# GPIB combiner

*Up to six standard g.p.i.b. controllers can be simultaneously connected to a set of standard peripherals. A typical application is the sharing of one disc unit and one printer between several CBM Pet computers.*

**by David J. Greaves**

The GPIB, HPIB, IEEE-488 or IEC625 bus is well known and widely used because of the ease with which it may be used to interconnect many peripheral devices. GPIB-compatible devices are made by many manufacturers and range from printers and disc units to industrial power controllers, network analysers and digital multimeters. A dozen or so of these devices are simply attached to one computer or controller using the specially designed connectors which can be plugged into each other. The 24 wires which make up the bus simply run in parallel from one device to the next. The main disadvantage of the bus is that only one controller may be connected to the string of peripherals at one time. However, with the GPIB combiner described in this article, up to six controllers can be linked via separate 24-core cables. A single 24-core cable leads from the combiner to the peripherals which are "daisy-chained" in the usual manner.

The prototypes for this design have been in use for over three years where a varying number of CBM Pet computers have needed to be connected to a varying number of printers and disc units. They have been very successful and the users are rarely aware of the existence of the combiners. However, the Pet does not use all of the functions which are provided for in the IEEE definition of the bus (in particular the serial and parallel poll sequences) and I cannot guarantee that the combiner will work with controllers built by other manufacturers.

Table 1 shows the designations of the 24 lines which make up the bus. The 16 signal lines all use negative logic, i.e. zero volts is logic one. At each device on the bus, each line is driven by an open-collector buffer and at each buffer there is a two resistor arrangement to pull up the line to about +3V when the transistor is off (Fig. 1). In this way all lines of the bus are nodes of a wired-or gate. That is, any device can pull a particular line low, but it is only when all devices are not pulling a line to zero that it floats high. The high state is also the idle state for all lines.

The operation of most of the lines of the bus is not altered by the functioning of the combiner and so a full description of the bus is not presented here. An accurate and readable description of the working of the bus is given in the references.

The bus functions by entering various modes in which different devices have control over different lines. For most communications, only four states are used. These are:

● **Idle mode:** this is the quiescent state which must exist before any communications can occur. All lines are high except possibly for remote enable (DEN) which is hardwired low is some systems. In particular both NDAK and NRFD are high, a state which can always be used to detect the idle mode.

● **Device talker mode:** in this mode the controller is receiving a string of bytes from a device connected to the bus. The bytes are transferred along the bus on the eight data lines $DA_0$-$DA_7$. Handshaking is performed using the three wires NDAK, NRFD and DAV. The talking device drives DAV while the controller drives NDAK and NRFD. When the talker comes to send the last byte in the string, it also pulls EOI (end of identify) low. EOI is normally high. The controller recognizes this signal and will then prepare to terminate the data transfer and return to idle mode.

**Table 1. GPIB line designations**

| Pin name | Type | Mnemonic | Name |
|---|---|---|---|
| 1 | Data | $DA_0$ | Data 0 (least significant) |
| 2 | Data | $DA_1$ | Data 1 |
| 3 | Data | $DA_2$ | Data 2 |
| 4 | Data | $DA_3$ | Data 3 |
| 5 | Control | EOI | End of identity |
| 6 | Handshake | DAV | Data valid |
| 7 | Handshake | NRFD | Not ready for data |
| 8 | Handshake | NDAK | Negative data acknowledge |
| 9 | Control | IFC | Interface clear |
| 10 | Control | SRQ | Service request |
| 11 | Control | ATN | Attention |
| 12 | Control | REN | Remote enable |
| 13 or A | Data | $DA_4$ | Data 4 |
| 14 or B | Data | $DA_5$ | Data 5 |
| 15 or C | Data | $DA_6$ | Data 6 |
| 16 or D | Data | $DA_7$ | Data 7 (most significant) |
| 17 or E | | GND | Ground |
| 18 or F | | GND | Ground |
| 19 or H | | GND | Ground |
| 20 or J | | GND | Ground |
| 21 or K | | GND | Ground |
| 22 or L | | GND | Ground |
| 23 or M | | GND | Ground |
| 24 or N | | GND | Ground |

● **Device listener mode:** this is similar to the talker mode except the controller and device swap control of the lines NRFD, NDAK, DAV, $DA_0$-$DA_7$ and EOI. Hence bytes are transferred from the controller to the device. (Many devices such as printers are listen-only devices.)

