Timing data transfer

A simple technique for measuring the speed of data transmission between microcomputers.

by Philip Barker Ph.D.

Undoubtedly, although fibre-optic transmission systems are growing rapidly in importance, the most popular techniques for interlinking localized computers are still based upon the use of some form of wire cabling. The speed of transmission that can be achieved with cable systems depends upon the type of cable used, the nature of the interface circuits that are employed and higher level factors such as the type of software data exchange protocols and error checking that is performed on the data.

We have recently been involved in the interlinking of a variety of different microcomputer systems[1,2]. The work that has been undertaken was orientated towards an investigation of the use of multiple microprocessor networks as a means of improving the user interface with microcomputer database systems: experiments designed to measure the speed of data transmission between some of the component computers arose as an ancillary interest and gave rise to a simple technique for measuring data transfer speed.

Measuring procedure

During the transmission of data between two micros, one acts as the transmitter while the other acts as the receiver of data. A third microcomputer, attached to the transmitter, can be used to measure the duration of the data transmission transaction, and is referred to as the timer. The experimental arrangement is depicted schematically in Fig. 1(a). Communication between the timer and the transmitter is via an appropriate i/o port within the latter: if such a port is not available, a specially designed memory-mapped interface can be fitted. For simplicity, the systems to be described are all based upon a suitable i/o port within the transmitter and, in all cases, the ports that have been used provide t.t.l. compatible signal levels. Most of the experiments have involved the use of a MOS Technology 6522 Versatile Interface Adapter (VIA)[3].

The measuring process depends upon the transmitter changing the status of an i/o line just before the commencement (and just after the termination) of data transmission – see Fig. 1(b): a program running in the timer monitors the status of this i/o line. When it detects the high-to-low transition it starts counting upwards from zero, continuing until the program subsequently detects the low-to-high transition which indicates the end of data transfer. The value of the count contained within the timer can then be used to compute the data transmission period, T, which may be achieved by the use of a previously prepared calibration graph(s). Alternatively, the known execution times of the program instructions can be used to calculate a loop cycle speed for the timer program, which can then be used as a multiplicative conversion factor.

The timer program used in the measurements is shown in Table 1. It is written in 6502 assembler code, which is subsequently run on a Commodore PET microcomputer[4,5]. It could easily be converted to run on other 6502 based systems (KIM, APPLE, AIM, etc.) by changing the addresses of the data direction register (DDR), user port (USER), print subroutine (PRINT) and the value assigned to the location counter at the start of the assembly.

The program uses the PET's 6522 VIA pin (PA0) for its connection to the transmitter. Zero page locations 0, 1 and 2 are used to store the count value. Once the program has been activated, it goes into a wait state until the status of pin PA0 goes low: as soon as this happens it enters its counting state until forced out of this when PA0 goes high again. Notice that prior to entering the wait state the program disables all interrupts (using the SEI instruction) to prevent the c.p.u. being called upon to perform any other ancillary tasks (for example, keyboard scan, clock update) while the data transmission period is being measured. Once the timing loop has terminated, system interrupts are again enabled (via the CLI instruction).

To test the program, an arrangement similar to that shown in Fig. 1(c) was used. The timer program was employed simply to measure the length of time for which a debounced switch circuit was held on: after each timed interval the contents of memory locations 0, 1 and 2 were examined and a total count value then computed. The results of some typical experiments are presented in Table 2(a). A graph of the total loop count was then plotted against the elapsed time as recorded by the stopwatch, these results being shown in Fig. 2(a). The timer program's loop execution time, as derived from the graph, is thus $36/(22.7\times10^5)$ which is equivalent to 1.6×10^{-5} seconds.

The alternative approach to estimating the timer program's loop cycle time depends upon a knowledge of the speed of execution of each of its component instructions. These are usually tabulated in programming manuals or hardware system specifications[6] for the 6502 chip. For the instructions involved in the timer program the relevant values are:

LDA	4 cycles
AND	2 cycles
BNE	2 cycles
INC	5 cycles
BNE	3 cycles
Total:	16 cycles

The BNE instruction can take 2, 3 or 4 cycles – depending upon whether the branch is taken and whether the branch operation involves crossing a page boundary. Since no page boundaries are crossed the values to be used in this case are 2 and 3. A value of 2 is used for the first BNE instruction since this branch is never taken – at least, until the end of the interval

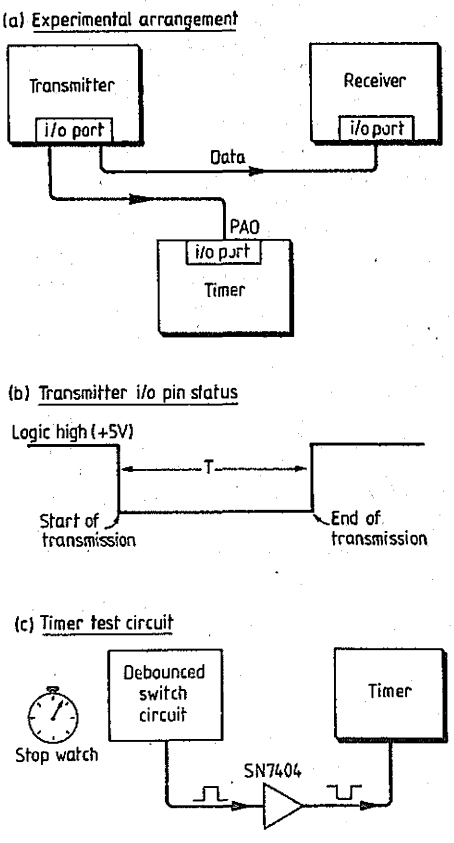


Fig. 1. Measurement technique using microcomputer as timer. Changes in status of i/o time at (b) determine counting period, test circuit for timer being shown at (c).

