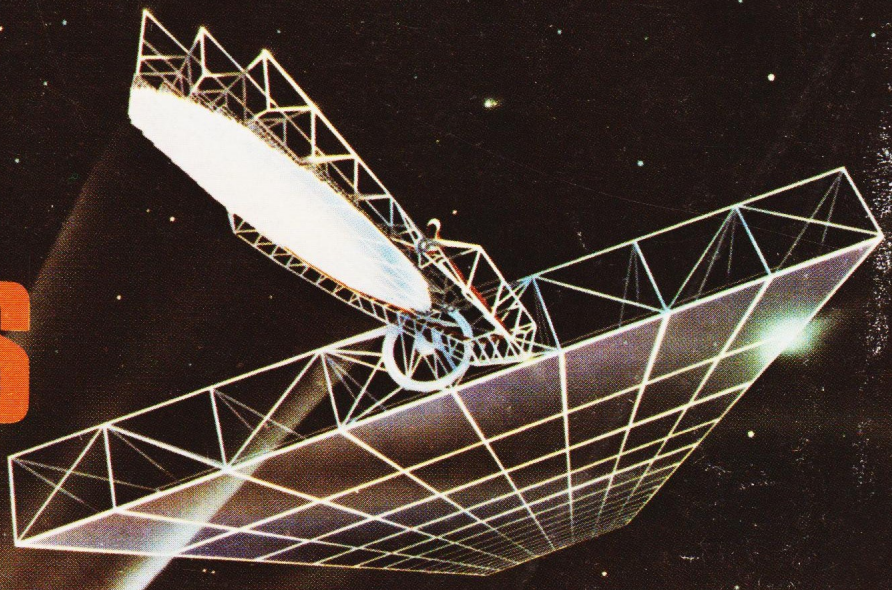
Australia $1.10 New Zealand $1.20 Malaysia $4.50

PRACTICAL ELECTRONICS

JANUARY 1981

65p

SOLAR POWER SATELLITES



PE **STARSPINNER** PART 1

Also... **INTERFACING COMPUKIT** PART 1

Interfacing COMPUKIT

Part 1 D.E.Graham

THE COMPUKIT UK 101 is one of the few personal computers with 8K BASIC and full keyboard that does not have an input/output port for interfacing external devices. In this series we propose a remedy for this in the shape of an Address Decoding and Port Module which plugs directly into the Compukit's expansion socket. It is also Superboard II compatible.

The Module has been designed with flexibility in mind, and as well as housing an MC6821 Parallel Interface Adaptor (PIA), which gives two 8-bit input/output ports, the board also provides 7 uncommitted address-decoded *read,* and 14 decoded *write* lines, each of which may be used with interfaces of the reader's choice, and a pair of specially decoded lines that will directly interface an AY-3-8910 or 8912 PSG. In addition there is on-board address decoding for a further 6 blocks of 16 memory locations; again these are completely uncommitted, and each could be used to enable devices with up to 16 independent registers, such as the 6522 Versatile Interface Adaptor, details of which will be given later in the series. The board also houses an independent 5 volt regulated power supply which may be used to run a limited number of external circuits.

During the series the principles of interfacing the Compukit using various devices will be developed, and circuits will be given for a range of interfaces that may be plugged directly into the Decoding Module. Amongst these will be featured interfaces for joysticks, l.d.r. light sensors, 7-segment l.e.d. displays, audio generators, power controllers, and D/A and A/D converters. Software support for each will also be discussed.

The first part of the series is devoted to the Decoding Module itself.

DECODING PRINCIPLES

The 16 address lines of the Compukit can be confugured in 2^{16} or 65,536 different ways, or in other words it can address 65,536 different memory locations. To pick out just one of these, gating circuitry must be used. The circuit in Fig. 1.1, employing a single 16-input AND gate, would give a high output if, *and only if,* each of the address lines was simultaneously high. The Compukit's address lines are active-high, so that the circuit could be used to provide a Chip Select signal when the address FFFF hex (or 65,536) was put on the address bus by the Compukit's CPU. Different addresses could be decoded by simply placing inverters between chosen address lines and the gate inputs. Putting an inverter in lines A0 and A4, for example, would decode for the address FFEE hex (65,519 decimal). In Table 1.1 we give a listing of a hex to decimal/decimal to hex converter that may prove useful for calculating addresses on the Compukit.

COMPONENTS . . .

DECODING MODULE

Resistors

R1, R2	1k ¼W (2 off)
R3	220 1W
R4–R6	10k ¼W (3 off)

Capacitors

C1	2200µ 15V
C2, C5	100n disc cer. (2 off)
C3	10µ 15V
C4	100µ 15V
C6	1000µ 15V
C7	10µ 15V

Diodes

D1–D4	1N4001 (4 off)
D5	5V 1·2W Zener

Integrated Circuits

IC1	74LS133
IC2, IC4	74LS138 (2 off)
IC3	74LS154
IC5–IC7	74LS04 (3 off)
IC8	74LS03
IC9	6821
IC10	7805

Miscellaneous

2 off	40-pin d.i.l. socket
5 off	16-pin d.i.l. socket
2 off	24-pin d.i.l. socket
4 off	14-pin d.i.l. socket
1 off	40-pin d.i.l. plug
2 off	22-pin 0·1in. edge connector
2 off	25-pin 0·1in. edge connector
1 off	24-pin d.i.l. plug
2 off	16-pin d.i.l. plug
2 off	8T28 to plug into Compukit

Ribbon cable
1A 20mm slo-blo fuse and p.c.b. holder
push-to-make switch.

The ENABLE line of Fig. 1.1 could be used to trigger a data latch (such as the 7475) to latch data appearing instantaneously on the CPU's data bus for use by some external device. In practice, in order to ensure that the ENABLE pulse comes at exactly the right instant, it is desirable to make it conditional on Compukit's Ø2 clock line going high. Fig. 1.2 shows a circuit using a 17-input AND gate that would decode for the address EF18 hex (61028 decimal). This is a

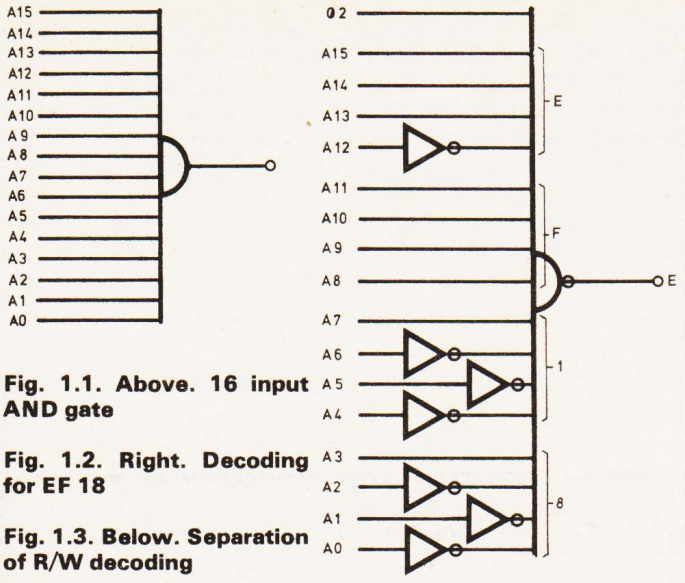
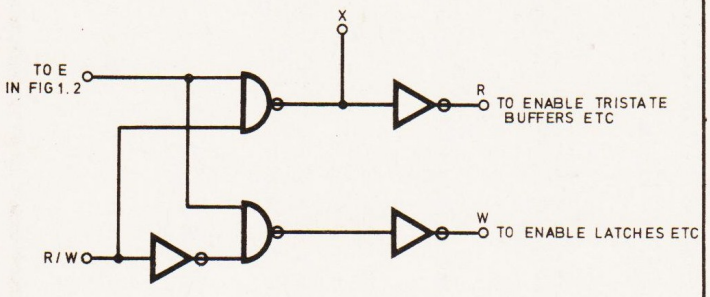


Fig. 1.1. Above. 16 input AND gate

Fig. 1.2. Right. Decoding for EF 18

Fig. 1.3. Below. Separation of R/W decoding



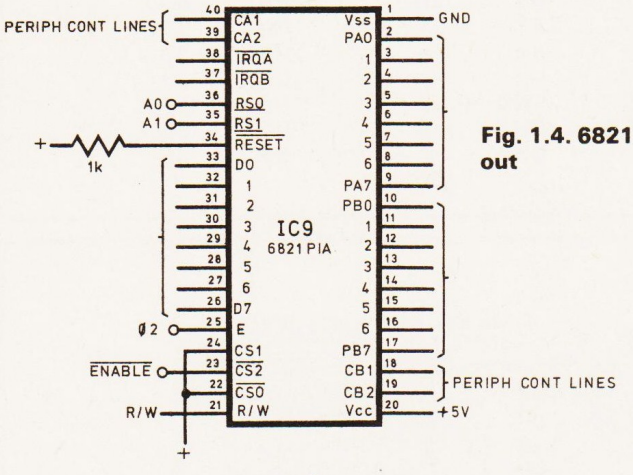


Fig. 1.4. 6821 pin-out

```
90 REM HEX-DEC-HEX CONVERTER
95 REM PE UK101 INTERFACING PROG NO 1
100 FORA=1TO16:PRINT:NEXT
110 PRINT,"HEX-DEC-HEX CONVERTER"
115 PRINT:PRINT:PRINT:PRINT
120 PRINT"    IS DATA HEX OR DECIMAL ?"
125 INPUT"    ENTER H OR D";Y$
130 IFY$="D"THENGOSUB550:GOTO165
140 IFY$="H"THENGOSUB550:GOTO350
150 PRINT:PRINT"    NOT RECOGNISED: ENTER AGAIN"
160 GOTO120
162 REM
163 REM DEC TO HEX ROUTINE
164 REM
165 PRINT:PRINT:PRINT
166 INPUT"    DECIMAL DATA PLEASE";N
168 IFN=0THEN350
170 A=INT(N/4096)
180 A1=A*4096
190 B=INT((N-A1)/256)
200 B1=B*256
210 C=INT((N-A1-B1)/16)
220 C1=C*16
230 D=N-A1-B1-C1
240 X$="0123456789ABCDEF"
250 PRINT,"HEX EQUIVALENT=  ";
260 PRINTMID$(X$,A+1,1);
270 PRINTMID$(X$,B+1,1);
280 PRINTMID$(X$,C+1,1);
290 PRINTMID$(X$,D+1,1)
300 GOTO165
350 REM
360 REM HEX TO DEC ROUTINE
370 REM
390 PRINT:PRINT:PRINT
400 INPUT"    HEX DATA PLEASE";H$
402 IFH$="0"THEN165
403 IFLEN(H$)<>4THENPRINT:PRINT"    4 DIGIT FORMAT ONLY":GOTO400
405 N=0
410 X$="0123456789ABCDEF"
420 FORJ=1TO4
430 FORI=1TO16
440 IFMID$(H$,J,1)=MID$(X$,I,1)THEN460
450 NEXTI
455 PRINT:PRINT"    CHARACTER NOT IDENTIFIED - RE DO"
456 GOTO390
460 N=N+(I-1)*16↑(4-J)
470 NEXTJ
480 PRINT,"DECIMAL EQUIVALENT=  ";N
490 GOTO390
500 END
550 PRINT:PRINT:PRINT"    NOTE THAT ENTERING A ZERO WHEN"
560 PRINT"    DATA IS REQUESTED REVERSES FUNCTION"
570 RETURN
```

Table 1.1 Hex/Dec. and D/H converter program

Table 1.2. Compukit's Memory Map showing gaps

Address	
0000-02FF	Scratchpad RAM for operating system
0300	Start of Basic Workspace
1FFF	End of On-board RAM
9FFF	End of Possible Ram expansion
A000-BFFF	Basic Interpreter
D000-D3FF	Video RAM
DF0D	Polled keyboard
F000, F001	ACIA serial port
F800-FFFF	Monitor ROM

quite arbitrary address, and clearly any of the Compukit's 65,536 addresses could be decoded in this way; although of course since 17-input AND gates are not readily available, one would be forced to use a combination of gates to achieve the same effect in a practical circuit.

There are two further factors which must be considered in decoding for an interface, both of which relate to the R/W (Read/Write) signal. The circuit of Fig. 1.2 will give an output at any time that the address 61208 appears on the address bus. Thus, executing POKE 61208, X or Y=PEEK (61208), would both cause an output from the decoding circuit. But in most applications it is useful to distinguish between read and write operations. If, for example, we are using the signal to trigger a set of latches to give a data output, we will only want this to occur in response to a POKE command, whereas if it were used to turn on a tristate buffer for the input of data to the CPU, we would want this to occur exclusively in response to a PEEK statement.

Differentiation between the two can be achieved by using the R/W line at Compukit's expansion socket. This goes high during a Read Cycle, and low during a Write Cycle. The configuration in Fig. 1.3 would derive two separate Chip Select lines from the output of the circuit in Fig. 1.2, one for a Read to the address 61,208, and one for a Write. As may be seen, even though the two resulting decoded lines share the same address in the Compukit's memory map, they could be used for entirely different purposes. The Write might be used to trigger latches driving a D/A converter, while the Read might

trigger tristate buffers to feed the CPU with the counting registers of an external clock, for example.

Finally, in our decoding circuitry we must include a means of controlling the DD or Data Direction line of the Compukit. This determines the direction in which data is allowed to pass through the two 8T28 data buffers on the Compukit's main board. Note, incidentally, that while these two i.c.s are essential in any use of the data bus at the expansion socket, they are not provided in the basic UK101 kit, and must be purchased separately. With the DD line high, data can pass from the CPU to the expansion socket, but not in the reverse direction. When it is low, on the other hand, the converse is true. With no external signal on this line, it is kept high by Compukit's on-board resistor network R9 R74. If we did not service the DD pin at the expansion socket, we could successfully write data to external devices with the circuit of Figs. 1.2 and 1.3, but even though the R line of Fig. 1.3 would go high when a PEEK(61208) was executed, no data from the tristate buffers, or whatever else was enabled, would actually get to the CPU data bus. This could be remedied by connecting point X in Fig. 1.3 directly to the DD pin of the expansion socket. This would bring DD low only when a Read instruction was carried out at the given address, and the associated interfaces could then be both written to, and read from, in a satisfactory manner.

THE DECODING MODULE

The Decoding Module requires a 128 byte address block, a requirement easily met within the Compukit's memory map. This is reproduced in table 1.2, and it may be seen that the Compukit possesses unused blocks at COOO-CFFF, D400–DEFF, DFO1–EFFF and F100–F7FF hex. For reasons of simplicity we have chosen to locate the module between EF80 and EFFF hex (61,312–61,439 decimal). This falls immediately below the serial port at F000 hex. An address map of the major 8 blocks of the module is given in Table 1.3.

Table 1.3. Address Map of Module

Base Address of (Hex)	Block (Dec)	Block Number	Function
EF80	61312	BLO	Base address for 8 decoded lines
EF90	61328	BL1	Base address for PIA block
EFA0	61344	BL2	Free Block
EFB0	61360	BL3	Free Block
EFC0	61376	BL4	Free Block
EFD0	61392	BL5	Free Block
EFE0	61408	BL6	Free Block
EFF0	61424	BL7	Free Block

The board uses a combination of edge connectors and d.i.l. sockets for external connections, and the pin-outs of these are given in Tables 1.4–1.8. Edge connector SK1 carries the 40 leads from the Compukit's expansion socket, and the wiring between these should be kept as short as possible. The 40-pin socket SK2 allows for further expansion of the Compukit, and has the same pin-out as Compukit's own expansion socket. The two 16-pin d.i.l. sockets SK3 and SK4 carry ports A and B of the PIA, respectively, together with associated control and power supply lines.

The decoded lines produced by the Decoding Module are taken out through the 24-pin d.i.l. socket SK5, carrying six Write and two Read lines, and the 2 x 25 pin edge connector SK6 which carries the remainder. Both SK5 and 6 also carry Vcc and the data bus, and in addition SK6 carries address lines A0–A3, Φ2, $\overline{NMI}$, $\overline{IRQ}$ and $\overline{RESET}$ to allow full use of the six 16-byte blocks.

Table 1.4. Connections to edge connector SK1.

	UPPER ROW		LOWER ROW	
SK1 pin	Function	Connection to compukit exp. soc.	Function	Connection to compukit expansion
1	A2	12	GND	40
2	A1	13	GND	39
3	AO	14	GND	38
4	A3	15	GND	37
5	A4	16	n/c	—
6	A5	17	n/c	—
7	A6	18	R/W	32
8	$\overline{IRQ}$	1	O2	31
9	$\overline{NM1}$	2	A15	27
10	DD	3	A14	26
11	DO	4	A13	25
12	D1	5	A12	24
13	D2	6	A11	23
14	D3	7	A10	22
15	Spare	11	A9	21
16	A8	20	GND	30
17	A7	19	GND	29
18	n/c	—	GND	28
19	n/c	—	D7	33
20	GND	8	D6	34
21	GND	9	D5	35
22	GND	10	D4	36

Table 1.6. SK3 and 4 of PIA.

1	GND	16	ADO
2	CA1	15	AD1
3	CA2	14	AD2
4		13	AD3
5	GND	12	AD4
6		11	AD5
7		10	AD6
8	Vcc	9	AD7

SK4—PORT B of PIA is identical

Next month we will deal with the circuit operation of the Decode Module, showing the printed circuit board layout and component overlay. We shall also cover the operation of the PIA, and the construction and testing of the Decoding Module; and will look at the inputting of data to the COMPUKIT, both via the PIA, and sets of tristate buffers.

Table 1.7. Connections to SK5.

GND	1	24	W10
GND	2	23	D6
GND	3	22	D4
D7	4	21	D1
D5	5	20	D3
DO	6	19	W12
D2	7	18	W14
$\overline{W11}$	8	17	GND
W13	9	16	GND
W15	10	15	GND
Vcc	11	14	R5
Vcc	12	13	R4

Table 1.8. Connections to SK6 edge connector.

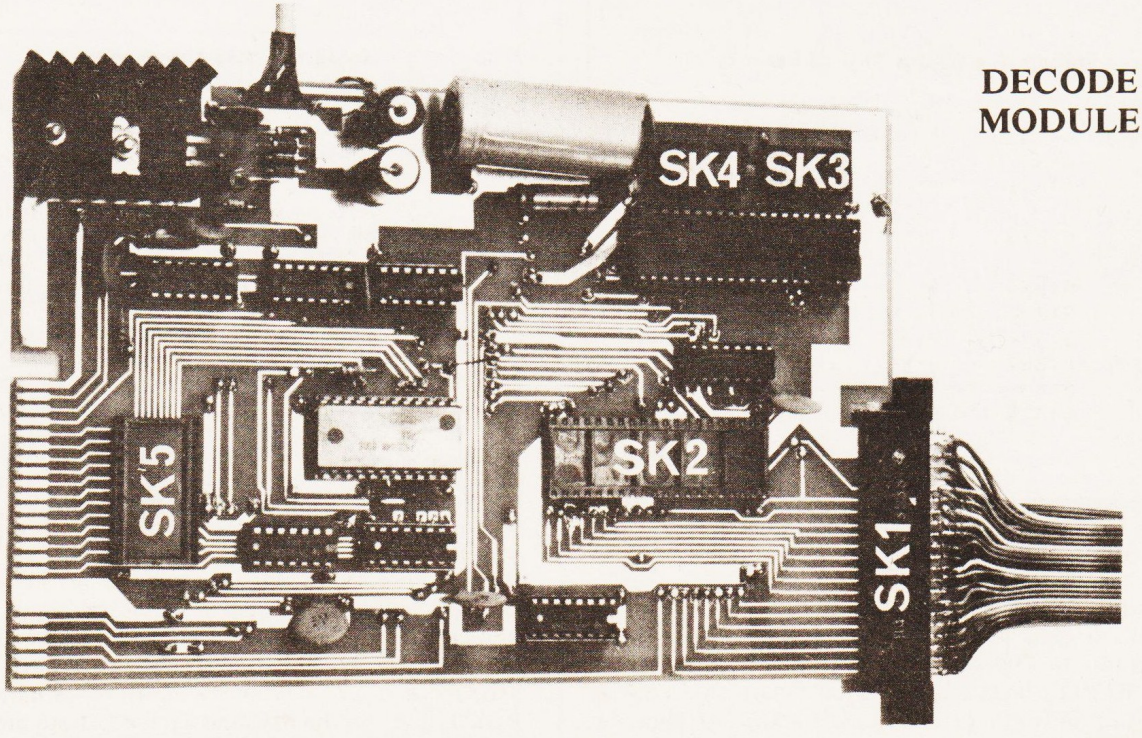
	upper (Component side)	lower
1	Vgg	$\overline{RESET}$
2	Ø2	W7
3	$\overline{IRQ}$	W8
4	BC1	R7
5	BD1R	RO
6	W1	R1
7	$\overline{WO}$	R/W
8	$\overline{W2}$	GND
9	$\overline{W3}$	GND
10	$\overline{W4}$	D7
11	$\overline{W7}$	D6
12	W9	D5
13	A3	D4
14	A2	DO
15	A1	D1
16	AO	D2
17	GND	D3
18	GND	Vcc
19	GND	Vcc
20	BL4	GND
21	BL3	GND
22	R3	GND
23	R2	BL6
24	BL7	BL5
25	NM1	BL2

Table 1.9. Address within Block 1. * Note that all but the three lines with an asterisk are uncommitted, and may be used with interfaces of the reader's choice, but that, as may be seen, a number of others have been earmarked for projects within the series.

Address	Write		Read	
61327	W15	To SK5 for 4 digit 7-segment display	—	
61326	W14		—	
61325	W13		—	
61324	W12		—	
61323	W11	To SK5	—	
61322	W10		—	
61321	W9	To SK6	—	
61320	W8	D/A converter	—	
61319	W7	A/D converter	R7 A/D converter	
61318	W6*	Audio (data)	R6* Audio (data)	
61317	W5*	Audio (address)	R5 to SK5	
61316	W4	To SK6	R4	
61315	W3		R3	
61314	W2		R2	To SK6
61313	W1	inverted	R1 inverted	
61312	WO	inverted	RO inverted	

Table 1.10. Selection of Base Address of Decoding Module. * "O" indicates inverter in use.

State of A13	A12	A11	Address hex of 128 byte block	Comments
1	1	1	FF80–FFFF	These two already used by monitor.
1	1	0	FE80–FEFF	
1	0	1	EF80–EFFF	If pads are left untouched, the module assumes this slot.
1	0	0	E780–E7FF	6 possible relocation sites for Module
0	1	1	DF80–DFFF	
0	1	0	DE80–DEFF	
0	0	1	CF80–CFFF	
0	0	0	CE80–CEFF	



DECODE MODULE

Interfacing COMPUKIT

Part 2 D.E. Graham

THIS MONTH we discuss the use of the MC 6821 PIA on the Decoding Module, and look at the way in which digital data may be input to Compukit using both this, and sets of tristate buffers.

The 6821 PIA

Motorola's 6821 PIA provides two 8-bit input-output ports together with interrupt and peripheral control facilities. It achieves this through the operation of six internal registers; three for each port. One of these carries the actual data passing through the port, a second determines the direction in which data is carried, while the third is a control register largely associated with the generation of interrupts, and the operation of control lines to peripheral devices. Somewhat unusually, although the chip uses six registers, it occupies only four addresses, and as a consequence accessing its registers is not as straightforward as it might be.

It circumvents the problem by using bit 2 of each control register to determine whether the other two addresses refer to the data direction register or to the port data register. Table 2.1 gives the disposition of the six registers, and the four corresponding addresses used in the Decoding Module. Accessing the two control registers is quite straightforward. The command POKE 61341, W will place the value W into the control register of port A, while POKE 61343, W will do the same for port B. In order to access the data direction register, bit 2 of the control register must first be set to zero. POKE 61341, O will achieve this for port A. The command POKE 61340, Y will then configure port A for input if Y=O, or for output if Y=255. Bit 2 of the control register must then be set to 1 (eg. by POKE 61341, 255) to achieve access to the port itself, and this completes the initialisation procedure. PEEKing or POKEing to 61340 can now be used for input or output of data.

This is not really as difficult as it sounds, and Table 2.2 gives the four complete sets of commands for configuring either port for 8-bit input or output. Note that on Reset, or at switch-on, all registers are automatically zeroed, so that a number of commands in this table may be omitted in certain situations; and of course it is only usually necessary to configure the ports once at the beginning of a program, and all further Reads or Writes can be accomplished with a single PEEK or POKE.

In the examples discussed so far, the data direction register has been loaded with either a zero (for input) or 255 (for output). In practice each of the 8 bits of the two ports is individually controllable in this respect, with one binary digit of the data direction register controlling one bit of the corresponding port; a zero signifying input, and a 1 an output. Thus whilst placing 255 (11111111 binary) in the data direction register will configure all 8 bits for output, the number 15 (00001111 binary) would cause the top 4 bits to be set for input, and the lowest 4 for output, and so on.

Although the registers of ports A and B are addressed in an identical fashion, the two ports differ electrically in certain respects. Both have a drive capability of two TTL gates. But on input, the output circuitry of port B adopts a tristate condition, whereas that of port A does not. Port A's inputs are accordingly taken high by internal pull-up resistors, and require an effective resistance of 1k or less to earth in order to render them low. The voltage sensitivities of the two ports is also somewhat different. Port A takes voltage lower than about 1·4 as a logical zero, and higher than about 1·6 as a logical 1. whereas the two corresponding voltages for port B are about 0·7 and 3·0.

In a later article in this series we will look at the use of the peripheral control lines of the 6821. We now turn to the construction and testing of the Decoding Module.

Table 2.1. 6821 Registers

Address	Control Bit Reg A	Reg B	Function	
61340	1	X	Port Data Register	
61340	0	X	Data Direction Register	Port A
61341	X	X	Control Flag Register	
61342	X	1	Port Data Register	
61342	X	0	Data Direction Register	Port B
61343	X	X	Control Flag Register	

Table 2.2

OUTPUT

For output on Port A P=61340	For output on Port B P=61342	Function
POKE P+1, 0	POKE P+1, 0	codes for data direction register
POKE P, 255	POKE P, 255	sets data direction to output
POKE P+1, 255	POKE P+1, 255	code for peripheral register

Further commands of *POKE P, W* will now place W on Port A or B

INPUT

For input on Port A P=61340	For input on Port B P=61342	Function
POKE P+1, 0	POKE P+1, 0	code for data direction register
POKE P, 0	POKE P, 0	sets data direction for input
POKE P+1, 255	POKE P+1, 255	code for peripheral register

calls of PEEK (P) will now return the data at port A or B

CIRCUIT OPERATION

A full circuit of the Decoding Module is given in Fig 1.5. Device IC1, a 13-input NAND gate, provides the complete decoding of the base address of the system. As the circuit stands, with a single inverter in address line A12, this is EF80 hex (61312 dec). It will be seen however, that a number of pads are provided to engage two further inverters in address lines A11 and A13. This allows the user, by cutting tracks and wiring between the pads on any of the three lines A11-A13 to set the base address at 7 other possible locations. The various permutations are given in Table 1.10. It should be noted that the top two of the 8 possible base addresses fall within the UK101's 2K monitor, and should therefore be avoided! The remaining 6 base addresses and accompanying blocks are unused by both the Compukit and the Superboard II. Since however, all software for the series will assume a base address of EF80, it may be simplest to leave all three sets of pads unaltered.

The output of IC1 is taken to IC2, a 74LS138 3-to-8 line decoder, which decodes address lines A4–A6 to produce eight 16-byte blocks (BL0–BL7). The decoding of these blocks has not been made conditional of Ø2 since they are generally intended for use with multi-register devices which often possess a separate master Enable pin, which when connected to Ø2 satisfies all timing requirements of the chip. Some devices, such as the 6522 VIA, require the chip select lines to settle before Ø2 goes high; so that if the output of IC2 had been made conditional on Ø2, no settling time would be given, and the device would not respond to the CPU's attempts to select it.

Device IC3, a 74LS154 4-to-16 line decoder, provides the sixteen address-decoded Write lines in response to BL0, R/W and Ø2. The outputs of both the 74138 and the 74154 are active-low, and a number of IC3's outputs have been inverted for convenience of subsequent use. Lines W5 and W6 have, in conjunction with R6, been further decoded to provide a pair of signals BDIR and BC1 for use with an AY-3-8910 or 8912 programmable sound generator. This decoding is all that the PSG requires to both write to, and read from its full complement of registers, and details of the use of these lines will be given later in the series.

IC4 (74LS138) decodes 8 Read lines in response to address lines A0–A3, $\overline{BL0}$. R/W and Ø2. The reason why 16 Write lines have been made available, and only 8 Read lines, is largely the additional Write requirement imposed by the use of 7-segment l.e.d. readouts, to be featured later. The addresses of these 24 lines are given in Table 1.9. As will be seen, a number of these have been ear-marked for particular projects in the series.

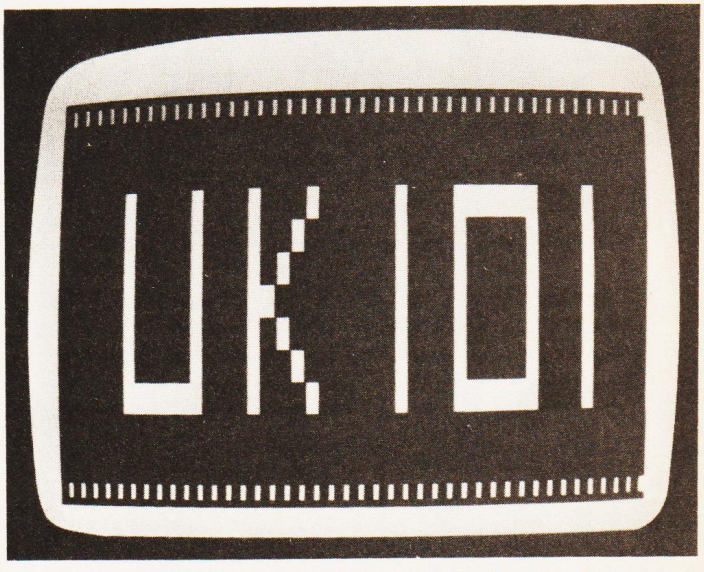
The NAND gate IC8A is used to produce the DD control signal. This is brought low when a Read is carried out at any of the addresses within the 128-byte block used by the module. The reason why it is not simply connected directly to the inverted R/W line is that this would cause the data bus of the CPU to receive extraneous noise from all interfaces during every memory Read cycle, whereas with the present circuit, interfaces are only given access to the CPU when the 128-byte block is called up.

As may be seen from Fig. 1.5, the 6821 PIA (IC9) is selected by the $\overline{BL1}$ line. Address lines A2 and A3 are wired to high-Enable Chip-Select pins 22 and 24, so placing the four addresses used by the PIA at the top of Block 1 (61340–61343 dec). Lines A0 and A1 are used internally by the PIA for decoding within the 4-byte block. Since block 1 is 16 bytes wide, there is clearly room for a further 3 PIAs to be located within it. This could be accomplished by wiring additional PIAs exactly as for IC9, but with inverters in either A2 or A3 or both; although it would probably be easier to locate them in one of the 6 unused blocks (BL2–BL7) whose signals appear on SK6.

The PIA's pin 25 is a master Enable against which all of its operations are timed. It is connected directly to the Ø2 clock. Pin 34 is a 6502-compatible $\overline{RESET}$ line. This has not been taken to the UK 101's Reset circuitry; one of the reasons for this being that the Compukit $\overline{RESET}$ signal does not actually appear at the expansion socket. Pin 34 is taken instead to a pair of pads intended for the connection of a pushbutton which may be used to simultaneously reset all devices wired to the Decoding Module. Capacitor C4 is used to give a power-on Reset. It does this by simply holding the $\overline{RESET}$ line low for a fraction of a second after power-up. A similar technique can be used on the Compukit itself: connecting 100 µF from pin 40 of the 6502 to earth will ensure that the D/C/W/M? message appears instantly at switch-on.

The Decoding Module power supply circuitry is quite straightforward, and employs a 7805 regulator to produce +5 volts at about 500 mA, and a zener stabiliser to give −5 volts at about 30 mA. The Module draws about 100 mA from the positive supply, leaving ample in reserve for driving external devices. The negative supply is generally intended for use with dual polarity analogue i.c.s that will be encountered in D/A and A/D conversion later in the series, and is not used by the Module itself.

The power supply requires a 9-0-9V a.c. transformer rated at 1A. It should be possible to tap Compukit's 3A transformer for this purpose providing it is not already heavily loaded.



