data will be serially shifted out of the register (at Q7) as the MIDI word, minus the start bit.

IC4a is effectively another stage of the shift register, tacked on the end. Its function is to produce the MIDI Start bit, which is always a low logic level, and to ensure a nominal HIGH output when the interface is not transmitting.

Again, the shaped Strobe pulse instigates the timing, in this case by clearing IC4a. The start bit is then terminated by the arrival of the next clock pulse and data from IC3. The last bit of this data will always be a HIGH, due to the Serial Data input (DS) of IC3 being tied to the +5 volt rail. Therefore, we automatically have a Stop bit (logic high), and the correct logic during the rest period between transmitted bytes.

The MIDI data stream appears with the correct logic polarity at the Q output of IC4a, however the Q-bar output is used due to the inverting nature of the output buffer Q1. This buffer provides the current sourcing for two standard 5mA MIDI loops.

An Acknowledge signal is sent to the printer port from the Q-bar output of IC4b. This line will return to a low logic level when the timing sequence is completed. The computer then knows that the MIDI word has been transmitted (or "printed"!), and will apply another Strobe pulse if it has the next byte ready on the parallel data lines. A transmit indication is provided from the Acknowledge line via the action of Q2, which illuminates LED1 during the cleared state of IC4b.

A power-on reset (or in this case; set) pulse is delivered to IC4a and IC4b by the charging action of C4 via R11, as the supply rail rises to its full value. The 5 volt regulator IC5 and associated circuitry have been included to allow the interface to be run from a plugpack or any convenient voltage source. Therefore the unit has the independence of a printer, although it's not nearly as heavy! Naturally, this regulated supply (IC5 etc.) may be omitted in favour of an accessible 5-volt source from the computer.

Software

Programming the Universal MIDI Interface is extremely simple. This is due to the printer driver program which is inherent in most computers supporting a

printer port. The BASIC instructions "PRINT" or "LPRINT" will automatically address the printer port (in this case, the MIDI interface), and activate the Strobe and Acknowledge lines. The character (CHR$) statement should be used so as to avoid ASCII codes, and a semi-colon (;) added to prevent carriage returns. For example, the MIDI status byte for "Note On" is sent as:

LPRINT CHR$ (144);

If a *large* MIDI system is to be implemented, the program may need to be written in machine code. This is because the execution time of a BASIC program may become significant when transmitting intense MIDI data. The interface itself is capable of transmitting a continuous stream of data bytes, without timing delays.

For the majority of applications, a BASIC program will offer more than adequate performance from the MIDI interface. As a starting point, a couple of simple examples can be found at the end of this article.

Construction

The Universal Interface is quite easy to construct, for all of the components except the MIDI output sockets and indicator LEDs are contained on one PCB measuring only 45x110mm (code: 88ms1).

Before any construction actually begins, the PCB should be checked for any bridged tracks due to incomplete etching. Quite a thorough check is worthwhile at this point, for this may prevent the infamous "frazzled constructor syndrome" when a PCB fault is encountered in final testing! Also, the corners of the PCB may need to be trimmed to clear the lid mounting posts.

The easiest way to begin construction is to tackle the lower profile components first. This allows the PCB to lie evenly with the copper side facing upwards while the component legs are being soldered. The overlay guide should be carefully followed for the correct component orientation, and the usual static and earthing precautions taken for the CMOS logic chips.

After mounting the larger components, short lengths of wire may be soldered to the appropriate PCB pads for later connection to the front panel sockets and LEDs. The 12-way ribbon cable may also be soldered to the appropriate pads, although the order of the wires will depend on the terminating connector arrangement.

Many PCs use a DB25-type socket for the parallel printer port, the most likely connector required to terminate the 12-way ribbon cable is a DB25-type line plug. When such a PC is interfaced to a printer with a Centronics-type connector, a DB25 to Centronics adaptor cable is used. Therefore, if the interface is required to connect *directly* at the point where a Centronics equipped printer would be, a Centronics-type (or a 36-way Amphenol 57N series!) socket is necessary.

It's worth bearing in mind the relative cost when choosing between the two connectors, for the Centronics socket tends to be around four times the price of a suitable DB25 plug. In fact, a Centronics socket costs around the same as the parts required for the entire MIDI Interface.