Department of Computer Science, Teeside Polytechnic.

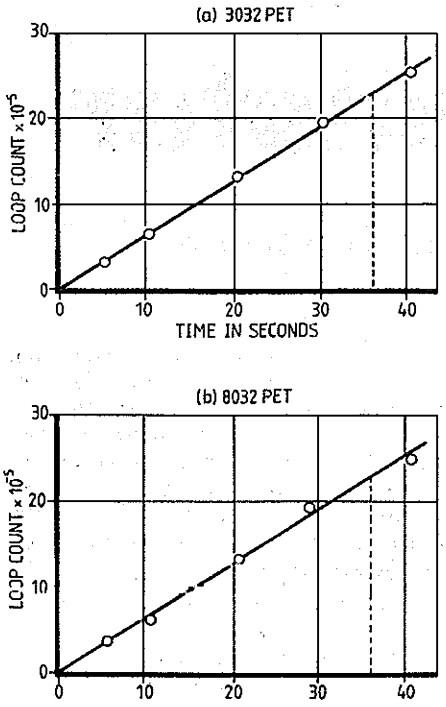


Fig. 2. Graph of time against total loop count gives loop execution time for 3000 and 8000 series PET micros.

being timed. A value of 3 is used for the other BNE instruction since this branch is virtually always taken – except when SUMH and SUMU are incremented.

The approximations used in the above formulation for the total number of cycles are reasonable since, if one assumes that the microcomputer clock speed is such that one cycles takes one microsecond, then the loop cycle time is easily calculated to be 1.6×10^{-5} seconds. This is in reasonable agreement with the value derived by the graphical approach.

In the data transmission experiments both a 3000 and an 8000 series PET have been used as a timer. Converting the 3032 program (see Table 1) for operation on the 8032 computer only required changing the address of the PRINT routine from $CA1C to $BB1D. Timing experiments analogous to those performed with the 3032 computer could then be conducted with the 8032 system. The results are shown in Table 2(b) and are presented graphically in Fig. 2(b). From this graph, the loop cycle time can be estimated as $36/(22.9\times10^5)$, that is, 1.6×10^{-5} seconds. This agrees closely with the value observed for the similar program running on the 3032.

From the experiments described in this section it is easy to see that the timer program offers a convenient means of measuring data transmission speeds. It is limited, however, in that the smallest time interval it could measure would be about 16 microseconds, which means that in our experiments we could not measure transmission speeds faster than about 16 Mbytes/s (PET-to-INS8060 transfer) or 2048 Mbytes/s (PET-to-PET transfer). However, because the transmission speeds involved in our systems are well below these limits this inherent limitation of the timer is of little concern.

Some data transfer measurements

Three different examples of microcomputer interconnection are described here. Two of these involve the use of parallel interfaces: in one case the standard IEEE-488 port is used[7] while in the other the direct linking of i/o ports is employed. The third example involves the use of a serial interface involving the use of a one byte buffer.

PET-to-PET transfer. The arrangement of the equipment for this transfer operation is shown schematically in Fig. 3. In this experiment, the data lines associated with the IEEE port of the 8032 were directly linked to the corresponding data lines of the IEEE port on the 3032 PET. A third PET system, another 3032 (not shown), was used as the timer. The transmitter then used its PA6 output line to interconnect with the timer's PA0 input pin. Some of the other user-port lines within the transmitter and receiver were employed as control lines to effect the handshaking of the data presented on the lines of the IEEE port. Four control lines were used: DAV (data valid), EOT (end of transmission), ACK (data acknowledge) and RFD (ready for data), implemented via user port connections PA0, PA2, PA1/CA1, and PA3 respectively. In the case of the ACK signal a choice between CA1 and PA1 at the transmitter end of the link could be used to decide whether this was (CA1) or was not (PA1) latched.

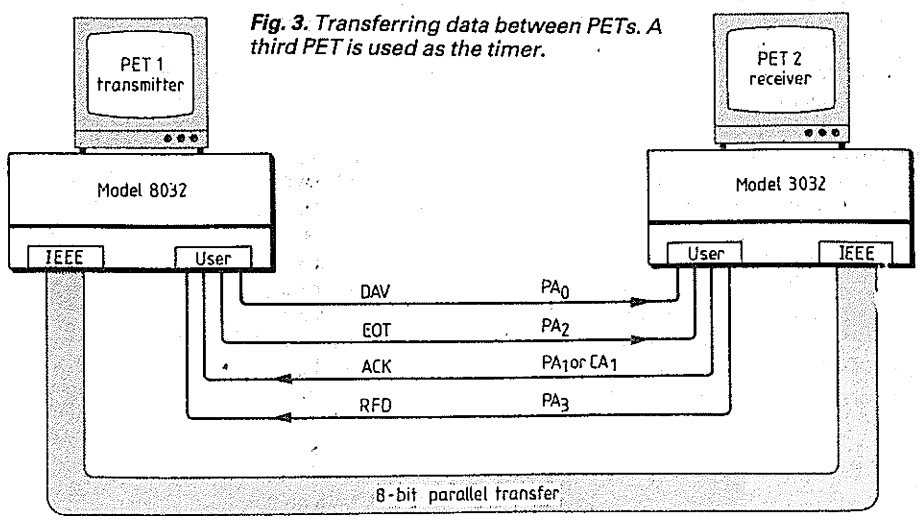
The details of the transmitter and receiver programs (in both BASIC and assembler) are given elsewhere[1], and may be used to send the contents of memory locations $2000 through $2FFF across the data link from PET1 to corresponding locations within PET2. Inspection of locations 0,1 and 2 in the timer yielded the results shown in Table 3(a). The transmission experiment was repeated five times giving an average count value of 26369, which gives a transmission time for the experiment of $26369\times1.6\times10^{-5}$, or 0.422 second. Since a total of 4096 bytes was transferred during this interval, the average transmission speed was therefore 9706 bytes/s.