Large characters produced by the joystic drawing routine

SK1 — TO COMPUKIT EXPANSION SOCKET — 1 off 2 × 22 pin edge connector
SK2 — 40 PIN DIL — ALLOWS FOR FURTHER EXPANSION OF COMPUKIT
SK3 & SK4 — 16 PIN DIL SOCKET — PORTS A & B OF PIA PIA
SK5 — 24 PIN DIL SOCKET — DECODING MODULE OUTPUT
SK6 — 1 OFF 2 × 25 PIN 0·1" EDGE CONNECTOR — DECODING MODULE OUTPUTS

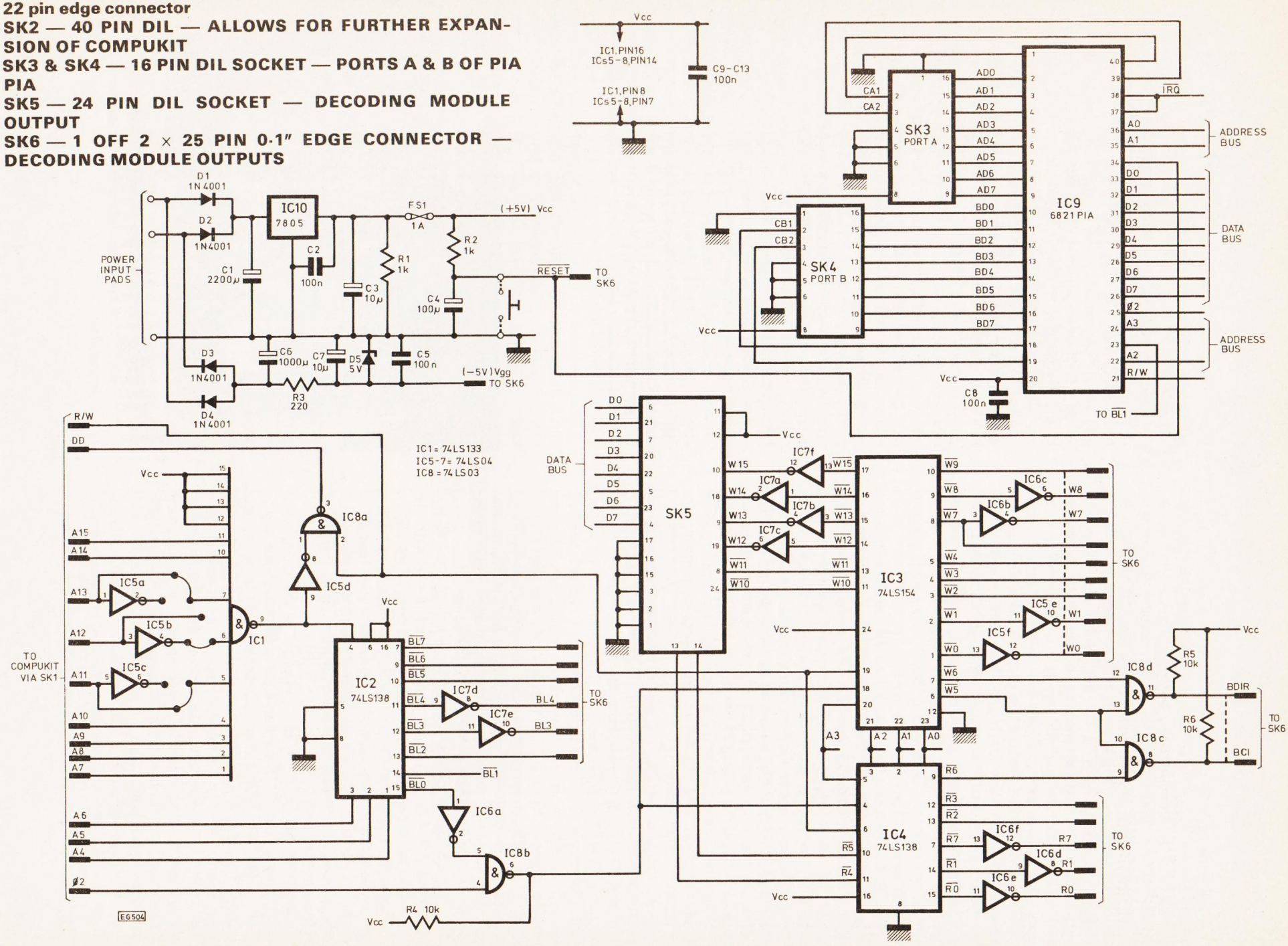


Fig. 2.1. Full circuit diagram. See Points Arising in this issue, concerning Part 1

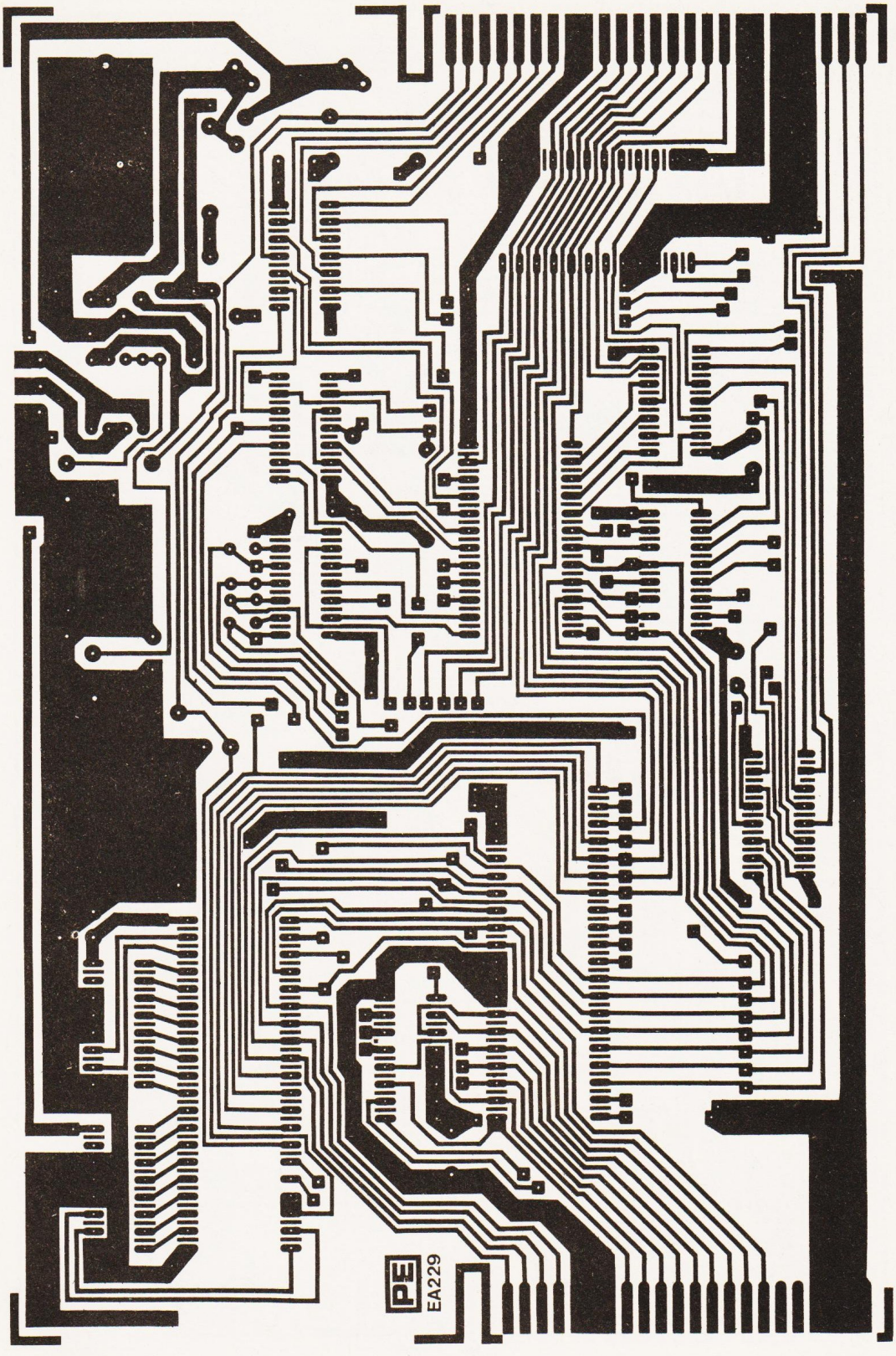


Fig. 2.2. Copper-side p.c.b. layout (actual size)

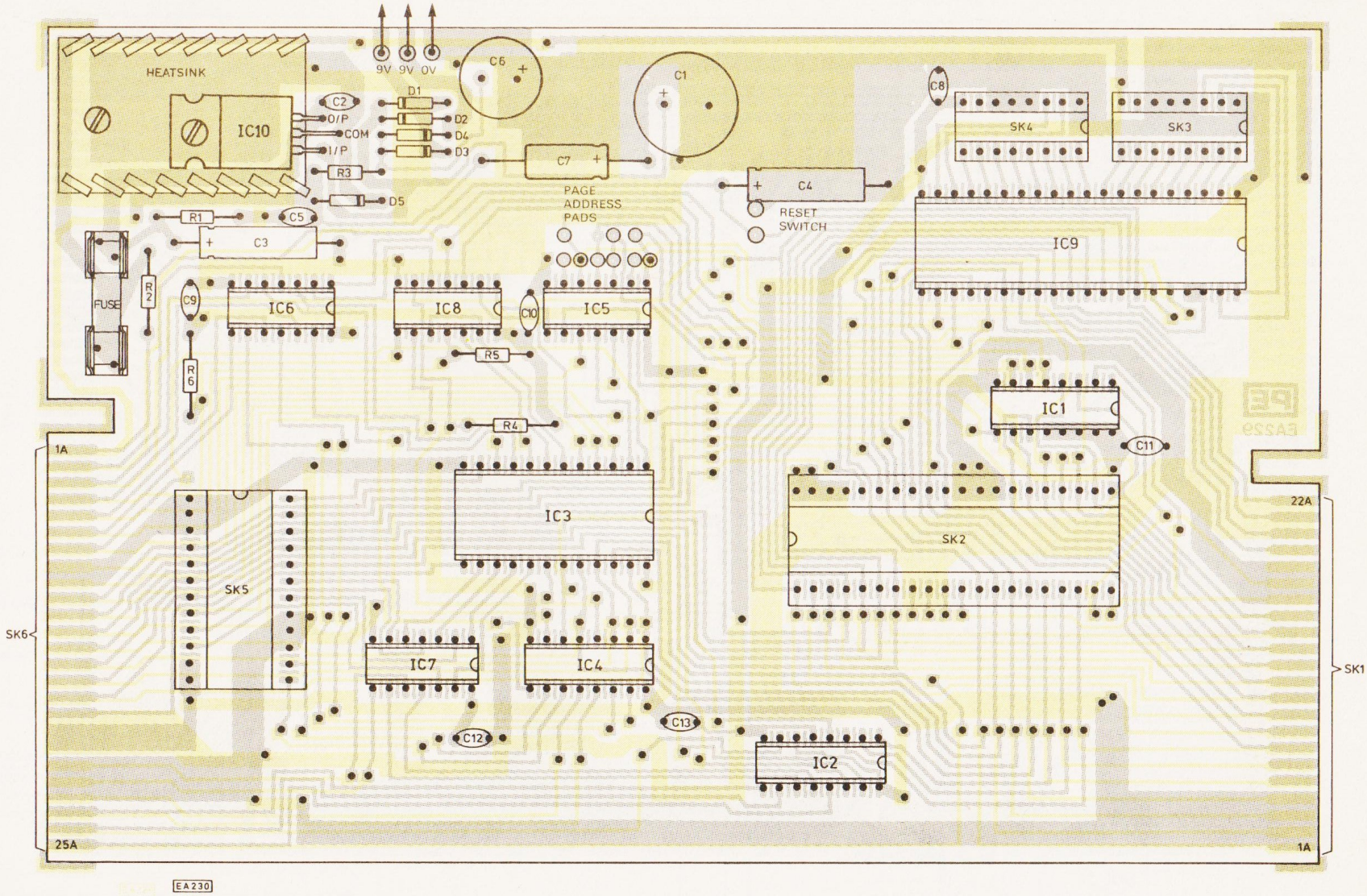


Fig. 2.4. Component layout. A complete kit of parts for the Decoding Module, including i.c.s, p.c.b., headers, edge connectors, ribbon cable etc., (but excluding transformer & 8T28s) is available from: Technomatic Ltd., 17 Burnley Road, London NW10. Price: £27.50 exc. VAT. Post free

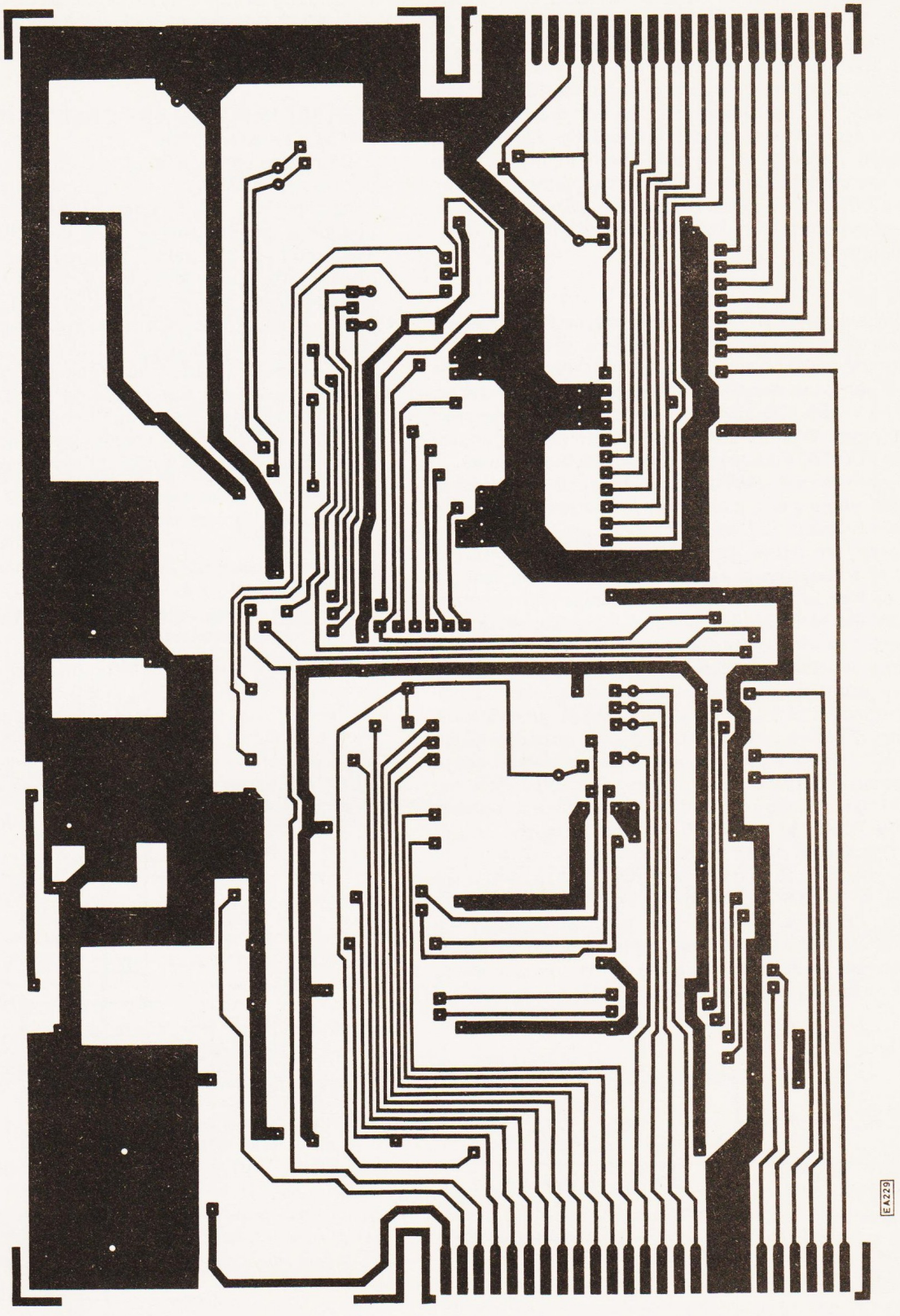


Fig. 2.3. Component-side p.c.b. layout (actual size)

CONSTRUCTION

Construction of the Decoding Module should pose no real difficulties to those experienced with soldering. It might be useful to put in all i.c. holders and other components before tackling the through-holes. The positions of these are indicated in the component overlay drawing, and may be identified in almost all cases as holes surrounded by a square rather than a round track on both sides of the board. All through-pins should be separately soldered on both sides of the board, and a check made that *both* joints are in tact at the end of the operation. The voltage regulator, IC10, should be mounted on the heat-sink with a 6BA bolt or similar.

Note that if it is intended to tap Compukit's PSU transformer to supply the Module, the connection should be made *directly* to the transformer tags, and *not* to the three pads on Compukit's board. This latter precaution will reduce mains hum due to the earth loop, keeping it within reasonably acceptable limits.

TESTING

After checking with an ohm-meter that the PSU lines are not shorted to earth, connect the Module to the 9–0–9 V supply with the fuse and all i.c.s except IC10 removed. If this produces 5 volts, the remaining i.c.s may be inserted, taking particular care with IC9, since this device may be damaged by high voltages. With all i.c.s. inserted, the Module should draw about 100mA (measured in series with the 1A fuse).

Next, connect the Module to Compukit's expansion socket. This should not cause any significant change in current consumption. If Compukit "locks up" at this point and refuses to Reset, then there is probably a short in the Ø2, R/W, data or address lines, or the DD line has been brought permanently low. Check for these possibilities with an ohm-meter (with the Module disconnected from Compukit and from its power supply), or by disconnecting the leads to SK1 one at a time until the fault clears.

When all appears to be well, connect a high impedance voltmeter to pin 16 of SK3. This should read about 4·5 volts. Ensure that IC6 has been reset, and then execute POKE 61340, 255. This sets port A to output, and should cause the voltage on pin 16 to drop to about 0·2 volts. Now execute POKE 61341, 255 (to call up the peripheral register) followed by POKE 61340, 255 (to put 1s in the output register). This should cause the voltage on pin 16 to rise to about 4·5 volts, and should indicate that ICs 1, 2, and 9 are operational. If pin 16 refuses to go high on resetting the 6821, this indicates a fault not associated with the decoding. If it is high, but failed to go low on POKEing 61340, then the fault could either be in the connections to the 6821 or a failure in the decoding.

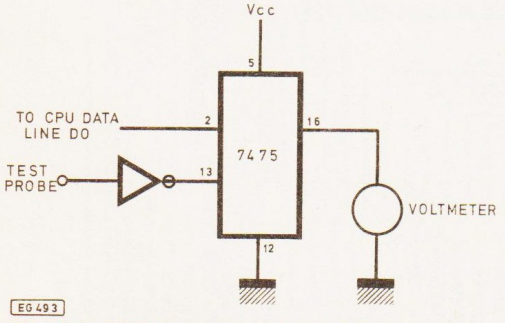


Fig. 2.5. 7475 Latch for testing the module

To check the decoding you will either need an extremely good oscilloscope, or a bistable latch. Fig. 2.5 shows a 74LS75 latch and inverter wired up for this purpose. The first point to check is the output of IC1. Connect this to the test probe, and POKE any address within the 128 byte block used (ie 61312–61439) with a 1 and then with a zero alternately. The voltmeter on the latch output should follow this, so verifying that IC1 is decoding the base address. The sixteen outputs of IC3 may also be tested with the probe. To test for the full operation of the Read circuitry a tristate buffer should be used (see below). Some indication as to the functioning of IC4 may be obtained by using the data latch of Fig. 2.5. This should read a '1' when connected to the outputs of IC4 when PEEK commands are executed at the appropriate address.

DIGITAL INPUT TO THE COMPUKIT TRISTATE BUFFERS

Digital data may be input to Compukit either directly using the Decoding Module's PIA, or indirectly through the use of tristate buffers. The function of the latter is simply to keep peripheral circuits isolated from the CPU data bus until the exact moment that data is required from it. Fig. 2.6 shows a tristate buffer buffering a single data line. With X high data will pass freely to the CPU, but when it goes low the buffer adopts a high impedance state. Eight such devices would be required to buffer a full 8–bit data bus, in which case the Enable lines X would all be connected to the same address-decoded line from the Decoding Module. A PEEK to the given address would then allow the CPU to read the data at the buffer inputs, and as soon as the Read was complete, the CPU would return the buffers to a high impedance state.

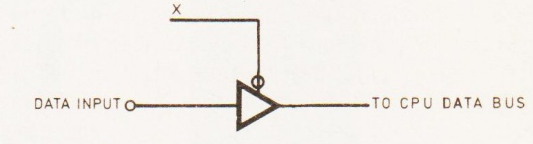


Fig. 2.6. Tristate buffer on single data line

The chief advantage of using this method over the PIA are its lower cost, and the fact that it does not require initialisation before use. This must of course be balanced against the extra complexity of wiring involved. There are also a number of more complex buffers on the market (eg the 74241) which are easier from the wiring point of view, but this is offset by their greater cost per bit.

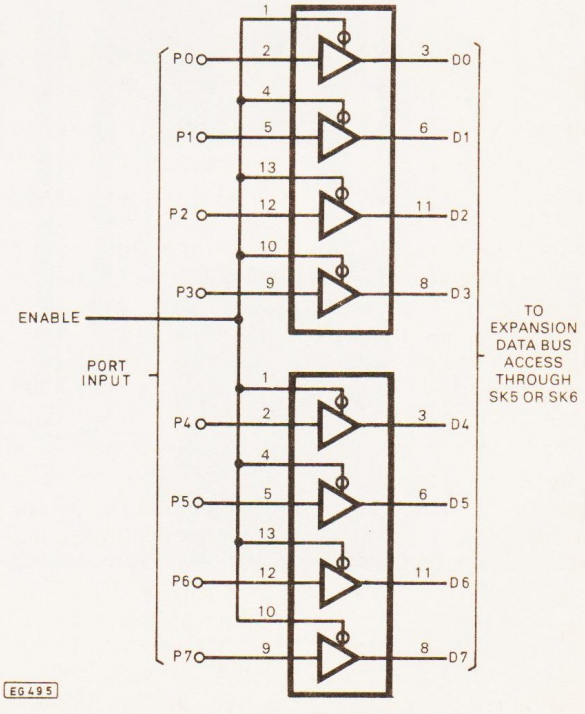


Fig. 2.7. 74LS125 8 bit port

In the event that more input ports are required than the Decoding Module PIA provides, Fig. 2.7 gives the circuit of a pair of 74LS125s wired as an 8-bit input port that may be used directly with the Module. The Enable, being active-low, is directly compatible with the Decoding Module's non-inverted Read lines.

SWITCHES AND JOYSTICKS

The 6821 and PIA or a 74125 port can be easily used for the input of information from switches or push-buttons. Fig. 2.8 shows one way in which this may be achieved. The resistor value of 1k used here is appropriate for either port of the PIA or for a 74125 port.

Interfacing a games joystick for digital operation is also easily accomplished (the more complex interfacing of joysticks for analogue control will be dealt with in a later article). The two-axis joystick consists of two linear potentiometers of about 100k mounted at right angles, and can be conveniently interfaced in a similar manner to the switches. Fig. 2.9 gives the circuit for a control box containing 4 push-buttons and a joystick. The 1k resistors may be mounted inside the box, and a single 10-strand ribbon lead used to connect this to the Decoding Module.

A truth table for the four least significant bits of this circuit is shown in Table 2.3. As may be seen, there are nine possible configurations, including the four diagonal directions. The great advantage which the use of switches and joysticks confers over the polled keyboard for games and other uses is that it is possible to activate any number of switches etc. simultaneously without blocking the input. In the case of the joystick alone this simply means that "diagonal" instructions can automatically be accepted as well as "vertical" or "horizontal" ones. But it also implies that any combination of the four push-buttons may be simultaneously acted upon.

It is also a simple matter to extract the relevant information from the binary data at the port. The easiest way to do this is to use the extremely useful AND operator in Compukit's BASIC. If this operator is used on decimal numbers rather than on expressions, it converts those numbers to binary, and behaves like a set of eight 2-input AND gates. Thus the instruction PRINT 13 AND 25 will give the result 9.

Converting 13 and 25 to binary and ANDing them we can see why.

```
13   00001101
25   00011001
 9   00001001
```

The only bits which are 1 in both numbers are the first and the fourth, which gives the binary representation of 9. This function makes decoding of the joysticks and other digital data an extremely easy matter.

To test whether the joystick is in the "up" position one can simply PEEK the corresponding port and AND the result with the binary number 00000010. Clearly, the result will only be non-zero if the second bit of the joystick data is also non-zero; ie if it is in the "up" position. Similar operations can be performed for the other three bits, and since each operates completely independently of the others, the diagonal positions will automatically be catered for. Thus the following four program lines could be used to move a cursor in eight different directions across Compukit's screen. X and Y are the horizontal and vertical screen positions, and A the address of the port.

```
100  IF(PEEK(A) AND 1) >0 THEN Y=Y+1
110  IF(PEEK(A) AND 2) >0 THEN Y=Y-1
120  IF(PEEK(A) AND 4) >0 THEN X=X-1
130  IF(PEEK(A) AND 8) >0 THEN X=X+1
```

In order to demonstrate these principles in action, a simple program is given for screen writing, using the joystick and push-button circuit of Fig. 2.9. The joystick is used to move a flashing cursor to any point on the screen, and the four buttons have the following functions: 1 (at bit 4) Draw, 2 Erase, 3 Change character, 4 Clear screen. The program, which is listed in Table 2.4 is extremely short, but allows intricate graphics work to be executed on the screen.

The techniques used here could obviously be implemented in a number of different ways for games purposes, and one could easily add a second joystick to allow for two-person games. In next month's issue, when we will be discussing digital output techniques, we will give circuitry and

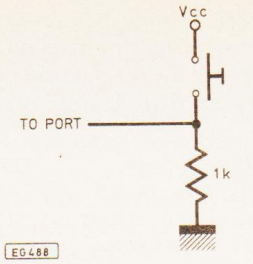


Fig. 2.8. (Left) push button inputs

Fig. 2.9. (Below) Joystick control box

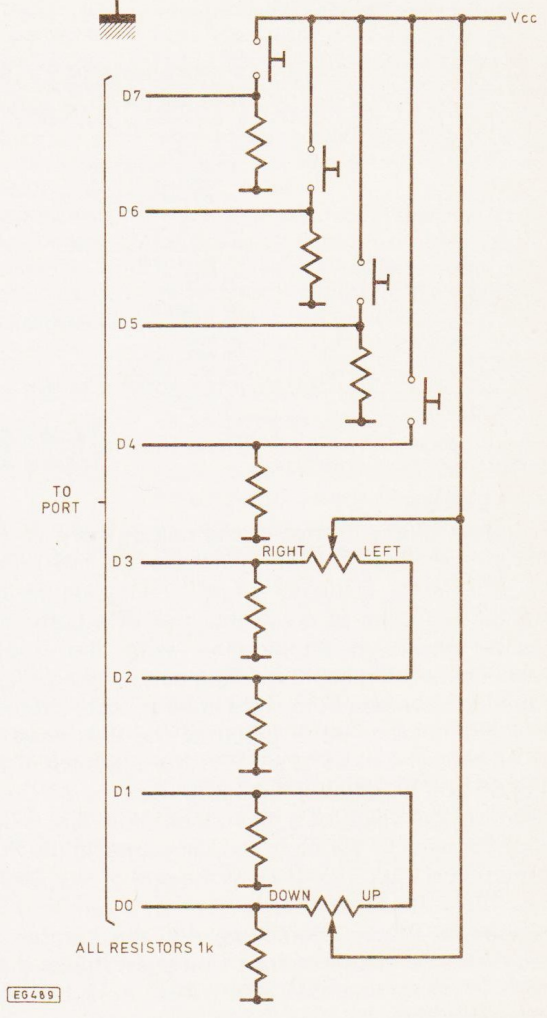


Table 2.3. Truth table for Joystick

Position of Joystick	Four Least Significant Bits			
	P3	P2	P1	P0
Centre	0	0	0	0
Down	0	0	0	1
Up	0	0	1	0
Left	0	1	0	0
Right	1	0	0	0
Down and Left	0	1	0	1
Down and Right	1	0	1	0
Up and Left	0	1	1	0
Up and Right	1	0	1	0

soft-ware for adding a 7-segment l.e.d. display to indicate the position of the cursor in the above program. This is a useful complement and enables the program to be used to set up graphics work for transferral of both BASIC and 6502 code programs.

```
OK
LIST
 80 REM P.E. INTERFACING UK101 PROG 2
 90 REM JOYSTICK DRAWING ROUTINE
 95 REM (WITHOUT LED DISPLAY)
 100 FORI=0TO15:PRINT:NEXT
 110 V=53260
 120 X=23:Y=8
 125 VP=V+X+64*Y
 130 P=61340
 140 C=161
 150 POKEP+1,0:POKEP,0
 160 POKEP+1,255
 170 Q=PEEK(P)
 180 IF(QAND64)=0THEN200
 184 C=C+1:IFC=191THENC=128
 186 POKEVP,C
 188 FORI=0TO300:NEXT
 200 IF(QAND1)>0ANDY<15THENY=Y+1
 210 IF(QAND2)>0ANDY>0THENY=Y-1
 220 IF(QAND4)>0ANDX>0THENX=X-1
 230 IF(QAND8)>0ANDX<47THENX=X+1
 240 IF(QAND128)>0THEN100
 250 VP=V+X+64*Y
 260 C1=PEEK(VP)
 265 POKEVP,35
 267 FORT=0TO100:NEXT
 270 POKEVP,C
 280 IF(QAND16)>0THEN170
 290 IF(QAND32)>0THENPOKEVP,32:GOTO170
```

Table 2.4. Left

```
 300 POKEVP,C1
 310 GOTO170
OK
LOAD

OK
LIST
 30 REM P.E. INTERFACING UK101 PROG 3
 40 REM UK101 LOGIC TESTER
 50 REM NOTE - RETURN KEY GIVES SCREEN
 55 REM RECORD OF LOGIC STATES
 60 V=54125
 70 P=61340
 80 POKEP+1,0:POKEP,0
 90 POKEP+1,255
 100 FORI=0TO15:PRINT:NEXT
 110 PRINTTAB(13);"UK101 LOGIC TESTER"
 115 PRINT:PRINT
 120 PRINTTAB(11);"D7 D6 D5 D4 D3 D2 D1 D0"
 130 PRINT:PRINT
 200 POKE530,0
 210 FORI=0TO7
 220 Q=PEEK(P)
 230 POKEV-3*I,((QAND(2↑I))/(2↑I))+48.1
 250 NEXT
 270 POKE530,1
 280 POKE57088,223
 290 IFPEEK(57088)=247THEN120
 300 GOTO200
OK
LOAD
```

Table 2.5. Above

LOGIC TESTER

It is a very simple matter to use one or more of the ports developed with the Decoding Module for the purpose of testing logic state in digital circuitry. This simply involves connecting test clips to the eight lines of a particular port, and using these in conjunction with the appropriate software. Table 2.5 gives the listing of a program which displays the logic states of the eight lines of port A of the PIA. If 74125s are to be used in place of the PIA, then line 70 should be changed so as to set P to the appropriate address, and line 80 and 90 deleted. Line 230 uses the AND operator to determine the value of a given input line, and it performs this once for each of the eight data lines, using the FOR loop at program line 210. The 48·1 at the end of line 230 should really be 48·0. Its function is to convert a zero or 1 into the ASCII code for those characters, and the number 48 will normally achieve this; but since Compukit thinks that 2↑16 is 65536·2 (rather than 65536·0), 48·1 must be used so as to convert correctly.

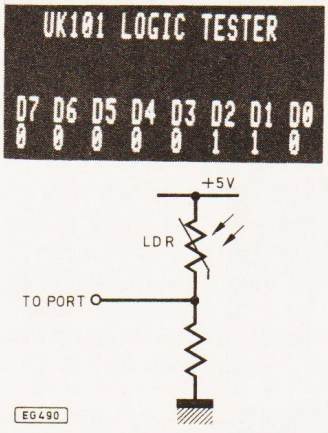
The photographs show Compukit's screen during the running of the program. Pressing Return at any time causes the current logic states to be printed as a record higher up the screen as may be seen. Note that since the program is written in BASIC, the response time is relatively slow.

OTHER INPUTS

There is of course no limit to the variety of devices that can be interfaced using the 6821 PIA or a 74125 port. In fact any device capable of producing a transition between about 0·5 and 3·5 volts (less for port A of the PIA) can be directly connected to them. We shall briefly look at the implementation of light and sound detectors, although here, as in many other cases, far greater information can be derived from such sources using analogue to digital conversion techniques, to be treated later in the series.

Light dependent resistors such as the ORP 12 are amongst the easiest light detectors to interface because their parameters change over many decades for relatively small changes in illumination. It is even possible to wire these directly to the port using a 1k resistor to ground as shown in Fig. 2.10. If it is required to detect lower levels of light than this circuit permits, a simple transistor amplifier may be used as in Fig. 2.11. With the high gain 2N2926G transistor this is capable of detecting very low levels of light.

If desired, a potentiometer could be inserted in the base circuitry of the transistor so as to provide some degree of level control. The circuit of Fig. 2.11 differs from that of Fig. 2.10 in that it takes the port low when light is present. This effect can easily be reversed in software by subtracting the data read at the port from 255. This will cause each of the eight inputs of the port to be effectively active-low rather than active-high.



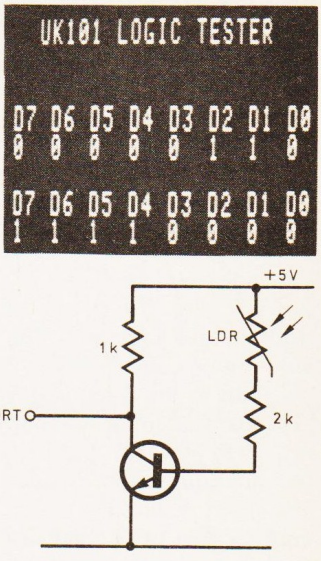
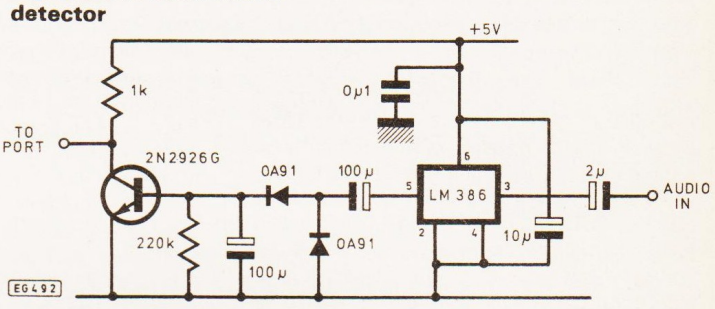
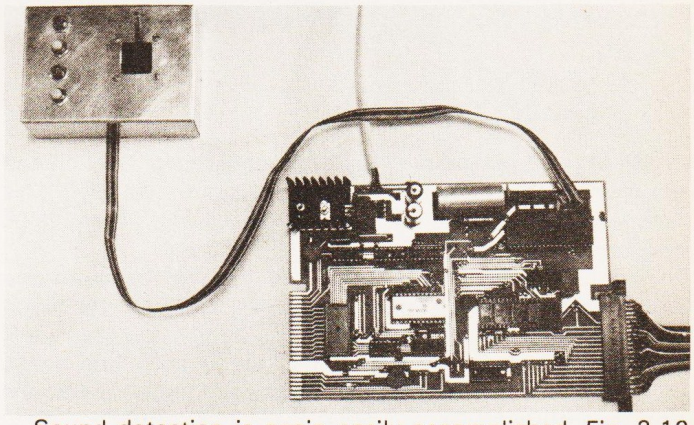


Fig. 2.10. (Above) LDR directly connected

Fig. 2.11. (Right) High Sensitivity light detector

Fig. 2.12. (Below) Audio detector





Sound detection is again easily accomplished. Fig. 2.12 gives the circuit of a simple audio detector using an LM386 audio amplifer, which will run happily at 5 volts. The amplifier is sufficiently sensitive for the circuit to operate with a high output dynamic microphone, but if greater sensitivity is required, a preamplifier should be used. In setting up this and other digital interfaces it will be found that the Logic Tester program of Table 2.5 can be used to provide a convenient way of monitoring the state of the port in use.

Next month we will look at the use of the PIA and data latches to output digital data from the Compukit, and will discuss various applications including an IC tester for devices of the 7400 series.

Interfacing COMPUKIT

Part 3 D.E.Graham

THIS month we shall be looking at the output of data from Compukit in digital form through the 6821 PIA and through sets of latches, to control devices from relays to 7-segment displays; and will cover applications such as a 7400 series i.c. tester, and full interfacing for the *P.E. Speech Synthesis Unit.*

DIGITAL OUTPUT

The Decoding Module described in parts 1 and 2 of the series allows the Compukit to output up to 16 parallel bits of data through the MC6821 PIA. The Module also provides a number of address-decoded Write Enable lines which may be used to activate sets of latches, to provide an alternative data output. These may be connected to Compukit's data bus via SK5 or 6 of the Decoding Module; and when enabled will store data appearing instantaneously on the bus, and make it available for use at the latch outputs. This output will be maintained until the latch Enable is retriggered by the appropriate address-decoded Write line from the Module.