The order of the Strobe, Acknowledge, and Data bit lines are the same for both styles of connectors. The Strobe is pin 1, Acknowledge is pin 10, and Data bits 0-7 are pins 2-9 respectively. The difference is simply the ground connections, which are 18-25 for the DB25, and 19-30 for the Centronics connector.

Some computers (e.g. IBM PCs and compatibles), may check the Busy and Paper End lines during their print routine. In this case, the Paper End line (pin 12) should be grounded and the Busy line (pin 11) tied to the Acknowledge (pin 10). The most convenient position to connect these links is at the DB25 or Centronics connector.

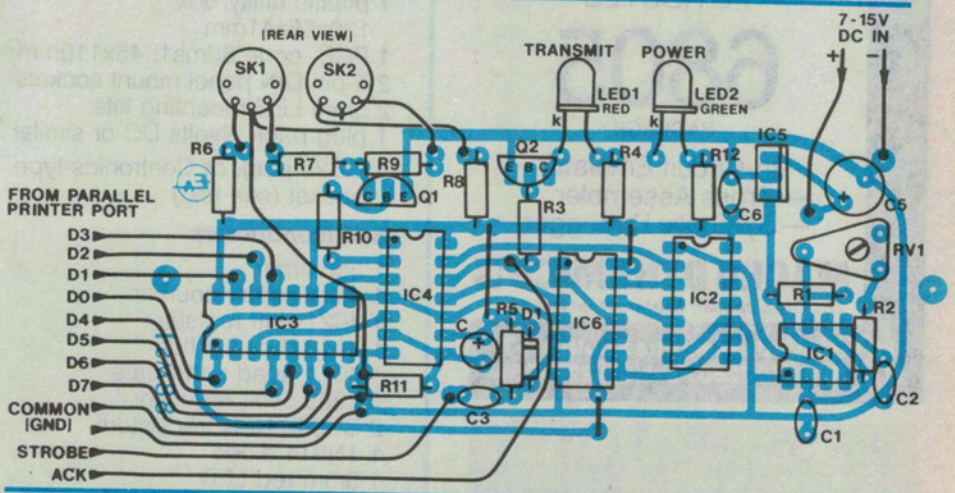
The final stage of construction is to prepare the box for the mounting of the PCB and other components. The Dynamark front panel may be attached, and holes drilled for the LEDs and DIN sockets. Care should be taken at this point, for drill bits have a nasty habit of destroying front panels.

Holes are then drilled for the power supply lead (if applicable) and the PCB mounting screws. The ribbon cable enters under the lid via a slot filed in the top of the box body. Finally, all of the parts are mounted in the box, and inter-

When BASIC isn't!

There is a trap when using some of the more recent forms of BASIC (Microsoft GW Basic etc.). If ASCII code 13 is sent via the printer port, the program interprets this as a printer Carriage Return and immediately sends a Line Feed code (10). We can hardly blame the software for this, as it believes that a *printer* is connected to the parallel port, rather than a Midi interface!

This problem may be tackled in a couple of ways. The first (and preferable) solution is to consult the computer's software manuals for a method of suppressing this automatic Line Feed. The second method is to use an offset when a data code of 13 is required. For example, a synthesizer with 32 selectable voices (programs) can be tricked into selecting program 13 by sending code 45 (an offset of 32), or code 77 (an offset of 64). A 64 voice capability may be treated in a similar way; that is, sending an offset of 64 (code 77 for voice 13).



Wiring and PCB overlay: The power supply components (IC5 and C5) may be omitted if an external 5 volt supply is to be used.

wired according to the overlay diagram.