PET-to-INS8060 (SC/MP) transfer. The experimental arrangement for this transfer is shown in Fig. 4. Notice that the pins used for interfacing the INS8060 are t.t.l.-compatible[12,13] and so could be directly connected to the appropriate user port pins of the PET. This approach was not used because we wished to investigate the additional programming overhead associated with using a serial-in-serial-out (siso) register as a buffer.

The way in which this interface works is as follows. The PET uses its serial shift register (which is a part of the 6522 VIA) to put data (serially) into the SN74LS91 buffer. When this operation has been completed, the PET signals 'data valid' to the SC/MP through the latter's SENSE-B input line. The SC/MP then generates clock pulses (on its FLAG-0 line) and strobes the data out of the buffer into its extension register via its serial input pin (SIN). After eight strobe pulses, the SC/MP acknowledges receipt of the data via its FLAG-1 line, which is attached to the PET's PA1 input pin. When the transmitter has passed across all the data, it signals the end of transmission by driving the EOT line high, which causes an interrupt in the SC/MP, causing it to jump to a special interrupt handling routine. Notice that because of the SC/MP architecture (and the mode of operation of the PET shifter) the passage of a data byte from transmitter to receiver causes bit reversal. Thus, it is important for the transmitter to reverse the bit pattern of all the data bytes before they are transmitted. This is done (for all of the data) prior to entry to the data transmission loop and so the time required to do this does not contribute to the data transfer interval. The programs for the transmitter (in 6502 assembler) and the receiver (in INS8060 assembler) are presented elsewhere[1].

The SC/MP system used for the experiments had available only 256 bytes of ram in which to store data. In view of this, only a limited volume of data could be transferred to it. The results for the transfer of 256 bytes of data from the PET (locations $2000 through $2OFF) to the SC/MP are presented in Table 3(b): the average value for the count is 12443 which corresponds to a data transfer interval of 0.199 and, since only 256 bytes were transmitted, the average data transfer rate was thus 1286

Fig. 3. Transferring data between PETs. A third PET is used as the timer.



bytes/s. Notice that the ratio of 9706 (parallel transfer) to 1286 (serial transfer) is 7.55. As might be expected, byte serial transfer is about eight times slower than byte parallel exchange.

Z80-to-PET transfer. For these experiments a SOFTBOX system was used[10]. This is essentially a plug-in hardware device that is designed to provide the PET microcomputer with access to the CP/M operating system[2]: the control software necessary to run the system is supplied on a 5.25in floppy disc: a disc unit is thus essential in order to use the SOFTBOX interface. The way in which the unit attaches to the PET's IEEE bus is illustrated schematically in Fig. 5. As can be seen from this diagram, for these experiments, an 8032 PET was used as a timer – interconnection of the two PETs was achieved via the PA0 user port line on each of the machines.

Within the SOFTBOX is housed a Z80 microprocessor that runs at a clock speed of 4MHz. In addition, there are 60 Kbytes of ram and rom to store the CP/M BIOS code[2]. The Z80 communicates with the PET's IEEE bus via two Intel 8255 peripheral support chips[11], the interconnections between the PET and the Z80 being illustrated in Fig. 6. When the Z80 system become active, it takes over control of the PET's disc drive and printer (see Fig. 5). The PET itself then acts as a dump terminal to the Z80 system.

