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.

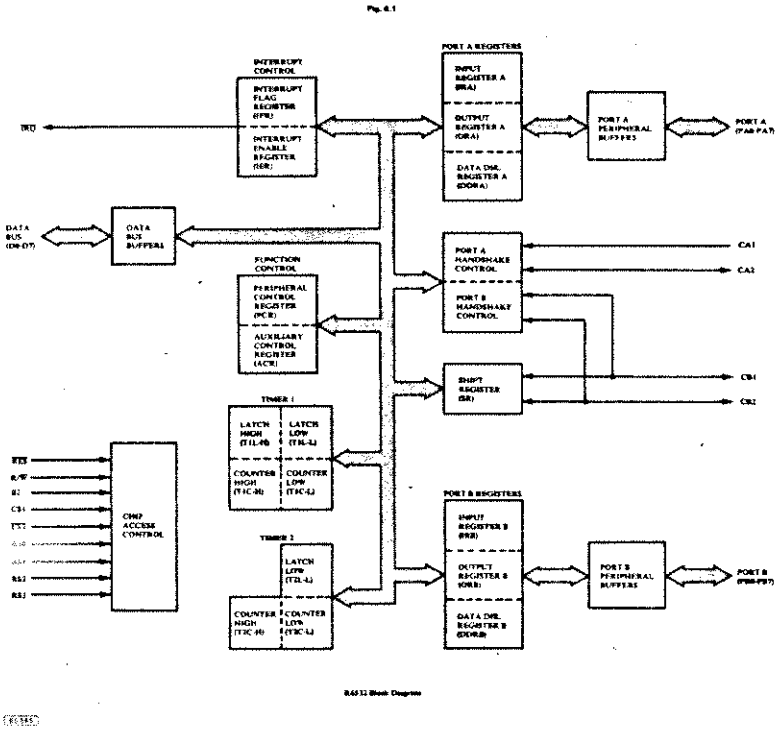


Fig. 6.1. 6522 Block diagram

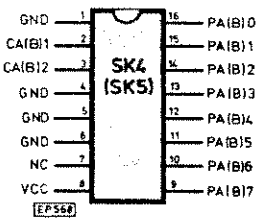


Fig. 6.3. Pin-outs of SK4 and SK5 of Analogue Board. SK4 carries Port A of VIA, SK5, Port B

