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.

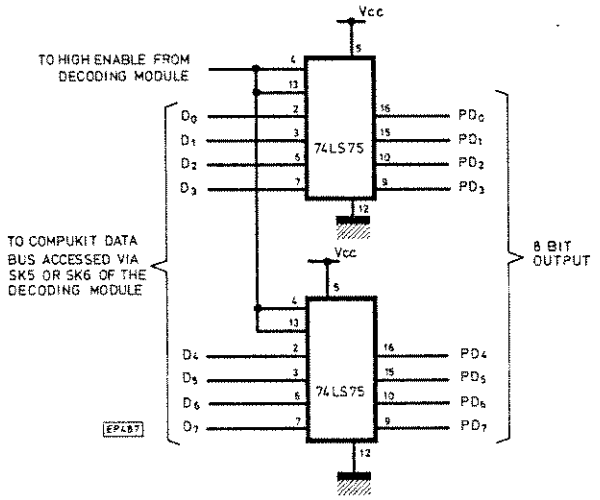


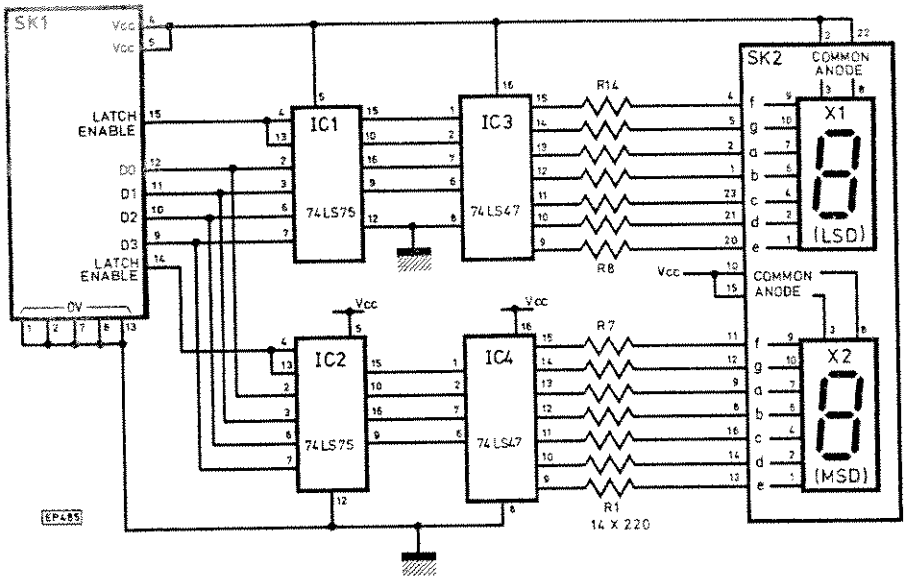
Fig. 3.1. 74LS75 8-bit Port

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.



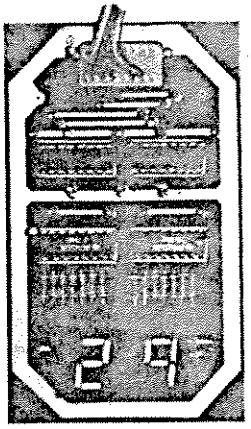
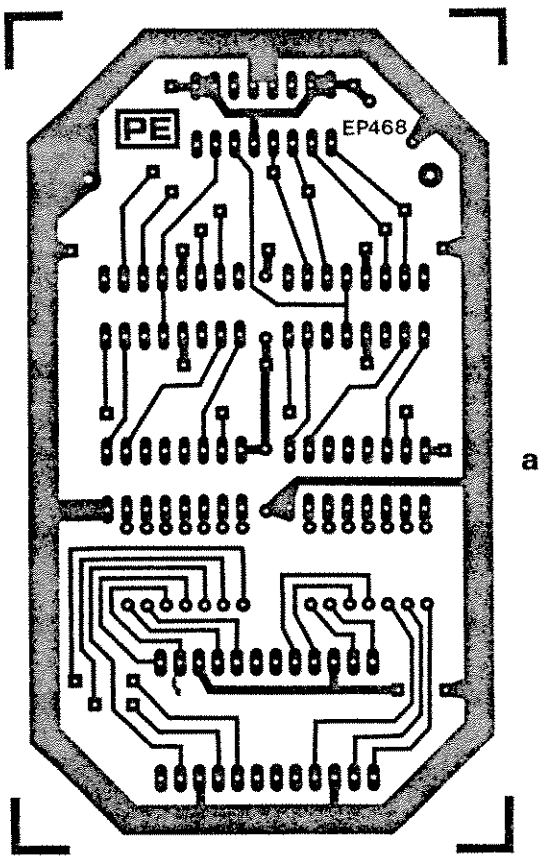