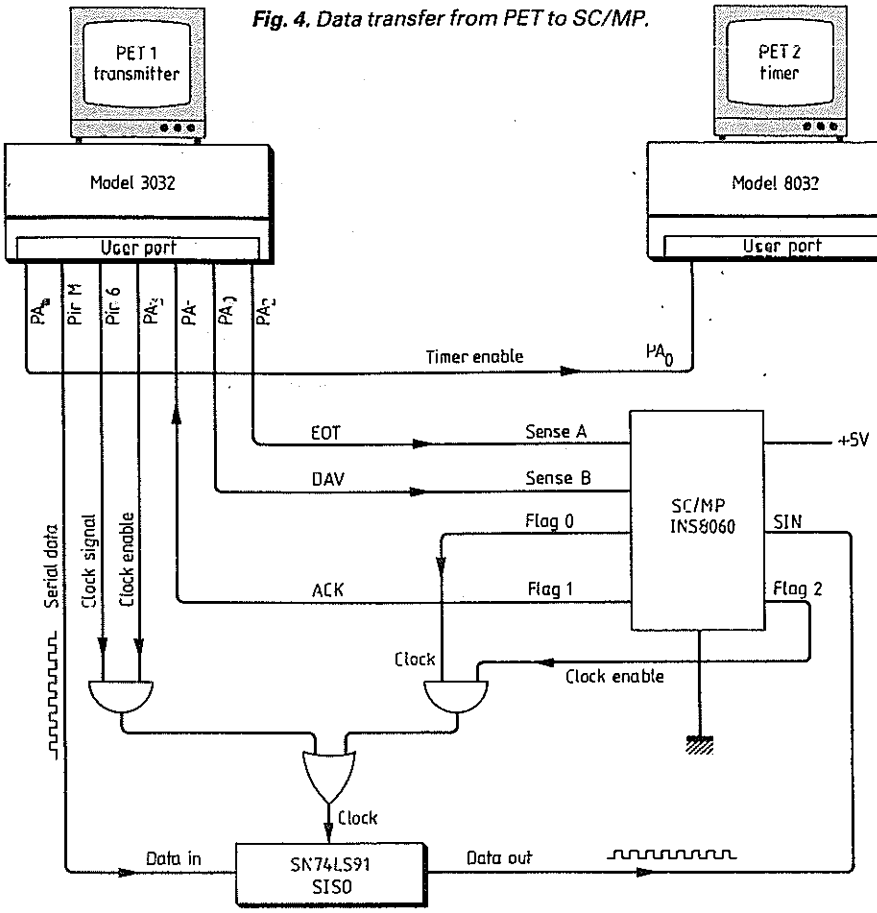
In addition to storing the CP/M BIOS code, the rom contained in the SOFTBOX provides many other useful routines. They may all be accessed by user programs running on the Z80 memory space via a series of jump vectors located at address $F003 and above. Two useful entry points within the rom store are PEEK and POKE – the POKE routine transfers data from the Z80 memory space across to that of the PET via the IEEE bus, and PEEK is complementary to POKE. This entry point can thus be employed to move data in the reverse direction – from the PET back to the Z80. In both cases, the Z80 register pairs BC, DE and HL are employed to hold the relevant transfer parameters: B and C specify the size of the memory image involved, while the relevant source/target addresses are held in DE (for the PET) and HL (for the Z80). Details of the architecture of the Z80 (and Intel 8080) are given[3]. A simple program for performing timed data transfer from the Z80 across to the PET is depicted in Table 4.

The program is written in 8080 assembler. First, a 4096 byte region of memory is initialized. Each byte within the defined area (with base address SBDATA) is set to the arbitrarily selected value $AB. The POKE entries immediately following the initialization loop then set the data direction register of the PET and also put the signal level on PA0 to logic high. The call to the dynamic debugging tool (DDT) is then used to check that the data area has been set up correctly; it also provides a processing interrupt that enables the timer program running on the 8032 to be put into a wait state. When the program in the Z80 is restarted it uses a series of POKE commands to (a) start the timer counting, (b) pass across the data to the PET, and subsequently, (c) switch off the timer by forcing the 3032's PA0 line into a high logic state. As in the previous experiments, processor interrupts are disabled prior to the data transfer steps and re-enabled immediately after it.

The timer count values extracted from the 8032 zero page locations (0, 1 and 2) are presented in Table 3(c). As was the case in the other experiments, measurements were repeated five times in order to check their reproducibility, giving an average count value of 40273. The data transfer interval calculated from this value is thus 0.644s which corresponds to a transfer rate of 6360 bytes/s.

The program presented in Table 4 runs within the environment of the CP/M dynamic debugging package[2], which was used to take advantage of the interrupt

Fig. 4. Data transfer from PET to SC/MP.



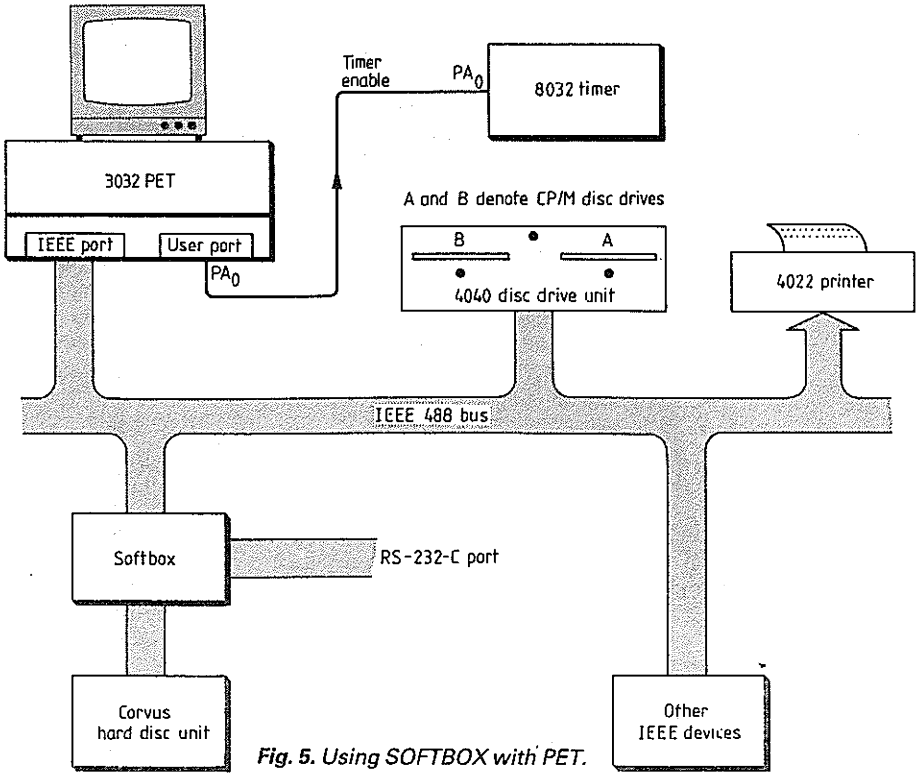


Fig. 5. Using SOFTBOX with PET.