From the range of t.t.l. latches available for this purpose, we have selected the commonly used 74LS75 quad latch. It is somewhat less convenient to use than more complex devices such as the 74116, which provides reset facilities and allows for active-low Enable, but has the advantage of relatively low cost, and is readily available in LS. Fig. 3.1 shows a pair of 74LS75s wired to provide an 8-bit port that connects directly to the Decoding Module. Note that only the Module's active-high Enable lines are suitable for connection to this latch, such as for example, W12, 13, 14 or 15 at pins 19, 9, 18 or 10 of SK5.

Note also, that if latches etc. are to be powered as well as enabled by the Decoding Module, then more than one earth connection should be made to it.

SEVEN-SEGMENT L.E.D.S

Either port of the PIA, or a 74LS75 port may be used to control 7-segment l.e.d.s using an intermediary as decoder-driver such as the 7447.

The circuit of Fig. 3.2 is for a two-digit Display Unit that plugs directly into SK5 of the Decoding Module to provide Compukit with digital readout, whilst leaving the PIA completely free for other uses.

The circuit uses a pair of 74LS75s to drive a pair of 74LS47 decoders which in turn drive a pair of FND 507 low cost common anode displays. These plug into a single 24-pin d.i.l. socket mounted on the board. The 7475s are both connected to the lowest four bits of the data bus, and are activated by two *separate* Enable lines from the Module. This procedure makes software control easier than it would have been had both l.e.d.s been run from the full 8-bits at a single address.

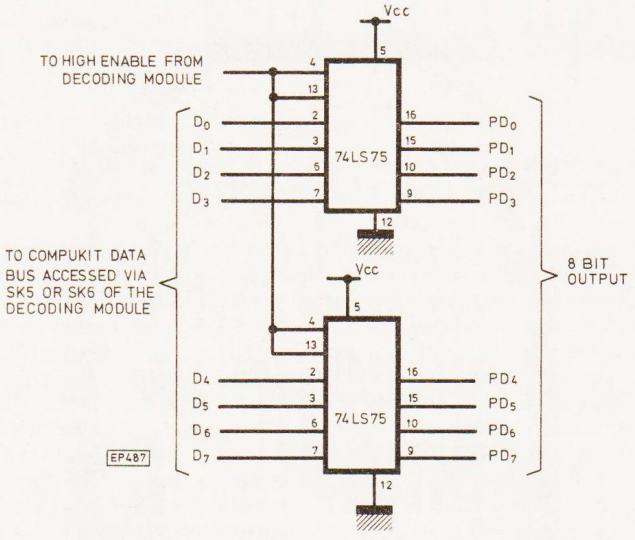


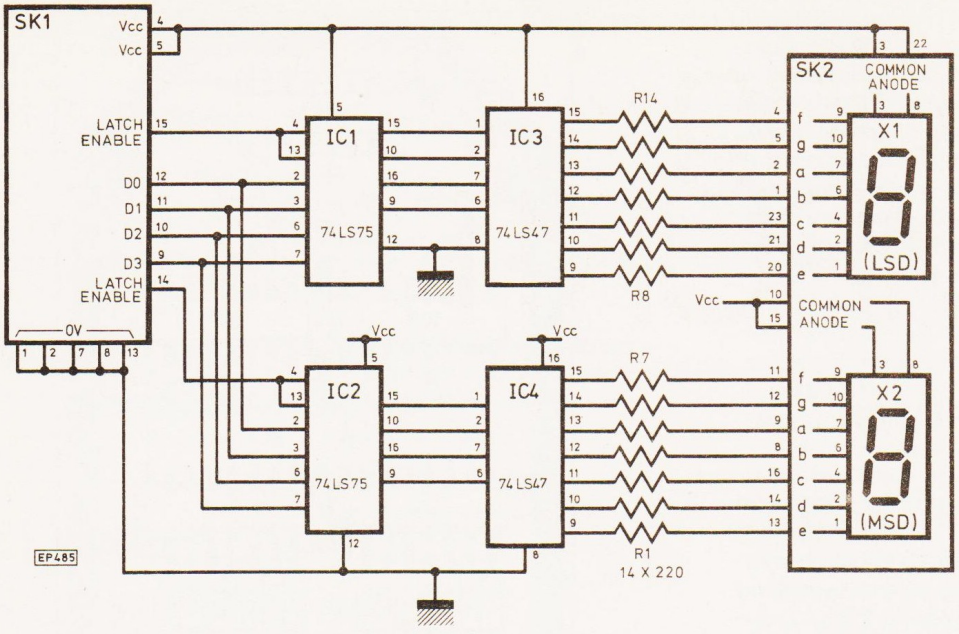
Fig. 3.1. 74LS75 8-bit Port

Figs. 3.3 and 3.4 give p.c.b. artwork and component overlay for the Display Unit. All connections to the board are via a single 16 pin d.i.l. socket which connects to SK5 on the Decoding Module, providing data bus, Write Enables and 5 volt supply. See Table 3.1 for pin connections.

The unit should not draw much more than 200mA when displaying "88" so that it should be possible to run a pair of these boards from the Decoding Module power supply, to give a four digit readout (in which case pins 15 and 14 of SK1 on the second display board should be connected to pins 18 and 10 of SK5 on the Decoding Module). If two Display Units are used however, the Decoding Module power supply will not be able to support the next add-on board of the series, to be introduced next month.

Using the Display Board is extremely simple. The command POKE 61324, X: POKE 61325, Y will display the number YX on the display (where Y and X are integers between 0 and 9).

Table 3.2 gives a program that will count seconds from 0 to 99 in decimal on the l.e.d. display located at 61324 and 5. Note how the two count digits are separated out in lines 130 and 140, and POKEd to the two separate addresses. It should be possible to achieve timing accuracies of up to about 0·1 per cent with this program by adjusting the length of the waiting loop on line 160.



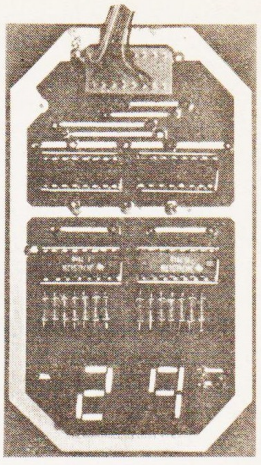
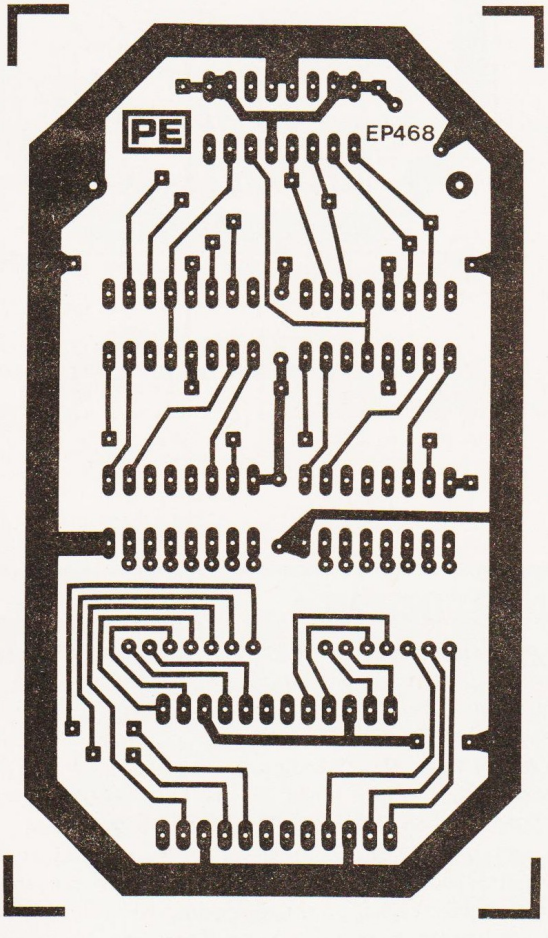


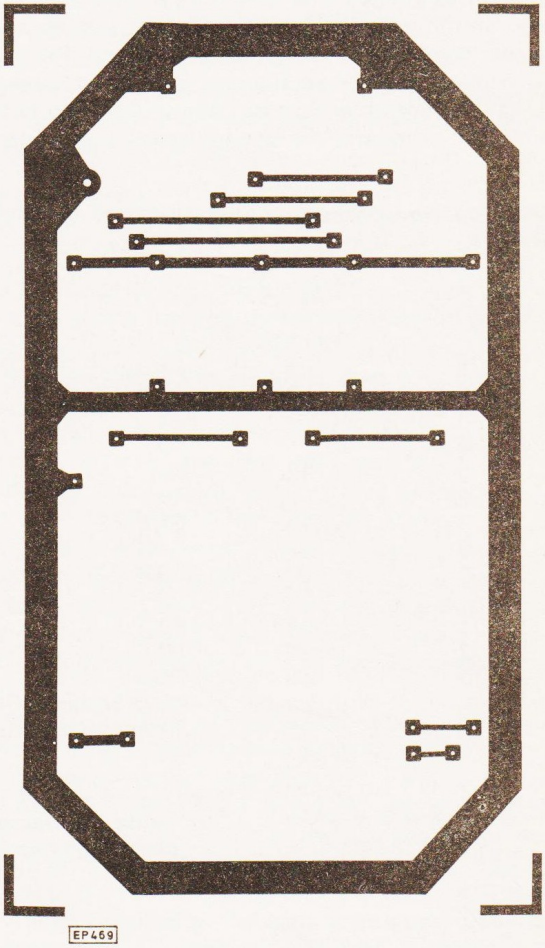
Fig. 3.2. 7-segment display circuit

Fig. 3.3. (Below), p.c.b. for Display Unit (actual size). (a) Copper side (b) Component side



a

b



Incidentally, although the count is only taken up to 9 on each digit in this program, the 7447 will in fact decode for the hex values A-F using the symbols shown in Table 3.3, so that it would be possible to display values up to FF hex (or 255) with the Display Board, albeit at some loss in ease of readout.

By employing similar techniques, it is possible to add a routine to the joystick screen writing program given last month so as to provide a digital readout of the screen address of the cursor. This is a useful facility in setting up graphics work, and the use of l.e.d.s for outputting the data is particularly convenient in that it leaves the full screen clear for graphics development. The full program is given in Table 3.4. As may be seen, the bulk of the work is carried out in a routine starting at line 400. This causes the two-digit display to show a sequence consisting of the first, second and third pairs of digits of the 6-digit screen address currently occupied by the cursor. The display then goes blank before repeating the sequence.

AUDIO OUTPUT

There are many ways in which Compukit can be interfaced for the production of sound. About the simplest is to connect a single bit of an output port directly to an audio amplifier, and then generate a series of pulses in software. Fig. 3.5 gives a circuit that can be connected to one bit of either port of the 6821 or to a 74LS75 latch. When used with the following program it will produce a square wave output at about 140Hz with a mark to space ratio of about 1:3 on all bits of Part A on the PIA.

```
100 P=61340
110 POKE P+1,0 : POKE P,255
120 POKE P+1,255
130 POKE P,255
140 POKE P,0
150 GOTO 130
```

The output frequency is limited by the speed of Compukit's BASIC interpreter, though this may be enhanced somewhat by using variables (eg X and Y) in place of the 0 and 255 of lines 130 and 140, and giving these the appropriate values at the start of the program; and by adding the contents of lines 140 and 150 to that of line 130.

If higher frequencies are required, it is necessary to resort to programming in 6502 machine code, which can then be accessed from BASIC using the USR(X) call. Table 3.5 gives an assembler listing of such a program. It was assembled on the UK101 Assembler/Editor. Column 1 of the listing gives dummy line numbers; the second gives the actual hex ad-

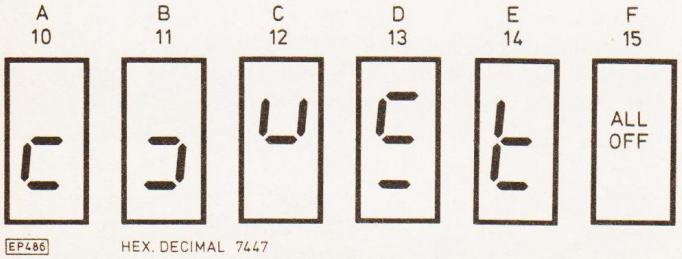
Table 3.1 Connections of Display Board to Decoding Module

SK1 on Display Board	To SK5 on Decoding Module	Function
1	1	GND } Not connected if 10-strand ribbon cable is used
2	2	GND }
3	/	n/c
4	11	Vcc
5	12	Vcc
6	/	n/c
7	15	GND
8	16	GND
9	20	D3
10	7	D2
11	21	D1
12	6	D0
13	/	GND
14	9	Write Enable (MSD)
15	19	Write Enable (LSD)
16	/	n/c

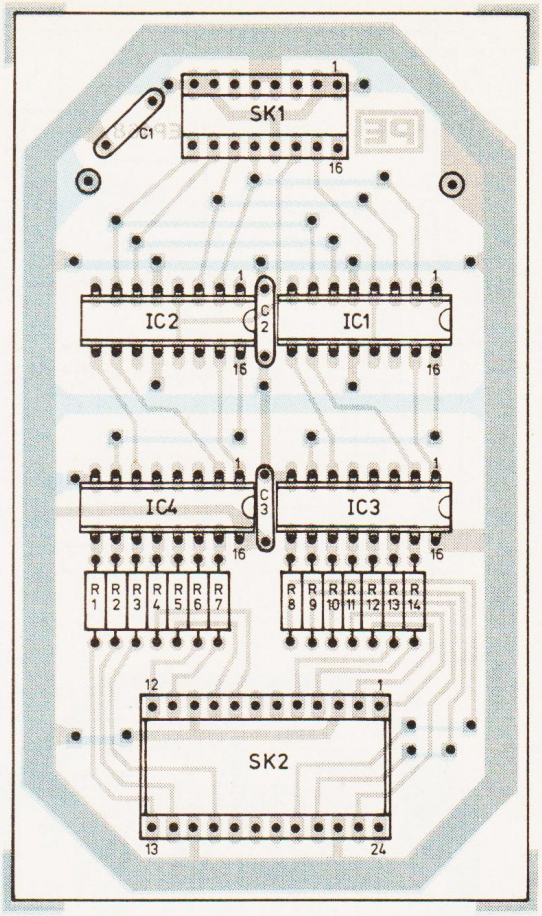
```
80 REM INTERFACING UK101 PROGRAM 4
90 REM 0-99 COUNTER ON61324/5
100 P=61324
120 FORA=0TO99
130 A1=INT(A/10)
140 A2=A-10*A1
150 POKEP,A2
155 POKEP+1,A1
160 FORT=1TO1100:NEXT
170 NEXT
180 GOTO120
```

Table 3.2. Seconds counting program

Table 3.3. 7447 symbols



EP486 HEX. DECIMAL 7447



EP470

Fig. 3.4. Component layout for Display Unit

COMPONENTS . . .

DISPLAY MODULE

Resistors

R1–R14	220 ¼W (14 off)

Capacitors

C1–C3	100n low voltage disc. cer. decoupling (3 off)

Semiconductors

IC1, IC2	74LS75 (2 off)
IC3, IC4	74LS47 (2 off)
X1, X2	FND 507 (2 off)

Miscellaneous

SK1	16 pin d.i.l.
SK2	24 pin d.i.l. (for displays)

16 pin d.i.l. sockets (4 off)
Printed circuit board
16-pin d.i.l. header
10-strand ribbon cable

Constructors' Note

A complete kit of parts is available from **Technomatic Ltd., 17 Burnley Rd., London NW10.**

```
80 PEM INTERFACING UK101 PROGRAM 5
90 REM JOYSTICK DRAWING ROUTINE
95 REM WITH LED SCREEN POSN READOUT
96 REM POKED TO 61324/5
100 FORI=0TO15:PRINT:NEXT
110 V=53260
120 X=23:Y=8
125 VP=V+X+64*Y
126 VE=VP:V1=0:V2=0
130 P=61340
140 C=161
145 J=5
150 POKEP+1,0:POKEP,0
160 POKEP+1,255
165 R=61324:REM LED ADDRESS *******
170 Q=PEEK(P)
180 IF(QAND64)=0THEN200
184 C=C+1:IFC=191THENC=128
186 POKEVP,C
188 FORI=0TO300:NEXT
210 IF(QAND2)>0ANDY>0THENY=Y-1
220 IF(QAND4)>0ANDX>0THENX=X-1
230 IF(QAND8)>0ANDX<47THENX=X+1
240 IF(QAND128)>0THEN100
250 VP=V+X+64*Y
260 C1=PEEK(VP)
265 POKEVP,35
270 B=B+1:IFB=5THENB=0:GOTO400
275 FORT=0TO100:NEXT
277 POKEVP,C
280 IF(QAND16)>0THEN170
290 IF(QAND32)>0THENPOKEVP,32:GOTO170
300 POKEVP,C1
310 GOTO170
400 IFJ<1THENPOKER,255:POKER+1,255:J=5:VE=VP:V2=0:GOTO500
410 VE=VE-V2*10↑(J+1)
420 V1=INT(VE/(10↑J))
430 VE=VE-V1*10↑J
440 V2=INT(VE/(10↑(J-1)))
450 POKER,V2:POKER+1,V1
460 J=J-2
500 GOTO277
```

Table 3.4. Screen Writer program with readout

```
  5 0000           ;ASSEMBLY LISTING OF SQUARE WAVE GEN
  6 0000           ;AUDIO OUTPUT ON PORT A OF PIA AT 61340
  7 0000           ;RELOCATABLE PROGRAM BASED AT 0230 HEX
 10 0000           START=$0230
 20 0230          *=START
 30 0230          ME1=START-3
 40 0230          ME2=START-2
 50 0230          ME3=START-1
 60 0230 A900      LDA #00
 70 0232 8D9DEF  STA 61341
 80 0235 A9FF    LDA #255
 90 0237 8D9CEF  STA 61340
100 023A 8D9DEF  STA 61341
110 023D AC2E02  STT LDY ME2
120 0240 AE2D02  STU LDX ME1
130 0243 CA      A1 DEX
140 0244 D0FD    BNE A1
150 0246 88      DEY
160 0247 D0F7    BNE STU
170 0249 8D9CEF  STA 61340
180 024C E901    SBC #1
190 024E D0ED    BNE STT
200 0250 CE2F02  DEC ME3
210 0253 D0E8    BNE STT
220 0255 60      RTS
```

Table 3.5. Assembler Listing for Audio output

```
60 REM INTERFACING UK101 PROGRAM 6
70 REM SQUARE WAVE GENERATOR
75 REM USES USR ROUTINE TO PIA AT 61340
77 REM REQUIRES ACCOMPANYING 6502 CODE PROGRAM
80 PRINT:PRINT:PRINT:PRINT
90 PRINT,"SQUARE WAVE GENERATOR"
95 PRINT:PRINT
100 POKE11,48
110 POKE12,2
120 PRINT"  FREQ 1-255 (255 = LF)"
130 INPUTA
135 POKE557,A:POKE558,A
137 PRINT"  DURATION 1-255 (0 OR 255 = LONG)"
140 INPUTB
145 POKE559,B
200 X=USR(X)
210 GOTO80
```

Table 3.6. Basic control program for above

dress in memory; the third, the instruction sequence in 6502 code; and the right hand column gives the assembly language listing, with standard 6502 nemonics. STT, STU and A1 are dummy labels used during assembly.

The program uses data stored in 022D and 022E hex to determine the time period of its output, and the contents of 022F to determine the duration of sound output. It then outputs a square wave on all 8 bits of port A of the PIA, each bit differing by one octave from the next.

To enter the program, column 3 of the listing could be input manually via Compukit's monitor, placing A9 at 0230 hex, and so on, up to 60 at 0255 hex. Alternatively this string of data could be POKEd into the appropriate addresses using a program in BASIC, though addresses and data would first have to be decimalised.

Once the values are in, the short BASIC program in Table 3.6 may be run to access the machine code program via the USR(X) call. Using this set-up the output frequency as measured at bit 0 of the PIA may be controlled from about 2Hz to 20KHz. At bit 7 it is 1/128 of this.

Because the machine code program is located at an unused space before 0300 hex (the start of Compukit's BASIC file space), it is safe from any attempts to erase it (except by switching off). Even a Cold Start will not shift it.

There is of course one obvious limitation to this method of sound production: it ties up the CPU for the whole duration of sound output. We shall examine more economical means of sound production next month.

OTHER OUTPUTS

The PIA and any 7475 port may also conveniently be used for control purposes, with each of the eight bits controlling a separate device. Power handling is easily achieved in such applications through the use of relays. Fig. 3.6 gives a circuit for relay operation from a single bit of such a port. It should be noted that if this circuit is used with port A of the PIA, the relay contacts will be closed even when the PIA is set to input (as it is on Reset), since port A output buffers are not tristate; this is not the case with port B. An alternative way of driving relays from either of the ports is to use a driver i.c. such as the 7416 hex inverting driver, or the 7417, its non-inverting equivalent. These devices will sink up to 40mA per bit, and their open collector outputs may be used with supply voltages up to 15V; although the chip supply on pin 14 must not exceed 5 volts. These i.c.s are also ideal for driving l.e.d. indicators (see Fig. 3.7) and opto-isolators from the PIA.

Software for this type of application is easily written even if all 8 bits of a port are simultaneously in use. One approach to this is to use a function such as A1×1 + A2×2 + A3×4 + A4×8 + A5×32 + A6× 64 + A7×128, where A1 to A8 are variables which take the value of zero for low output on the appropriate bit (relay off), and 1 for high output (relay on)—or the reverse if a non-inverting driver is used as in Fig. 3.7. All that is then required of the control program is to allocate ones or zeros to the 8 variables as desired, and then to POKE this function to the appropriate port.

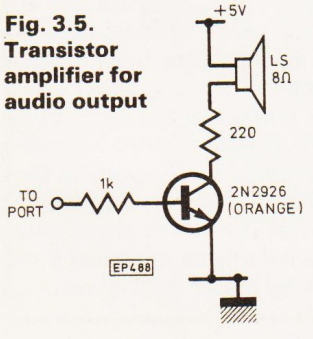
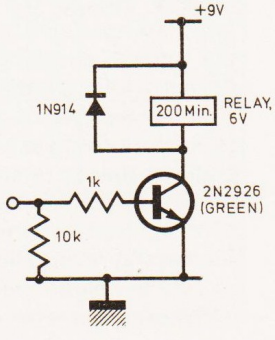


Fig. 3.5. Transistor amplifier for audio output

Fig. 3.6. Relay operation



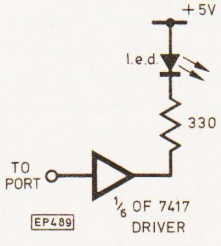


Fig. 3.7. L.e.d. indicator

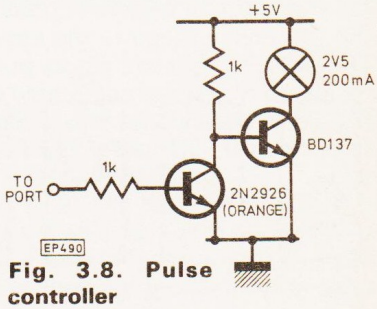
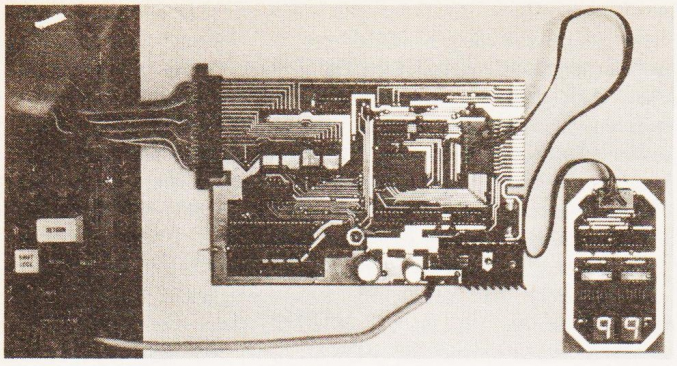


Fig. 3.8. Pulse controller



It is also possible to use each bit of a port to achieve a degree of analogue control by using pulse techniques. This involves generating variable duty cycle pulses in software, and POKEing these to a given bit of an output port which is connected to a current amplifier. In order to illustrate this method Fig. 3.8 gives a circuit for controlling the brightness of a 2·5 volt 200mA torch bulb. A short test program to drive this from any bit of port A of the PIA is given below:

```
50 REM DUTY CYCLE LAMP CONTROL
90 P=61340
100 POKE P+1,0: POKE P,255
110 POKE P+1,255
120 INPUT "BRIGHTNESS 0 (BRIGHT) TO 10 (OFF); X
130 POKE P,0
140 FOR A=1 TO 10 : NEXT
150 POKE P,255: FOR A=1 TO 10*X: NEXT
160 GOTO 130
```

The program requests a number from 0 to 10, and controls the brightness of the lamp accordingly; decimal numbers in the range 0 to 1 giving greatest illumination. This means of control suffers somewhat from the relatively low pulse frequency obtainable in interpreted BASIC, and from the fact that it monopolises the CPU during power output. In many respects a more satisfactory means of achieving analogue power control is to use D/A conversion techniques, details of which will be given next month.

7400 SERIES IC TESTER

The ability of the PIA to configure any of its 16 bits for either input or output allows it to be used as the basis for a t.t.l. i.c. tester. To show how this may be achieved, we give details of a tester suitable for most 14 pin i.c.s of the 7400 series. It will in fact work with all those using pins 7 and 14 for power supply connection, and whose gates are configured symmetrically across the middle, as are those of the 7400 and 7420, for example, but not the 7415, whose third gate has 2 inputs on the LH side and one input and its output on the RH side. The principles used may be extended to cover most 7400 devices with 14, 16 or 18 pins, though removing the symmetry condition will increase data and software complexity.

For the 14 pin tester, a 14 pin d.i.l. socket is wired to a pair of 16 pin headers as shown in Fig. 3.9 which plug directly into SK3 and 4 of the Decoding Module. Pins 7 and 14 of the test socket are taken to ground and Vcc respectively, and the remaining 12 go to the lowest 6 bits of ports A and B. The six 4·7k resistors act as pull-up resistors on the port B inputs, so that when confronted with a high impedance state (as would be the case when testing a 74125 tristate buffer for example), the tester consistently detects a high logic state.

To perform a test, the i.c. is plugged into the socket, and the program listed in Table 3.7 is run on Compukit. This first requests the user for the i.c. number, and checks program lines 5000 onwards to see if it has data on the device. If it does, it sets up the ports for input and output on the appropriate pins and then performs a series of logic tests on the device, checking responses against data held for that device. It then prints out the results.

If the number of the i.c. is not known by the user (or he lacks the energy to enter it), a zero may be entered when the chip number is requested. This causes the program to perform its complete repertoire of tests in sequence until the i.c. is recognised; whereupon the device number is printed out. Successful recognition can of course only be achieved if the i.c is fully operational, and if it is one whose data is included in the program.

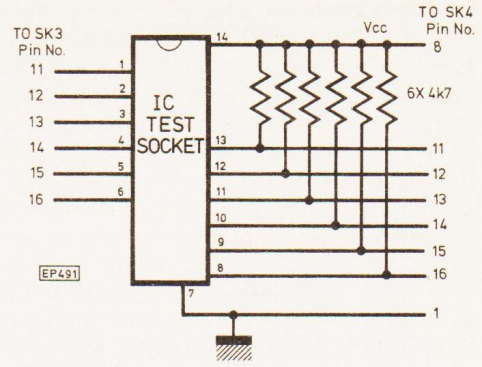


Fig. 3.9. I.c. tester connections

```
2 REM INTERFACING UK101 PROGRAM 7
5 REM UK 101  IC TESTER
10 FORI=1TO16:PRINT:NEXT
15 PRINT,"UK101 7400 SERIES IC TESTER"
20 RESTORE
70 P=61340
80 F=0
90 PRINT:PRINT:PRINT
100 PRINT"    ENTER NUMBER OF DEVICE"
110 PRINT"    OR ZERO, IF NOT KNOWN"
120 INPUTD
130 IFD=0THEN280
140 IFD<7400 OR D>74999THEN170
150 IFD>7499 AND D<74000THEN170
160 GOTO200
170 PRINT:PRINT
180 PRINT"    ONLY 7400 SERIES PLEASE"
190 GOTO20
200 READ D1
210 IFD1>-1THEN240
220 PRINT:PRINT"    NO DATA AVAILABLE ON  THIS DEVICE"
230 GOTO20
240 IFD1<>DTHEN200
250 PRINT:PRINT:PRINT"    DATA AVAILABLE ON  ";D1
260 PRINT
270 GOTO300
280 READD1
285 IFD1<0THEN950
290 IFD1<7400THEN280
300 REM
320 READR
340 POKEP+1,0
350 POKEP,63-R
360 POKEP+1,255
370 POKEP+3,0
380 POKEP+2,63-R
390 POKEP+3,255
400 READS
410 IFS>255 OR S<0THEN770
420 READT
430 F=F+1
440 POKEP,S
450 POKEP+2,S
460 IF(PEEK(P) AND R)<>TTHEN870
470 IF(PEEK(P+2) AND R)<>TTHEN870
480 GOTO400
770 IFD<>0THEN800
780 PRINT:PRINT"    DEVICE RECOGNISED AS  ";D1
790 GOTO830
800 PRINT
810 PRINT"    ";F"TESTS COMPLETED ON";D1
820 PRINT"    DEVICE OK"
830 FORI=1TO5000:NEXT
840 GOTO20
870 IFD=0THEN280
900 PRINT"     **************************"
910 PRINT"    ";D1;"FAILS ON TEST   ";F
920 GOTO20
950 PRINT:PRINT"    ********************"
960 PRINT"    DEVICE NOT RECOGNISED"
970 GOTO20
5000 DATA7400,9,54,0,36,9,18,9,0,9
5010 DATA7402,36,0,36,18,0,9,0,27,0
5020 DATA7404,21,42,0,0,21
5030 DATA7408,9,54,9,36,0,18,0,0,0
5040 DATA7420,1,62,0,60,1,58,1,46,1,30,1,0,1
5060 DATA7432,9,54,9,36,9,18,9,0,0
5070 DATA74125,9,54,9,36,9,18,9,0,0
10000 DATA-1
```

Table 3.7. I.c. tester program

As the program stands it will test the i.c.s 7400, 02, 04, 08, 20, 32 and 125; though since data for each device is handled in a single data line, it is a relatively easy matter to add data for further similar devices. When using the i.c. identification routine, it should be remembered that a number of devices in the 7400 series are logically similar (such as the 7404, 5, 6 and 16 for example). In such cases the program will simply print out the type number of the first device that it comes across whose data correctly matches the i.c. under test.

In order to facilitate additions to the program enabling it to test further devices, we will examine the derivation of the data used for testing the 7400. All the relevant data for this is stored in line 5000 of the program:

5000 DATA 7400, 9, 54, 0, 36, 9, 18, 9, 0, 9

The first number after the device code is used for setting up the PIA so as to input data from those pins of the i.c. which carry output, and to output data to those which carry input. The program is also so arranged that one only has to consider the LH side of the i.c. in setting up the data—ie pins 1–6: the RH side is automatically catered for providing that the i.c has the required symmetry.

Fig. 3.10 gives a pinout of the 7400, with the port connections made by the test socket. From this it may be seen that pins 1, 2, 4 and 5 are inputs, and 3 and 6 are outputs. From the way in which the PIA ports are configured, ones are used to denote PIA outputs, and zeros input. The code used for configuring the ports in this program follows this, so that for the 7400 this is 9 (ie (0×32) + (0×16) + (1×8) + (0×4) + (0×2) + (1×1)).

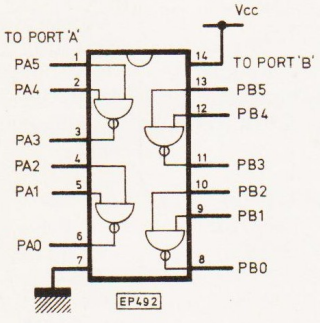


Fig. 3.10. 7400 pin-out for tester

Table 3.8 Truth Table for 7400 IC Tests

PIA Bit	=	PA5	PA4	PA3	PA2	PA1	PAØ	Code
Decimal Bit Value	=	32	16	8	4	2	1	
First Test	input	1	1	X	1	1	X	54
	output	X	X	Ø	X	X	Ø	Ø
Second Test	input	1	Ø	X	1	Ø	X	36
	output	X	X	1	X	X	1	9
Third Test	input	Ø	1	X	Ø	1	X	18
	output	X	X	1	X	X	1	9
Fourth Test	input	Ø	Ø	X	Ø	Ø	X	Ø
	output	X	X	1	X	X	1	9
Key:	1 = High Ø = Low X = Ignored							

The numbers which follow the port code in the data line are arranged in pairs; two for each test. The first of each specifies the test parameters, and the second the required result. Four sets of tests are used on the 7400 (see Table 3.8). The first test takes each input high, and looks for a low output. Since ones are used to denote high PIA output on any given bit, the code for the first test is 54 (ie (1×32) + (1×16) + (1×4) + (1×2)). The result of the test should be zeros on bits 8 and 1. The code for this is (0×8) + (0×1), or zero. Had the required result been high outputs from the i.c. on these two pins, as it is in the remaining tests, the result code would have been (1×8) + (1×1), or 9.

The remaining test codes are constructed in a similar way, as may be seen from Table 3.8, and the program stops testing a device when, as it looks for the next test code it confronts a new device number. When performing the whole repertoire of tests, or when searching for a particular device number, it uses the −1 in line 10000 as an end indicator.

USING THE PIA TO CONTROL SPEECH OUTPUT

In the *December 1980* issue of *P.E.* Dr Berk described a Speech Synthesis Unit which can be used to provide a microcomputer system with a vocabulary of 24 or 64 spoken words. This unit cannot be easily interfaced to Compukit because Compukit possesses no user port; and even when the unit is interfaced via the 2114 memory sockets, word timing problems are encountered because such an arrangement provides no way of monitoring the Busy signal from the Speech Unit.

The PIA on the Decoding Module provides a simple way of fully interfacing the Speech Synthesis Unit to Compukit, and at the same time gives us an opportunity to examine the use of the peripheral control facilities provided by the 6821. These will be used in the present instance to monitor the state of the Synthesis Unit, and inform Compukit when the unit is busy outputting a word, and when it is ready for the next. But first we will look at the simpler question of how to interface the Speech Unit to the PIA without monitoring the Busy signal.

PIA INTERFACE

The Speech Unit requires six parallel data lines to specify the word to be output. These may be connected to the lowest six bits of either port of the PIA. Apart from the Busy line, which we shall ignore for the moment, there are two other lines to consider: Latch and Start. The former requires a positive-going edge to cause the 74174 data latches on the Speech Interface Board to capture data on its bus. The requirement for the Start line is that it be taken low to trigger the output of a word, and left in that state until the word is completed.

These requirements may be met using the remaining two bits of the chosen port of the PIA. Bit 6 of the port should be connected directly to the Latch Enable, and bit 7 to the Start line. Software can then be used to control their status.