Testing and programming

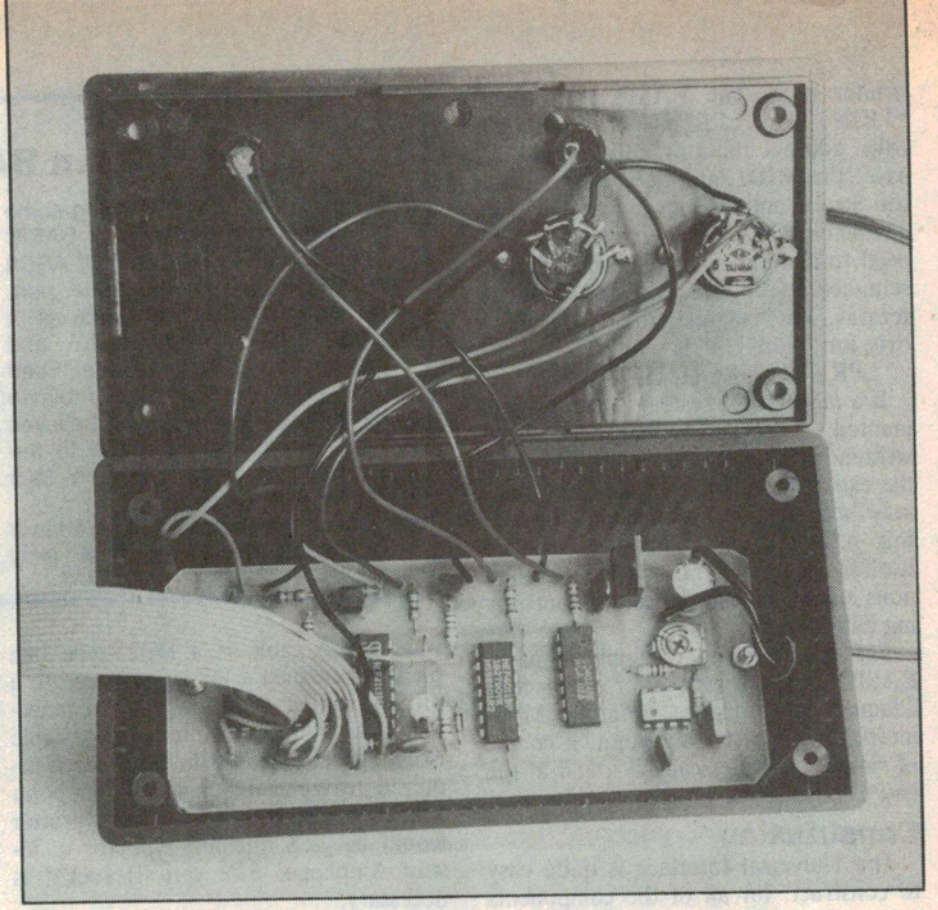
The initial tests on the MIDI Interface should be completed with only the power supply connected. Use a multimeter or CRO to check the 5 volt supply and the "power-on set" condition of IC4a (pin 1 HIGH). The master clock may be set to 250kHz by adjusting RV1, while monitoring the output with a CRO or (preferably) a frequency counter.

If the above mentioned test instruments are not available, the clock frequency can be adjusted on a trial and error basis. Assuming the circuit is working correctly, the Interface should be connected to suitable MIDI instrument and the printer port of a PC. Then run a simple, repetitive program (such as the supplied listing) while adjusting the clock frequency trimpot RV1 for reliable operation. Note that the MIDI channel number of the instrument must match the MIDI channel number encoded in the program.

The asynchronous nature of MIDI information means that if the transmission of note data is interrupted, the instrument will effectively "hang" until it receives further instructions. Therefore, when the instrument rejects data arriving at an incorrect rate (wrong setting of RV1), notes will continue to sound until matching "note off" bytes are received.

This can be quite confusing when the selected sound on a synthesizer has a natural decay; the notes will fade away despite the lack of "note off" messages. If the instrument is 8-voice polyphonic (8 separate oscillators) for example, each oscillator may eventually be assigned to a note that is not sounding. Now, *some* synthesizers may not respond to the 9th "note on" data, leaving the instrument effectively "dead".