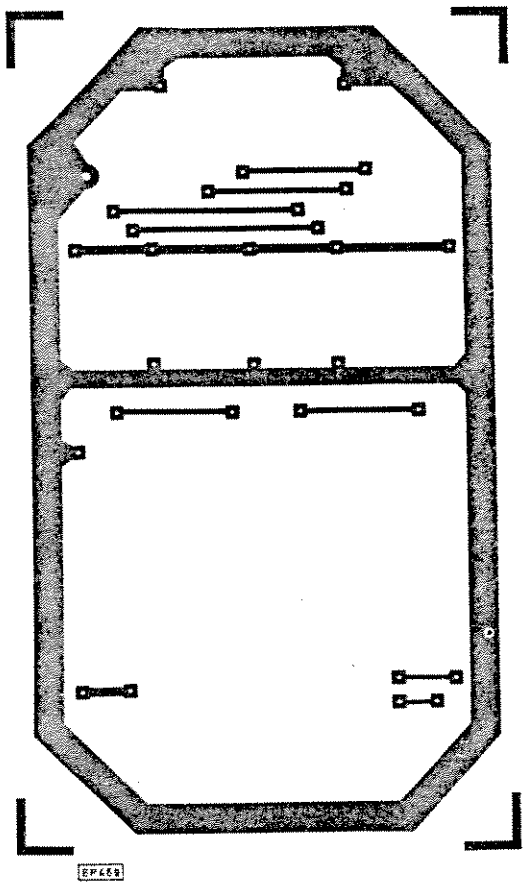
Fig. 3.2. 7-segment display circuit

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-

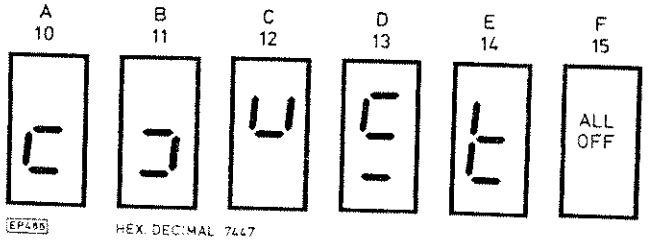
Table 3.1 Connections of Display Board to Decoding Module