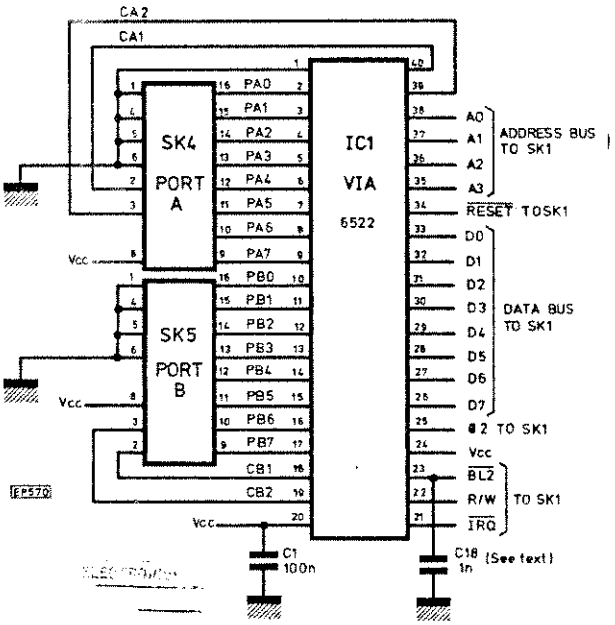


Fig. 6.2. 6522 section of the Analogue Board

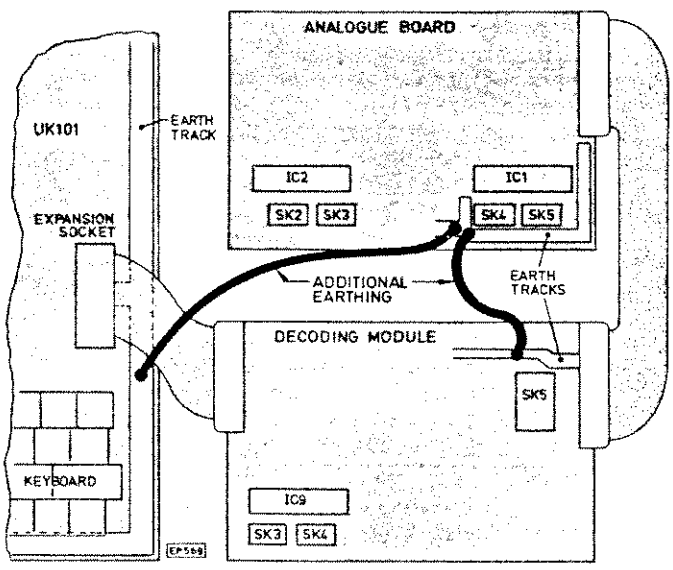


Fig. 6.4. Additional earthing leads to Analogue Board

Table 6.1. 6522 handling program

```
80 REM INTRFACING UK101 PROGRAM 12
85 REM 6522 HANDLING PROGRAM
90 DIMA$(15)
95 P=51*4
100 FORA=0TO15
110 READA$(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
```

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.

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

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

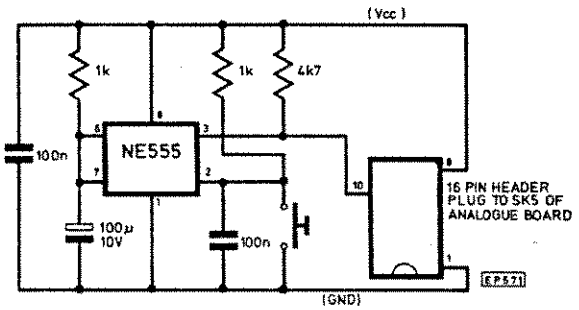
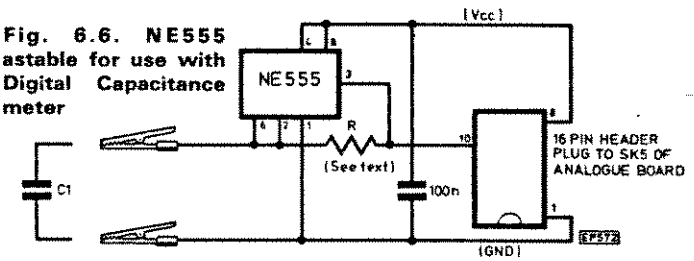


Fig. 6.6. NE555 astable for use with Digital Capacitance meter




```
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, Ø2and 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)"('F' PRINTS - SPACE BAR EXITS)"
95 PRINTTAB(15)"  ACCURACY +/- 20 HZ  "
96 PRINT: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$=STR$(N):IFN1=1THENN$=N$+" 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

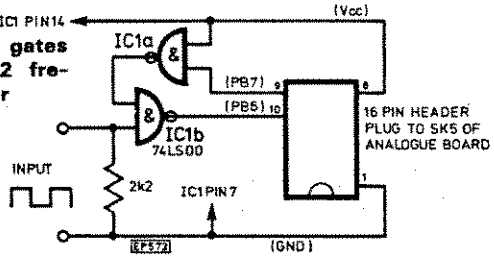


Fig. 6.7. Nand gates used in T1–T2 frequency counter

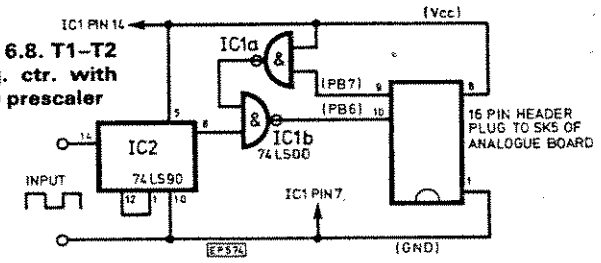


Fig. 6.8. T1–T2 freq. ctr. with ÷10 prescaler

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=1TO15: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.