Despite this complicated sequence of events, the solution is very easy. Simply turn the instrument off, and turning it on again will reset the oscillators. Another solution is provided by lines 20 to 60 of the program in this article, which



PARTS LIST

1 plastic utility box, 130x68x41mm
1 PCB, code 88ms1, 45x110mm
2 5-pin DIN panel mount sockets
2 5mm LED mounting kits
1 plug pack, 9volts DC or similar *
1 DB25 plug, or Centronics-type socket (see text)

Semiconductors

1 555 timer
1 4024 binary counter
1 4021 shift register
1 4013 dual flip flop
1 4011 quad nand gate
1 7805 5volt regulator *
2 BC547 NPN transistors
1 1N914 diode
1 5mm red LED
1 5mm green LED

Capacitors

1 220pF ceramic
1 270pF ceramic
2 10nF metallised polyester
1 10uF 16VW electrolytic (PC mount)
1 33uF 25VW electrolytic * (PC mount)

Resistors (all 0.25W, 5%)

1 x 180Ω, 4 x 220Ω, 1 x 330Ω, 1 x 1kΩ, 1 x 4.7kΩ, 1 x 10kΩ, 2 x 22kΩ, 1 x 100kΩ 1 x 5kΩ horizontal trimpot

Miscellaneous

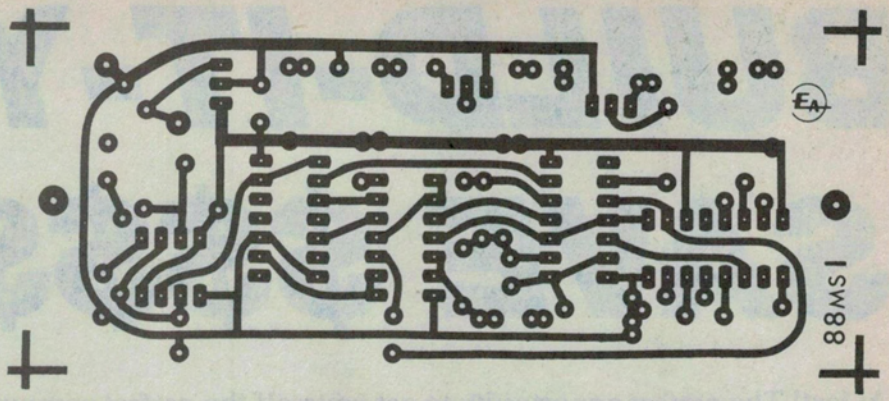
12-way ribbon cable, nuts and bolts, hookup wire, Dynamark front panel.
* Note: parts for optional power supply.NN

sequentially sends a "note off" message to each possible note position. This is a little more convenient, and should be run at the beginning of each sequence when there is a danger of data rejection.

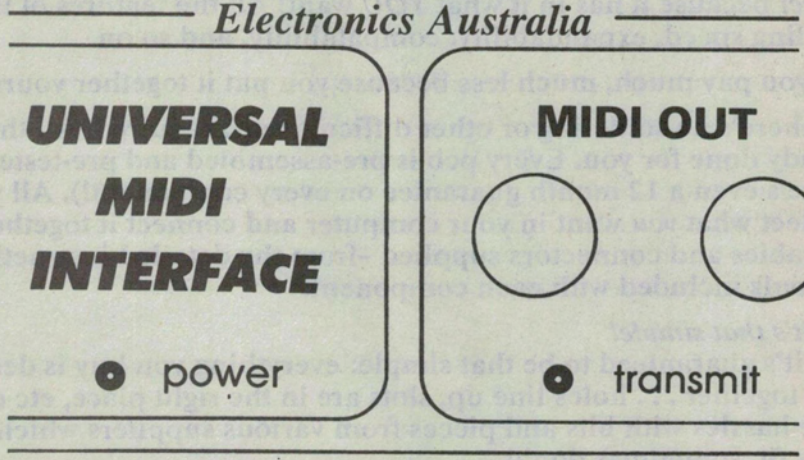
The next section of the program (lines 110 to 170) sounds each note in turn for a period as set by line 140. This will complete a cycle of the possible notes of an average synthesizer (as set by line 120), and is ideal for setting the clock frequency without test instruments.

The last program (lines 200 to 300) demonstrates the MIDI program change capability. This will "play" a couple of notes then change to the next program (or sound), working its way through 32 possibilities (maximum range 127: see line 200). The "INPUT A1$" line is included between programs so the next section will run after a Carriage Return key.

These programs have been included as a starting point for more adventurous programming, which is only limited by your imagination and software skills. However, simple tasks such as a defined series of program changes, or a repetitive bass melody only require a few quick lines of BASIC programming.



Above: Full size reproduction of the PCB artwork.



The front panel artwork for the Universal MIDI Interface, shown full size.