Table 1. Program for using PET as timer, convertible for use with other 6502 micros.

```
0001 0000                ;TIMER
0002 0000                ;*****
0003 0000                ;
0004 0000                ;USING PET AS A TIMER
0005 0000                ;
0006 0000                DDR=59459
0007 0000                USER=59471
0008 0000                PRINT=$CA1C
0009 0000                SUML=$0000
0010 0000                SUMH=$0001
0011 0000                SUMU=$0002
0012 0000                      *$5000
0013 5000 A9 00          START LDS     #$0
0014 5002 8D 43 E8             STA     DDR     ;ALL PINS AS INPUT
0015 5005 A9 00                LDA     #$0
0016 5007 85 00                STA     SUML    ;ZEROISE LOW COUNT
0017 5009 85 01                STA     SUMH    ;ZEROISE HIGH COUNT
0018 5008 85 02                STA     SUMU    ;ZEROISE ULTRA COUNT
0019 500D 78                   SEI             ;DISABLE INTERRUPTS
0020 500E AD4F E8        WAIT  LDA     USER
0021 5011 29 01                AND     #$01    ;WAIT FOR PA0
0022 5013 D0 F9                BNE     WAIT    ;TO GO LOW
0023 5015 EA                   NOP
0024 5016 AD4F E8        TIMER LDA     USER
0025 5019 29 01                AND     #$01
0026 501B D0 15                BNE     END
0027 501D E6 00                INC     SUML    ;INCREMENT LOW COUNT
0028 501F D0 F5                BNE     TIMER
0029 5021 E6 01                INC     SUMH    ;INCREMENT HIGH COUNT
0030 5023 D0 F1                BNE     TIMER
0031 5025 E6 02                INC     SUMU    ;INCREMENT ULTRA COUNT
0032 5027 D0 ED                BNE     TIMER
0033 5029 58                   CLI
0034 502A A9 3B                LDA     #<ERR
0035 502C A0 50                LDY     #>ERR
0036 502E 20 1C CA             JSR     PRINT
0037 5031 00                   BRK
0038 5032 58             END   CLI
0039 5033 A9 59                LDA     #<TUP
0040 5035 A0 50                LDY     #>TUP
0041 5037 20 1C CA             JSR     PRINT
0042 503A 00                   BRK
0043 503B 0D             ERR   .BYTE $D,$A, 'ERROR – INTERVAL TOO
                               LONG',$D,$A,$00
0043 503C 0A
0043 503D 45 52
0043 5056 0D
0043 5057 0A
0043 5058 00
0044 5059 0D             TUP   .BYTE $D,$A, 'END OF TIMES INTERVAL', $D,$A,$00
0044 505A 0A
0044 505B 45 4E
0044 5070 0D
0044 5071 0A
0044 5072 00
0045 5073                      .END
```

facilities provided by the RST 7 instruction. To prove that the environment provided by DDT did not influence the speed of transmission, a further experiment was conducted. This necessitated re-writing the transfer program in such a way that the RST 7 calls could be dispensed with. Instead, the same effects were achieved through appropriate use of the CP/M BDOS routines for console output (CONOUT) and input (CONIN). CONOUT was used to display a prompt character on the 3032 screen. The Z80 processor then went into a wait loop until a pre-defined escape character (*) was typed on the 3032 keyboard. When the prompt character was displayed, the 8032 timer was started and the escape character then typed – thereby releasing the Z80 for its data transfer activity. The results obtained using this approach are listed in Table 3(d). As there is no significant difference between the results in Tables 3(c) and 3(d) we conclude that the DDT package did not influence the speed of execution of the program shown in Table 4.

A final set of experiments was conducted to see if the speed of transfer for PET-to-Z80 transmission was the same as that which was observed for Z80-to-PET transfer. To do this a new program was written, similar to that shown in Table 4, except that, instead of using the SOFTBOX POKE entry, it used the PEEK routine for block data transfer. The results of this set of experiments are presented in Table 3(e). Comparing these results with those of Tables 3(c) and 3(d) suggests that transfer in this direction is about 10% slower – probably due to the different ways in which the PEEK and POKE firmware is implemented within the SOFTBOX unit.

It is interesting to observe that parallel data transfer using the standard IEEE-488 bus (Z80-to-PET) is about 30% slower than that encountered in the other parallel transmission technique (PET-to-PET) that was used. This discrepancy is probably due to the additional overhead associated with the need to specify listener/talker addresses when transmitting data over an IEEE bus.

The maximum speed of transmission that can be measured using this simple method is given by the relationship

Table 2. Results of timer calibration.

(A) 3032 PET

TIME	SUMU	SUMH	SUML	Total	Rounded Total × 10^{-5}
5	04	BE	38	310,840	3.1
10	09	AB	19	633,625	6.3
20	13	96	77	1,283,703	12.8
30	1D	1F	A3	1,908,643	19.1
40	26	04	97	2,491,543	24.9
Weight	65,536	256	1		

(B) 8032 PET

TIME	SUMU	SUMH	SUML	Total	Rounded Total × 10^{-5}
5	05	17	6B	333,675	3.3
10	09	75	64	619,875	6.2
20	13	32	8C	1,258,124	12.6
30	1D	25	79	1,975,673	19.7
40	26	36	AB	2,504,363	25.0
Weight	65,536	256	1		

Table 3. Results of timing data transfer.