● **Command mode:** it is likely that more than one device will be connected to the bus, and so before communications can take place the controller must transmit information to select one (or possibly more) of the devices present. Also, many physical devices have several logical registers within them and these too must be distinguished. The solution is to give each physical device a distinct device address (or device number). The different registers within a device are also given a number – the secondary address. Both halves of the total address are integers in the range 0 to 30. It is this information that is sent in command mode. Command mode can always be identified by ATN (attention) being low.

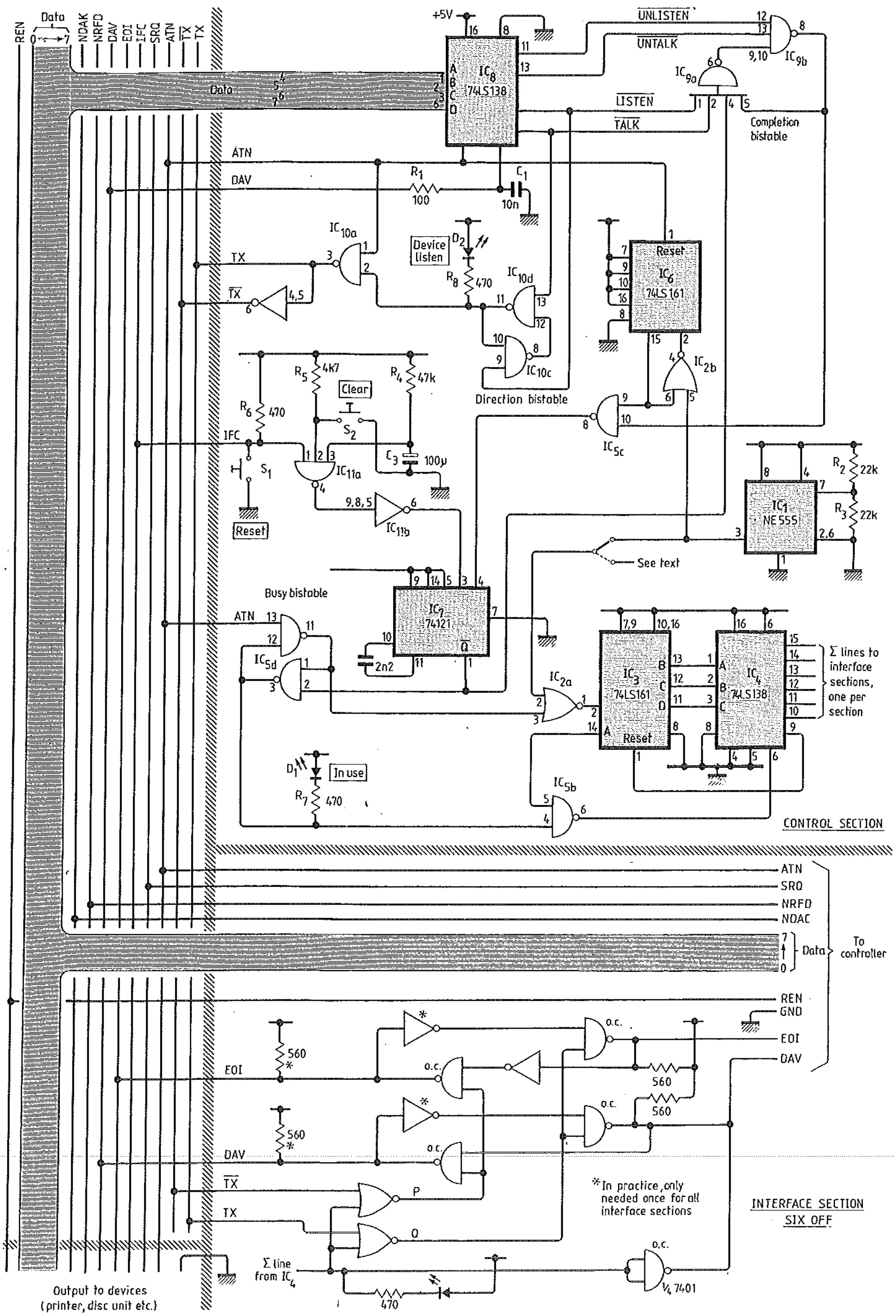
## A typical transaction

A transaction is the complete sequence of events from leaving to returning to idle mode. Table 2 shows the steps of a typical sequence – the reading of one line from an open disc file or a paper tape reader. Ten separate events have been identified starting in the idle mode at event one. For each event shown, the lines DAV, NRFD and NDAK complete one handshake cycle and the byte shown on $DA_0$-$DA_7$ is transferred from one device to another. The byte on $DA_0$-$DA_7$ is true when DAV (data valid) is low. All transactions start when ATN goes low as in event two.

*Event 2:* the controller puts ATN low to wake up all devices and command mode is deemed to have been entered. All devices have to respond at this point since they do not yet know which of them is to be addressed in the coming transaction. Each device temporarily enters device listener mode (although this is more properly called acceptor handshake).

The controller now sends out a command byte selected from Table 3. In this example the transaction will be data coming from a device to the controller, so the

*Fig. 1. In full circuit of the combiner only ▶ one control section is needed but a separate interface section is needed for each user.*


REN
Data
NDAK
NRFD
DAV
EOI
IFC
SRQ
ATN
TX
TX
+5V
IC8
74LS138
UNLISTEN
UNTALK
LISTEN
TALK
IC9a
IC9b
Completion bistable
R1
100
C1
10n
IC10a
Device listen
D2
R8
470
IC10d
IC10c
Direction bistable
Reset
IC6
74LS161
IC2b
IC5c
R5
4k7
R4
47k
R6
470
Clear
S2
C3
100μ
S1
Reset
IC11a
