Chapter Six
Interfacing

In the context of computing, the term interfacing is taken to mean connecting any type of external equipment - including printers, disc drives and tape recorders. In this chapter, however, the subject is restricted to considering the BBC Micro's A to D convertor and the user port. The presence of these two facilities within the BBC Micro make it suited for a wide range of 'serious' tasks such as controlling experiments and other machinery, and making measurements automatically. The BBC Micro is ideally for this sort of application; its graphics capabilities can be put to good use displaying results or the current state of the equipment being controlled; the high speed calculating power of the BASIC can be used to process the measurements, and even the sound generator can be used to process measurements, and even the sound generator can be used to draw attention to an abnormal condition! This more serious side of the BBC Micro isn't entirely devoid of lighthearted applications. The A to D convertor can be used to connect a pair of joysticks or 'paddles' that bring a whole new dimension to playing games! Both the user interface and the A to D convertor are fitted only to the Model B machine.

In the first part of this chapter we consider the A to D convertor. This is a fairly straightforward device from the programmer's point of view in that the MOS and BASIC provide commands to handle the device and it is fairly easy to use even from assembly language. The difficulty with the A to D convertor lies in the electronics that you connect it to rather than the programs that you write afterwards. Unless you know a little about practical electronics, that is, can use a soldering iron and recognise a resistor from a diode, then you would be well-advised to connect only off-the-shelf extras such as games paddles made by Acorn to the A to D convertor. The same is true for the user port, which also presents another level of difficulty in that the VIA which provides the user port is a very complex device. And to make matters worse, the BASIC and the MOS provide nothing to control it.

However, the effort spent in mastering it is certainly well worth it because it is extremely versatile.

The A to D convertor

Most of the things that we measure are *analog* in the sense that the results of the measurements range over a scale that is for all practical purposes continuous. Another way of thinking about this is to imagine the graph of the measurement with time. The graph of an analog quantity would be free to wander up and down without any restrictions. By contrast, the graph a *digital* quantity is restricted to a finite number of values and is, therefore, always subject to some limitation that causes it to progress in *jumps*. For example, the graph of the number of computers manufactured per month cannot go up by a fractional amount! Digital quantities are normally easy to measure using a computer. For example, in the case of the number of computers produced per month, all you would have to do is arrange for the computer to add one to a variable every time a computer was produced. The exact details of how the BBC Micro would be informed that another computer had been produced is part of the subject studied in the second half of this chapter - digital interfacing. Measuring analog quantities using a computer is slightly more difficult and really requires a special piece of hardware called an A to D convertor (Analog to Digital convertor). This takes an electrical signal and turns it into a number that can be read by the computer. Of course, to use an A to D convertor to measure any quantity, it must be first converted to an electrical signal that is proportional to the quantity. A device that carries out such a conversion is called a *transducer*. For example, a

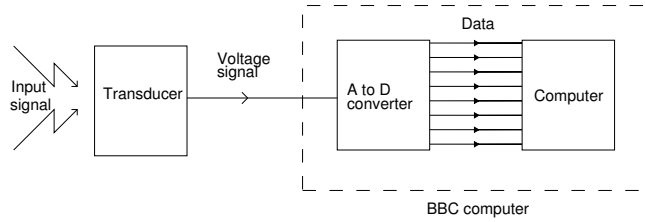

Fig. 6.1. Computer measuring system

photo cell, or a photo diode, is a transducer that converts light intensity to a voltage. Thus, a complete analog measuring system consists of three components - the computer, an A to D convertor and a transducer (see Figure 6.1). As the BBC Micro (Model B) comes equipped with an A to D convertor you can guess that most of our problems lie in the transducer. However, before considering transducers it is necessary to look at the capabilities and limitations of the BBC Micro's A to D convertor.

The A to D convertor used in the BBC Micro (a µPD7002) is a four channel 12-bit convertor. This means that it can select any one of four inputs to convert. However, it is important to notice that it can be engaged in converting only one of the four inputs at any one time! The '12-bit' part of the specification tells us how accurately the conversion is carried out. A 12-bit number lies in the range 0 to 4095 and this indicates how finely the A to D convertor divides up its input range. If the largest voltage that can be input to the A to D is Vmax then it is not difficult to see that the entire input range is divided into 4095 parts. This means that the smallest voltage change that can be detected is Vmax/4095 or roughly .02% of the maximum reading. When you compare this to the normal accuracy of 3% of maximum reading of conventional 'moving-needle' meters you will realise that 12 bits of accuracy is very acceptable. Unfortunately, you cannot suppose that just because the A to D convertor returns results accurate to 12 bits that all of them reflect the quantity being measured. Allowing for noise etc., 10 bits of accuracy is all you can expect unless you take special care. Even so, 10 bits gives a range of 0 to 1024 which is an accuracy of roughly .1% of the maximum reading. This is still very good.

Apart from the number of channels and their accuracy, A to D convertors also differ in how fast they can produce a result. The A to D convertor in the BBC Micro takes 10 ms per conversion. This is not very fast and limits the sort of application that the BBC Micro can be used for. If an input signal varies during this conversion time the final result will not adequately reflect the input signal. In other words, any important variations in the signal must occur over a time that is roughly twice as long as the conversion time. (To be precise a signal can only be digitised accurately by the BBC Micro if its highest frequency component is less than 50 Hz.) If all four channels are in use then the situation is even worse. As each channel takes 10 ms for a conversion and each channel is treated in turn, any one channel is read only once every 40 ms. Although the speed on conversion limits the

range of signals that can be converted accurately, it is perfectly fast enough for joystick signals, position measurement, temperature measurement etc.