A: PET TO PET TRANSFER

Expt. No.	SUMU	SUMH	SUML	Total
1	00	66	FE	26,366
2	00	67	00	26,368
3	00	67	00	26,368
4	00	67	00	26,368
5	00	67	00	26,368
Weight		256	1	26,369 (Av)

B:PET TO SC/MP TRANSFER

Expt. No.	SUMU	SUMH	SUML	Total
1	00	30	A1	12,449
2	00	30	96	12,438
3	00	30	9B	12,443
4	00	30	95	12,437
5	00	30	A1	12,449
Weight		256	1	12,443 (Av)

C: SOFTBOX TO PET – CASE A

Expt. No.	SUMU	SUMH	SUML	Total
1	00	9D	5B	40,283
2	00	9D	4F	40,271
3	00	9D	4F	40,271
4	00	9D	4D	40,269
5	00	9D	51	40,273
Weight		256	1	40,273 (Av)

D: SOFTBOX TO PET – CASE B

Expt. No.	SUMU	SUMH	SUML	Total
1	00	9D	49	40,265
2	00	9D	51	40,273
3	00	9D	4A	40,266
4	00	9D	4B	40,267
5	00	9D	61	40,289
Weight		256	1	40,272 (Av)

E: PET TO SOFTBOX

Expt. No.	SUMU	SUMH	SUML	Total
1	00	B1	C2	45,506
2	00	B1	B2	45,490
3	00	B1	CC	45,516
4	00	B1	D5	45,525
5	00	B1	BF	45,503
Weight		256	1	45,508 (Av)

$S = V/1.6 \times 10^5$ bytes/s, where V is the volume of data (in bytes) that is passed.

In the experiments that have been described above a fairly expensive timing element was used – far too costly to dedicate solely for timing measurements. However, where such machines are used as general laboratory tools[12] an approach of this type is not unreasonable. Indeed, in the machines used in our laboratory the timer software shown in Table 4 is permanently

held in a rom module fitted to the microcomputer's memory-expansion sockets. This rom module also contains a variety of other useful firmware that is frequently required for other laboratory applications; for example, terminal emulation, data smoothing, pattern matching and so on.

Those situations that do not permit the use of a general purpose laboratory microcomputer (as described above) would require a less costly approach – easily achieved through the use of less expensive single board microsystems. Indeed, we have used a KIM micro[13] to perform exactly the same measurements that were undertaken by the 3032 and 8032 timer systems – at about one seventh the cost. If need be, further substantial cost reductions for the timer system could be achieved by simply wiring up a 6502 c.p.u., a 6522 VIA, some rom and a simple read-out system.

The author is grateful to Small Systems Engineering Ltd (UK) for their encouraging help and invaluable assistance during the preparation of this paper. He is also

continued on page 58

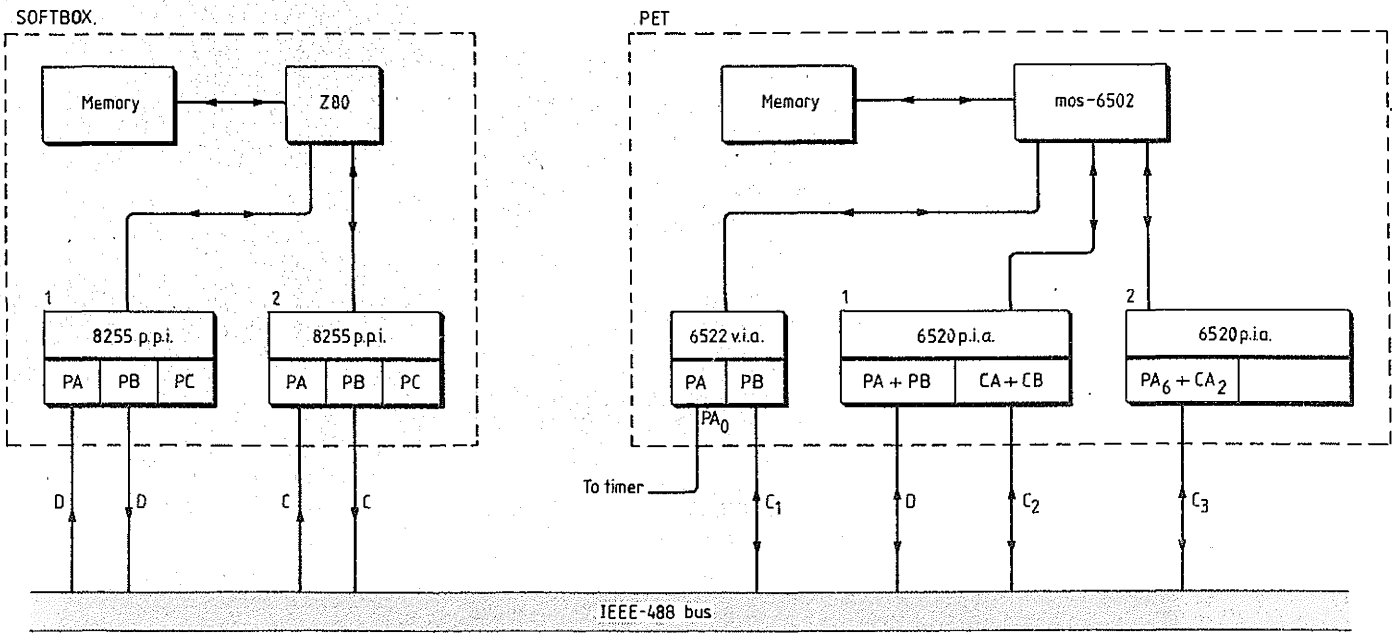
The author

Philip Barker is a Principal Lecturer in the Department of Computer Science at Teesside Polytechnic. He is a graduate of the University of Wales, a Member of both the ACM and the IEEE, and a Fellow ofboth the British Computer Society and the Royal Society of Chemistry. His research interests lie in the area of human-machine interaction, and he has undertaken a number of studies of the user interface with computer systems. The research topics in which he is currently interested include: author languages for computer assisted instruction (CAI), applications of CAI to the problems of the disabled, query languages for database systems, intelligent interfaces and image processing for videodisc systems.