To output the word "four" for example, the following commands would be executed after initialising the PIA for output on all 8 bits of a port:

POKE A, 128+ 4
POKE A, 64 + 4

A is 61340 for port A of the PIA (or 61342 for port B). The first command takes the Start line high, the Latch low, and places the number 4 on the bottom 6 data lines. The second command triggers the latch by taking bit 6 high, and initiates speech output by taking bit 7 low, while maintaining the data on the lowest 6 lines. Technically the Start signal should come marginally later than the Latch, but the above

arrangement appears to work well at both ports of the PIA.

The program below will output the full 24 word vocabulary of the Speech Unit using the techniques described above:

```
100 A=61340
110 POKE A+1,0
120 POKE A,255
130 POKE A+1,255
140 FOR B=0 TO 23
150 POKE A,128+B
160 POKE A,64+B
170 FOR C=1 TO 1500: NEXT
180 NEXT
```

As Dr Berk has suggested, if the Busy line cannot be monitored, a timing loop, such as that given in program line 170, must be used to allow one word to finish before the next begins. One difficulty with this approach is that since the words are of differing length, spaces between them will also be variable. In the present program there are considerable pauses after the 10 digits in order that much longer expressions such as "times minus" may be spoken without interruption. This problem may be overcome by using one of the PIA's peripheral control lines.

PERIPHERAL CONTROL WITH THE PIA

Each port of the 6821 PIA has two so-called peripheral control lines: CA1 and CA2 on port A, and CB1 and CB2 on port B. These have been taken out to pins 2 and 3 of SK3 and 4 respectively, of the Decoding Module.

CA1 and CB1 are input only lines, and may be programmed to set a flag, and optionally cause an interrupt when transitions occur on them.

CA2 and CB2 may be programmed as either inputs or outputs. In the discussions which follow we shall be using the CA1 (or CB1) line, and if the reader requires data on the slightly more complex CA2 and CB2 lines he is referred to the Motorola data sheet on the 6821.

The key to understanding the 6821's peripheral control facilities lies in its two control registers CRA and CRB, mentioned briefly last month. One of these registers is dedicated to each port, and both have a similar structure. Table 3.9 gives their format. As may be seen, bits 1 and 0 determine the mode of operation of control lines CA1 and CB1, whilst bit 7 is the flag which monitors the status of the relevant control line.

Table 3.10 gives the four possible states of the lowest two bits of the control register, and their effect on CA1 (or CB1). It will be seen that bit 1 determines whether the flag is set on a high or a low transition of the control line, while bit 0 determines whether the interrupt line $\overline{IRQ}$ (connected directly to Compukit's $\overline{IRQ}$ pin via the Decoding Module) will be activated (taken low) when the flag is set or not. In the discussions which follow we shall not be employing the interrupt facility (which requires handling-routines in 6502 code). The use of interrupts will be treated in a later issue in connection with the 6522 VIA.

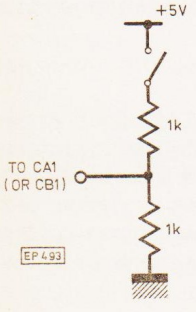


Fig. 3.11. Resistor chain to test flag

Table 3.9 Structure of PIA Peripheral Control Registers

Bit	7	6	5 4 3	2	1 Ø
Control Register A (CRA) at 61341	$\overline{IRQ}$ (CA1)	$\overline{IRQ}$ (CA2)	CA2 Control	DDRA Access	CA1 Control
Control Register B (CRB) at 61343	$\overline{IRQ}$ (CB1)	$\overline{IRQ}$ (CB2)	CB2 Control	DDRB Access	CA2 Control

Note: DDRA — The Data Direction and Peripheral Register Access through bit 2 of the control Register was treated last month.

Table 3.10 CONTROL OF INTERRUPT INPUTS CA1 AND CB1

CRA–1 (CRB–1)	CRA–Ø (CRB–Ø)	Interrupt input CA1 (CB1)	Interrupt Flag CRA–7 (CRB–7)	MPU Interrupt Request $\overline{IRQA}$ ($\overline{IRQB}$)
Ø	Ø	↓ Active	Set high on ↓ of CA1 (CB1)	Disabled— $\overline{IRQ}$ remains high
Ø	1	↓ Active	Set high on ↓ of CA1 (CB1)	Goes low when the interrupt flag bit CRA–7 (CRB–7) goes high
1	Ø	↑ Active	Set high on ↑ of CA1 (CB1)	Disabled— $\overline{IRQ}$ remains high
1	1	↑ Active	Set high on ↑ of CA1 (CB1)	Goes low when the interrupt flag bit CRA–7 (CRB–7) goes high

Notes: 1) ↑ indicates positive transition (low to high)
2) ↓ indicates negative transition (high to low)
3) The interrupt flag bit CRA–7 is cleared by an MPU Read of the A Data Register, and CRB–7 is cleared by an MPU Read of the B Data Register.
4) Of CRA–Ø (CRB–Ø) is low when an interrupt occurs (interrupt disabled) and is later brought high, $\overline{IRQA}$ ($\overline{IRQB}$) occurs after CRA–Ø (CRB–Ø) is written to a "one".

TESTING THE PERIPHERAL CONTROL FLAG

We can now proceed to test the operation of the control line CA1. To do this, first connect the resistor network of Fig. 3.11 to pin 2 of SK3 of the Decoding Module. Then reset the PIA using the Reset push button on the Decoding Module, and run the following program:

```
100 A=61340
120 POKE A+1,0
130 POKE A,255
140 POKE A+1,254
200 PRINT (PEEK (A+1) AND 128)
220 FOR Z=1 TO 100: NEXT
300 GOTO 200
```

This initialises port A for output as usual, but in line 140, it places 254 into control register A. It may be recalled from last month (see Table 2.1) that the control registers for ports A and B may be directly accessed, and are located at 61341 and 61343 respectively. With 254 in CRA, bit 1 of the register will be high, and bit 0, zero. This sets up the conditions shown in the third row of Table 3.10: ie the interrupt is disabled, and the flag will be set high on a positive going transition of CA1.

The program then monitors the contents of bit 7 of the control register. It should print out a string of zeros, indicating that the flag is not set. Closing the switch in the circuit of Fig. 3.11 will set the flag, and cause the program to print out a series of 128s.

The flag will remain set (and the 128s will continue to register) even when the switch is reopened, in order that the CPU can monitor the flag when it wishes without a risk of missing the control line signal. To clear the flag, one must read (or write to) the associated data register of the PIA. Thus if we insert the line:

210 X=PEEK (A)

into the above program, the screen will register only one 128 after any switch closure, subsequently registering zeros, and indicating that the flag has been reset.

PERIPHERAL CONTROL OF SPEECH OUTPUT

We are now in a position to use control line CA1 (or CB1 which functions similarly) with the Speech Board to inform Compukit when speech output is complete. The Speech Board Busy signal goes low when speech begins, returning high when a word is completed. The Busy line may thus be connected directly to CA1, and the data 1 placed in bit 1 of CRA, and 0 in bit 0. This will cause the flag at bit 7 to go high at the end of each word output, and the program can simply inspect this bit, and enter a waiting loop until it returns high, before outputting the next word. The full connections for the interface are given in Fig. 3.12.

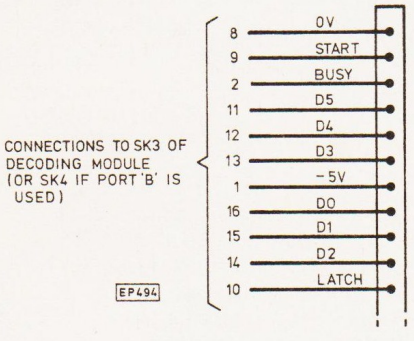


Fig. 3.12. Connections of speech board to SK3

```
70 REM INTERFACING UK101 PROGRAM 8
80 REM SPEECH ROUTINES
100 A=61340
120 POKEA+1,0
130 POKEA,255
140 POKEA+1,254
200 FORA1=1TO16:PRINT:NEXT
210 PRINT,"1. VOCABULARY"
220 PRINT,"2. COUNTING"
230 PRINT,"3. READOUT"
240 PRINT,"4. DICE"
250 PRINT:PRINT:PRINT
260 INPUTAB
270 ONABGOTO300,450,600,750
280 GOTO200
300 PRINT:PRINT,"VOCABULARY"
320 FORWW=0TO23
330 GOSUB2000
340 NEXT
350 GOTO200
450 PRINT:PRINT
460 INPUT"   COUNT TOTAL";AD
470 FORW=0TOAD
480 GOSUB1000
490 FORAE=1TO1000:NEXT
500 NEXT
510 GOTO200
600 PRINT:PRINT:PRINT
610 PRINT,"INPUT NUMBER STRING"
620 INPUTW$
625 W$="0"+W$
630 GOSUB1070
640 GOTO200
750 REM DICE
760 PRINT:PRINT:PRINT
770 PRINT,"DICE ROUTINE"
780 PRINT,"ENTER RANDOM NUMBER"
790 INPUTA6
800 A7=RND(-A6)
810 PRINT:PRINT"   PRESS RETURN FOR EACH THROW"
820 PRINT,"   PRESS ↑ TO EXIT"
830 POKE530,1
840 POKE57088,223
850 A8=PEEK(57088)
860 IFA8=239THENPOKE530,0:GOTO200
870 IFA8<>247THEN840
880 WW=INT(1+6*RND(8))
890 GOSUB2000
900 GOTO840
910 REM
920 REM
1000 REM
1050 W$=STR$(W)
1060 PRINTW$
1070 FORW1=2TOLEN(W$)
1080 WW$=MID$(W$,W1,1)
1085 WW=VAL(WW$)
1090 GOSUB2000
1100 NEXT
1200 RETURN
1900 REM
1910 REM
1920 REM
2000 REM SPEECH POKE
2050 POKEA,128+WW
2060 POKEA,64+WW
2070 A1=PEEK(A+1)
2090 A3=A1AND128
2100 IFA3=0THEN2070
2110 FORA4=1TO100:NEXT
2120 RETURN
OK
```

Table 3.11. Speech handling program

SPEECH PROGRAM

These principles have been put into practice in the program listed in Table 3.11. It performs four different functions according to a menu printed out at the start. "1" gives the full 24 word vocabulary of the board. "2" counts to any predetermined number. "3" speaks a string of figures entered via the keyboard; and "4" initiates a dice routine in which a random number from 1 to 6 is spoken at each press of the Return key.

Incidentally, this last routine uses an apparently unpublished facility of the RND function on Compukit. Calling RND(X) will give a random number *independent* of the value of X for all positive integers. But if X is negative then this has the effect of "seeding" the random number generator to a position *dependent* on the value of X, and can thus be used to start off the generator at a new point.

All four of the routines described make use of one or more of the digit and speech handling routines at lines 1000 and 2000. The first splits up a variable W into a sequence of digits WW (or if entered at 1070 does the same for a string of characters W$). These are then output sequentially by the speech handling routine at line 2000, which simply "speaks" the digit WW, waiting until speech output is complete before a Return is executed either to the first subroutine or to the main body of the program. This is achieved in lines 2070–2100 by examining bit 7 of the PIA control register for a 1.

These two routines should be found useful in other applications of the Speech Board.

NEXT MONTH we will introduce an Analogue Board which plugs directly into the Decoding Module to provide Compukit with an AY–3–8910 Programmable Sound Generator, a D/A converter, an 8 channel A/D converter, and a 6522 Versatile Interface Adaptor, allowing counting and timing operations, as well as providing a second 16 bit input/output port.

Interfacing COMPUKIT

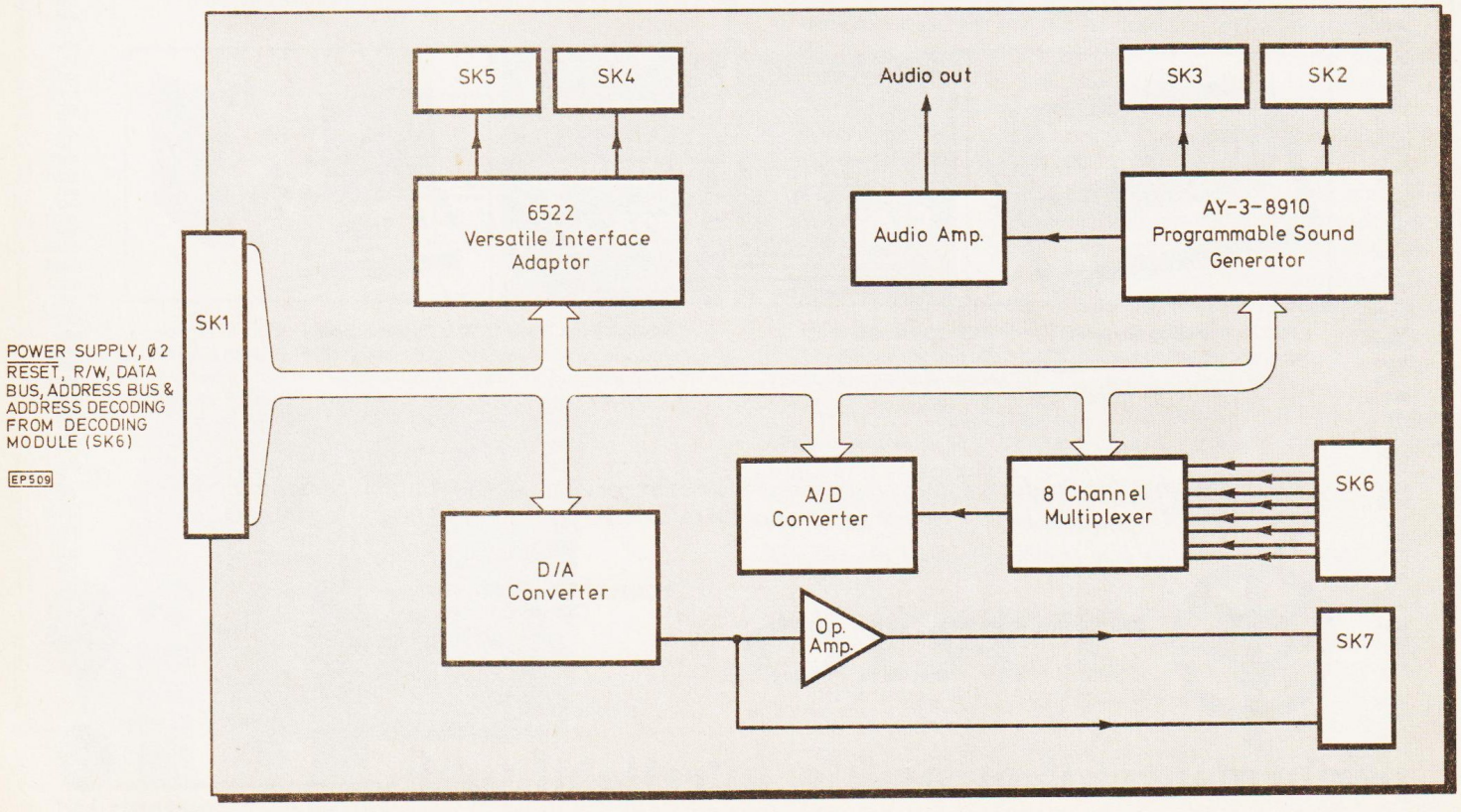
Part 4 D.E.Graham

THIS MONTH we introduce a second companion board to the Decoding Module, and examine the implementation of analogue output from the Compukit.

ANALOGUE, TIMING AND AUDIO BOARD

This is a double sided p.c.b. of the same dimensions as the Decoding Module, which connects directly to it via a single edge connector to provide the Compukit with a range of facilities. The board is powered by the Decoding Module's dual 5 volt power supply, and, as may be seen from Fig. 4.1, contains four separate sections: A D/A converter and operational amplifier taken out to SK7; an 8 channel A/D converter accessed through SK6; an AY-3-8910 Programmable Sound Generator and audio amplifier whose output is taken to a number of pads at the edge of the board, and whose two 8 bit ports are accessed through SK2 and 3; and a 6522 Versatile Interface Adaptor providing a number of counting and timing facilities, as well as a further 16 bits of parallel port. Connections to the 6522 are made via SK4 and 5. Sockets SK2, 3, 4, 5, 6 and 7 are all of the 16-pin d.i.l. variety.

4.1. Block diagram of Analogue Board



CONSTRUCTION

This should prove to be fairly straightforward. It is probably easiest to solder in i.c. sockets first, followed by discrete components, and finally the through-pins (as indicated on the component overlay in Fig. 4.4). Before inserting the i.c.s, test that the correct supply voltage appears at the appropriate pins of all i.c. sockets. The Analogue Board connects to the Decoding Module via a 2 × 25-pin 0·1 inch edge connector SK1. This is wired to SK6 on the Decoding Module as shown in Table 4.1. This wiring should be kept to a few inches in length. Precautions against static damage must be exercised when dealing with i.c.s 1, 2, 6, 8 and 10 since these devices may be easily damaged by static charges.

We will cover the testing of the four parts of the board in the particular sections dealing with each functional unit. Details of the PSG 6522 and A/D converter will be given in forthcoming issues. Now we deal with D/A conversion.

D/A TECHNIQUES

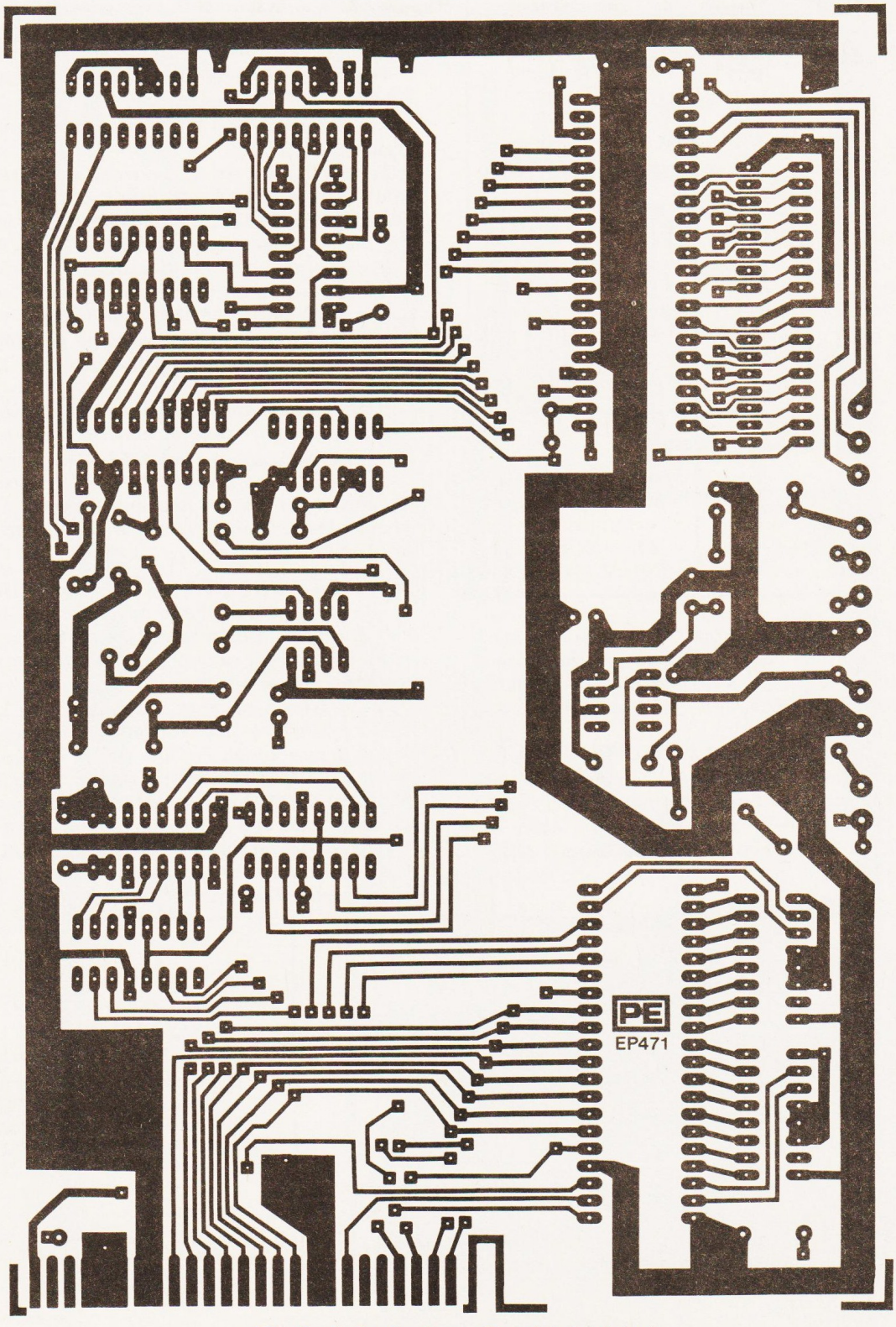
In its simplest form a D/A converter may consist of a chain of resistors joined to a parallel output port. Fig. 4.5 shows a 4 bit D/A converter that could be connected directly to the

outputs of a 7475 quad latch. The output voltage would range from a fraction of a volt for zero data to about 4 volts for the decimal value 15. The resistors would need to be one per cent tolerance types to avoid abrupt changes in voltage occurring when different sections of the chain are brought into play, as in the major transition which occurs from 7 to 8 for example.

The configuration could be doubled up to produce an 8 bit converter, but resistor tolerances would become more critical. Also, if a PIA port was to be used with such a converter, higher value resistors would be required because of the relatively low drive capability of its output. This would further necessitate the use of a d.c. amplifier to produce a usable analogue output.

ZN425 MONOLITHIC CONVERTER

It is of course possible to get around these problems, and particularly the problem of conversion accuracy, by using a



4.2. P.c.b. for Analogue Board (actual size)

Table 4.1 Connections Between SK1 of Analogue board and SK6 of Decoding Module.

SK1 pin number	Upper		Lower	
	SK6 pin number (upper)	Function	SK6 pin number (lower)	Function
1	1	Vgg(−5V)	1	RESET
2	2	Ø2	2	W7
3	3	1RQ	3	W8
4	4	BC1	4	R7
5	5	BDIR	—	NC
6	—	NC	—	NC
7	—	NC	7	R/W
8	—	NC	8	GND
9	—	NC	9	GND
10	—	NC	10	D7
11	11	W7	11	D6
12	—	NC	12	D5
13	13	A3	13	D4
14	14	A2	14	D0
15	15	A1	15	D1
16	16	A0	16	D2
17	17	GND	17	D3
18	18	GND	18	Vcc
19	19	GND	19	Vcc
20	—	NC	20	GND
21	—	NC	21	GND
22	—	NC	22	GND
23	—	NC	—	NC
24	—	NC	—	NC
25	25	NMI	25	BL2

monolithic D/A converter i.c. From the variety of such devices on the market we have chosen to use the Ferranti ZN425 for a number of reasons. In particular it is readily available at a reasonable price, and operates from a 5 volt supply.

Fig. 4.6 gives a block diagram of the sections of the 425 used in D/A conversion. Essentially it consists of 8 data switches which are activated by an external port or latches. These switch a precision R-2R network to an on-chip 2·5 volt reference source to produce an analogue output on pin 14. This is typically 2·555 volts for all bits on, and 3mV for all bits off.

PRACTICAL D/A CIRCUIT

Fig. 4.7 gives the full circuit of the D/A section of the Analogue Board. This consists of a pair of 74LS75s wired to form an 8 bit data latch. The latch enables are taken to the W line on the Decoding Module, which corresponds to an address of 61320. The 8 parallel outputs of the latch are connected directly to the ZN425, which performs the conversion of the latched data within 1μ sec. The analogue output (DA) appears at pin 14 of the 425, and is fed to the non-inverting input of IC11, a 741 operational amplifier. Both DA, and the output of the op. amp. (DAA) are taken out to SK7, which also carries both polarity supply connections and ground.

The op. amp. circuit has two associated variable resistors. VR1 is used for zeroing, and has been given an extended offset capability, and VR2, which controls the gain between about 1 and 2.

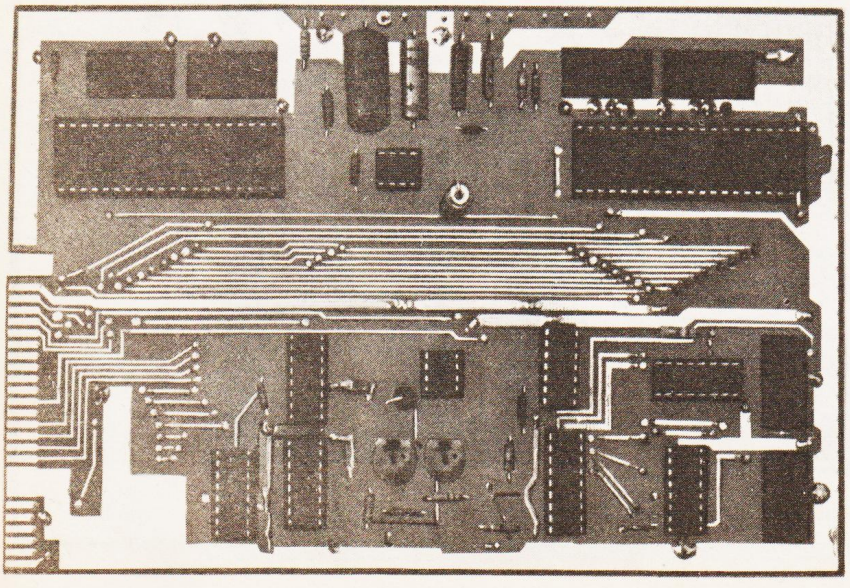
To test the converter, connect a voltmeter between pin 14 of SK7 and earth (pins 1, 5 or 6 of SK7). Execute the command **POKE 61320, 0,** and adjust VR1 to give zero volts on the meter. Now execute **POKE 61320, 255.** This should cause the meter to read somewhere between 2·5 and 4 volts, depending on the setting of VR2. The system is now operational, and POKEing intermediate values to 61320 should yield intermediate voltage readings with a linear correspondence (providing the gain has not been set too high).

If the voltage does not vary with differing data, a voltage check should be made on the DA output of the converter (pin 14 of IC6, or pin 16 of SK7). If this does not alter when data is POKEd to 61320, then checks should be made on the outputs of the two latches IC4 and 5. These should also change when different values are POKEd to 61320.

APPLICATIONS OF THE D/A CONVERTER

The DAA output of the converter unit at pin 14 of SK7 may be used in a wide variety of different applications. It could be used for example to feed a servo amplifier controlling a d.c. motor which could variously drive a graph plotter, a steering mechanism, or a robot's left leg.

More simply it may be used to drive power controllers of one kind or another. For low power d.c. operation, a simple current amplifier of the type shown in Fig. 4·8 may be connected to the DAA output of the converter unit. This will vary the brightness of a 2·5V lamp according to the data POKEd to 61320. To set this up, first execute POKE 61320, 0, and adjust the zero offset (VR1) so that the bulb is just extinguished. Then execute POKE 61320, 255. This should



CONSTRUCTOR'S NOTE: NEW MONITOR IN EPROM

During the development of this series the screen editor written by Nigel Climpson and published in PE was found to be extremely useful. This editor is now available as the CE1 monitor in a 2716 EPROM for £12·50 + VAT and p&p, from **Technomatic Ltd.** It replaces the UK101 2K monitor ROM, and also contains useful routines such as a rapid screen clear.

illuminate the lamp brightly, and VR2 may then be adjusted to achieve best control over the full range of data.

A program of the type listed below will be found useful in setting up the converter for the above, and for other applications:

```
80 REM TEST ROUTINE FOR A/D CONVERTER
100 A=61320
120 INPUTX
140 POKEA,X
160 GOTO100
```

It simply requests a number, which should be an integer between zero and 255, and POKEs this to the converter.

TRIAC CONTROLLER

If a.c. or pulsed d.c. control is required, then the converter may be used to drive a Triac or Thyristor. There are many ways in which this may be achieved, but perhaps the most straightforward is to use the DAA output of the converter to vary the brightness of a l.e.d. indicator, which itself illuminates a light dependent resistor placed at a strategic

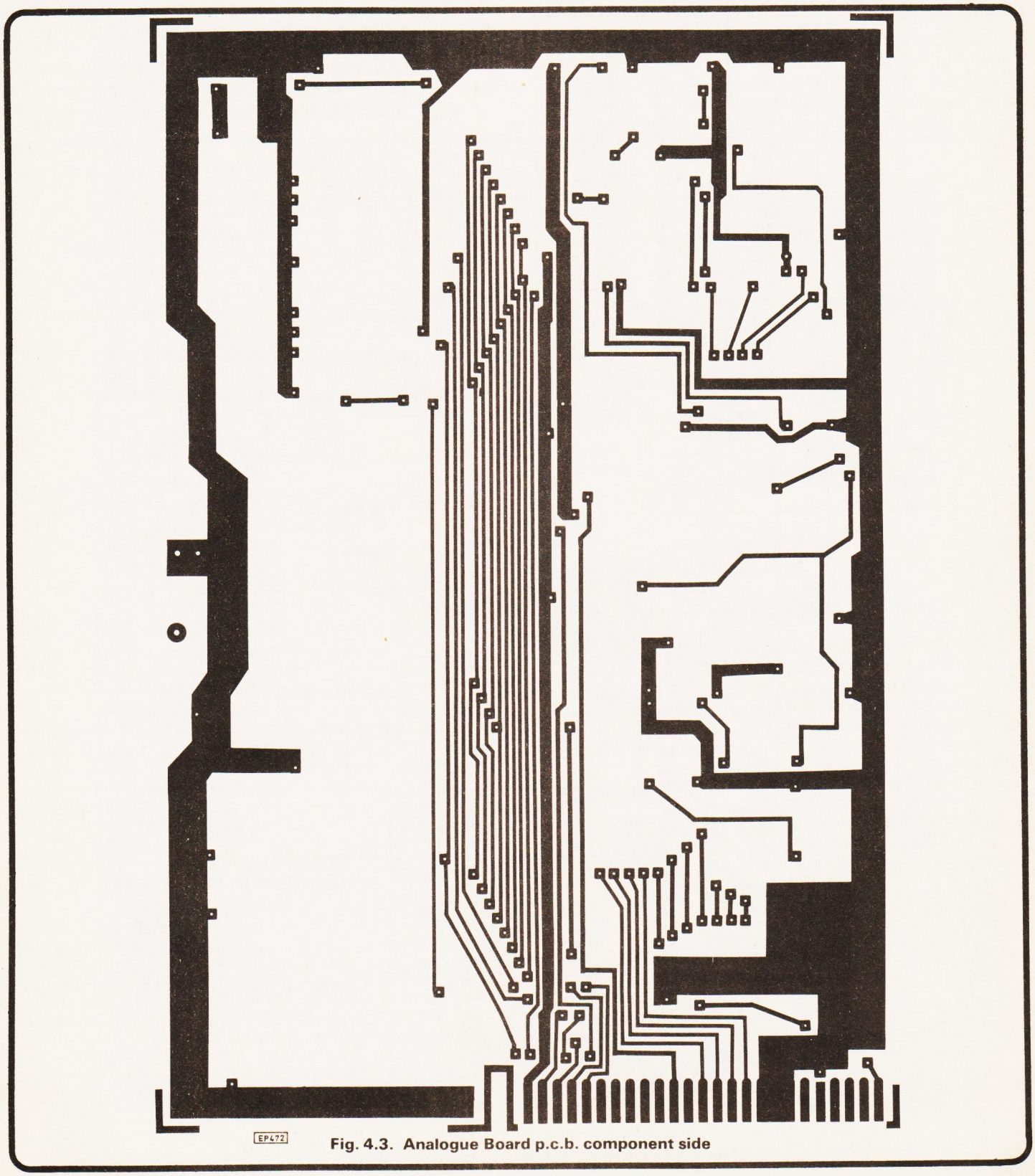


Fig. 4.3. Analogue Board p.c.b. component side

point in a triac or thyristor controller circuit. This has the great advantage of completely isolating the computer system from the mains. Alternatively, a patent opto-isolator such as the TIL112 may be used. In either case the l.e.d. may be directly driven by the DAA output of the converter as in Fig. 4.9.

Fig. 4.10 gives an experimental circuit for a power controller using the l.d.r. method. The phase shift for the triac is produced by the R1/C1 network, with the l.d.r. altering the charge time of C1. R2, R3, and C2 help to reduce hysteresis and flicker, common diseases of this type of controller, though the latter is *not* completely eliminated. L1 and L2 are inductors each formed by winding about 100 turns of wire of a half inch former. Perhaps the most vital part of the circuit is the R4/C3 network. This prevents spikes in the supply line from destroying the triac.

The l.e.d. and series resistor are connected between the DAA output of the converter and Vcc. The l.e.d. should be

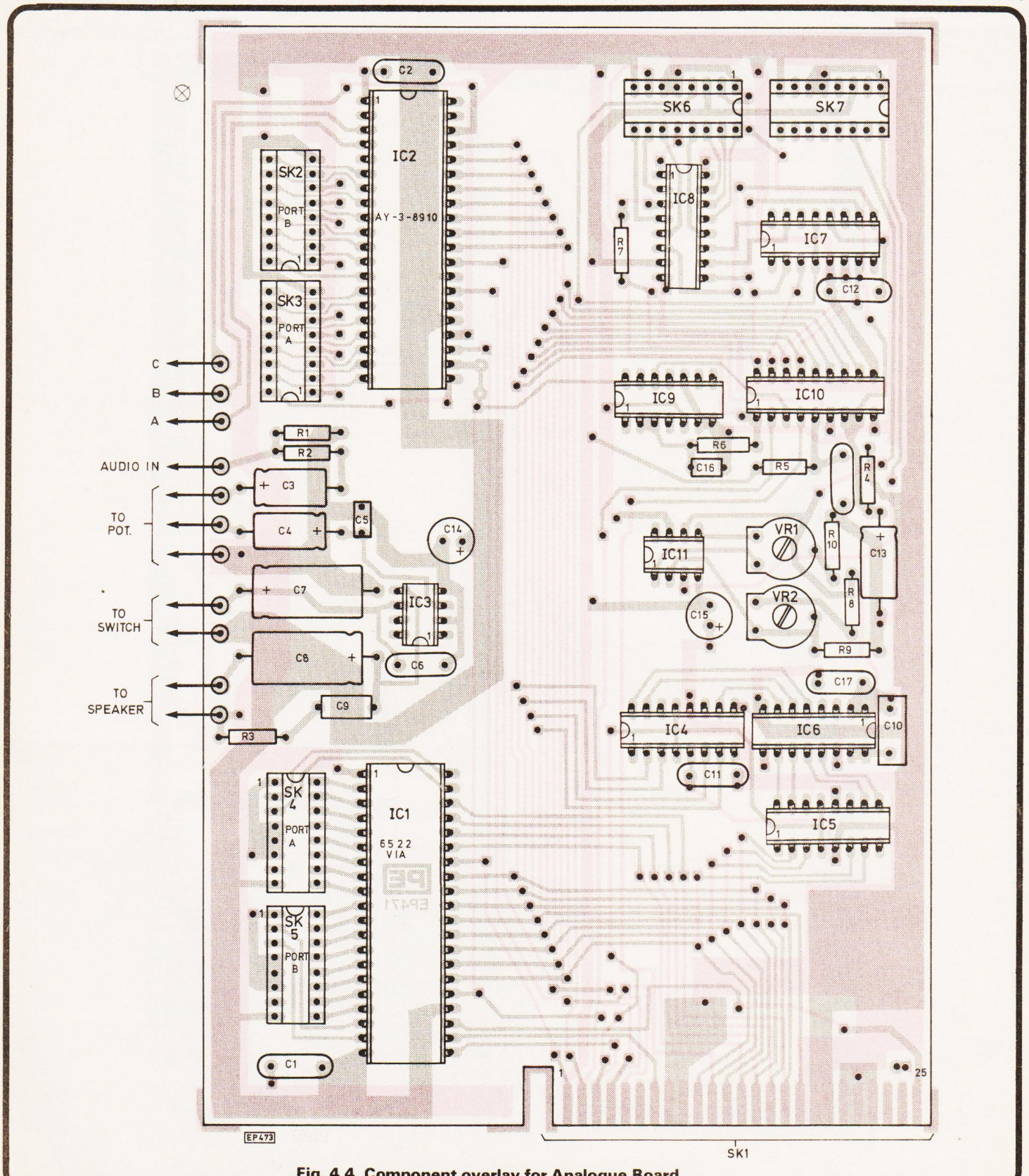


Fig. 4.4. Component overlay for Analogue Board

taped to the l.d.r., and the pair mounted in a *completely* light-tight container.