IC11b
R2
22k
R3
22k
IC1
NE555
See text
Busy bistable
IC7
74121
2n2
IC5d
IC2a
IC3
74LS161
IC4
74LS138
Σ lines to interface sections, one per section
D1
In use
R7
470
IC5b
CONTROL SECTION
ATN
SRQ
NRFD
NDAC
Data
To controller
REN
GND
EOI
DAV
560
o.c.
P
Q
*In practice, only needed once for all interface sections
INTERFACE SECTION
SIX OFF
Σ line from IC4
470
¼ 7401
Output to devices
(printer, disc unit etc.)

upper three bits of the command byte are 010. Since the device number that data is being read from is eight, the other five bits are 01000. This byte is communicated to all devices on the bus using a standard DAV, NRFD and NDAK handshake.

*Event 3:* this event occurs only when a secondary address is needed. Again a command byte is selected from table three, but this time the top three bits are 011. This tells the device that the low-order five bits are the secondary address to be used for the forthcoming data transfer. Only the device which recognized its device number during event two will need the secondary address, but all devices handshake the bus while ATN is still low – they are watching out for their own device number. It is because more than one device can be handshaking the bus at one time that the open collector, wired-or structure is used.

At the end of event three the controller puts ATN high and the remainder of the devices which were not selected return to idle mode.

*Event 4:* since this is an example of a device talking to the controller, the controller is now waiting for the first data byte from the addressed device. It sets up NRFD and NDAK and waits for DAV to go low signifying that the byte is present on $DA_0$-$DA_7$. If DAV does not go low within a reasonable length of time then the controller can only assume that device eight is broken, switched off or non-existent.

*Events 5-8.* The controller goes on reading bytes for as long as it likes. The device sourcing the data cannot stop sending bytes even if it has run out of data – it must pad with nulls. However it can indicate that it would like to stop with the EOI line. On the CBM Pet the Basic variable ST is set to 64 when an EOI is detected. If the program then continues fetching further data with, say, an INPUT# or GET# statement then ST is set to 2.