Table 4. Program for timing transfer from 280 to PET.

```
3000  =          TOPET   EQU  3000H      ;TARGET ADDRESS FOR DATA
2000  =          SBDATA  EQU  2000H      ;SOURCE ADDRESS OF DATA
E84F  =          UPORT   EQU  59471      ;PET USER PORT ADDRESS
E843  =          DDR     EQU  59459      ;PET DATA DIRECTION REGISTER
0001  =          N1      EQU  1
F069  =          POKE    EQU  0F069H    ;SOFTBOX POKE ROUTINE
1000  =          NSEND   EQU  4096       ;NUMBER OF BYTES TO SEND
0100                     ORG 100H
0100  3E30       BEGIN   MVI A,30H       ;GENERATE TEST DATA
0102  210020             LXI H,SBDATA    ;LOAD SOURCE ADDRESS
0105  36AB       FILL:   MVI M,0ABH      ;MOVE VALUE TO MEMORY
0107  23                 INX H
0108  BC                 CMP H          ;ALL DONE?
0109  C20501             JNZ FILL
                ;
                ;INITIALISE TIMER
                ;
010C  010100             LXI B,N1
010F  1143E8             LXI D,DDR
0112  215101             LXI H,DDRVAL
0115  CD69F0             CALL POKE      ;SET PET DDR
0118  010100             LXI B,N1
011B  114FE8             LXI D, UPORT
011E  215201             LXI H,TMRSTOP
0121  CD69F0             CALL POKE      ;SET PA0 HIGH
0124  FF                 RST 7          ;CALL TO DDT
0125  00         BRK1:   NOP            ;PUT TIMER IN WAIT STATE
                ;
                ;SEND DATA TO PET MICROCOMPUTER
                ;
0126  F3                 DI             ;DISABLE INTERRUPTS
0127  010100             LXI B,1
012A  114FE8             LXI D, UPORT
012D  2155301            LXI H,TMRGO
0130  CD69F0             CALL POKE      ;START TIMER
0133  010010             LXI B,NSEND
0136  110030             LXI D, TOPET
0139  210020             LXI H,SBDATA
013C  CD69F0             CALL POKE      ;SEND DATA TO PET
013F  010100             LXI B,1
0142  114FE8             LXI D,UPORT
0145  215201             LXIH,TMRSTOP
0148  CD69F0             CALL POKE      ;STOP TIMER
014B  FB                 EI             ;ENABLE INTERRUPTS
014C  FF                 RST 7          ;CALL DDT
014D  00         BRK2:   NOP
014E  C30000             JMP 0          ;WARM START
0151  01         DDRVAL: DB  1
0152  01         TMRSTOP: DB  1
0153  00         TMRGO:  DB  0
0154                     END
```

Fig. 6. Arrangement for transferring data between 280 in SOFTBOX and 6502 in PET.



C - IEEE control lines
D - IEEE data lines
p.p.i. - programmable peripheral interface
PA, PB, PC etc. - Ports A, B, C

Table 1. Example of how the memory map may be changed when more than 16K of ram is used.

```
FORTH HEX
SMAX DUP @ 4000 + SWAP !        ( allow more data stack)
S0 DUP @ 4000 + SWAP !          ( move data stack)
SP! SMAX DUP @ 4000 - SWAP !    ( reset data stack)
R0 DUP @ 4000 + SWAP !          ( move return stack)
TIB DUP @ 4000 + SWAP !         ( move terminal input buffer)
' FIRST @ DUP @ 4000 + SWAP !   ( move Forth virtual memory buffers)
' LIMIT @ DUP @ 4000 + SWAP !   ( IE 'FIRST' and 'LIMIT')
FIRST DUP PREV ! USE !          ( point virt. memory pointers to virt. memory)
DPMAX DUP @ 4000 + SWAP !       ( move limit of dictionary up)
DECIMAL                         ( return to decimal arithmetic)
```

and the counter carry being set (refresh quantum finished), processor action is suspended by a dummy direct-memory-access cycle which guarantees a non-memory-access cycle.

Parity checking

Capacitance used to store data in dynamic rams is so small that naturally occurring charged particles (alpha particles) have a charge great enough to corrupt data should they hit a cell. Improved coatings on dynamic-ram dies have reduced this effect to give an error rate below 0.1%/1000h for 16K dynamic memories[5]. It is impractical to include error correction in small 8bit memories but parity checking to halt the processor when an error occurs is not.

An odd-parity bit, generated by an LS280 parity checker when a byte is written into memory, is stored with the other eight bits. During the write-cycle the parity-ram data output is in its high-impedance state and the floating EO input is high. The parity device output is clocked into the ram input and correct parity is looked for when memory is read. On reading, the data output drives the parity checker and the error signal is passed to the error latch with the row-address strobe signals. If an error exists, the RAS line concerned is latched, a led indicates which memory bank contains the error, and the processor halts.

Memory speed and drive

Input characteristics of dynamic ram are quite different from those of t.t.l. Ram inputs are capacitive, which especially affects signals common to many inputs like RAS, CAS and WE, and they require little direct current. When driven directly from low-power Schottky t.t.l. these inputs can cause considerable overshoot that can result in exceeding device specifications and longer access times through the time taken for the voltages to level out.