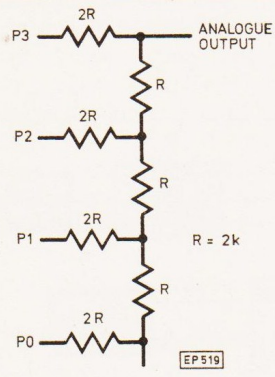
To set up the circuit, VR1 of the converter should be set to give zero volts between DAA and ground on execution of POKE 61320, 0. VR2 should then be adjusted to give a smooth range of control. Some adjustment of R1, 2 and 3 may be necessary to effect this.

THYRISTOR CONTROLLER

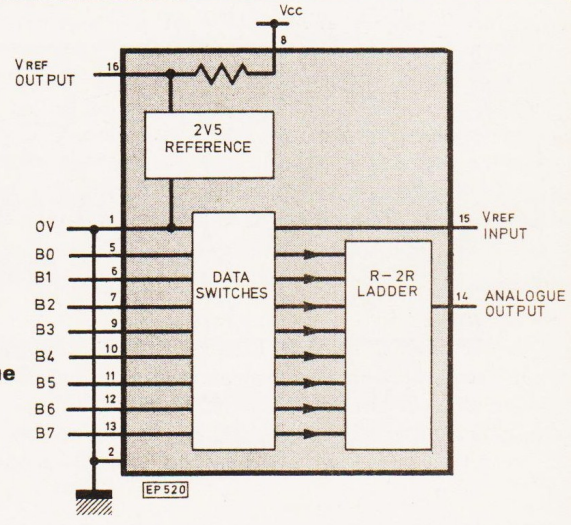
In the author's experience, far more satisfactory power control is achieved using thyristors rather than triacs. One advantage of the thyristor is the ease with which unijunction transistor delay circuits may be used with them; and secondly they cannot suffer from asynchronous firing in the two directions of current flow, as may occur with the triac, and which is indeed one of the factors causing flicker in the controller of Fig. 4.10.

Fig. 4.11 gives the circuit of a thyristor controller which may be used to vary the power to some 12 volt d.c. device for currents up to two or three amperes. Control using the 500k resistor is smooth and flicker-free. An l.d.r. driven by an l.e.d. from the D/A converter may be introduced in a number of ways into this circuit. About the simplest is to take the l.d.r. from point X to earth via a resistor in the range 20 to 100k. To obtain smooth control it will be necessary to adjust the 500k pot in conjunction with VR1 and VR2 on the Analogue Board. Again, however, it should be stressed that this is an experimental circuit, and some adjustment of values may be necessary to obtain the best performance.

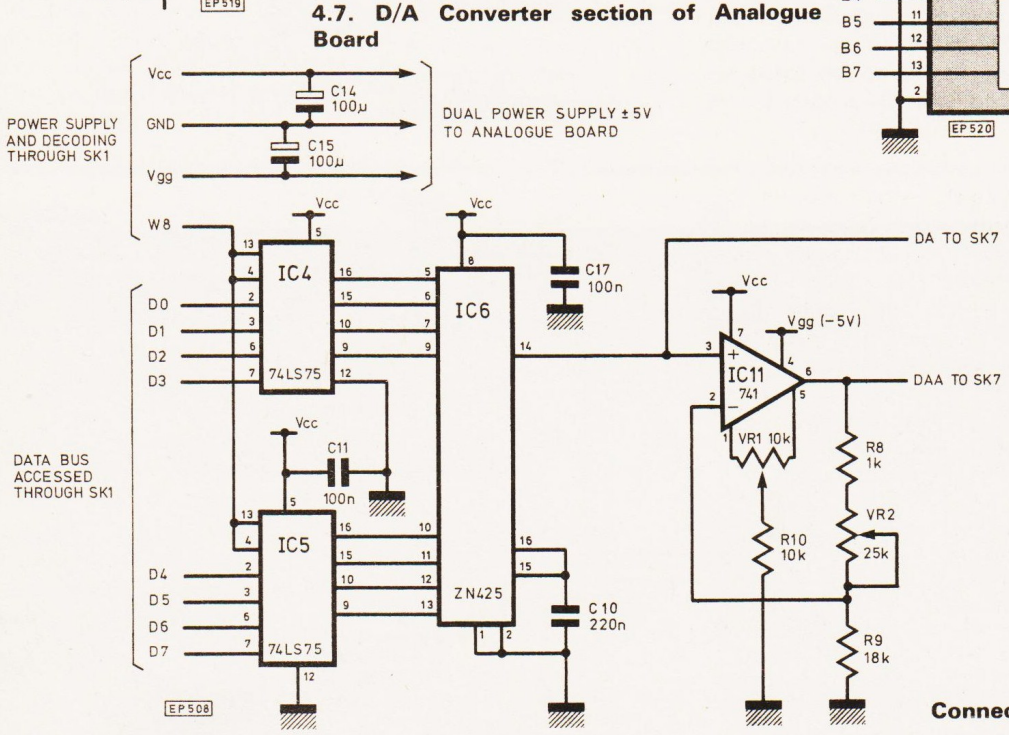
If it is desired to use this circuit for power control at a higher voltage, then it should be possible to increase the supply voltage, and adjust the Zener diode dropper resistor R1 accordingly. If a.c. control is required, then the load



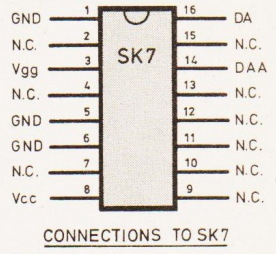
4.5. R/2R D/A converter



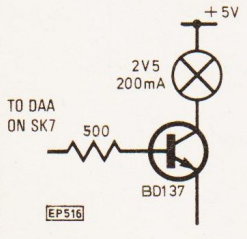
4.6. ZN425 block diagram



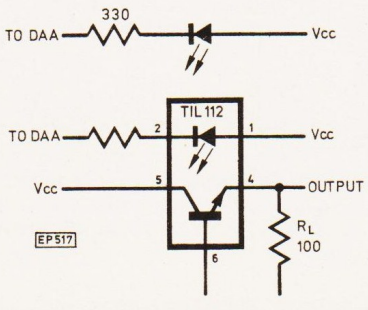
4.7. D/A Converter section of Analogue Board



Connection of l.e.d. to D/A converter

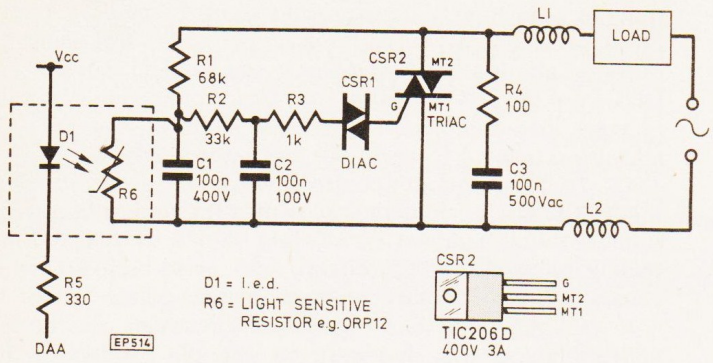


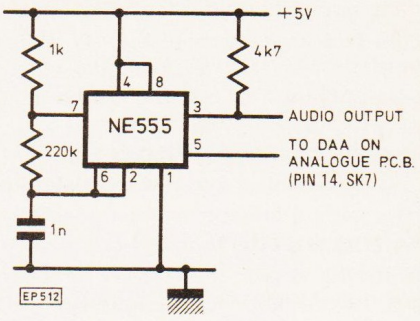
4.8. BD137 Current Amplifier



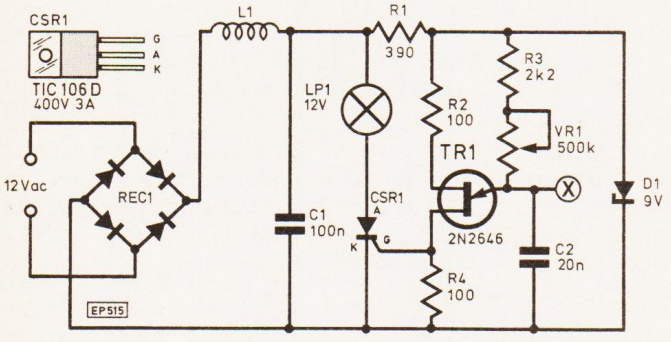
4.9. L.e.d. and Resistor (to opto-isolator)

4.10. Triac Power Controller. All resistors are 1 Watt

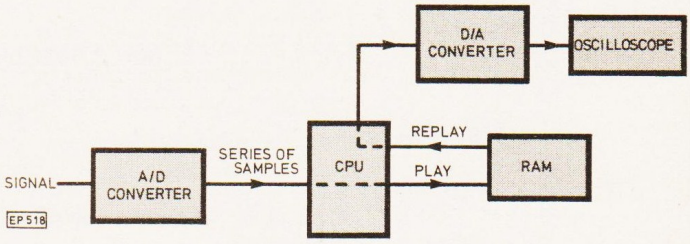




4.12. NE555 signal generator



4.11. Thyristor Power Controller



4.13. Block diagram of storage oscillocope

COMPONENTS . . .

Resistors

R1, R6, R8	1k (3 off)
R2	100k
R3	10
R4	390
R5	82k
R7	3k9
R9	18k
R10	10k

Potentiometers

VR1	10k preset
VR2	25k preset
VR3	100k log + switch

Capacitors

C1, C2, C6, C11, C12, C17	100n disc ceramic (6 off)
C3, C4, C13	10µ/10V (3 off)
C5	1n
C7, C14, C15	100µ/10V (3 off)
C8	200µ/10V
C9	47n mylar
C10	220n mylar
C16	50n mylar

Integrated Circuits

IC1	6522
IC2	AY-3-8910
IC3	LM386
IC4, IC5, IC7	74LS75 (3 off)
IC6	ZN425
IC8	4051
IC9	74LS90
IC10	ZN427
IC11	741

Miscellaneous

P.c.b.
SK1 2 x 25 0·1in. edge connector
SK2–SK7 16-pin d.i.l. sockets (6 off)
40-pin d.i.l. sockets (2 off)
16-pin d.i.l. sockets(5 off)
8-pin d.i.l. sockets (2 off)
14-pin d.i.l. sockets
14-pin d.i.l. sockets
length of 40 strand ribbon cable

Constructors' Note

A complete kit of parts, excluding loudspeaker, is obtainable from **Technomatic Ltd., 17 Burnley Road, London NW10**

should be placed in series with the a.c. supply feeding the bridge rectifier. Additionally the reader is referred to the many power control circuits that have appeared in *P.E.* in the past, and to the useful book on the subject by *D. Marsden*, entitled *110 Thyristor Projects.* The use of one of these with a controlling l.d.r. or opto-isolator should meet most individual requirements; though it should be noted that the recently published circuit for the *Slave Light Dimmer (P.E. Feb. 1981)* is not suitable for this purpose.

AUDIO OUTPUT

For some purposes it may be found useful to run an audio generator from the Analogue Board D/A converter, or from a R-2R converter running from an unused port, and buffered with an operational amplifier similar to that used on the Analogue Board. In either case the DAA output (or similar) may be used directly with i.c.s such as the NE566 function generator or the NE555 timer. Fig 4.12 gives a circuit for audio production using the 555. The DAA line from pin 14 of SK7 is used to directly drive the control pin (pin 5) of the 555. With the components specified this will give outputs in the range 5 to 10kHz. VR1 should be set to null output for zero data, and VR2 to maximum gain. This will result in outputs of about 10kHz for 255, and about 5kHz for data of around 80. If zero is POKEd to the converter, the generator ceases to oscillate, so providing a convenient means of switching off audio output.

FURTHER APPLICATIONS

The D/A converter on the Analogue Board may also be used for directly handling audio and other waveforms. It can, for example, be used in the direct generation of virtually any conceivable waveform. The program below produces a staircase output at the DA and DAA pins of the converter:

```
100 A=61320
110 INPUT "SAMPLE RATE: TRY 5"; C
120 FOR B=1 TO 255 STEP C
130 POKE A, B
140 NEXT
150 GOTO 120
```

Using BASIC for this purpose limits the output frequency of any waveform generated to a few Hz or so. For higher frequency outputs, the program would have to be executed in 6502 code. It would be a relatively simple matter to write a short routine in 6502 code that successively output the contents of a block of memory to the D/A converter. The block could then be filled beforehand, using a POKE routine in BASIC, with any desired waveform, e.g. sin, square, triangular step, etc. The short 6502 code program could then be accessed via the USR(X) call to output the data at any given speed.

Using similar techniques in conjunction with an A/D converter it would be possible to write software for a storage facility for an oscilloscope. The A/D converter would sample a given waveform, and store the data in a given block of RAM. The D/A converter could then be used to output the sequence repeatedly, and at any frequency and repetition rate, so as to provide a permanent display, with the option of recall facilities, etc. See Fig. 4.13.

Next month we will look at the use of the PSG on the Analogue Board, and discuss applications such as a 14-note organ operated from the UK101 keyboard. Details will also be given on the use of the Programmable Sound Generator as a 3-channel D to A converter

Interfacing COMPUKIT

Part 5 D. E. Graham

THIS month we will be looking at the Programmable Sound Generator on the Analogue Board. A separate page is also included which elaborates on the use of SK4 and SK5 of the Decoding Module.

THE AY-3-8910 PROGRAMMABLE SOUND GENERATOR

Fig. 5.1 gives the circuit of the audio section of the Analogue Board. This consists of a General Instrument AY-3-8910 Programmable Sound Generator feeding an LM386 audio amplifier which produces 200mW into an 8 Ohm load. The audio circuitry is based on a design from G.I.'s data manual.

The 8910 is a sixteen register device containing three independent tone generators, a variable pitch white noise generator, mixers and an envelope controller. Details of the operation of the very similar 8912 were given in the *September 1980* issue of *P.E.*, and for this reason a full device description will not be given here. For additional information on the device the reader is referred both to that article, to G.I.'s data manual on the chip, and to the article *Micro Sounds* in *Personal Computer World, October 1980.*

Although the 8910 is a sixteen register device, it is intended for CPUs which use a shared address and data bus, and for this reason cannot be used directly with 6502 based machines. This difficulty may, however, be circumvented by using the data bus to carry data which the 8910 can use either as true data or as an address, depending on the state of its control lines (which are designed to inform it of such things). One has then simply to set up the correct configuration on its control lines before furnishing it with appropriate data on its shared bus.

This task is performed by IC8c and IC8d on the Decoding Module. The required configuration of the two control lines (coded BDIR and BC1) is given in Table 5.1. From this it may be seen that to select any one of the P.S.G.'s 16 internal registers, both BDIR and BC1 must go high, and at the same time the number of the register required must be placed on the data bus by the CPU for the P.S.G. to read. Once the operation is completed, both BDIR and BC1 must be taken low. If it is then required to write data into the particular register just called up, BDIR must be taken high. Any data on the data bus will then be written into the given P.S.G. register.

The two signals BDIR and BC1 are produced by the Decoding Module in response to the R6, W6 and W7 lines, as may be seen from the circuit of Fig 5.2. This is organised so that reading and writing to the P.S.G. is performed in the following way. To place the value 100 (decimal) into P.S.G. register 4, one simply executes:

POKE 61317, 4
POKE 61318, 100

The first command calls up register 4. The second places the data into it. To read the contents of register 7, execute:

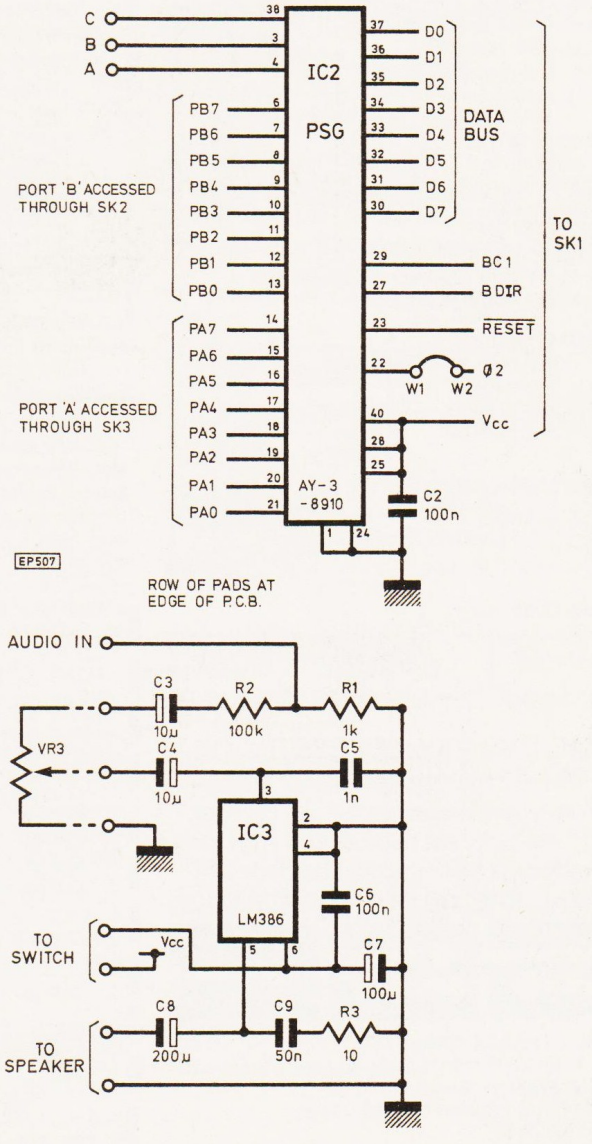
POKE 61317, 7
PRINT PEEK (61318)

This calls up register 7, then reads from it.

TESTING THE PSG

When setting up the PSG for the first time, connect the pads as shown in Fig 5.3, and switch on S1. Check that the audio amplifier is working by setting the volume control VR3

Fig. 5.1. PSG section of Analogue Board. W1 and W2 allow alternative clock drive (1–2MHz) if desired.



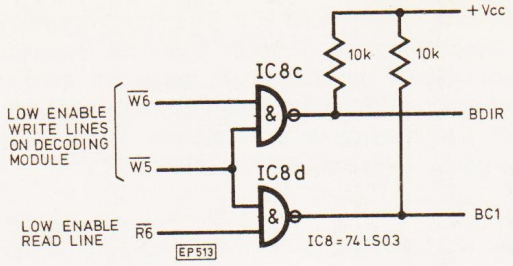


Fig. 5.2. PSG decoding circuitry on Decoding Module Board.

Table 5.1. PSG Control line functions

Function	BDIR	BC1
Inactive	Ø	Ø
CPU read from PSG	Ø	1
CPU write to PSG	1	Ø
Latch address of PSG register	1	1

Fig. 5.3. Connection of PSG pads for normal operation.

Fig. 5.4. Connection of PSG pads for normal operation on channel C, with analogue output on channels A and B.

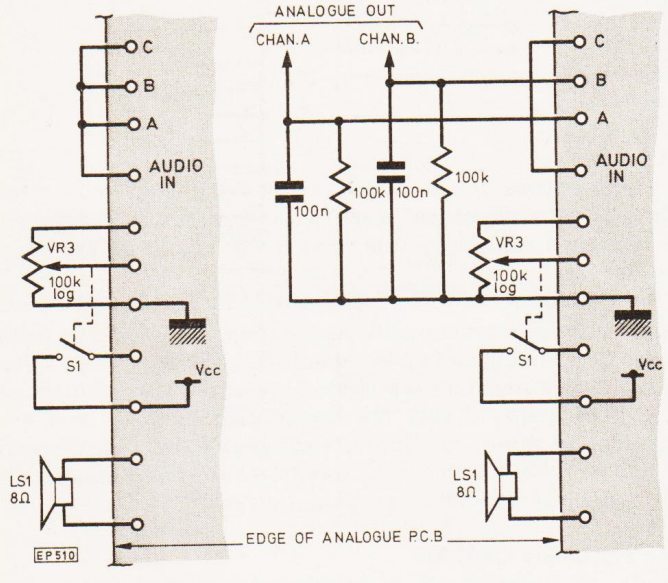


Table 5.2. Manual operation of Programmable Sound Generator.

```
50 REM INTERFACING UK101 PROGRAM 9
80 REM AY-3-8910 MANUAL ENTRY PROGRAM
100 FORS=1TO16:PRINT:NEXT
110 PRINT,"AY-3-8910 POKE/PEEK ROUTINE"
120 PRINT,"USING ADDRESSES 61317 & 61318
124 PRINT,"(NOTE DATA > 255 CAUSES A READ)"
125 PRINT:PRINT
130 A=61317:D=61318
135 REM*********************************
140 REM ZERO ALL REGISTERS
150 FORX=0TO15
160 POKEA,X
170 POKED,0
180 NEXT
190 REM*********************************
200 REM POKE/PEEK ROUTINE
210 INPUT"    REGISTER";A1
220 POKEA,A1
230 INPUT"    DATA";D1
240 IFD1>255THEN300
250 POKED,D1
260 GOTO210
300 PRINT,"CONTENTS OF REG";A1;" :   ";PEEK(D)
310 GOTO210
```

to full volume, and touching the wiper pin. If this produces hum, then run the program shown in Table 5.2. This has been written for manually controlling the contents of the P.S.G.'s registers. It first zeroes all registers, and then requests a register number. When this is entered, it requests the data that is to be put into it (any integer form 0 to 255). If a number greater than 255 is entered, the program will read from the register instead of writing to it, and print out the results.

Run the program and use it to examine the contents of the P.S.G.'s registers: all should be at zero. Data may then be placed in registers 0 to 13, and a check made to verify that the data has been properly stored. Remember here however, that certain registers (1, 3, 5, 6, 8, 9, 10 and 13) are only 4 or 5 bits wide, and attempting to place the value 255 say, into register 1 will cause the value 15 to be stored in that 4-bit register.

If registers 0 to 13 do not read or write correctly, first check the status of all lines to the P.S.G. (supply, Ø2, etc.). Next check the functioning of the control lines BDIR and BC1 with a 74LS75 latch (or similar) in the manner indicated in part 2 of the series. Their states should be checked against those given in Table 5.1. If these are incorrect, then the decoding circuitry should be checked. If they are functioning correctly, then a way should be devised of testing the P.S.G. chip itself. This could be done using the PIA on the Decoding Module, with one port connected to the 8 data/address lines of the P.S.G. and two lines of the other port supplying the BDIR and BC1 signals.

Once the registers read and write correctly, it should be possible to produce some sounds. The fastest way to make noise is to enter 15 into register 8 after resetting the P.S.G. This should produce white noise whose colour may be altered by placing data into the lowest 5 bits of register 6. To stop the noise, place 254 into register 7. If this is followed by placing 100 into register 0, a pure tone should be produced whose note may be varied by changing the data in registers 0 and 1.

If audio is not forthcoming, but the registers read and write correctly, then the audio circuitry should be checked, and pins 4, 3 and 38 of the P.S.G. examined for audio output.

THE P.S.G.'s REGISTERS

In order to produce more subtle sounds, it is necessary to be acquainted with the functions of the P.S.G.'s full set of registers. These are represented diagrammatically in Table 5.3.

Perhaps the most important register is number 7, the master enable. This is organised, somewhat inconveniently, to be active-low, so that placing a zero into it enables all tone and noise channels, 255 silences them all, while 254 enables tone on channel A only, etc. Note frequencies must then be set using the first six registers (two for each channel); and then amplitudes (registers 8, 9 and 10 for channels A, B and C respectively). Data from 0 to 15 in these registers produces differing amplitudes, while data of 16 in any of the three registers puts the associated channel over to envelope control.

Registers 11 and 12 control the time period of the envelope, while 13 controls, its shape, and whether it is repeating or not. For more precise details of this register the reader is referred to one of the three works cited earlier, but in Table 5.4 we give a selection of useful data for this register.

Registers 14 and 15 are two 8 bit ports, whose function is controlled by the top two bits of register 7 (0 for input, 1 for output). The connections to these two ports have been taken

out to the 16 pin d.i.l. sockets SK2 and 3. See Table 5.5 for connections. It will be seen that they have been made similar to those for the PIA on the Decoding Module, although it should be noted that the two sockets face the opposite direction to those on the Decoding Module.

Register	Function	BIT							
		7	6	5	4	3	2	1	0
R0	Channel A tone period	8-bit fine tune A							
R1					4-bit coarse tune A				
R2	Channel B tone period	8-bit fine tune B							
R3					4-bit coarse tune B				
R4	Channel C tone period	8-bit fine tune C							
R5					4-bit coarse tune C				
R6	Noise period				5-bit period control				
R7	$\overline{\text{Enable}}$	$\overline{\text{In/out}}$		$\overline{\text{Noise}}$			$\overline{\text{Tone}}$		
		I/O B	I/O A	C	B	A	C	B	A
R8	Channel A amplitude			Env en	4-bit amplitude				
R9	Channel B amplitude			Env en	4-bit amplitude				
R10	Channel C amplitude			Env en	4-bit amplitude				
R11	Envelope period	8-bit fine tune							
R12		8-bit coarse tune							
R13	Envelope profile				4-bit control				
R14	I/O port A	8-bit parallel port							
R15	I/O port B	8-bit parallel port							

Table 5.3. Programmable Sound Generator registers.

Table 5.4. Selected functions of register 13 of PSG —envelope control

Data	Function
Ø	Non repeating, fast attack, slow decay
4	Non repeating, slow attack, fast decay
8	Repeating, fast attack, slow decay
12	Repeating, slow attack, fast decay
14	Repeating, slow attack, slow decay

Table 5.5. Connections to SK2 and SK3
SK3 = Port A of PSG
SK2 = Port B of PSG

Pin	Function	Pin	Function
1	GND	9	P7
2	NC	10	P6
3	NC	11	P5
4	GND	12	P4
5	GND	13	P3
6	GND	14	P2
7	NC	15	P1
8	VCC	16	P0

MAKING SOUNDS

We now give one or two sequences that may be entered using the program in Table 5.2 to produce some simple sounds. The short sequence below will produce a continuous note:

7—248
8—15
0—100

The number to the left is the register number, that to the right, the data to be placed in it. The first line sets register 7 to give tone output on all three channels, the second sets maximum volume on channel A, while the third selects a note. Entering alternative values into register 0 will change the note, and placing data in register 1 will considerably lower its pitch. If this sequence is followed by:

9—15
2—102

the note will change in timbre, becoming richer as the results of the beating of notes from channels A and B.

It is also possible to place this composite tone under envelope control. The following sequence achieves this:

8—16
9—16
12—30

The first two entries place channels A and B under envelope control, while the third determines the envelope period. Subsequently placing zero into register 13 (even if it already contains zero) will produce a one-shot decaying chime, which may be repeated by placing further zeros into 13. Alternatively, placing an 8 into 13 will produce a repeating chime or electronic piano note. This note may be further enriched by adding sounds from channel C.

For a different effect, if the following data is entered after the P.S.G. has been reset (either by rerunning the program, or by pressing the Reset button on the Decoding Module), the sound of an explosion will be produced.

6—31
7—7
8—16
9—16
10—16
12—20
13—0

Further entries of 13—0 will repeat the effect, while entering 13—8 will cause continuous repetition. Register 12 determines the decay rate, and 6 the colour of the white noise.

The program in Table 5.2 is useful for testing the P.S.G. and for experimenting with simple sound effects, but a richer variety of effects can be obtained with a little practice when the P.S.G.'s registers are altered under program control using FOR loops to vary the frequencies, timbres, and amplitudes of notes. To give an example of this, the program listed in Table 5.6 produces a two part effect by varying the frequency of channel A in different ways.

KEYBOARD ORGAN

As an example of a somewhat different usage, the program listing in Table 5.7, which should just squeeze into a 4K machine, is for a 14 note organ operated from the Compukit keyboard. When the program is run, a four part menu appears on the screen:

KEYBOARD ORGAN PROGRAM
CONTENTS
1. To play press P (or GOTO 2000).
2. To hear press H (or GOTO 4000).
3. To record press R (or GOTO 5000).
4. To load press L (or GOTO 6000).
NOTE: Space Bar Exits Routines

If P is pressed, the program requests a note period (try around 30 for this) and then a further integer which determines the difference between the two frequencies used for the organ note. The organ may then be played using keyboard letters W–I and S–K. Once 100 notes have been played, or if the space bar is pressed, the program will exit this routine, and return to the menu.

Table 5.6. Dynamic sound effects.

```
70 REM INTERFACING UK101 PROGRAM 10
80 REM SOUND EFFECTS PRODUCED BY
85 REM VARYING DATA IN PSG REGISTER A
90 REM INITIALISATION********************
100 J=61317
110 K=J+1
120 POKEJ,7:POKEK,248
130 POKEJ,8:POKEK,15
140 POKEJ,0
200 REM FIRST EFFECT*********************
220 FORA=0TO128
230 POKEK,A-64*INT(A/64)
240 POKEK,128-A
260 POKEK,A
270 NEXT
290 POKEK,0
300 FORZ=1TO100:NEXT
380 REM SECOND EFFECT********************
390 FORB=1TO10
400 FORA=0TO255STEP10
420 POKEK,A
440 NEXT
450 NEXT
500 POKEK,0
600 INPUT"  AGAIN";YY$
610 IFLEFT$(YY$,1)="Y"THEN200
```

Table 5.7. Keyboard organ using the UK101 and its QWERTY keyboard.

```
20 REM INTERFACING UK101 PROGRAM 11
30 REM AY-3-8910 KEYBOARD ORGAN
50 QA=61808
55 QD=QA+1
60 K=57088
65 DIML(100)
100 PRINT:PRINT:PRINT:PRINT:PRINT,"KEYBOARD ORGAN PROGRAM"
110 PRINT:PRINT," CONTENTS"
120 PRINT:PRINT:PRINT"  1. To play press P (or GOTO 2000)."
130 PRINT:PRINT"  2. To hear press H (or GOTO 4000)."
140 PRINT:PRINT"  3. To record press R (or GOTO 5000)."
150 PRINT:PRINT"  4. To load press L (or GOTO 6000)."
155 PRINT:PRINT"     NOTE  Space Bar Exits Routines"
160 INPUTY$
170 IFY$="P"THEN2000
180 IFY$="H"THEN4000
190 IFY$="R"THEN5000
200 IFY$="L"THEN6000
220 PRINT,"NOT RECOGNISED"
225 PRINT,"ENTER AGAIN PLEASE"
230 GOTO160
2000 REM ORGAN
2003 GOSUB8000
2007 PRINT:PRINT:PRINT
2020 Z=0
2021 INPUT"  TONE QUALITY 0,1,2 ETC";XX
2022 PRINT:PRINT,"KEYBOARD READY":PRINT:PRINT:PRINT
2030 POKE530,0
2040 POKE530,1
2050 POKEK,247
2060 P=PEEK(K)
2080 FORA=9TO15
2090 READB
2100 IFB=PTHENL=A:A=20
2110 NEXT
2120 RESTORE
2122 IFA>19THEN2220
2125 POKEK,239
2130 P=PEEK(K)
2135 FORA=2TO8
2140 READB
2145 IFB=PTHENL=A:A=20
2150 NEXT
2155 RESTORE
2220 POKEK,253
2230 IFPEEK(K)=239THEN3600
2240 IFA<19THEN2030
3000 POKEQA,2
3005 POKEQD,L*5+40
3010 POKEQA,0
3015 POKEQD,L*10+80+XX
3020 POKEQA,13
3030 POKEQD,0
3040 IFL=L1THEN2050
3045 L(Z)=L:L1=L
3050 Z=Z+1
3200 IFZ<101THEN2030
3500 DATA127,191,223,239,247,251,253
3600 POKEQD,0
3602 S=Z-1
3605 POKE530,0
3610 GOTO100
3990 REM AUTO REPLAY ROUTINE
4000 PRINT:PRINT:PRINT:PRINT:PRINT
4001 GOSUB8000
4003 POKEQA,2
4005 PRINT,"AUTO REPLAY"
4010 PRINT,"SEQUENCE LENGTH ";S
4012 PRINT,"FILE NAME  ";FL$
4015 PRINT:PRINT:PRINT
4020 INPUT"REPEATING? Y OR N";R$
4030 INPUT"NOTE LENGTH? 0 - 2000";NL
4100 FORZ=0TOS
4105 POKEQA,2:POKEQD,L(Z)*5+40
4110 POKEQA,0:POKEQD,L(Z)*10+80+XX
4115 POKEQA,13:POKEQD,0
4120 FORA=1TONL:NEXT
4123 POKE530,1
4125 POKE57088,253:IFPEEK(57088)=239THEN4140
4127 POKE530,0
4130 NEXT
4132 POKE57088,253:IFPEEK(57088)=239THEN4140
4135 IFR$="Y"THEN4100
4140 POKEQD,0
4145 POKE530,0
4150 GOTO100
4990 REM SAVE ROUTINE
5000 PRINT:PRINT:PRINT,"FILE CREATION"
5002 INPUT"    ENTER FILE NAME";FL$
5004 PRINT:PRINT:PRINT
5010 PRINT,"SET RECORDER, AND PRESS"
5020 PRINT,"ANY KEY"
5030 INPUTX$
5040 SAVE
5044 PRINT"ZZZ"
5045 PRINTFL$
5050 PRINTS
5055 PRINTXX
5060 FORZ=0TOS
5070 PRINTL(Z)
5080 NEXT
5090 POKE517,0
5095 PRINT:PRINT:PRINT,"RECORDING COMPLETE"
5100 GOTO100
5990 REM LOAD ROUTINE
6000 PRINT,"NOTE SEQUENCE LOADER"
6002 INPUT"    NAME OF FILE REQUIRED";FL$
6004 PRINT:PRINT:PRINT
6010 PRINT,"START TAPE, & PRESS ANY KEY"
6020 INPUTX$
6030 LOAD
6032 INPUTFF$
6034 IFFF$<>FL$THEN6032
6036 PRINT,"FILE   ";FF$;"  FOUND"
6040 INPUTS
6045 INPUTXX
6050 FORZ=0TOS
6060 INPUTL(Z)
6070 NEXT
6075 POKE515,0
6080 PRINT:PRINT:PRINT,"LOAD COMPLETE"
6090 PRINT,"PLAY?"
6130 GOTO100
8000 REM AY-3 INITIALISE
8110 POKEQA,0:POKEQD,0
8120 POKEQA,1:POKEQD,0
8130 POKEQA,3:POKEQD,0
8140 POKEQA,7:POKEQD,248
8150 POKEQA,8:POKEQD,16
8160 POKEQA,9:POKEQD,16
8170 POKEQA,11:POKEQD,0
8180 POKEQA,12:INPUT" PERIOD";PE:POKEQD,PE
8200 RETURN
```

If H is then entered, a replay routine is initiated providing an opportunity to hear again the sequence just played, and to alter note length and replay speed. The replay may be put in a repeating mode if desired.