*Event 9:* when the controller has had enough data it again puts ATN low to re-enter the command mode. All devices set up NRFD and NDAK in order to receive a command byte. The controller sends 5FH which is the untalk command.

*Event 10:* finally, the controller puts ATN high. All lines are now in idle mode ready for the next transaction.

## Logically connected transactions

Consider the case where a program has been written on the controller to transfer a file from one device to another, one byte or line at a time. This might be the case when listing a disk file on a printer. In the original IEEE488 standard there is provision for such a process to be set up by the controller and run as a single transaction. However, the average user will not bother to consult lengthy texts to find out how to do this. Instead he will write a simple program such as

```
40 INPUT#1,A$
50 PRINT#2,A$
60 GOTO 40
```

This uses two separate transactions for each line of the file and although this is not particularly undesirable, it does cause the bus to pass through idle mode many times during the execution of the program.

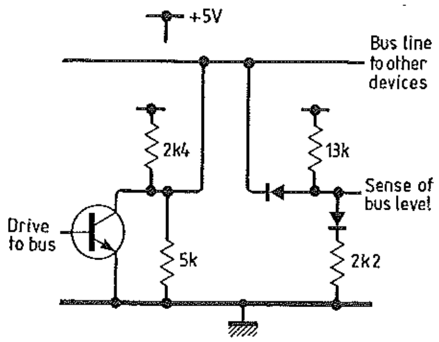



*Fig. 2. Electrical connection used at each device on each line to sense and influence the status of the line.*

From the point of view of the combiner the only way that it can detect that a user has finished with the bus is the presence of the idle mode. However, in this example, the user has not finished and will be immediately submitting another transaction. It is undesirable that another user should interrupt and gain control of the printer (say) during an idle period half-way through the first user's job.

The combiner overcomes this problem by providing a special re-admit period. This is an interval of about 1.5 seconds after a particular user has finished a transaction during which that user alone can submit further transactions. If he has no further work to do then the next controller that is queued is allowed to proceed; or if there is none, the bus becomes free.

There is a disadvantage of the re-admit period: when two users are quite separately using two peripherals it works out that one of them is continually waiting for a 1.5s gap in the other's transaction stream. Only then can they swap roles.

## Circuit description

The combiner consists of eight sections: six interface sections, one control section and a power supply. The sections are interconnected by an internal bus and most of this bus also leads directly off to the peripheral devices. Figure 1 shows one interface section, the control section and the internal bus.

The interface sections are very simple as all except two of the signal lines from the controller lead directly through it to the internal bus. DAV and EOI are only connected through to the internal bus when a particular interface section is enabled by the control section. EOI must normally be isolated because of a limitation on the early Pet computers. To prevent excessive flicker on the Pet screen when scrolling was taking place, a programmable output pin was connected so that the screen could be blanked, under software control, during scrolling. Unfortunately, when all of the outputs from the computer's three versatile interface adaptors (v.i.as) had been assigned, there was none left to use for this blanking signal. Instead, the pin that controls EOI on the g.p.i.b. connector was used for this purpose as well. The two consequences are that the screen sometimes blanks when using the bus and that the computer produces an extraneous EOI pulse every time the screen scrolls. The EOI half of the interface section is therefore provided to stop these spare pulses from interrupting other users' transactions.

The DAV half of the interface section is provided so that the combiner can pull a

**Table 2. Typical sequence of bus events making reading up a bus transaction: reading the line BLUE <CR> from device 8, register 4, in this example.**

| Event | Mode | Control of DAV, EOI $DA_0DA_7$ | NRFD NDAK | Data on $DA_0DA_7$ | ATN | EOI | Comments |
|---|---|---|---|---|---|---|---|
| 1 | idle | – | – | 00-all high | high | high | Quiescent state |
| 2 | command | controller | all devices | 48H | low | high | Controller requests that device 8 should talk |
| 3 | command | controller | all devices | 64H | low | high | 'Secondary address 4' |
| 4 | talker | device 8 | controller | 42H | high | high | Data B |
| 5 | talker | device 8 | controller | 4CH | high | high | transfer L |
| 6 | talker | device 8 | controller | 55H | high | high | U |
| 7 | talker | device 8 | controller | 45H | high | high | E |
| 8 | talker | device 8 | controller | 0DH | high | * | return |
| 9 | command | controller | all devices | 5FH | low | high | Controller commands UNTALK |
| 10 | idle | – | – | 00 | high | high | |

*If this were the last byte in a file, EOI would be low.