To reduce ringing, some form af matching is required. Series matching is most appropriate since it does not increase static loading. The ideal driver would produce a slightly under-damped response but because t.t.l. drive characteristics are asymmetric a compromise had to be made in the resistance value. Control signals are driven from LS37 clock drivers to ensure adequate drive toward the 5V rail. Resistance values are not critical for this relatively slow memory and the original even worked faultlessly with no damping resistors and standard LS00 drive.

On analysing the timing requirement of the ram/M6809 interface I noticed that the most readily available 200ns rams leave a lot of spare time – so much so that these devices could theoretically be run with a 666ns cycle time instead of the standard 1μs. This was, of course, tried. Not only was it tried with the faster M6809A processor but also with the standard device. In both cases functioning was faultless. This is not to say that all 1MHz parts will run at higher speeds but certainly 200ns access time rams will work at 1.5MHz. So for the cost of a new crystal the through-put of the system was improved by 50%.

Peripherals

To ensure that 1MHz peripheral devices such as the 6821 peripheral-interface adapter and the 6850 communication-interface adapter operate correctly, the memory-ready signal (MRDY) is used. Whenever peripherals are addressed MRDY is held false by an LS122 monostable multivibrator which extends the memory-access time. An M6850 communication device forms the RS232 interface and the clock frequency for it is crystal derived. Currently the 1.5MHz c.p.u. clock only allows 1800bit/s and an external baud generator is an attractive proposition. Both −5 and +12V supplies are used for the RS232 interface. Current from the −5V supply is so low that the RS2 driver has an active current limiter; the +12V drive is resistive.

Many of you will not have an RS2 terminal and will wish to use a separa keyboard and domestic tv. The keyboa interface will accept any 7bit parallel inp signal with active-low most-significant-b and active-low-going strobe and reque signals. Two spare hand-shake lines on th p.i.a. and an output port could form Centronics-type printer port.

An EF69364A video i.c. provides timir signals necessary for a 625-line tv; 96364B device will provide signals time for 525-line tv. Control code for the vide i.c. is supplied through an LS157 qua two-to-one-line multiplexer and for norm display characters (p.i.a. B D_7=0) a fixe control code is set. When control chara ters (hexadecimal 0 to F) are used th p.i.a. supplies the relevant code throug the multiplexer (p.i.a. B D_7=1) to th EF69364. As the c.r.t. gun scans th screen, the EF69364 selects the characte to be displayed from the display ram an latches it into an LS273.

The video i.c. was designed for use wit ram that has separate data input and out put lines (2101 ram) so the circuit wa modified to allow 2114 rams with commo i/o to be used. Character-code from LS27 and row information from 69364 i supplied as an address to a character ron (a specially programmed 2716 eprom) Each character position is allocated a 7 wide-by-12-high character block.

Referring to last month's article, th signal name at pin 6 of IC_{41} is active lo and should read $\overline{R}$, as should the sign name at the junction of IC_{47} pin 2 and IC pin 3. On page 57, pins 13, 12 and 5 of th LS175 should be labelled Y_0, Y_1 and Y respectively.

A set of three programmed roms is avai able from Brian Woodroffe at 63 Queensferry Road, Edinburgh for £23.5 inclusive. Technomatic (see advertiser index) will supply all i.cs mentioned in thi article.

Disc-drive interfacing is described in th next article.

References

5. E. Westfield, Memory system strategies fo soft and hard errors, Wescon '79.

continued from page 48

indebted to Keith Frewin, who wrote the SOFTBOX software, for providing roms 385 and 386.

References

1. P. G. Barker, Data Transmission Between Micros, *Electronics and Computing Monthly*, 2(5), 1982, 21-25 and 46-49.
2. P. G. Barker, Introducing CP/M, *Electronics and Computing Monthly*, 1982, in press.
3. A. Osborne and J. Kane, An Introduction to Microcomputers: Volume 2 – *Some Real Microprocessors*, Osborne & Associates Inc, California, 1978, Chapter 10, pp29-49.
4. Commodore Business Machines Ltd., CBM PET 3032N Professional Computer User's Manual, Publ. No. 320856-3, June 1979.
5. Commodore Business Machines Ltd., CBM PET Series 8000 User's Guide, Publ. No. 320894, 1981.
6. MOS Technology Inc., MCS6500 Microcomputer Family Hardware Manual, Publ. No. 6500-10A, 2nd Edition, January 1976.
7. E. Fisher and C. W. Jensen, PET and the IEEE-488 Bus [GPIB], Osborne/McGraw-Hill, California, 1980.
8. National Semiconductor Corporation, *SC/MP* Technical Description, Publ. No. 4200079B, September 1976.
9. I. Williamson and R. Dale, Understandin Microprocessors with the Mk 14, The Mac millan Press, Bristol, 1980.
10. Small Systems Engineering Ltd., 2-4 Can field Place, London NW6 3BT, UK SOFTBOX User Manual, Revision 3, 1981.
11. J. Kane and A. Osborne, An Introduction t Microcomputers: Volume 3 – Some Rea Support Devices, Osborne & Associate Inc., California, 1978.
12. P. G. Barker, Computers in Analytical Chemistry, Pergamon Press, Oxford, 1982, in press.
13. MOS Technology Inc., KIM-1 Microcomputer Module User Manual, Publ. No. 6500-15B, 2nd Edition, August 1976.