There is also a facility to save on tape the sequence of digits representing the notes played, and, using a load routine, to reload them on a subsequent occasion.

The save routine uses a file name system, and when a tape is reloaded, the program asks for the name of the file required, and will ignore all other files that it comes across. A vital part of the SAVE/LOAD routines is the PRINT "ZZZ" statement in line 5044. This data (ie, ZZZ) is subsequently read and ignored by the load routine, and ensures that the loading of data is not confused by random noise or unwanted data preceeding the file name. The sequence of data used by the routines is as follows: File Name, Length of Note Sequence, and Timbre Indicator. This data is followed by a sequence of numbers representing the notes played.

SEPARATE USE OF THE 8910's D/A CONVERTERS

The 8910 P.S.G. contains three internal D/A converters, one of which is used for each channel. These may be employed separately for 4 bit D/A conversion, to supplement the ZN425 converter on the Analogue Board. One could, for example, use channels A and B each as 4 bit converters, while retaining channel C for audio output. Fig. 5.4 shows the wiring of the 8910 pads on the Analogue Board for this contingency. Channels A and B are then controlled simply by registers 8 and 9 which, under normal circumstances, would have controlled audio output level. Now, data from 0 to 15 in these registers will determine the d.c. voltage on pads A and B, while data of 16 will place the d.c. voltage under

envelope control. The envelope facility might prove to be particularly useful in a number of different applications. Each of the 3 channels of the P.S.G. could be used to control the level of banks of lights for example, in which case each or all of them could be brought under envelope control to give a slow fade, etc. In such an application each channel output could be made to drive a 741 op. amp. followed by an opto isolator and thyristor or triac controller. It should be noted, however, that the PSG's 4-bit converters may not provide sufficient resolution for some applications.

When using the D/A converter in this way the state of register 7 should be borne in mind. The converter will function whether the associated channel's noise or tone outputs are enabled or not. But with both disabled a non-linearity appears in the transition from 0–1. This is minimised by enabling tone and noise, although the latter places some noise on the d.c. output. The effect can be reduced by increasing the capacitor taken from the pad outputs to ground.

Constructor's Note

A C60 cassette tape containing all the numbered programs of this series is available from *Technomatic Ltd.*, (see advertiser's index), at £3·50 + VAT and p & p.

Next month we shall explore the facilities by the 6522 Versatile Interface Adaptor on the Analogue Board, and will discuss such applications as frequency counters and real time clocks.

PIN CONNECTIONS TO SK3 AND SK4 OF THE DECODING MODULE

One or two readers have asked for specific pin connections for interfacing the devices described in part 2 of the series.

Interfaces such as the light sensor, sound detector or joystick control box described in part 2 of the series, or the l.e.d. indicator or relay output described in part 3 are accessed directly through SK3 or SK4 of the Decoding Module.

The pin connections of these two sockets (SK3 for port A and SK4 for port B) are identical, and were given in Table 1.6 of part 1 of the series.

Devices which require a single port line, a ground and +5V line (such as the l.d.r. circuit of Fig. 2.10 in part 2 of the series) should be connected as indicated in Fig. 5.5.

Pin 8 of SK3 and 4 carries Vcc (+5 volts), pin 1 ground, and pin 16 data line DØ of the port (sometimes labelled PAØ). If the header wired as in Fig. 5.5 is plugged into SK3 of the Decoding Module, and the Logic Tester program of Table 2.5 (part 2 of the series) is run, DØ of the screen display should register a zero for dark conditions, and a 1 in daylight.

The joystick control box of Fig. 2.9 of part 2 connects directly to a 16-pin header that may be inserted into SK3 or SK4 of the Decoding Module. The wiring for this is given in Fig. 5.6. If the header is plugged into SK3 it may be used in conjunction with the screen drawing program listed in Table 2.4 of part 2.

Similar pin connections are made to SK3 or SK4 when using the PIA for digital output. As an example, Fig. 5.7 gives the pin connections for simple audio output from the PIA. Again use is made of the 5 volt line supplied by the Decoding Module. In this instance it is used to power the 2N2926 audio amplifier. This circuit should be used in conjunction with the programs in Table 3.5 and 3.6 of part 3 of the series.

Other input and output applications use similar connections to SK3 and SK4.

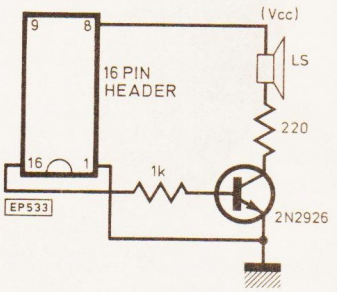


Fig. 5.4. Pin connections to 16-pin header for simple audio output from PIA

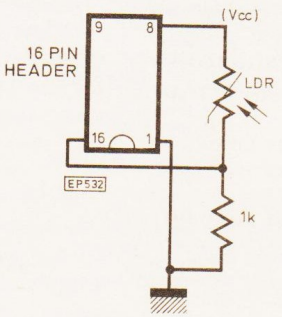


Fig. 5.5. Pin connections to 16-pin header for use with LDR. The header plugs into SK3 or SK4 of the Decoding Module

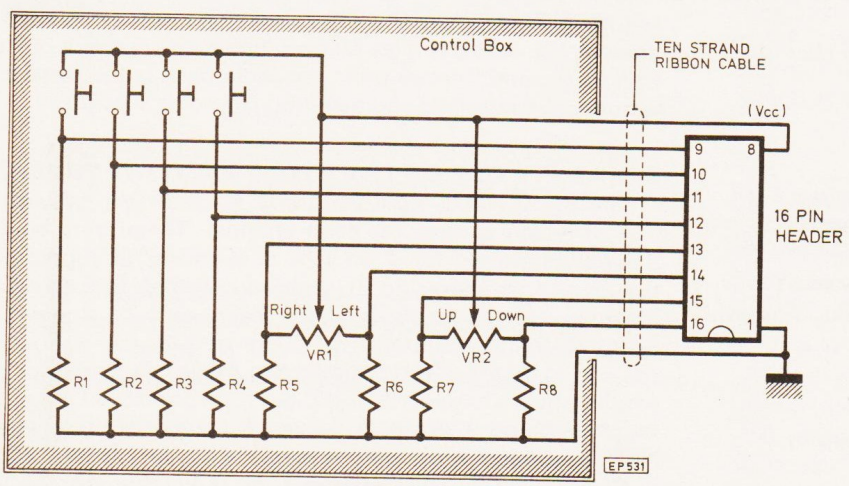


Fig. 5.6. Joystick control box giving connections to 16-pin header to be plugged into SK3 of the Decoding Module

Interfacing COMPUKIT

Part 6 D. E. Graham

THIS month we shall be looking at applications of the 6522 Versatile Interface Adaptor.

6522 VERSATILE INTERFACE ADAPTOR.

The third section of the Analogue Board consists of a 6522 Versatile Interface Adaptor. This is an extremely useful 16 register device, providing two programmable 8 bit input/output ports with handshaking and interrupt control, two 16 bit timers, and a serial input/output port. Fig 6.1 gives its schematic diagram.

Because of the complexity of this chip, we are forced to be highly selective in the treatment of it here, and have chosen to concentrate on the use of its timing and counting registers. In this context we will explore a number of practical applications including a signal generator, a digital frequency meter, a digital capacitance meter, and a real time clock.

For a treatment of aspects of the 6522 not covered here, the reader is referred to the bibliography at the end of the article.

6522 INTERFACE

Fig. 6.2 gives the circuit of the 6522 section of the Analogue Board. The device is selected by the $\overline{BL2}$ line from the Decoding Module, giving it a base address of 61344 decimal. All external connections to the 6522 are made via the 16 pin d.i.l. sockets SK4 and SK5 on the Analogue Board (see Fig. 6.3 for pin connections). These have been made similar to those of SK3 and SK4 of the Decoding Module for the purposes of interchangeability.

The 6522 is a much more demanding device to interface than the 6821 PIA; and the timing and state of the chip select, Ø2 and R/W signals and ground connections are much more critical. For this reason, as suggested earlier, leads between the Analogue Board and the Decoding Module should be kept to a few inches in length. It is also necessary to enhance the earthing between the 6522 and IC2 on the Decoding Module, and between the UK101 earth track close to its expansion socket and the 6522. This is accomplished by soldering two leads as indicated in Fig. 6.4. For the same reason a 1n capacitor (C18) not provided for in the published p.c.b. design, but included in the kit supplied by *Technomatic Ltd.*, must be taken from the 6522 chip select line (pin 23) to ground (or Vcc). This is most easily accomplished by soldering the capacitor between pins 23 and 24 of the 6522 on the underside of the board. This capacitor has the effect of delaying the chip select pulse sufficiently to coincide with the arrival of the slightly misshapen and overworked Ø2 and R/W signals.

TESTING THE 6522

After checking the supply to the socket of IC1, the 6522 should be inserted, and the program listed in Table 6.1 run. This displays the states of the 6522's sixteen registers. Even after a Reset, these should contain a variety of data. If all registers read the same, then the chip is not interfacing correctly, and checks should be made on pins 1, 20, and 22-38 of IC1 for a faulty connection or short circuit, either on the Analogue Board itself, or in the connections to the Decoding Module.

Once the registers appear to read correctly, data may be written to them using the program of Table 6.1. After displaying a readout of each of the 6522's registers, the program requests a register number, and data to be entered in it. And after performing the writing operation, will display the new states of all registers. When using this program in initial tests, it should be remembered that by no means all of the 6522's registers can be directly written to. To test the operation of the 6522 for both Read *and* Write operations, a convenient pair of registers to use is 2 and 3. These should each accept any integer from 0 to 255, and the program should display the value entered, on subsequent readout.

This is also an appropriate point to test the Reset function. Pressing the Reset button on the Decoding Module should reset certain (but not all) of the 6522's registers. As a test, it should be found that registers 2 and 3 are set to zero by this operation. Because of the complexity of the VIA it is advisible to reset it in this way before carrying out any of the experiments below using the two parallel ports or the timers; although as mentioned earlier, the Decoding Module Reset line is automatically activated at power-on.

USE OF THE PARALLEL PORTS

The 6522 has a pair of 8 bit input/output ports very similar to those of the 6821 PIA, though because of its extra provision of registers it is much easier to configure than the latter. Table 6.2 gives the configuration of the 6522's sixteen registers. From this it may be seen that registers 0 and 1 are the data registers for ports B and A respectively; note the reversal of order here. Registers 2 and 3 (the Data Direction Registers) determine whether each bit of port B and Port A Data Registers are set for input or output. This is very similar to the functioning of the 6821's Data Direction Registers.

To set port B for output on all 8 bits, the value 255 should be placed in register 2. This may be accomplished either by executing the command POKE 61346, 255, or by using the program in Table 6.1 to enter the data manually. In either case this should set the port B data pins (ie pins 9-16 of SK5) to zero volts. To output data through the port, that data should then be written to register 0 (at 61344). Thus for example the two commands:

```
POKE 61346, 255
POKE 61344, 15
```

will cause an output on port B of 15; or in other words, pins 16, 15, 14 and 13 of SK5 will read a one (about 4·5 volts), while pins 12, 11, 10 and 9 will read a zero (a few tens of millivolts). The configuration of port A follows a similar pattern, with the Data Register (register 1) at 61345, and the Data Direction Register (register 3) at 61347.

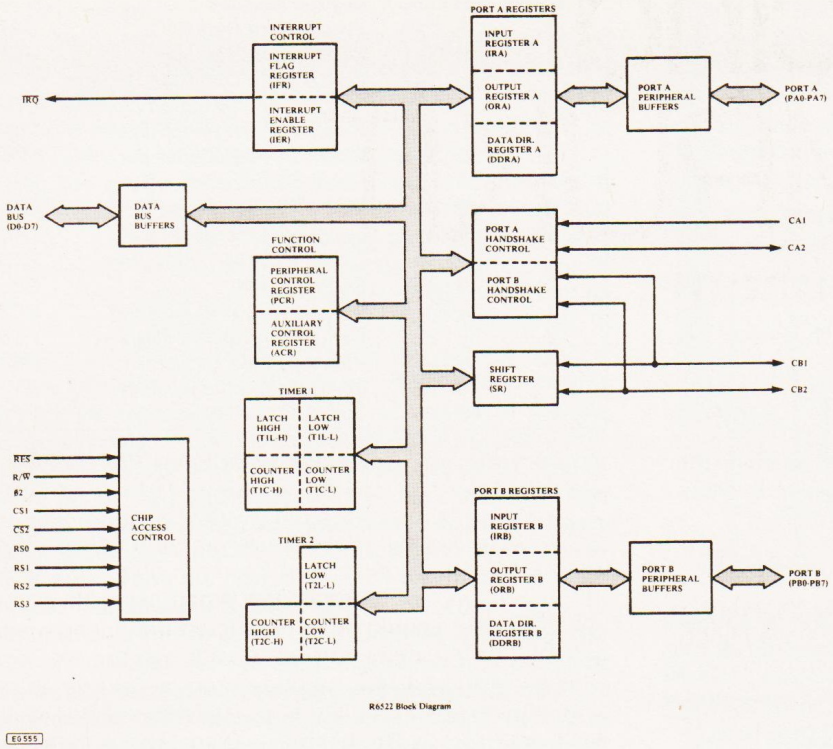


Fig. 6.1. 6522 Block diagram

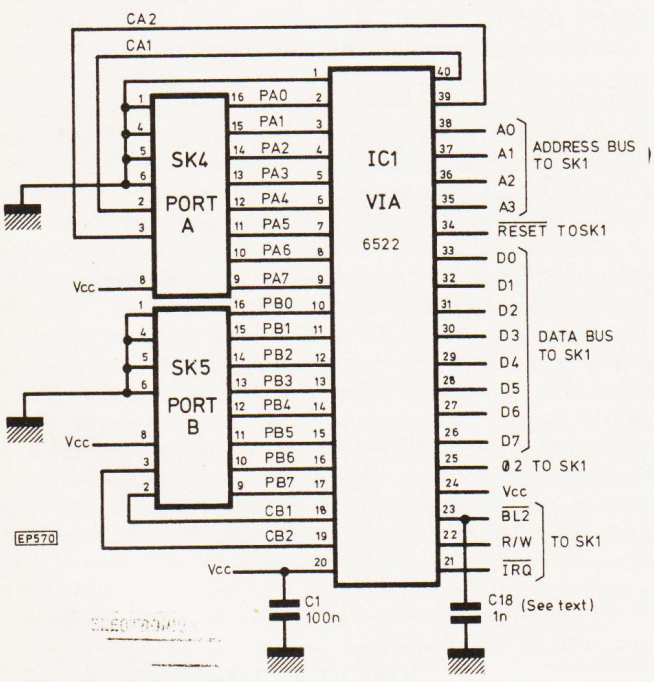


Fig. 6.2. 6522 section of the Analogue Board

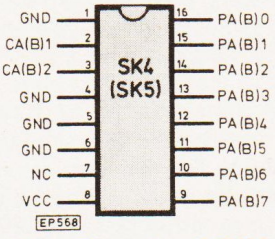


Fig. 6.3. Pin-outs of SK4 and SK5 of Analogue Board. SK4 carries Port A of VIA, SK5, Port B

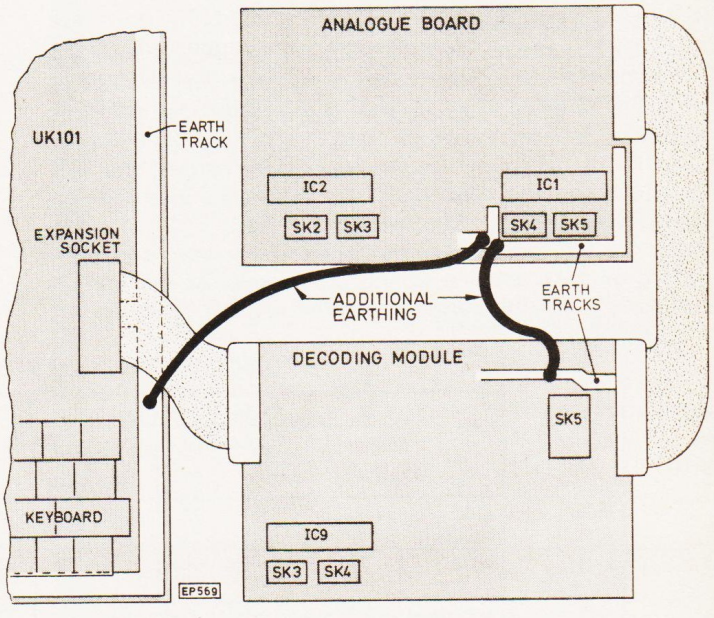


Fig. 6.4. Additional earthing leads to Analogue Board

Table 6.1. 6522 handling program

```
80 REM INTRFACING UK101 PROGRAM 12
85 REM 6522 HANDLING PROGRAM
90 DIMA$(15)
95 P=61344
100 FORA=0TO15
110 READA$(A)
120 NEXT
130 PRINT
140 FORA=0TO15STEP2
145 PRINTTAB(9)CHR$(140);
150 PRINTTAB(10)A;TAB(15)A$(A);
160 PRINTTAB(20)PEEK(P+A);TAB(25);CHR$(140);
170 PRINTTAB(26)A+1;TAB(31)A$(A+1);
180 PRINTTAB(36)PEEK(P+A+1);
185 PRINTTAB(41)CHR$(140)
190 NEXT
200 PRINT
250 INPUT" REGISTER";R
255 IFR>15THEN130
260 INPUT" DATA";D
270 POKEP+R,D
280 GOTO130
300 DATADRB,DRA,DDB,DDA,T1L,T1H,T1L,T1H
310 DATAT2L,T2H,SR ,ACR,PCR,IFR,IER,DRA
```

Table 6.2. The 6522's registers

Register Number	Address (Decimal)	Register Function	Nemonic
0	61344	Port B Data Register	DRA
1	61345	Port A Data Register	DRB
2	61346	Port B Data Direction Register	DDA
3	61347	Port A Data Direction Register	DDB
4	61348	Timer T1 Low Order Byte	TIL
5	61349	Timer T1 High Order Byte	TIH
6	61350	Timer T1 Low Order Byte	TIL
7	61351	Timer T1 High Order Byte	TIH
8	61352	Timer T2 Low Order Byte	T2L
9	61353	Timer T2 High Order Byte	T2H
10	61354	Shift Register	SR
11	61355	Auxiliary Control Register	ACR
12	61356	Peripheral Control Register	PCR
13	61357	Interrupt Flag Register	IFR
14	61358	Interrupt Enable Register	IER
15	61359	Port A Data Register	DRA

Data input on each port is configured by setting the corresponding bits in the Data Direction Register to zero. Thus the commands:

POKE 61347, 0
PRINT PEEK (61345)

will print the value of the data at port A. This will of course be a decimal representation of the binary states of the 8 input lines at pins 16-9 of SK4.

The 6522 does offer one extra facility over the 6821 PIA in the input mode: it may be configured to latch data in response to level changes in the peripheral control lines CA1 and CB1; and for details of how this is accomplished the reader is referred to the 6522 data sheets.

Since the pin connections to SK4 and SK5 of the Analogue Board have been made similar to those serving the PIA on the Decoding Module, devices such as the Joystick Control Box described in parts 2 and 5 of the series may be plugged directly into either of the 6522 ports if desired. In order to run the Joystick from port B of the 6522, for example, the Joystick header should be plugged into SK5 of the Analogue Board, and the Joystick program of Table 3.4 of Part 3 may be run with the following alterations:

130 P=61344
150 POKE P+2, 0
160 Deleted

THE 6522's TIMERS

The 6522 contains two 16 bit counters which may be used for timing and/or counting operations. The two counters, usually referred to as T1 and T2 are somewhat different in operation, and each may be used in a number of different modes. Mode selection is carried out by setting the appropriate bits in the Auxillary Control Register and the Interrupt Enable Register. See Table 6.3.

Timer T1 will only count at the Ø2 clock rate, and may therefore only be used in timing operations. At the end of each count it may be variously conditioned to cause an interrupt and/or to switch the polarity on PB7 (data line 7 of port B of the VIA, accessed through pin 9 of SK5). This counter may also be configured either in a one-shot mode, in which case only a single interrupt, or a single change in polarity occurs on PB7; or it may be configured in the free running mode, in which case a continuous series of interrupts may be produced, and a continuous square wave train of predetermined frequency may be output at PB7.

Timer T2 can be conditioned to count either at the Ø2 rate, or to count the pulses appearing on PB6 (data line 6 of port B of the VIA, accessed through pin 10 of SK5). When a predetermined number of pulses from either source has been counted, T2 can cause an interrupt to signal this event. Timer T2 may also be conditioned to produce clock pulses for the 6522's shift register, though there is not the space here to discuss this latter feature.

The precise operation of the two counters is best understood with reference to specific applications. Timer T2 is the least complex of the two, and we will look at this first.

EVENT COUNTER

Since T2 is able to count incoming pulses on PB6, it may be used as a simple event counter. Only one bit (bit 5) of the Auxilliary Control Register controls the operation of T2. When it is at zero, T2 counts at the Ø2 rate. Setting it to one on the other hand causes it to count pulses on PB6. For an event counter therefore, the value 32 should be loaded into the ACR (register 11 at 61355), so as to give a one at bit 5. For initial tests this may be done manually with the program of Table 6.1. It now remains to set the counter operating. This is accomplished by loading data into T2's latches. The latches are a pair of 8 bit registers that may be loaded by writing data to registers 8 and 9 of the VIA. For the purposes of the event counter it is probably easiest to write 255 into each of these. This may be done using the manual entry program. Writing 255 into register 8 will have no apparent effect, but when register 9 is written to, the data in the two latches will automatically be loaded into the counting registers of T2, and the count started. Reading registers 8

Register	Nemonic	FUNCTION OF EACH BIT							
		7	6	5	4	3	2	1	0
11	ACR	T1 Control		T2 Control	Shift Register Control			Port B Latch Enable	Port A Latch Enable
12	PCR	CB2 Control			CB1 Control	CA2 Control			CA1 Control
13	IFR	IRQ	T1	T2	CB1	CB2	Shift Register	CA1	CA2
14	IER	Interrupt Control	T1	T2	CB1	CB2	Shift Register	CA1	CA2

Table 6.3. Function of Auxiliary Control, Peripheral Control, Interrupt Flag, and Interrupt Enable Registers of the 6522

and 9 has the effect of reading the value of the ongoing count, and with no signal on PB6, this should continue to be 255 for each register.

To use the event counter to count switch closures, a debouncing circuit is essential. Fig. 6.5 gives a suitable circuit using an NE555 in the monostable mode. It is supplied by a 5 volt line from SK5. Each time that the push button is closed T2's counting registers will decrement by one.

By using the program in Table 6.4, setting up the event counter on T2 may be streamlined a little. This loads the ACR with 32, and places 255 into registers 8 and 9, and then continuously monitors their contents. If any change is detected, it prints out the new contents, suitably deducted from 65535, so as to effect a count up rather than down. If no pulses are present, a single zero will be printed. Such a counter may of course be used with any device producing a useable voltage transition, such as a photocell or Geiger counter for example, though in both cases the signals would need to be suitably conditioned before application to PB6. In the case of an l.d.r. light sensor, a single transistor amplifier driving pin 2 of the NE555 monostable of Fig 6.5 should prove to be adequate.

A SIMPLE FREQUENCY COUNTER

Similar principles may be used to create a simple frequency counter. Table 6.5 gives a program which achieves this. The signal to be measured is applied to PB6, and the number of pulses occurring during a one second interval is counted, so giving the frequency in Hz. The counting period is determined by the assigned value of T in line 105. By trimming this value, accuracies of up to about 0·1% should be possible, which will be sufficient for many applications.

The frequency under test is calculated in line 180 of the program from the data in the counting registers of T2; and then the subroutine at line 2000 is entered. This POKEs the value of N (the frequency measured) directly on to the screen at a location assigned in line 2200. The subroutine is then re-entered at line 2220 to POKE up the units of measurement. In this case these are Hz only, but in a later program the same routine is used to provide an auto-ranging readout. The routine at 2000 is quite portable, and may be used for POKEing to the screen any assigned variable, or if entered at 2220 will POKE up any assigned string variable. For use on the Superboard, it should only be necessary to alter the value of S, the screen address, in line 2200.

The test signal (applied to PB6) should be a t.t.l. compatible pulse train of frequency from a few Hz up to about 65 kHz. For greater precision at low frequencies a longer waiting loop could obviously be employed; while for higher frequencies a shorter waiting loop could be used, but accuracy would be likely to decline as a result. In any case the maximum useable frequency is determined by the 6522's timing characteristics, which require pulses on PB6 to be at least 2 µs in duration, and of 2 µs separation. For frequencies above 250 kHz therefore, a prescalar should be used. Also it should be noted that if it is required to use this counter arrangement for measuring sine rather than square waveforms, some form of input conditioning will be required to square the waveform, such as a high gain amplifier.

DIGITAL CAPACITANCE METER

The frequency counting facilities provided in the case above may be conveniently employed to produce a digital capacitance meter. This uses an NE555 timer i.c. in the astable mode to generate a frequency dependent on the unknown capacitor. The output of the 555 is fed to PB6 of the 6522, and is counted for one second intervals by timer T2.

Fig. 6.6 gives the circuit of the tester, which may be built on a small piece of Veroboard, and is powered by the 5 volt supply available at SK5, which also carries the PB6 line.

The output frequency of the 555 in this circuit is determined by the expression F =0·7/(R×C1), where C1 is the value of the capacitor under test. The program for the tester employs this fomula to print out the value of C1 in nanofarads. It allows for any value of R to be used, and requests the value employed to be entered at the outset; although once a value (or series of values) has been decided upon, it may be easiest to write it into the program.

Fig. 6.5. NE555 monostable debounce circuit for use with event counter

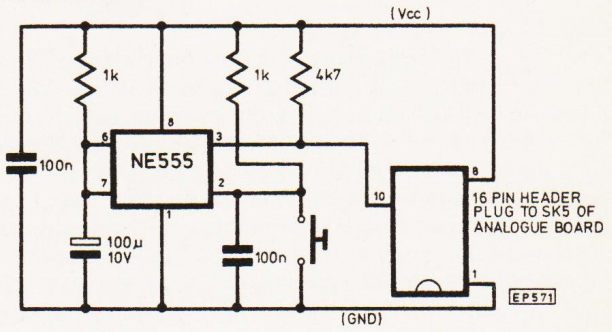
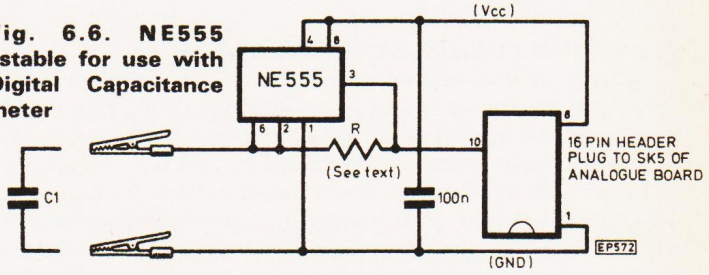


Fig. 6.6. NE555 astable for use with Digital Capacitance meter




```
70 REM INTERFACING UK101 PROGRAM 13
80 REM 6522 EVENT COUNTER
90 REM COUNTS PULSES ON PB6
95 FORZ=1TO16:PRINT:NEXT
96 PRINT,"6522 EVENT COUNTER"
98 PRINT:PRINT:PRINT:PRINT
100 P=61344
110 POKEP+11,255
120 POKEP+8,255
130 POKEP+9,255
140 Y1=Y:Z1=Z
150 Y=PEEK(P+8)
160 Z=PEEK(P+9)
170 IF(Y=Y1)AND(Z=Z1)THEN150
180 PRINT,65535-Y-256*Z
200 GOTO140
```

Table 6.4. Event counter program

```
80 REM INTERFACING UK101 PROGRAM 14
90 REM SIMPLE FREQUENCY COUNTER USING 6522
95 REM SQUARE WAVE INPUT ON PB6
96 FORZ=1TO16:PRINT:NEXT
97 PRINT,"SIMPLE FREQUENCY COUNTER"
98 PRINTTAB(11)"('P' PRINTS - SPACE BAR EXITS)"
99 PRINT:PRINT:PRINT:PRINT:PRINT
100 P=61344
105 T=1110
110 POKEP+11,255
120 POKEP+8,255
130 POKEP+9,255
140 FORQ=0TOT:NEXT
150 Y=PEEK(P+8)
160 Z=PEEK(P+9)
180 N=65535-Y-256*Z
200 GOSUB2000
220 S=S+10
230 N$="HZ"
240 GOSUB2220
250 GOSUB3000
300 GOTO110
2000 REM
2200 S=54110
2210 N$=STR$(N)
2220 L=LEN(N$)
2240 FORM=1TOL
2250 POKES+M,ASC(MID$(N$,M,1))
2260 NEXT
2265 IFPEEK(S+M)=32THEN2280
2270 POKES+M,32:M=M+1:GOTO2265
2280 RETURN
3000 REM
3020 POKE530,1
3030 POKE57088,253
3040 IFPEEK(57088)=239THENPOKE530,0:END
3050 IFPEEK(57088)=253THENPRINT:PRINT
3060 POKE530,0
3080 RETURN
```

Table 6.5. Simple frequency counter program

The circuit appears to give best results with R = 1mΩ (giving a range from about 100p to about 100n, or R = 100k (giving a range from about 500p to 1m). Using lower values of R (eg 10k or 1k) to extend the range to higher values of capacitor unfortunately corrupts the mark to space ratio of the output, causing a false reading—although it might be possible to compensate for this somewhat in calculating the capacitor value. Another way to extend the range would be to use a longer waiting loop. A ten second loop, achieved by setting T to about 10000 in line 105, would extend the range at the high end by a factor of 10.

TIMER T1

Timer T1 is a slightly more complex device than T2, and makes use of four count and latch registers, where T2 uses only two. It also differs significantly in being able to count only at the Ø2 rate, but is able to toggle PB7 each time a count is completed. Thus while T2 can be conveniently conditioned to *count* pulses on an external line (ie PB6), T1 can *produce* an external square wave output; and in fact by wiring PB6 and PB7 together (and taking a 1kΩ load resistor to earth) counters T1 and T2 could be cascaded to give a 32 bit counter. As with timer T2, we will use specific examples to illustrate its operation.

PRECISION FREQUENCY GENERATOR

With minimal conditioning operations T1 may be set up to produce a square wave output on PB7 (pin 9 of SK5) of frequency between 10Hz and 250kHz.

T1's counting mode is determined by bits 6 and 7 of the Auxilliary Control Register. See Table 6.7. With bit 6 at one, the counter operates in the free running mode, while a zero will put it in the one-shot mode. Bit 7 determines whether there will be an output on PB7 (bit 7 = 1) or not (bit 7 = 0). Incidentally when PB6 or PB7 are selected for use with T2 or T1 this takes precedence over their use as input/output data channels for port B.

From Table 6.7 it may be seen that placing a one at bits 6 and 7 of the ACR conditions T1 for continuous output on PB7. This is arranged in such a way that the output of PB7 is inverted at the end of each count.

It only remains to place the required values for the count in T1's latch registers. These are written to at registers 4 (low order byte) and 5 (high order byte); and as with T2, writing the high order byte automatically causes the two values to be loaded into the counter, and counting to begin. When T1's counting registers reach zero, the polarity on PB7 is inverted, and the contents of the two latch registers again loaded into the counters, and so on.

This operation may be set up using the manual entry program of Table 6.1. As a test, place 192 into register 11 (setting bits 6 and 7 of the ACR), followed by 2 into register 4, and 0 into register 5. This should produce a square wave output on PB7 of period 8 µs.

The length of the period is given by the expression' T = 2(256A + B + 2) µs; where A is the high order byte, and B the low order. The initial multiplication factor of 2 arises because of the toggling of the output effectively dividing the frequency by 2, and the additional 2 in the expression is associated with the time taken to load the counters etc.

It should be noted here that because of the configuration of the 6522's registers, the latches of timer T1 are *written to* at registers 4 and 5, but are *read* at 6 and 7. Data entered in register 4 actually appears on reading register 6, and similarly for 5 and 7. Reading registers 4 and 5 gives the current value of the count in progress. In the author's experience register 4 is particularly sensitive to good earthing of the 6522 and to good timing of chip select, Ø2 and R/W, and it is advisable to check that data written into register 4 does actually appear at register 6. As a test, try for example the values 0 then 1. If these fail to write in correctly, a check should be made that C18 is in place, and that the extra earth connections described earlier are properly made—some experimentation may be required here.

```
60 REM INTERFACING UK101 PROGRAM 15
70 REM 6522 DIGITAL CAPACITANCE METER
80 PRINT"   DIGITAL CAPACITANCE METER"
85 PRINT"   USING NE555 DRIVING PB6 OF VIA"
86 PRINT:PRINT
90 INPUT"  R IN KILOHMS";R
100 P=61344
105 T=1110
110 POKEP+11,255
120 POKEP+8,255
130 POKEP+9,255
140 FORQ=0TOT:NEXT
150 Y=PEEK(P+8)
160 Z=PEEK(P+9)
180 HZ=65535-Y-256*Z
190 IFHZ<3THENPRINT"OUT OF RANGE":GOTO110
195 IFHZ<10THENPRINT,"LOW PRECISION RESULT"
200 C=700/(HZ*R)
210 C=1000*C
230 Q=4
240 IFINT(C/10^Q)<1THENQ=Q-1:GOTO240
250 C=INT(.5+C/10^(Q-1))*10^(Q-1)
260 PRINT,C;
270 PRINTTAB(22);"NANOFARADS"
300 GOTO110
```

Table 6.6. Digital capacitance meter program

Table 6.7. Functions of bits 6 and 7 of 6522 Auxiliary Control Register

bit 7	bit 6	Mode
0	0	T1 one-shot mode—Generate a single time out interrupt each time T1 is loaded. Output to PB7 disable
0	1	T1 free-running mode—Generate continuous interrupts. Output to PB7 disabled
1	0	T1 one-shot mode—Generate a single time out interrupt and an output on PB7 each time T1 is loaded
1	1	T1 free-running mode—Generate continuous interrupts and toggle output on PB7

Using the 6522 to toggle the output on PB7 allows the implementation of very precise frequency generation; although of course only frequencies corresponding to integral numbers of counts may be generated, and ultimately the accuracy is limited by that of the UK101's 8 MHz crystal, which on the author's machine was about 1 part in 3000.