**Table 3. Command byte meaning**

| $DA_7$ | $DA_6$ | $DA_5$ | $DA_4$ | $DA_3$ | $DA_2$ | $DA_1$ | $DA_0$ | Interpretation |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | n | n | n | n | n | Cause device nnnnn to go into talk mode |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | Untalk command |
| 0 | 0 | 1 | n | n | n | n | n | Cause device nnnnn to go into listen mode |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | Unlisten command |
| 0 | 1 | 1 | n | n | n | n | n | Use secondary address nnnnn |

Note:- 'nnnnn' is a five-bit binary number in the range 0 to 30. Device number 31 is illegal.

**Table 4. Integrated circuit list**

| IC | Type | +5V | GND | Usage |
|---|---|---|---|---|
| 1 | NE555 | 8 | 1 | Clock |
| 2 | 74LS02 | 14 | 7 | |
| 3 | 74LS161 | 16 | 8 | 'Which controller?' counter |
| 4 | 74LS138 | 16 | 8 | 'Which controller?' counter |
| 5 | 74LS00 | 14 | 7 | |
| 6 | 74LS161 | 16 | 8 | 'Re-admit' timer |
| 7 | 74121 | 14 | 7 | 'End of transaction' monostable |
| 8 | 74LS138 | 16 | 8 | Bus command detector |
| 9 | 74LS20 | 14 | 7 | Completion bistable |
| 10 | 74LS00 | 14 | 7 | Direction bistable et al. |
| 11 | 74LS10 | 14 | 7 | |

controller's DAV line low without pulling the DAV line of the internal bus low. The interface section holds this low unless it is enabled, in which case it connects it to the internal bus. When a controller wants to use the bus, the first thing it does is to test its DAV line to see if it is high. If the controller finds it low, it sits in a software loop until it does go high. This loop or "hang" is the basis of the combiner's queueing system. Because the interface sections are not normally enabled, when a controller wants to start a transaction it normally finds DAV low and goes into a hang.

Circuit $IC_1$ in control section is the master clock and runs at a few hertz. Assuming the busy bistable ($IC_{5a}$ and $IC_{5d}$) is clear, clock pulses are transmitted through $IC_{2a}$ into $IC_3$. $IC_3$ is a four bit counter and is decoded by $IC_4$ so that the interface sections are each enabled in a "round-robin" fashion. If a particular controller has a transaction ready it will see DAV go high as its interface section is enabled and put ATN low (event 1). ATN is connected to $IC_{5d}$ and this sets the busy bistable, stopping $IC_3$ at its present count.

Some Pet users have effected the facilities provided by this combiner purely by modifying the g.p.i.b. handling software within their computers. However, this approach has a number of drawbacks, among which is the problem $IC_{5b}$ is designed to solve. What can happen is that the software checks that the bus is in idle mode (i.e. not being used) and then starts its transaction with the assertion of ATN. Meanwhile, on another user's machine the same software can be running just a few microseconds further on. Because time must elapse between the execution of the idle mode check and the execution of the instructions which assert ATN, both controllers get the impression that the bus is free and both assert ATN at once. On the combiner described here, there is a safety period of one clock cycle after an interface section has been enabled and then disabled during which the controller can still assert ATN and gain control of the bus.