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=1TO1000:NEXT
170 NEXT
180 GOTO120
```

Table 3.2. Seconds counting program

Table 3.3. 7447 symbols



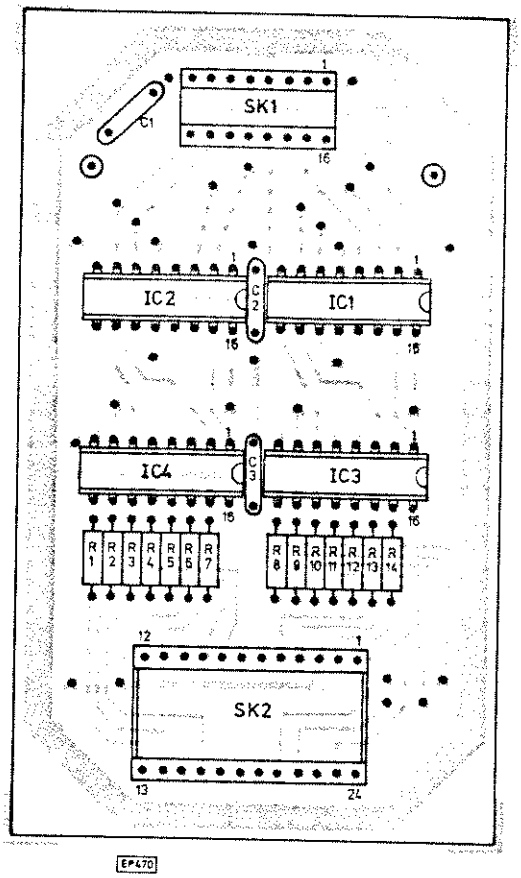


Fig. 3.4. Component layout for Display Unit

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 REM 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*10T(J+1)
420 V1=INT(VE/(10TJ))
430 VE=VE-V1*10TJ
440 V2=INT(VE/(10T(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

Fig. 3.5. Transistor amplifier for audio output

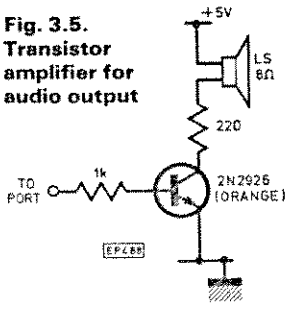
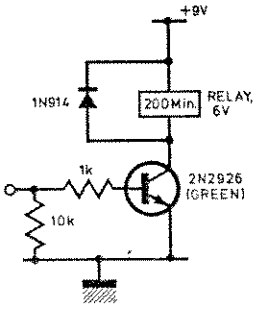


Fig. 3.6. Relay operation



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.

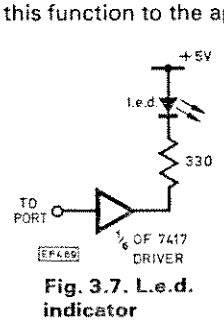


Fig. 3.7. L.e.d. indicator

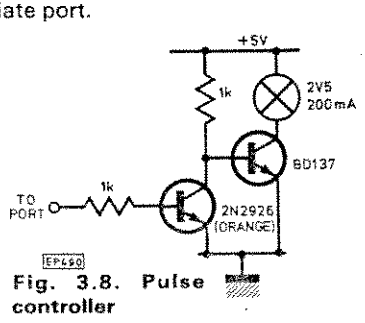
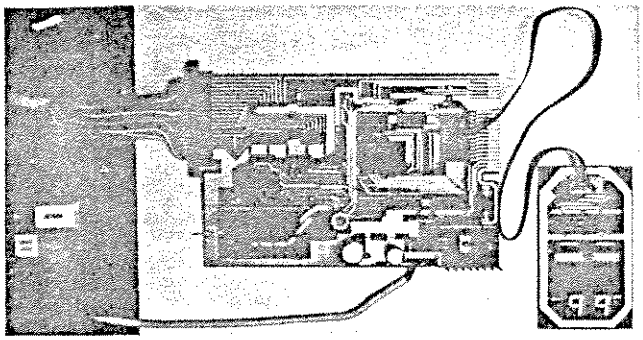


Fig. 3.8. Pulse controller



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.

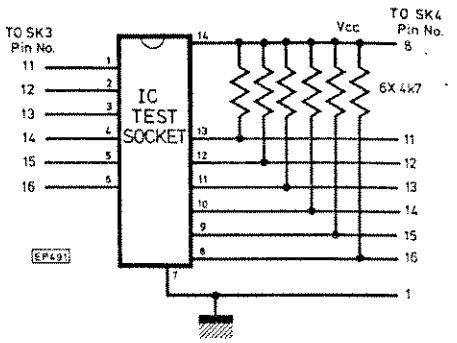


Fig. 3.9. I.c. tester connections

```
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)).

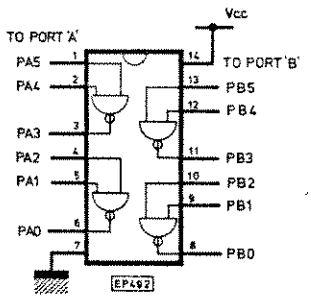


Fig. 3.10. 7400 pin-out for tester

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.

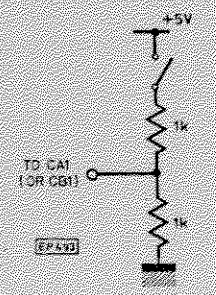


Fig. 3.11. Resistor chain to test flag

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.

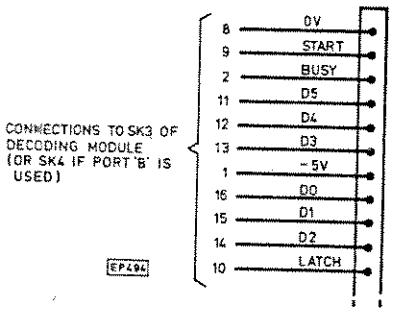


Fig. 3.12. Connections of speech board to SK3

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.

```
 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 INPUTA8
270 ONA8GOTO300,450,600,750
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)
2080 A3=A1AND128
2100 IFA3=0THEN2070
2110 FORA4=1TO100:NEXT
2120 RETURN
OK
```

Table 3.11. Speech handling program