The program listed in Table 6.8 implements the signal generator. It first requests a frequency, and then outputs the nearest frequency available (rounding down), and at the same time prints out the exact frequency actually produced (within the limits imposed by the 8 mHz crystal). It will be noted when using this program that at the high end of the scale, at 100 or 200 kHz, the choice available is somewhat limited. In fact one can either have a 4, 6, or 8 µs time period. At lower frequencies, much greater choice is available, and in the 1 kHz region for example, the frequency may be adjusted in steps of about 2 Hz.

FREQUENCY COUNTER USING T1 AND T2

One of the limitations to the accuracy of the frequency counter described earlier is the timing loop used to produce the one second delay during which the signal to be measured is counted. A more precise way to implement such a delay for higher frequencies is to use timer T1 in the one shot mode. T2 could then be made to count pulses on PB6 for the duration of T1's count. Stopping the count on T2 at the moment when T1 times out may be accomplished using a pair of NAND gates as shown in Fig. 6.7. The first gate (i.c. 1a) is used to invert the output of PB7 since this goes low rather than high during the count period. The signal to be tested is applied to the second gate (i.c. 1b) with the result that an output to PB6 is produced only during the counting time of T1.

```
80 REM INTERFACING UK101 PROGRAM 16
90 REM PRECISION SQUARE WAVE GENERATOR
95 REM USING 6522 VIA
100 P=61344
110 POKEP+11,192
115 PRINT:PRINT:PRINT:PRINT
120 PRINT:PRINT:PRINT,"PRECISION SQUARE WAVE"
130 PRINT,"GENERATOR - O/P ON PB7 OF VIA"
135 PRINT:PRINT:PRINT
140 PRINT:PRINT"  FREQ REQUIRED (10 - 250000 HZ)"
145 INPUTF
150 IFF>=10ANDF<=250000THEN180
160 PRINT:PRINT"   OUT OF RANGE - ENTER AGAIN"
170 GOTO140
180 N=(500000/F)-2
190 N1=INT(N/256)
200 N2=INT(N-256*N1)
205 N3=256*N1+N2
210 F1=500000/(N3+2)
215 PRINT:PRINT"  EXACT FREQ GENERATED =";
220 PRINTF1;" HZ"
225 PRINTTAB(20);"OR ";
230 PRINT1000/F1;" M SEC"
240 POKEP+4,N2
250 POKEP+5,N1
300 GOTO140
```

Table 6.8. T1 frequency generator program

A t.t.l. wave train is then applied at point X, and the program in Table 6.9 run. This puts the value 160 into register 11 of the VIA, setting bits 7 and 5 high, and conditioning T2 for input on PB6, and T1 for the one-shot mode with output on PB7. T2 and T1 are then both loaded with the value 65535 (by placing 255 into both low and high order bytes), so setting in motion both counters. After a wait, the program POKEs the input signal frequency to the screen with auto-ranging units, then repeats the sequence.

As with the frequency counter described earlier, signals must be below 250kHz, and be t.t.l. compatible. And again a prescalar may be used to extend the upper limit. As a simple example Fig. 6.8 shows a 7490 configured as a divide by ten counter which effectively extends the range up to about 2 mHz. The accuracy of this frequency counter (±20 Hz) is determined by the relatively short period (about 1/20 sec) during which the sample is taken. The only way around this with this method is to use software to extend the counting period to several count cycles of T1.

THE USE OF INTERRUPTS

One obvious application of the 6522 VIA in the personal computer context is the implementation of a real time clock. There are many ways of achieving this. About the simplest would be to set either T1 or T2 counting down at the Ø2 rate, and inspect its count registers periodically, calculating the time accordingly. Such a procedure however will tie up the CPU rather more than is necessary, rendering the simultaneous running of other programs difficult if accurate timekeeping is to be achieved.

An alternative approach which does not suffer from this drawback is to use the 6502's interrupt facilities. An interrupt provides a way of drawing the CPU's attention to a particular situation at the instant that it occurs, without the CPU having to waste time in waiting loops, perpetually examining the state of various registers for a given occurrence.

The 6502 has two interrupt lines: the $\overline{\text{NMI}}$ or non-maskable interrupt, and the maskable $\overline{\text{IRQ}}$ line. When either of these is taken low by some external device, the CPU will shelve what it is doing and start the execution of a separate set of routines. When it has completed these it will return to where it left off, and continue until further interrupts are sensed. In the case of the maskable interrupt, the CPU will only take notice of the $\overline{\text{IRQ}}$ line going low if the interrupt mask flag in the 6502's own status register is not set (ie at zero). This is not the case with NMI, which cannot be masked in this way, and is acted upon as soon as the CPU has completed the instruction that it was engaged upon when $\overline{\text{NMI}}$ was taken low.

Interrupt facilities provided by the $\overline{\text{IRQ}}$ line may be used in a number of ways. As an example, they might be used to signal to the CPU that new data had arrived at a particular

```
80 REM INTERFACING UK101 PROGRAM 17
90 REM FREQUENCY COUNTER USING T1 + T2
91 FORZ=1TO16:PRINT:NEXT
92 PRINT,"T1 T2 FREQUENCY COUNTER"
94 PRINTTAB(11)"('P' PRINTS - SPACE BAR EXITS)"
95 PRINTTAB(15)"  ACCURACY +/- 20 HZ  "
96 PRINT:PRINT:PRINT:PRINT
100 P=61344
120 POKEP+11,160
130 L2=255
140 L1=255
150 GOSUB4000
1000 F=(256*C1+C2)/(256*L1+L2+1.5)
1010 N=INT(F*100000+.5)/100
1020 N1=0:IFN<1THENN=1000*N:N1=1
1040 GOSUB2000
1050 S=S+10
1060 N$="KHZ":IFN1=1THENN$=" HZ"
1070 GOSUB2220
1080 GOSUB3000
1090 GOTO100
2000 REM
2200 S=54110
2210 N$=STR$(N)
2220 L=LEN(N$)
2240 FORM=1TOL
2250 POKES+M,ASC(MID$(N$,M,1))
2260 NEXT
2265 IFPEEK(S+M)=32THEN2280
2270 POKES+M,32:M=M+1:GOTO2265
2280 RETURN
3000 REM
3020 POKE530,1
3030 POKE57088,253
3040 IFPEEK(57088)=239THENPOKE530,0:END
3050 IFPEEK(57088)=253THENPRINT:PRINT
3060 POKE530,0
3080 RETURN
4000 REM
4100 POKEP+8,255
4110 POKEP+9,255
4120 POKEP+4,L2
4130 POKEP+5,L1
4142 FORZ=1TO1000:NEXT
4150 C1=255-PEEK(P+9)
4160 C2=255-PEEK(P+8)
4200 RETURN
```

Table 6.9. T1–T2 frequency counter program

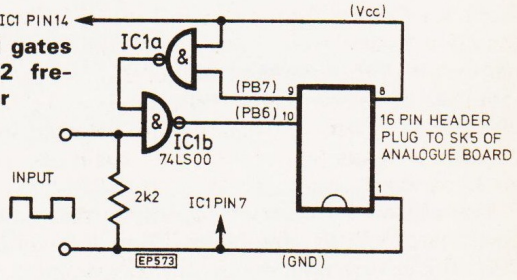


Fig. 6.7. Nand gates used in T1–T2 frequency counter

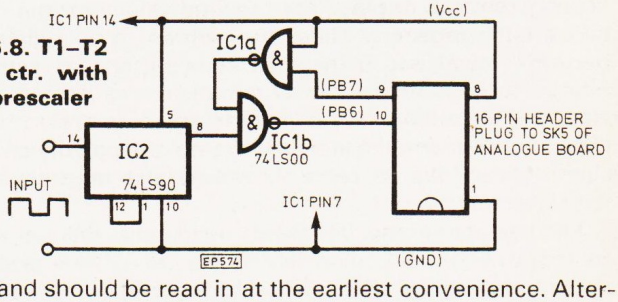


Fig. 6.8. T1–T2 freq. ctr. with ÷10 prescaler

port, and should be read in at the earliest convenience. Alternatively they might be used in the implementation of what is called an interrupt driven clock. Essentially this uses a counter to pull the $\overline{\text{IRQ}}$ line low at regular intervals, say 20 times every second. The interrupt routine can then count the number of times this occurs, and store the number of elapsed seconds, minutes and hours in RAM, returning to the main program when this has been done. Because interrupt handling routines are written in machine code, their execution times are usually very short. In the case of the interrupt-driven clock the CPU will only spend a few microseconds every 50 ms on timekeeping activities, so that this can provide a very efficient way of implementing a clock function on a microcomputer system.

Next month we will give a program which achieves this using the 6522 on the Analogue Board. But for now we will take a look at the general principles involved in using the IRQ interrupt on the UK101.

A SIMPLE INTERRUPT ROUTINE

As a simple illustrative example we will consider an interrupt program which just increments a given memory location when an IRQ interrupt occurs. And to keep the hardware very simple, we will take the $\overline{IRQ}$ line low with a push button connected between $\overline{IRQ}$ and ground (pins 1 and 8 of the UK101 expansion socket).

Interrupts can only be handled using machine code since there are no appropriate Basic commands, and in any case for interrupt servicing, speed of operation is usually of the essence. The central part of an interrupt program is the so-called interrupt service routine, to which the CPU is directed every time the $\overline{IRQ}$ line is brought low. This must first of all save the contents of the 6502's operating registers by pushing them on to the stack. The 6502 automatically saves its program counter and status register in this way, so that it is only necessary to include instructions for saving the Accumulator and X and Y registers; and even this is only necessary if their contents will be disturbed by the main body of the interrupt service routine.

In this case the main body of the routine will simply increment a given memory location. And when this is complete, the interrupt routine must restore the contents of the 6502's operating registers and finally execute a return to the program on which it was engaged prior to interrupt.

Table 6.10 gives a listing of an interrupt service routine which performs these functions. It is located at 0230 hex, and will cause the data stored at 022F hex to be incremented each time an interrupt is sensed.

Table 6.10. Disassembler listing of simple interrupt service routine

```
0230 48        PHA
0231 8A        TXA
0232 48        PHA
0233 EE2F02    INC $022F
0236 68        PLA
0237 AA        TAX
0238 68        PLA
0239 40        RTI
```

The first command PHA pushes the contents of the accumulator on to the stack. Next the contents of the X register are transferred to the accumulator, and are themselves pushed on to the stack for later retrieval. In this particular example these three commands are not in fact necessary, since the main body of the interrupt service routine does not use either the accumulator or the X register, and so their contents do not need to be protected in this way. The instructions are included here to illustrate the principle involved; and if the Y register is to be used in the service routine, then its contents should also be preserved in the same way.

The next command, INC 022F, increments the contents of 022F, while the three which follow, PLA, TAX and PLA, restore the contents of the X register and then the accumulator. Note that there is an effective reversal of order here, since the stack works on a first-in last-out basis. The final command, RTI, causes a return to the main program.

Before this interrupt service routine can be used, the UK101's interrupt system must be initialised in two ways. Firstly the 6502's IRQ mask must be cleared. This is performed by executing the instruction CLI. In the example we will do this by setting up a two instruction subroutine at 0228 hex. This involves placing 58 hex (CLI) at 0228, and 60 hex. (RTS) at 0229. We can then jump to this subroutine to clear the interrupt mask. Clearing the mask in this way must be carried out as a subroutine of the main program, since using a systems Reset on the UK101 to get back into Basic, automatically resets the IRQ mask again. This is why taking the IRQ line low has no effect during the normal running of Basic or machine code programs—IRQ is masked.

The second part of the initialisation procedure involves the UK101's IRQ vector. This is a location in RAM (01C0 hex, 448 dec) to which the CPU jumps when it accepts an IRQ request. We must load this location with an instruction that directs it to the interrupt service routine described above, which is located at 0230 hex. To do this, locations 01C0, 01C1 and 01C2 are loaded with 4C, 30 and 02 hex respectively, representing the instruction jump (JMP) to 0230.

We are now in a position to test the IRQ interrupt. To do this, the data in Table 6.11 should be entered at the locations indicated, using the UK101 Monitor. This effectively enters the CLI subroutine, the interrupt jump instruction at the IRQ vector, and the main interrupt service routine described above. When these are in, the BASIC program in Table 6.12 should be run. Lines 120-160 of this use the USR(X) call to clear the interrupt mask. The program then prints out a running count together with the contents of location 022F hex. It will be seen that this latter changes after the $\overline{IRQ}$ line has been brought low using the push button. It will of course not simply increment by one, since during the finite time that the button is pressed, the CPU will complete many cycles through the interrupt service routine. It will also be noted that during the time that the button is pressed, the BASIC program will pause, resuming operation when it is released.

In more usual interrupt applications, the interrupt would be initiated by a device such as a 6821 or 6522 which can be instructed to take the $\overline{IRQ}$ line high again as soon as the CPU acknowledges the interrupt. This prevents the CPU getting involved in an endless loop of interrupts, and at the same time clears the way for fresh interrupt requests. This is the case with the interrupt driven clock to be described in the next issue.

Table 6.11. Table of data for IRQ interrupt test

Address	Data	Function
01C0	4C	Replace Jump
01C1	30	Instruction at
01C2	02	IRQ Vector
0228	58	Clear
0229	60	Interrupt Flag
0230	48	
0231	8A	
0232	48	
0233	EE	Interrupt
0234	2F	Service
0235	02	Routine
0236	68	
0237	AA	
0238	68	
0239	40	

```
80 REM INTERFACING UK101 PROGRAM 18
90 REM BASIC PROGRAM FOR USE WITH
92 REM MACHINE CODE ROUTINES TO TEST
94 REM UK101 IRQ INTERRUPT
100 FORA=1TO16:PRINT:NEXT
105 PRINT,"IRQ TEST PROGRAM"
106 PRINT,"FOR USE WITH MACHINE CODE"
107 PRINT,"PROGRAM"
108 PRINT:PRINT"IF UK101 HANGS UP, CHECK MACHINE CODE"
120 POKE11,40
130 POKE12,2
160 X=USR(X)
162 PRINT"   IRQ MASK NOW DISABLED"
163 PRINT:PRINT
165 PRINT,"COUNT";"      022F"
166 PRINT
170 PRINT,B;"       ";PEEK(559)
180 B=B+1
190 FORC=1TO500:NEXT
200 GOTO170
```

Table 6.12. Basic program for interrupt test

NEXT MONTH, as well as the interrupt driven clock, we will cover analogue to digital conversion, and discuss applications of the 8 channel A/D converter on the UK101 Analogue Board.

BIBLIOGRAPHY

6522 Data Sheet.
L. Leventhal, 6502 Assembly Language Programming.
R. Zaks, 6502 Applications Book.
R. Zaks, Programming the 6502.

Interfacing COMPUKIT

Part 7 D.E.Graham

THIS MONTH we conclude the series with discussion of an Interrupt Driven Clock, and with applications of the Analogue to Digital Converter section of the *P.E.* Analogue Board. These include voltage, temperature, light and sound measurement with v.d.u. bar chart displays, plus a data logger and sampling oscilloscope. But first we continue the discussion of the use of interrupts, begun last month.

6522 INTERRUPTS

The interrupt lines of the 6821 on the Decoding Module and the 6522 on the Analogue Board have both been connected to the UK101's $\overline{IRQ}$ line so that interrupts from both these devices may be readily handled. Unfortunately space has not allowed a discussion of the use of 6821 interrupts during this series. The principles involved however are similar to those for 6522 interrupts, and for further information on the 6821 the reader is referred to the data sheets.

6522 interrupts are handled through two sets of registers, the Interrupt Flag Register or IFR (register 13 at 61357) and the Interrupt Enable Register or IER (register 14 at 61358). These are represented in Table 6.3 of part 6 of the series. From this it will be seen that one bit of the IFR and one bit of the IER are associated with each of the control lines CA1, CA2, CB1 and CB2, and one with each timer and the shift register.

The various bits of the IFR are automatically set by the 6522 when events such as the timing out of the timers, or particular transitions on the peripheral control lines occur, and these flags may be inspected periodically by the CPU. If the interrupt is used however, such continual polling is not necessary, and the CPU need only inspect the IFR once an interrupt has been sensed and accepted. The IFR will then tell the CPU which one of the 7 possible functions caused the interrupt.

To condition the 6522 to cause an interrupt request (ie by taking $\overline{IRQ}$ low) when any of its flags are set is achieved by previously setting the corresponding bit in the Interrupt Enable Register. This register uses bit 7 to indicate whether specified bits are to be set or unset. If bit 7 is a one, then writing to any other bit of the register will set that bit. Thus, placing 130 (=128+2) into the IER codes bit 7 with a one, so that the remaining data (2) will cause bit 1 to be set. No other bit positions will be affected. Unsetting bits in the IER is accomplished with bit 7 at zero, so that writing 65 to the IER will unset bits 6 and 0, again leaving the remainder unchanged. To clear the register completely the IER should be loaded with 127. Pressing the Reset button on the Decoding Module will also zero this register.

INTERRUPT DRIVEN CLOCK

As an example of the use of interrupts with the 6522 we will look at the implementation of an interrupt driven clock on the UK101.

This will employ timer T1 on the 6522 to produce an interrupt every 50ms, and the UK101 will be conditioned to respond to this by updating a series of memory locations in which it holds the current time in hours, minutes, seconds, and twentieths of a second.

Table 7.1 gives the assembler listing of a program which accomplishes this. It resides at 0230 hex in an unused area of RAM below the UK101's BASIC file space. The program falls into two parts: the interrupt service routine itself, from 0230 to 0278 hex; and the initialisation routine from 027D to 02A6 hex. This latter is executed only once at the outset, and we will look at this first.

The first five lines set a jump to 0230 instruction at the UK101's IRQ vector. Line 780 initialises the memory location TWENT that is used to count twentieths of a second. The next sequence is used to configure the 6522. Lines 790 and 800 set timer T1 in the continuous mode by placing 64 into the ACR. Lines 810 and 820 enable the T1 interrupt by loading the IER with 192 (=128+64), thus setting bit 6. The instruction CLI at line 830 clears the interrupt mask, while lines 840 to 870 load the low and high order bytes of the T1 latches with the values 78 and 195, so initiating the count. The count total is thus 195 × 256 + 78 (=49998). The 2 micro-secs delay inherent in the counter make this up to 50000, so giving a 50ms interrupt repetition rate. The final command executes a return from subroutine.

The interrupt service routine starting at 0230 first places the accumulator on the stack, and executes a read from the low byte of T1's count registers. This latter has the effect of automatically clearing the T1 interrupt flag, so taking $\overline{IRQ}$ high again in preparation for the reception of further interrupt requests. The program then decrements the location TWENT (at 022A hex), and checks to see whether it has reached zero. If not it restores the accumulator from the stack, and exits the interrupt routine. If zero has been reached on the other hand, it increments the location SECS (at 022B hex), checking to see whether 60 seconds have been counted, in which case it increments MINS, and so on.

To start the clock once the program has been entered, it is only necessary to execute the subroutine at 027D. This may be accomplished from BASIC using a USR(X) call, or from a machine code routine with a JSR command. We will access it from BASIC.

RUNNING THE INTERRUPT CLOCK FROM BASIC

The BASIC program listed in Table 7.2 consists of two parts. The first loads the clock program of Table 7.1. The second allows the current time to be inserted in the three locations SECS, MINS and HRS, and uses a USR(X) call to enter the clock initialisation routine, thus starting the clock. Lines 3000 onwards then cause the time to be POKEd to the centre of the screen. Line 2100 determines the screen

position of the display, and this should be altered for Superboard use.

The clock display program may be exited by pressing the space bar at any time, and as long as the UK101 Reset is not pressed, the interrupt clock will continue to operate. Other programs may then be loaded, run and saved as required without disrupting the clock, and these may if desired access the continually updated time by PEEKing locations 555, 556 and 557 for seconds, minutes and hours respectively. There seems to be just one proviso to this: programs which use more than the simplest configuration of GOSUB commands must be avoided, since due to an error in the UK101's firmware the IRQ vector has been located within the stack. As a result, subroutine calls can corrupt the interrupt vector at 01CO, and are in turn corrupted by it. It is for this reason that the program of Table 7.2 avoids subroutine calls, and makes use of flags F and F1 to get around the problems which this creates.

Once the clock has been started, the whole program may be erased with a NEW command, if desired, so as to save memory space, again without affecting the running of the clock.

As with other applications of the 6522, the accuracy of the interrupt clock is tied to that of the UK101's 8 MHz crystal. In my machine this caused the clock to run slow by about 2 secs every hour. This could of course be corrected by using a closer tolerance crystal. But a much simpler way is to alter the value loaded into the low byte of T1's latch register—this is the last data item (78) on line 210 of the program of Table 7.2. With this method it should be possible to achieve accuracies of the order of 1 sec in 16 hours, which should be sufficient for most foreseeable applications.

MACHINE CODE TO BASIC TRANSLATOR

Lines 100 to 340 of the program on Table 7.2, which enter the interrupt clock program in 6502 code using BASIC, were themselves produced by a translator program written for the purpose. Since this program can translate the contents of any section of memory into BASIC POKE statements, it can be used for saving data or programs stored anywhere in the machine. The program is listed in Table 7.3. It first requests the start and end address of the memory block to be recorded. These may be in hex or decimal, since a hex to decimal conversion routine is included in the program. Next it requests the line number that the BASIC program which it will write is to start from, and the title of the program. It then instructs the cassette recorder to be set to record, and writes the program directly on to it. The machine may then be cleared, and the cassette loaded and run as if it were a normal program.

ANALOGUE TO DIGITAL CONVERTER

The fourth section of the Analogue Board consists of a multiplexed 8 channel analogue to digital converter. The heart of this unit is a ZN427 monolithic A/D converter i.c. See Fig. 7.1. This uses the so-called successive approximation technique to achieve conversion times of some 20µs or less. This is a far more efficient means of conversion than the other commonly used method which simply causes a D/A converter to sequence its output from zero to 255, using a comparator to stop it when the output matches the analogue signal to be measured. In this case conversion may take as long as 255 clock cycles.

Fig. 7.2 gives the circuit of the A/D section of the Analogue Board. The 8 analogue inputs are taken through SK6, whose full pinout is given in Fig. 7.3. From here they are applied to IC8, a 4051 8 way analogue switch. This selects one of the 8 channels according to the logic states of its pins 6, 9, 10 and 11. These in turn are determined by the

```
 80 0000          ;INTERRUPT CLOCK AT 022B, C & D
 90 0000          START=$0230
100 0230          *=START
110 0230          TWENT=START-6
120 0230          SECS=START-5
130 0230          MINS=START-4
140 0230          HRS=START-3
150 0230          ACR=61355
160 0230          IER=61358
170 0230          TL=61348
180 0230          TH=61349
200 0230 48       BEGIN PHA ;INTERRUPT SERVICE ROUTINE
220 0231 ADA4EF   LDA TL
230 0234 CE2A02   DEC TWENT
240 0237 D038     BNE OUT
250 0239 A914     LDA #20
260 023B 8D2A02   STA TWENT
270 023E A901     LDA #1
280 0240 18       CLC
290 0241 6D2B02   ADC SECS
300 0244 8D2B02   STA SECS
310 0247 C93C     CMP #60
320 0249 D029     BNE OUT
330 024B A900     LDA #0
340 024D 8D2B02   STA SECS
350 0250 A901     LDA #1
360 0252 18       CLC
370 0253 6D2C02   ADC MINS
380 0256 8D2C02   STA MINS
390 0259 C93C     CMP #60
400 025B D017     BNE OUT
410 025D A900     LDA #0
420 025F 8D2C02   STA MINS
430 0262 A901     LDA #1
440 0264 18       CLC
450 0265 6D2D02   ADC HRS
460 0268 8D2D02   STA HRS
470 026B C918     CMP #24
480 026D D005     BNE OUT
490 026F A900     LDA #0
500 0271 8D2D02   STA HRS
510 0274 EA       OUT NOP
520 0275 EA       NOP
530 0276 EA       NOP
540 0277 68       PLA
550 0278 40       RTI
```

Table 7.1. Assembler Listing of Interrupt Clock

```
560 0279 00       BRK
570 027A 00       BRK
580 027B 00       BRK
590 027C 00       BRK
700 027D          ;INITIALISE CLOCK
710 027D A94C     LDA #$4C
720 027F 8DC001   STA $01C0
730 0282 A930     LDA #$30
740 0284 8DC101   STA $01C1
750 0287 A902     LDA #$02
760 0289 8DC201   STA $01C2
770 028C A914     LDA #20
780 028E 8D2A02   STA TWENT
790 0291 A940     LDA #64
800 0293 8DABEF   STA ACR
810 0296 A9C0     LDA #192
820 0298 8DAEEF   STA IER
830 029B 58       CLI
840 029C A94E     LDA #$4E
850 029E 8DA4EF   STA TL
860 02A1 A9C3     LDA #$C3
870 02A3 8DA5EF   STA TH
880 02A6 60       RTS
```

```
OK
LIST

 90 REM INTERFACING UK101 PROGRAM 19
100 REM 6522 INTERRUPT CLOCK AT 0230 HEX
105 REM LOCATIONS  560  TO  678
110 DATA 72 , 173 , 164 , 239 , 206 , 42 , 2 , 208 , 59 , 169
120 DATA 20 , 141 , 42 , 2 , 169 , 1 , 24 , 109 , 43 , 2
130 DATA 141 , 43 , 2 , 201 , 60 , 208 , 41 , 169 , 0 , 141
140 DATA 43 , 2 , 169 , 1 , 24 , 109 , 44 , 2 , 141 , 44
150 DATA 2 , 201 , 60 , 208 , 23 , 169 , 0 , 141 , 44 , 2
160 DATA 169 , 1 , 24 , 109 , 45 , 2 , 141 , 45 , 2 , 201
170 DATA 24 , 208 , 5 , 169 , 0 , 141 , 45 , 2 , 234 , 234
180 DATA 234 , 104 , 64 , 0 , 0 , 0 , 0 , 169 , 76 , 141
190 DATA 192 , 1 , 169 , 48 , 141 , 193 , 1 , 169 , 2 , 141
200 DATA 194 , 1 , 169 , 20 , 141 , 42 , 2 , 169 , 64 , 141
210 DATA 171 , 239 , 169 , 192 , 141 , 174 , 239 , 88 , 169 , 78
220 DATA 141 , 164 , 239 , 169 , 195 , 141 , 165 , 239 , 96
310 FORC=0 TO  118
320 READ I
330 POKE  560 +C, I
340 NEXT
2000 PRINT:PRINT:PRINT:PRINT:PRINT
2005 PRINT:PRINT,"INTERRUPT CLOCK"
2007 PRINT:PRINT:PRINT"  ENTER PRESENT TIME"
2010 INPUT"    HOURS";H
2020 INPUT"    MINUTES";M
2030 INPUT"    SECS";S
2035 POKE557,H
2040 POKE556,M
2045 POKE555,S
2050 POKE11,125
2060 POKE12,2
2070 X=USR(X)
2100 Z=53660
2110 FORQ2=1TO16:PRINT:NEXT
2200 REM MAIN CLOCK LOOP
2230 IFF1=0THEN3000
2240 F1=0
2250 IFF=2THEN6000
2260 GOTO4000
3000 REM CLOCK DISPLAY
3020 H=PEEK(557)
3030 M=PEEK(556)
3040 S=PEEK(555)
3100 H$=STR$(H)
3120 M$=STR$(M)
3140 S$=STR$(S)
3160 T$="  "+H$+":"+M$+":"+S$+"   "
3200 REM POKE ROUTINE
3210 FORI=1TOLEN(T$)
3220 Q=ASC(MID$(T$,I,1))
3230 POKEZ+I,Q
3240 NEXT
3500 REM SPACE BAR SCAN
3510 POKE530,1
3520 POKE57088,253
3530 IFPEEK(57088)=239THENF1=1
3540 POKE530,0
3545 IFF=2THEN5550
3550 GOTO2200
4000 PRINT:PRINT:PRINT"  DISPLAY ROUTINE EXITED"
4010 PRINT"  CLOCK CONTINUES"
4020 PRINT"  TO RESUME DISPLAY, RUN 2100
4030 POKE530,0
OK
```

Table 7.2. Interrupt Clock in BASIC

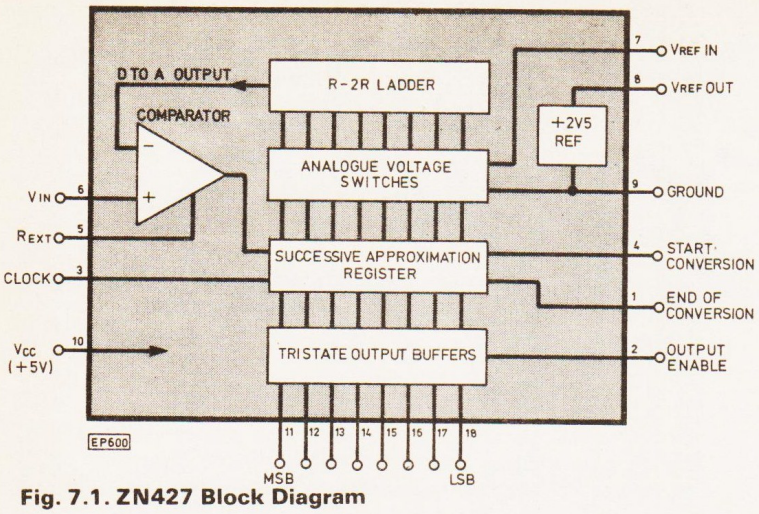


Fig. 7.1. ZN427 Block Diagram

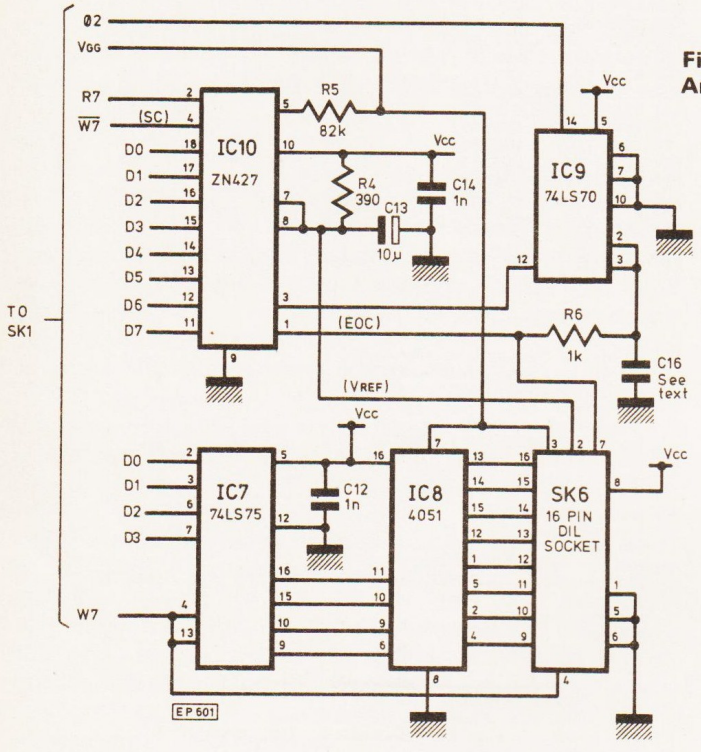


Fig. 7.2. A/D section of Analogue Board

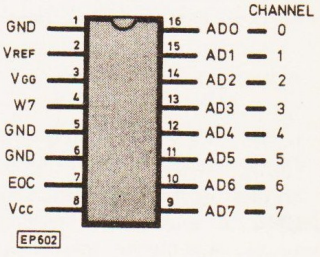


Fig. 7.3. Connections to SK6 on Analogue Board

quad latch IC7. This is enabled by line W7 from the Decoding Module. The 8 channels are selected by POKEing the channel number (0-7) to address 61319. Thus the command POKE 61319, 6 will connect channel 6 (at pin 10 of SK6) to the input of the converter.

The converter itself requires a low-enable Start Conversion pulse, and this is supplied by $\overline{W7}$ from the Decoding Module. This means that each time the multiplexer at 61319 is addressed, a new conversion sequence is initiated.

The clock input of the converter at pin 3 requires a frequency less than 600 kHz. In order to satisfy this condition, IC9 is used to divide the UK101 Ø2 clock by two, giving a clocking frequency of 500 kHz. Other timing conditions of the ZN427 are satisfied by delaying the operation of the divide by two counter until the ZN427 End of Conversion signal (EOC) goes negative. This is achieved by taking pin 1 of IC10 to pins 2 and 3 of IC9. In the prototype, this was further delayed by the insertion of C16 (50nF); and this capacitor was included in the component overlay diagram published in part 4 of the series. This extra delay time is not necessary for any of the applications described below, and it is recommended that C16 be removed from the board, so greatly enhancing conversion times.

The EOC signal produced by the ZN427 to indicate the completion of conversion has been taken to pin 7 of SK6, and may be monitored by the CPU if required, although conversion is so fast that there is barely time to monitor it in machine code (and certainly not in BASIC) before conversion is completed. Once conversion has taken place, the converter may be read at 61319. Thus the following two commands will print the value of the analogue input on channel 4.