The buffers in the interface sections are bidirectional and need to know whether it is a controller or a peripheral device which is driving DAV and EOI. This information is sent along the TX lines which are controlled by $IC_{10a}$ in the control section of the combiner. If ATN is low then it is command mode and the controllers drive DAV and EOI. $IC_8$ is a 3-to-8 line demultiplexer and provides a simple way of recognizing the command bytes that the controllers send. In fact it is a bit too simple since it does not work with device numbers in the range 16-30! A more complex detector must be used instead of $IC_8$ if it is desired to use these device numbers. The direction bistable ($IC_{10c}$ and $IC_{10d}$) is set depending on whether a talk or listen command byte is detected and when ATN again goes high at the end of the command mode (event 3 in the example) the TX lines take up the information from this bistable.

All the circuitry in the combiner remains passive until the command mode is again entered at the end of the transaction. When either the unlisten or untalk command byte is sent, $IC_8$ detects it and sets the completion bistable ($IC_{9a}$ and $IC_{9b}$). $IC_6$, another four bit counter, and $IC_{2b}$ together form a retriggerable monostable capable of high duty cycles. Whenever ATN goes low, $IC_6$ is reset, and when it goes high, clock pulses from $IC_1$ feed through $IC_{2b}$ causing it to count up until it reaches 15. This takes about 1.5 seconds and provides the re-admit period after ATN has gone high and the bus finally returned to the idle mode. If no new transactions are started by the enabled controller during this period, then the completion bistable will still be set and the busy bistable will be cleared. The circuitry which does this is $IC_{5c}$ and $IC_7$. However,

continued on page 62

*David Greaves is an engineering student at St John's College, Cambridge. He has designed and built many projects including a digital spectrum analyser and a powerful microcomputer system using three microprocessors. He is currently working on a polyphonic keyboard synthesiser. He is a three times winner of the Design Technology competition sponsored by Esso, a holder of an IEE Jubilee Scholarship and was once a contestant on BBC TV's 'Young Scientists of the Year' programme.*

*An enthusiast for most types of music, he plays the guitar. Other interests include archery, sailing, canoeing and badminton.*

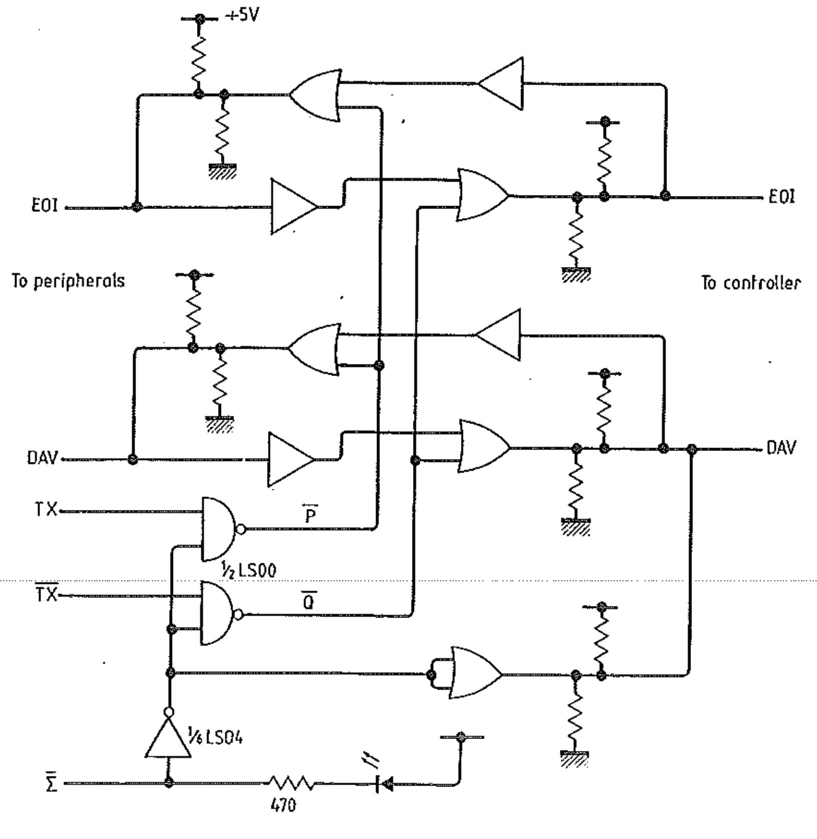
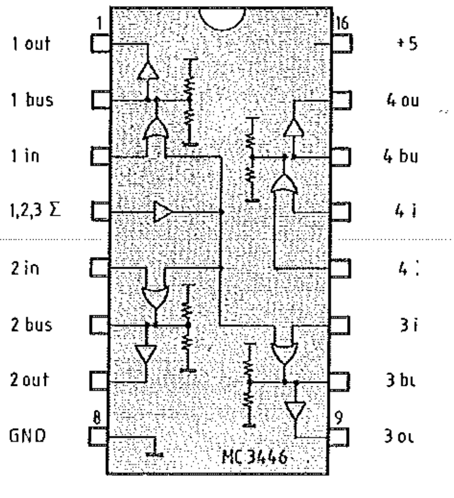