POKE 61319, 4
PRINT PEEK(61319)

The POKE and PEEK statements may of course be put within a FOR loop to sequence through each of the 8 channels:

FOR A = 0 TO 7
POKE 61319, A
PRINT PEEK(61319)
NEXT

```
 50 REM INTERFACING UK101 PROGRAM 20
 60 REM MACHINE CODE TO BASIC TRANSLATOR
 70 PRINT:PRINT:PRINT:PRINT
 72 PRINTTAB(6)"MACHINE CODE TO BASIC TRANSLATOR"
 74 PRINT:PRINT
 75 PRINT"     THIS PROGRAM PRODUCES A TAPED PROGRAM"
 76 PRINT"     IN BASIC FOR REPRODUCING ANY BLOCK OF"
 77 PRINT"     MEMORY LOCATIONS"
 90 PRINT:PRINT:PRINT
 92 PRINT"     ARE START AND END ADDRESSES OF MEMORY"
 93 PRINT"     BLOCK IN HEX"
 95 INPUTY$
 96 IFLEFT$(Y$,1)="Y"THEN600
 98 PRINT
100 INPUT"    START ADDRESS IN MEMORY";AS
105 PRINT
110 INPUT"    END ADDRESS IN MEMORY";AE
120 INPUT"    FIRST LINE NO OF BASIC PROG";NB
130 PRINT"    TITLE OF BASIC PROGRAM"
135 INPUTA$
140 SAVE
150 PRINT"    START RECORDER"
160 INPUT"    ENTER S TO START";B$
200 PRINTNB; "REM ";A$
205 PRINTNB+5; "REM LOCATIONS ";AS;" TO ";AE
210 FORA=0TO(AE-AS)STEP10
220 PRINTNB+10+A;" DATA";
230 FORB=0TO9
240 PRINTPEEK(AS+A+B);
250 IFAS+A+5=AETHENB=10
254 IFB<9THENPRINT",";
260 NEXT
270 PRINT
280 NEXT
400 Z=NB+100+10*INT((AE-AS)/10)
410 PRINTZ;" FORC=0 TO ";AE-AS
420 PRINTZ+10;" READ I"
430 PRINTZ+20;" POKE ";AS;"+C, I"
440 PRINTZ+30;" NEXT"
450 FORZX=1TO5000:NEXT
500 POKE517,0
510 FORA=1TO16:PRINT:NEXT
520 PRINT,"RECORDING COMPLETE"
550 END
600 PRINT"   START ADDRESS (HEX) IN MEMORY"
620 GOSUB900
630 AS=N
635 PRINT
640 PRINT"   END ADDRESS (HEX) IN MEMORY"
650 GOSUB900
660 AE=N
670 GOTO120
900 REM HEX CONVERSION
910 INPUTQ$
920 IFLEN(Q$)<>4THENPRINT:PRINT"  4 DIGIT FORMAT ONLY":GOTO900
930 N=0
940 X$="0123456789ABCDEF"
950 FORJ=1TO4
960 FORI=1TO16
970 IFMID$(Q$,J,1)=MID$(X$,I,1)THEN1010
980 NEXTI
990 PRINT:PRINT"  CHARACTER NOT IDENTIFIED - REDO"
1000 GOTO900
1010 N=N+(I-1)*16^(4-J)
1020 NEXTJ
1030 PRINT"   DECIMAL EQUIVALENT = ";N
1050 RETURN
OK
```

Table 7.3. Machine Code to Basic Translator Program

The conversion scale of the ZN427 is determined by the voltage applied to its pin 7. On the Analogue Board this is connected to the ZN427's precision 2·55 volt reference source, which results in a very convenient conversion scale of 10 mV per digit, so that an input of 2·17 volts for example will produce a digital value of 217, and so on.

EIGHT CHANNEL DISPLAY PROGRAM

The program in Table 7·4 may be used to monitor the input levels on all 8 channels of the converter. Analogue to digital conversion is carried out in lines 220 and 230, and the updated conversion result POKEd to the screen, after suitable processing. As may be seen from the photograph, a horizontal bar graph representation is also produced for each channel. This latter is only altered in subsequent passes if a change in input voltage is detected. The program may be exited by depressing the space bar.

VOLTAGE MEASURING

As it stands, the converter will measure voltages from 0 to 2·55 volts presented to the input pins of its 8 channels (ie pins 16 to 9 of SK6, corresponding to channels 0 to 7 respectively). For accurate conversion, each input should be loaded with an impedance of 10k or less. This is necessary to ensure an impedance match with the input circuitry of the ZN427, and also in order to avoid charge carry-over from one channel to the next as the multiplexer sequences through the channels. If for example a voltage is applied to channel 0, and the input of channel 1 is left floating, channel 1 will read a non-zero voltage as measured in the display program of Table 7.4. This is avoided by keeping input loads below 10k and by earthing unused inputs.

If it is required to increase the voltage range of any channel of the converter, this may be accomplished by using a resistor network. That shown in Fig. 7.4 gives a scale factor of two so that 5 volts will give a reading of 250. The resistors used should be 1% types or better, although if these are not available, lower tolerance resistors could be trimmed (by constructing series/parallel combinations) so as to give a reading of 127 or 128 when point X is connected to pin 2 of SK6. This is the ZN427's own 2·55 volt reference source. The function of C in the dropper network is to keep the input impedance low enough to match the ZN427. Similar techniques may be used to extend the range further.

ANALOGUE JOYSTICK

Each of the 8 channels of the converter may be used with a virtually infinite variety of sensors. We will give circuits for use with some of the more obvious. Fig. 7.5 gives a circuit for a two-direction joystick. Each joystick potentiometer is supplied by the 2·55 volt reference source, and 100nF capacitors are again used to keep the input impedance low. As the circuit stands the two joysticks are connected to channels 0 and 1, and very straightforward software could be used to move a cursor or other character to any point of the screen in response to the position of the joystick. With the analogue use of the joystick, as opposed to the digital applications discussed in parts 3 and 5 of the series, the cursor could immediately move to the selected spot without having to traverse all intermediate screen positions first.

LIGHT SENSOR

Fig. 7.6 gives the circuit for an l.d.r. light sensor. With R1 at 1k this produces a good range of output, from very dim conditions (a v.d.u. screen at 1 metre) to brightly lit interiors. Higher precision in dark conditions may be achieved by increasing the value of R1, while improving the range in very bright conditions will be limited by the current drawn.

```
100 REM INTERFACING UK101 PROGRAM 21
110 REM 8 CHANNEL A/D CONVERTER DISPLAY
120 FORA=1TO8:PRINT:NEXT
130 PRINTTAB(10)"8 CHANNEL A/D CONVERTER"
140 PRINT:PRINTTAB(13)"(SPACE BAR EXITS)"
150 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
160 FORA=1TO3000:NEXT
170 FORA=1TO16:PRINT:NEXT
180 GOSUB 470
190 Z=53335
200 B=61319
210 FORA=0TO7
220 POKEB,A
230 N=PEEK(B)
240 S=Z+A*128
250 GOSUB 300
260 GOSUB 560
270 NEXT
280 GOSUB 410
290 GOTO210
300 REM NUMERICAL DISPLAY ENTRY POINT
310 REM SCREEN POKE ROUTINE
320 N$=STR$(N)
330 L=LEN(N$):REM STRING DISPLAY ENTRY POINT
340 FORM=1TOL
350 Q=ASC(MID$(N$,M,1))
360 POKES+M,Q
370 NEXT
380 IFPEEK(S+M)=32THEN 400
390 POKES+M,32:M=M+1:GOTO 380
400 RETURN
410 REM SPACE BAR MONITOR
420 POKE530,1
430 POKE57088,253
440 IFPEEK(57088)=239THENPOKE530,0:END
450 POKE530,0
460 RETURN
470 REM TEXT HANDLING
480 B$="CHANNEL"
490 Z=53323
500 FORA=0TO7
510 N$=B$+STR$(A)
520 S=Z+A*128
530 GOSUB 330
540 NEXT
550 RETURN
560 REM BAR DISPLAY CALCULATION
570 N1=INT(N/10+.5)
580 N2=N1-N(A)
590 IFN2=0THENN(A)=N1:RETURN
600 IFN2<0THEN 650
610 FORQ1=N(A)TON1-1
620 POKES+6+Q1,161
630 NEXT
640 GOTO 680
650 FORQ1=N(A)TON1STEP-1
660 POKES+6+Q1,32
670 NEXT
680 N(A)=N1
690 RETURN
OK
```

Table 7.4. Eight Channel A/D Converter Display

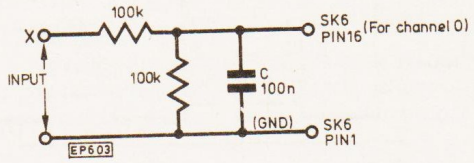


Fig. 7.4. Divide by two network

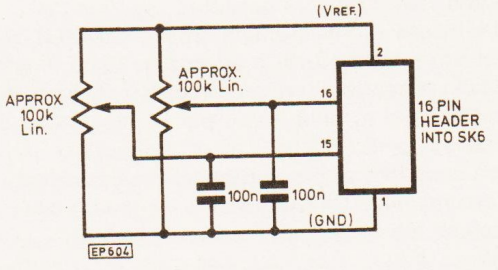


Fig. 7.5. Connection of 2 axis joystick to A/D converter

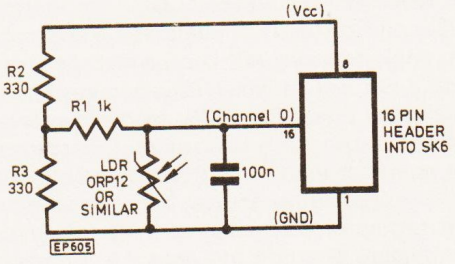


Fig. 7.6. Connection of light sensor to A/D converter

The left hand end of R1 could have been connected to the 2·55 volt reference source at pin 2 of SK6, but this is easily overloaded (maximum loading about 1k), and it is more prudent to use the pair of dropper resistors shown; or alternatively a 2·5 volt zener could be used. With the circuit of Fig. 7.6, quite small changes in light level may be discerned; and by using an l.d.r. on each channel, pattern recognition experiments may be undertaken. For such an application nine (or better still, sixteen) would be a more appropriate number of light sensors to use, in which case a second multiplexer could be added, either driven by one port of the PIA or VIA, or using a second 74LS75 latch, for which purpose the decoded line W7 has been taken out to pin 4 of SK6 on the Analogue Board. In such an application it should be noted that POKEing a one to bit 3 at 61319 renders the analogue switch IC8 open circuit.

TEMPERATURE SENSOR

Thermistors may be used for temperature measurement using a simple dropper resistor in the same manner as described for the l.d.r. application, although only fairly high resistance types are suitable, in order to avoid self-heating effects.

Far more precise measurement may be achieved for little extra outlay by using a temperature sensing diode such as the LM335Z. This is essentially a diode whose forward voltage is dependent on its temperature in a very linear manner, at the rate of 10mV per degree Kelvin. This means that its voltage increases by 10mV for every degree Centigrade above absolute zero (−273°C). Thus at 20°C its forward voltage will be 2·93. This voltage is unfortunately beyond the direct range of the A/D converter. But it is easily reduced with a resistor network such as that shown in Fig. 7.7. This will produce a reading of about 146 at 20°C.

To improve the reading precision, a d.c. amplifier with offset facility can be used between the sensor and the converter. Fig. 7.8 gives a circuit for such an arrangement. With this the voltage gain of the amplifier can be altered from unity to about 50 by adjusting VR1, and an offset of ±3 volts at maximum gain can be achieved by altering VR2. It should be noted however, that even at quite modest gain settings, the earth loop caused by using the UK101's mains transformer to drive the Decoding Module produces hum problems, and a small fluctuation appears on the reading. To avoid this, a separate 9-0-9 volt transformer should be used to power the Decoding Module.

To obtain the highest precision with the LM335Z (1°C error at room temperature) it should be used in conjunction with a 10k potentiometer as in Fig. 7.9, and the potentiometer adjusted to give an output as close to 2·982 volts as possible at 25°C.

The d.c. amplifier of Fig. 7.8 may also be used for increasing the sensitivity of the converter for use in other applications, such as d.c. voltage measurement etc.

AUDIO DETECTION

The circuit of Fig. 7.10 allows sound levels to be measured with the A/D converter. Like the temperature sensor amplifier of Fig. 7.8, it uses a 741 op. amp., but no earth loop hum problems arise in this case because no offset is required, and pin 2 can be taken down to earth. The values of R1 and C1 in the integrator following the diode detector circuit should be selected to give a time constant appropriate to the use to which the unit is to be put. Generally speaking this should be somewhat greater than the sample repetition time of the program driving the converter. In the case of the 8 channel display program of Table 7.4, a time constant of about one tenth of a second is appropriate, and may be

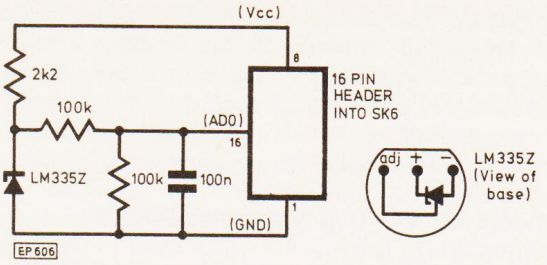


Fig. 7.7. Connection of Temperature Sensor to A/D converter

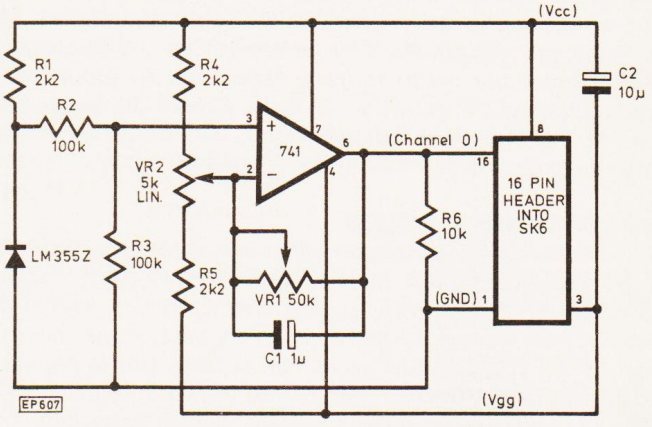


Fig. 7.8. Temperature Sensor with voltage amplifier

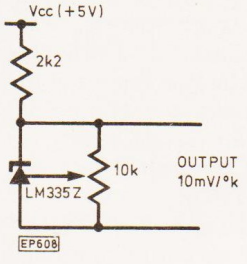


Fig. 7.9. Calibration of LM335Z temperature sensor

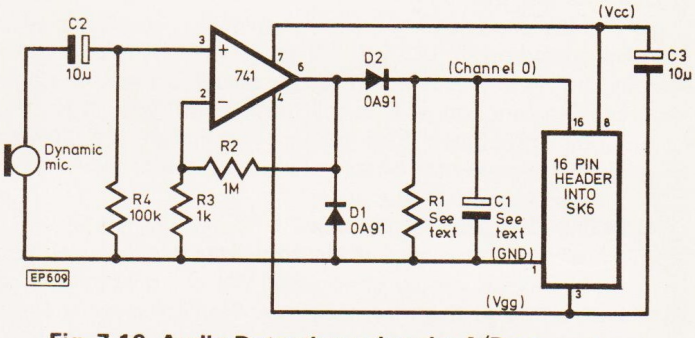


Fig. 7.10. Audio Detection using the A/D converter

achieved using 10k for R1 and 10µ for C1. For much higher sample repetition rates, such as those achievable with the sample 'scope program below, C1 may be removed completely for certain applications.

DATA LOGGER

The interrupt clock described earlier may be used in conjunction with the A/D converter to create a simple data logger. A program that accomplishes this is given in Table 7.5. To use it, load the interrupt clock program in BASIC of Table 7.2, then load the data logger supplement of Table 7.5. The latter makes use of routines in the former, including the interrupt clock itself, to implement the data logging function. When RUN is executed, the interrupt clock will operate as before, but now pressing the space bar will cause a transfer to the data logger. This requests a start time for logging in hours, minutes and seconds; then a repetition

```
4090 REM INTERFACING UK101 PROGRAM 22
4100 REM DATA LOGGER SUPPLEMENT
4110 REM TO BE LOADED ON TOP OF
4120 REM PROGRAM 19 (THE INTERRUPT
4130 REM CLOCK)
5000 REM DATA LOGGER
5020 F=2:Z=53276:NA=0
5100 FORQ2=1TO16:PRINT:NEXT
5110 PRINT,"DATA LOGGER"
5130 PRINT:PRINT:PRINT"   START TIME OF RECORDING"
5140 INPUT"   HOURS";SH
5142 INPUT"   MINUTES";SM
5144 INPUT"   SECONDS";SS
5150 PRINT:PRINT"   REPETITION PERIOD OF READINGS"
5160 INPUT"   HOURS";RH
5170 INPUT"   MINUTES";RM
5180 INPUT"   SECONDS";RS
5190 PRINT:PRINT"   NUMBER OF READINGS"
5200 INPUTNR
5210 DIMA(NR,8):DIMA$(NR)
5220 FORQ4=1TO16:PRINT:NEXT
5230 PRINT"  NO";TAB(11)"TIME";TAB(33)"DATA"
5500 GOTO2200
5550 REM ENTRY POINT
5560 IFS=SSANDM=SMANDH=SHTHEN5600
5570 GOTO2200
5600 REM DATA AQUISITION
5605 PRINT" ";NA+1;"..";
5610 PRINTT$;
5620 FORQ2=0TO7
5625 IFQ2=4THENQ3=-24:PRINT
5630 POKE61319,Q2
5640 A(NA,Q2)=PEEK(61319)
5650 PRINTTAB(23+6*Q2+Q3);
5660 PRINTA(NA,Q2);
5670 NEXT
5680 Q3=0
5690 PRINT:PRINT
5695 NA=NA+1:IFNA=NRTHEN6100
5700 CS=0:CM=0
5710 SS=SS+RS
5720 IFSS>59THENSS=SS-60:CS=1
5730 SM=SM+RM+CS
5740 IFSM>59THENSM=SM-60:CM=1
5750 SH=SH+RH+CM
5760 IFSH>=24THENSH=SH-24
5800 GOTO2200
6000 PRINT,"EXIT FROM DATA LOGGER"
6010 GOTO6110
6100 PRINT,"DATA LOGGING COMPLETE"
6110 PRINT,NA;"READINGS TAKEN"
6120 PRINTTAB(15)"TIME";T$
6125 PRINT:PRINT
6130 PRINT"TO DISPLAY CLOCK, RUN 2100"
6140 PRINT"TO RESTART LOGGER, RUN 5000"
```

Table 7.5. Data Logger Supplement

```
100 0000          ;SAMPLE SCOPE
110 0000          START=$0230
120 0230          *=START
130 0230          ;INPUT ROUTINE
140 0230 AE2E02 LDX START-2
150 0233 AD2F02 BB LDA START-1
160 0236 8D87EF STA 61319
170 0239 AC2D02 LDY START-3
180 023C 88       AA DEY
190 023D EA       NOP
200 023E EA       NOP
210 023F EA       NOP
220 0240 D0FA     BNE AA
230 0242 AD87EF LDA 61319
240 0245 9D9002 STA $0290,X
250 0248 CA       DEX
260 0249 D0E8     BNE BB
270 024B 60       RTS
280 024C 00       BRK
290 024D 00       BRK
300 024E 00       BRK
310 024F 00       BRK
400 0250          ;OUTPUT ROUTINE
410 0250 AE2E02 CC LDX START-2
420 0253 BD9002 DD LDA $0290,X
430 0256 8D88EF STA 61320
440 0259 AC2C02 LDY START-4
450 025C 88       EE DEY
460 025D EA       NOP
470 025E EA       NOP
480 025F EA       NOP
490 0260 D0FA     BNE EE
500 0262 CA       DEX
510 0263 D0EE     BNE DD
520 0265 4C5002 JMP CC
```

Table 7.6. Assembler Listing of Sample 'Scope

period in hours, minutes and seconds; and lastly the number of loggings required. This latter number sets the dimension of two arrays A$(I), which records the time of each reading, and A(I,J), which contains the 8 data values recorded at each reading.

The program then initiates the continued POKEing of the time to the centre of the top line of the screen, at the same time continually checking to see whether the start time has been reached. When it has, the A/D converter sequences through its 8 channels, and the resultant data is stored in the array for later use. The number of the reading, the time and the 8 data values are also printed on to the screen. See accompanying photograph. The program then waits for the duration of the repetition period, and again stores and prints the data measured. If hard copy or cassette recording of the data is required, an appropriate routine may be added at 6200 to be executed after data logging has been completed. And as with the interrupt clock itself, the space bar may be used to exit the data logger.

The logger program is highly flexible, and can take readings at intervals as short as 1 second, or as long as 24 hours, and the number of readings allowed is limited only by the memory size of the machine. A 4k machine will allow some 15 or 16 readings, while with 8k this rises to somewhere above 110. If memory size is a problem, a considerable saving could be made by storing the measured data individually in RAM by means of POKE statements to a protected area. Alternatively the program could be modified so as to print out, but not to store, the data, in which case there is no limit to the number of readings taken.

The uses to which the data logger may be put are legion. It could for example be used to monitor the temperature, noise, light and humidity levels in a given locality; and checks could be made on the effectiveness of thermal insulation by recording inside and outside temperature over a 24 hour period. It could also be used to gather data on the temperature dependence of a resistance, or to record the voltage decay curve of a CR combination of sufficiently long time constant. Additionally, the logger could be used in conjunction with the event recorder using T2 of the VIA described last month, to log the number of events of a given kind occurring within any period. Those events could variously be counts from a Geiger-Muller tube, passages across a photocell, the occurrence of sound levels above a given threshold etc. Once the data has been collected from whatever source, the UK101 can then be used to analyse it in any appropriate manner, and to present the results graphically if so desired.

WAVEFORM SAMPLING

The very fast conversion rate attainable with the ZN427 A/D converter—around 50,000 per second—allows the sampling and digital storage of waveforms at frequencies up to a few kHz. Using the converter a series of samples may be taken in very rapid succession, and stored in RAM, from where they may be analysed at leisure. For example, Fourier analysis might be performed on the data, or the sampled waveform could be replayed continuously through the D/A converter for steady display on an oscilloscope screen. Alternatively the UK101 VDU screen could be used to display the sampled waveform.

For such applications it is obviously essential to use machine code, since using BASIC would only allow some 100 or so samples to be taken every second. Table 7.6 gives an assembler listing of a two-part program which has a sampling and replay facility. The first part, which is located at 0230 hex, causes a series of samples to be taken and stored in RAM above 0290 hex. The sampling rate, the number of samples, and the channel number from which the samples are taken are determined by the contents of locations 022D, 022E and 022F respectively, and these may be set up before the program is run. The second part of the program, which begins at 0250 hex, allows the replay of the stored data in a continuous loop to the D/A converter located at 61320 dec. The number of samples to be replayed, and the replay rate, are determined by the contents of locations 022E and 022C respectively, and again these parameters may be set up before the routine is run.

A program in BASIC is given in Table 7.7 which loads the above program, and then controls the sampling and replay parameters with a series of INPUT statements. The program sets the channel number to 0, and the number of samples to 100 in lines 280 and 290. These may be altered if required, though 100 is the maximum number of samples allowed. It then requests a sample rate from 1 (fast) to 255 (slow), and after making the sample, requests a replay rate. It then replays through the D/A converter in a continuous mode, which can only be exited through a reset and warm-start.

The accompanying photographs show a 'scope trace of a 100Hz waveform that was sampled and replayed using this program. The original waveform is also shown. As may be seen from this, fidelity of reproduction is very good, even though no sample-and-hold circuitry has been used in the A/D conversion.

Sampling with this arrangement is fast enough for the lower end of the audio spectrum, and the circuit of Fig. 7.10 may be used in conjunction with the sampler to allow sampling of parts of speech signals and other audio waveforms.

As suggested above, the 100 data samples produced by the sampler may be analysed at leisure, or modified before replaying them through the D/A converter. If it is required to work on more than one set of samples, for comparison purposes for example, the 100 bytes of data may be transferred to another location using PEEK and POKE commands. The 100 bytes from 0290 hex would then be used simply as a temporary input buffer for the incoming data.

VDU SAMPLE SCOPE

If no suitable oscilloscope is available, or a larger display is required, the add-on program of Table 7.8 will allow for any 40 byte wide part of the sampled waveform to be displayed on the UK101 screen. When this program is loaded on top of the sample scope program of Table 7.7, it requests an offset factor to be entered (0-60) after the sampling routine has been executed, and POKEs up a display on the screen, using the characters 128-135 to achieve a vertical resolution of 128. The offset factor determines whereabouts along the 100-wide sample the display is taken from, and altering this moves the 40-byte wide sample window to left or right through the 100 bytes of data. If the space bar is depressed during the display, a new offset factor may be entered to change the position of the window. If a number greater than 60 is entered here, the program re-enters the sampling routine so that a new sample may be taken. The accompanying photograph shows the screen representation of the 100 Hz signal shown in the oscilloscope trace photographs referred to earlier. As may be seen, reproduction quality is quite good given the constraints imposed by the UK101's low resolution graphics.

CUSTOMISED WAVEFORM GENERATOR

The D/A section of the machine code routine referred to above may also be used on its own for the creation of customised waveforms. The 100 bytes of memory space above 0290 hex can be filled from BASIC using SIN, COS, LOG, RND, power or other functions of the user's choice. The replay routine in machine code may then be executed to generate the waveform repeatedly, and at speed, through the D/A converter at 61320.

The number of bytes of digitised waveform sent to the D/A converter before the sequence is repeated is determined by the data held in location 558 dec. This may be POKEd with any value from 1 to 100 so as to achieve a smooth follow on from the end of one displayed cycle of the waveform to the next.

Table 7.7. Sample Oscilloscope in BASIC

```
100 REM INTERFACING UK101 PROGRAM 23
110 REM SAMPLE SCOPE PROGRAM
120 GOSUB 360
130 REM
140 FORA=1TO16:PRINT:NEXT
150 PRINT,"SAMPLE SCOPE"
160 PRINT,"ON CHANNEL 0"
170 PRINT:PRINTQQ$
180 GOSUB270
190 POKE11,48:POKE12,2:X=USR(X)
200 PRINT:PRINT:PRINT"    SAMPLE MADE"
210 PRINT:PRINT
220 INPUT"    REPLAY RATE (1-255)";RR
230 PRINT:PRINT"    REPLAY IN PROGRESS"
240 PRINT"    USE RESET KEYS TO EXIT"
250 POKE556,RR
260 POKE11,80:POKE12,2:X=USR(X)
270 REM INITIALISATION
280 C=0:REM SET CHANNEL
290 N=100:REM SET NO OF SAMPLES (100 = MAX)
300 INPUT"    SAMPLE RATE 1 (HIGH) TO 255 (LOW)";R
310 IFR<1ORR>255THEN300
320 POKE559,C
330 POKE558,N
340 POKE557,R
350 RETURN
360 REM MACHINE CODE PROGRAM
370 REM LOCATIONS  560  TO  615
380 DATA 174 , 46 , 2 , 173 , 47 , 2 , 141 , 135 , 239 , 172
390 DATA 45 , 2 , 136 , 234 , 234 , 234 , 208 , 250 , 173 , 135
400 DATA 239 , 157 , 144 , 2 , 202 , 208 , 232 , 96 , 0 , 0
410 DATA 0 , 0 , 174 , 46 , 2 , 189 , 144 , 2 , 141 , 136
420 DATA 239 , 172 , 44 , 2 , 136 , 234 , 234 , 234 , 208 , 250
430 DATA 202 , 208 , 238 , 76 , 80 , 2
440 FORC=0 TO  55
450 READ I
460 POKE  560 +C, I
470 NEXT
480 RESTORE
490 RETURN
```

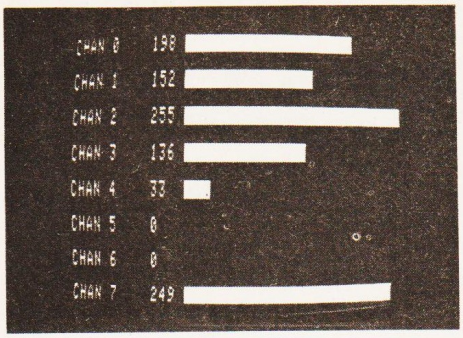
Table 7.8. Supplement to Sample Oscilloscope for Screen Display of Waveform

```
205 REM INTERFACING UK101 PROGRAM 24
206 REM GIVES SCREEN DISPLAY OF
207 REM SAMPLED WAVEFORM
208 REM LOAD ON TOP OF PROGRAM 23
210 GOTO7000
7000 REM SCREEN DISPLAY ROUTINE
7050 INPUTZZ
7060 IFZZ<0 OR ZZ>60THEN110
7070 ZZ=60-ZZ
7100 X=54222
7110 FORQ=1TO16:PRINT:NEXT
7120 S=656 +ZZ+40
7130 FORN=0TO40
7140 H=INT((PEEK(S-N))/2)
7150 H1=INT(H/8)
7160 H2=INT((H-H1*8))
7170 POKEN+X-64*H1,H2+128
7200 NEXT
7250 POKE530,1
7260 POKE57088,253
7270 IFPEEK(57088)<>239THEN7260
7280 POKE530,0
7290 GOTO7000
```

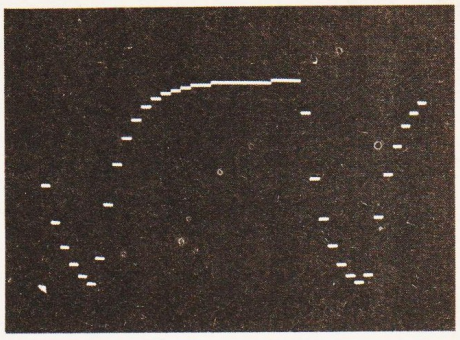
A NOTE ON THE USE OF NEW MONITORS

All programs in this series are fully compatible with the UK101's original monitor, and with Technomatic's CE1 monitor. But difficulties may arise with other new monitors when using the 3 or 4 programs that contain machine code routines located at 0230 hex, such as the Interrupt Clock and the Sample Scope program. The reason for this is that monitors such as Cegmon, Wemon and Compshop's new monitor use some of the space above 022A hex for scratchpad purposes. With Cegmon this space can be kept free by disabling the screen editor, and using the original screen handling routines. But to get around the problem more generally, an alternative set of programs have been prepared for the Interrupt Clock and Data Logger, and the Sample Scope, which relocate the machine code routines at the top of memory. There is no space to list these programs here, but they are included in the cassette containing the 24 numbered programs from this series, available from *Technomatic Ltd.*

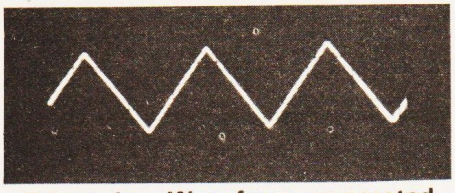
On Wemon and Compshop's new monitor, but not on the CE1 or Cegmon, the IRQ vector has also been relocated, and this must be accounted for in programs such as the Interrupt Clock and Data Logger, which make use of IRQ, by modifying lines 720, 740 and 760 of the assembler listing in Table 7.1. Further details on this modification are given in the data supplied with the cassette, since the shifted version of these programs must in any case be used with these two monitors.



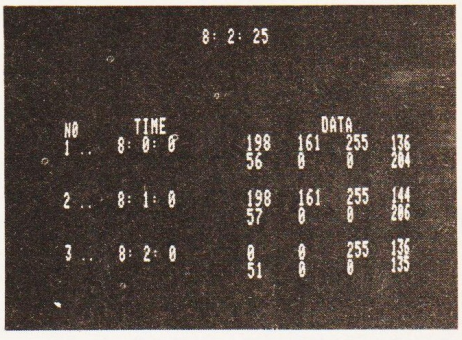
Screen Photograph of Eight Channel Display Program



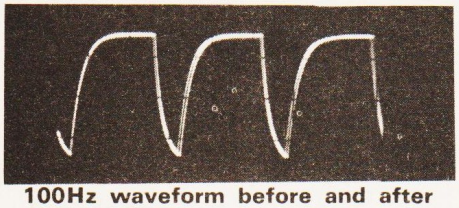
Screen display of sampled 100Hz waveform



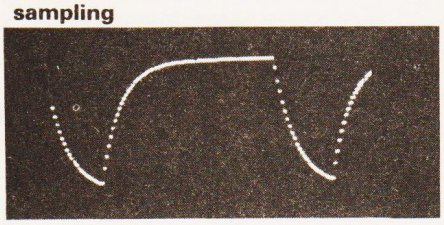
Triangular Waveform generated by the Customised Waveform Generator, and output through the D/A Converter



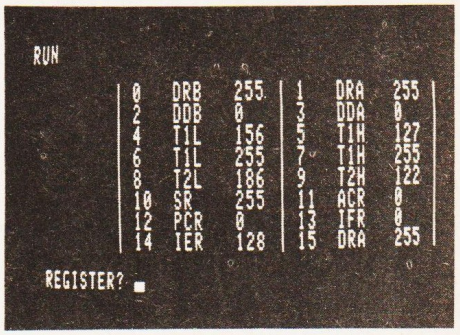
Screen Photograph of Data Logger



100Hz waveform before and after sampling



INTERFACING COMPUKIT



Screen photograph taken during the running of the 6522 Handling Program (Program 12) described last month

NEXT MONTH . . . we shall be publishing an **EPROM PROGRAMMER** which may be driven from this interface system.

Additionally, we will be taking a look at **Chromosonic's Colour Board** in use with the UK101

Constructor's Note

In order to improve stability in the offset adjustment on the D/A converter op amp IC11, R10 on the Analogue Board should be increased to 47k.

Filling the data space using the program below will produce an even triangular wave made up of exactly 100 samples:

```
2000 FORA=0TO99
2010 B=4×A
2020 IFA>49THENB=4×(99−A)
2030 POKE656+A,B
2040 NEXT
2050 GOTO300
```

The accompanying photograph shows the waveform produced using this program. As may be seen, its shape is good, and since it is constructed of a relatively large number of samples, its true step nature is not apparent in the photograph. It should be noted when creating other waveforms using the machine code replay sequence, that although the storage space for the waveform begins at 0290 hex, this location is the *last* rather than the *first* to be accessed, so that the *last* byte of waveforms placed into these locations for subsequent replay should be at 0290 hex, the first being at 0290+N-1, where N is the number of samples to be replayed.

CONCLUSION

Although this is the final article of the present series, it by no means exhausts the possible applications of the UK101 Decoding Module and Analogue Board. Further applications are planned for future issues of P.E., and it is hoped that ideas will also be contributed by UK101 and Superboard users through the columns of P.E.'s MicroPrompt. ★

The new MAPLIN CATALOGUE is out on December 5th

On sale in all branches of W H Smith from Dec 5th price £1

A massive new catalogue from Maplin that's bigger and better than ever before. If you ever buy electronic components this is the one catalogue you must not be without. Over 300 pages, it's a comprehensive guide to electronic components with thousands of photographs and illustrations and page after page of invaluable data. We stock just about every useful component you can think of. In fact, well over 5000 different lines, many of them hard to get from anywhere else. Hundreds and hundreds of fascinating new lines, more data, more pictures and a new layout to help you find things more quickly.

PLUS LOW PRICES
such as
Resistors from 1½p
Multimeters from £4.25
Ni-Cads from £1.15
Stereo Headphones from £3.49
Radios from £2.95
Over 300 Books
21 Different ranges of capacitors, etc., etc., etc.,

MAPLIN

Maplin Electronic Supplies Ltd.
All mail to: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Telephone: Southend (0702) 554155. Sales (0702) 552911.
Shops:
159-161 King Street, Hammersmith, London W6. Telephone: (01) 748 0926.
284 London Road, Westcliff-on-Sea, Essex. Telephone: Southend (0702) 554000.
Both shops closed Mondays.

Post this coupon now for your copy of our 1981 catalogue price £1.

Please send me a copy of your 320 page catalogue. I enclose £1 (Plus 25p p&p). If I am not completely satisfied I may return the catalogue to you and have my money refunded. If you live outside the UK send £1.68 or 12 International Reply Coupons.

I enclose £1.25

Name _____

Address _____

PE. 1.81