*Fig. 3. (a) Alternative circuit for the interface sections using the MC3446 integrated circuit (b) which is specifically designed for use with the g.p.i.b.*



# GPIB combiner *continued from page 28*

if a new transaction does start, either a talk or listen command will occur and reset the completion bistable. Then, when the readmit period completes, nothing will happen.

Because the re-admit period and "round robin" polling of the controllers both use similar circuits, there is an average latency of half the re-admit period on all one-off transactions, even if the bus is completely free. To overcome this, on one prototype a second NE555 counter was fitted running at about 150Hz. This was connected into $IC_{2a}$ instead of $IC_1$ thereby increasing the response time. The colossal disadvantage of this is that the pretty display caused by the interface section l.e.d.s flashing in turn is completely lost.

The two push buttons are termed clear and reset. Pushing "reset" generates an interface clear signal (IFC) and resets all devices on the bus. If any old-style Pet disc units are connected then they will need re-initializing. Pushing "clear" is generally a good way to get rid of a "bus-hogger".

### Alternative interface sections

An alternative circuit for each of the six interface sections, Fig. 3(a) uses the MC3446 integrated circuit of Fig. 3(b). The MC3446 contains four-line driver-receivers specifically designed for interfacing with the bus, their equivalent circuit being exactly as shown in Fig. 1. The driver transistor is guaranteed to sink 48mA and an absolute rating of 150mA is specified.

If very long lengths of cable are to be used, then buffers similar to those already on EOI and DAV can be made up for the other lines of the bus. These extra buffers must still be enabled only when the enable line from the control section is low and must buffer in the correct direction depending on the TX signal. That is, $DA_0$-$DA_7$ always go in the same direction as EOI and DAV, NDAK and NRFD always go in the opposite direction, ATN and REN always go from the controller to the peripherals and SRQ always goes from the peripherals to the controller. IFC is best neglected.

When one user is performing a large transaction such as printing a listing on a printer, other users who are queued and awaiting access to the peripherals may change their minds and decide to do something else that does not use that peripheral. In this case some means of exiting from the queue or "unhanging" their controller is required. In the case of the Pet computer, pressing buttons on the keyboard is to no avail, the only solution being to unplug the g.p.i.b. connector from the rear of the computer. To reduce connector wear, it may be preferable to fit four pole push-to-break buttons in series with the lines NDAK, NRFD, DAV and ATN. Operation of this switch has the same effect as removing the connector. The controller sees that DAV has gone high, asserts ATN, finds that NRFD and NDAK are both high (an error condition) and aborts its transaction.

### References:

Fisher, E. Jensen, C. W., The Pet and the IEEE 488 bus. Osborne/McGraw-Hill.

Jackson, P. J., IEC/IEEE data transmission bus interfaces, *Mullard Technical Communications*, vol. 14, no. 138, April 1978, p.290.

IEEE bus standard, by P. R. Ellefson, *Wireless World*, June/July 1980, pages 75-78.

## Improving colour television decoding

We regret that David Read's final articles describing a one-line PAL comb decoder have been postponed due to pressure on space. In the meantime you may be interested to read of the new PAL modifications proposed for North America, described on the previous page.