A to D software

There are three commands that can be used to control the operation of the A to D convertor, ADVAL, *FX 16 and *FX 17. ADVAL is a BASIC function that can be used to find out the last reading from any of the A to D channels. For example:

```
X=ADVAL(2)
```

will store the most recent result from channel 2 (the channels are numbered from 1 to 4) in the variable X. The value that ADVAL actually returns is in fact the A to D value multiplied by 16. The reason for this strange action is that it allows for future improvement in A to D convertors to 16 bits accuracy. Thus the value returned by ADVAL varies over the range 0 to 65520 and changes in steps of 16. The reason why ADVAL returns the most recent value rather than the current value from the channel concerned is that in normal operation the A to D convertor is continually converting each channel in turn. At the end of each 10 ms conversion period the result is stored in memory ready for later use. Thus, whenever the ADVAL function is used it is the last value stored in memory that is returned as the result. This is a very sensible arrangement as it saves having to wait for the requested channel to have its turn in the sequence of conversion but it is as well to be aware of the fact that the value returned can be as much as 40 ms old. If you are using less than four channels then, obviously, converting all four in turn could be a waste of much-needed time. The MOS command *FX 16 can be used to remove any of the channels from the sequence of conversion.

```
*FX16,0no channel is converted
*FX16,1only channel 1 is converted
*FX16,2channels 1 and 2 are converted alternately
*FX16,3channels 1,2 and 3 are converted in turn
*FX16,4each channel is converted in turn
```

Notice that if you want to cut down the number of channels involved in conversion you must use the lower numbered channels

first. If you want an up-to-date reading from any channel then you can use *FX 17,' channelnumber' which will start a conversion on the channel corresponding to ' channelnumber' Of course, following this command you will have to wait for 10 ms until the conversion is complete but it is the only way to get a completely up-to-date measurement. The function ADVAL (or the MOS OSBYTE call with A=&80 which is equivalent) can be used to find out when a channel has completed conversion. ADVAL(0) DIV 256 will give the number of the last channel to complete conversion. If no channel has completed conversion, then 0 is returned. So, if you initiate conversion on a particular channel, you should wait until its number is returned by ADVAL(0) 256. To see this in action try:

```
10 *FX 17,1
20 PRINT ADVAL(0) DIV 256
30 GOTO 20
```

You should see two zeros printed by line 20 indicating that channel 1 is still converting and then the sequence 1,2,3,4 over and over again as the normal sequence of conversion carries on.

This is all there is to the software to control the A to D convertor. Even from machine code it is simpler to use the OSBYTE equivalent of ADVAL, *FX 16 and *FX 17 rather than try to write your own subroutines. However, the ADVAL function does a little more than just deal with the A to D convertor. There are two *fire button* inputs provided on the analog input connector at the back of the BBC Micro and, as these are intended to be used with joysticks, they are also handled by the ADVAL function. The fire button inputs are digital inputs and so would be better described in the second half of this chapter. However, as they are handled by ADVAL it is worth saying that they can be used to detect whether or not a switch (usually a push button) is open or closed. The switch should be connected between the fire button input and a 0 V line provided on the same connector. (See the next section for more hardware information.) The state of the switches can be read by using:

X=ADVAL(0) AND 3

which returns a number with the following meaning:

X= 0 no switch closed
1 left switch closed
2 right switch closed
3 both switches closed

The other uses of ADVAL (i.e. with a negative parameter) are very useful but have nothing to do with the A to D convertor.

Hardware for the A to D convertor

So far the discussion has been about using the A to D convertor from the software point of view. Before you can get to this stage, however, you have to have solved the problem of connecting a suitable transducer to the channel of your choice. The easiest way to do this is to buy an off-the-shelf transducer such as a pair of joysticks from Acorn. If you want to try something a little more adventurous then it is not difficult to connect your own transducers. The only thing that you need to know is that the voltage input to the A to D convertor must lie in the range 0 to 1.8 V. It is very important to keep within these limits because the A to D convertor chip itself is very easily damaged by input voltages outside this range. This warning sounds a little frightening but there is an output available from the analog connector - Vref (pins 11 to 14) - which is 1.8 V and if you use this to supply any transducers that you are using then you cannot possibly exceed the maximum input voltage. The voltage Vref is used by the A to D convertor as a *calibration* voltage. In other words, an input equal to Vref will give the maximum reading from the A to D convertor. Thus, to change the reading of the A to D into Volts, you need to measure the actual value of Vref (using a meter) accurately. Once you know Vref then the reading in volts is given by:

```
Volts=ADVAL(N)*Vref/65520
```

It is important to notice the accuracy of the A to D convertor depends on the accuracy and stability of Vref - which is not very good. If you are trying to make accurate measurement then it is better to connect a good calibration voltage to one of the channels and compare the reading on all the other channels with it.

The most simple and most common transducer is a variable resistor (potentiometer) used to convert the position of its spindle to a voltage. This is the principle behind most joysticks and paddles. You can see the circuit for a joystick or games paddle in Figure 6.2. The potentiometers are connected between Vref and analog ground. The voltage from the slider depends on its position and this is fed to the A to D convertor input. (10K potentiometers are used because they are large enough not to take too much current from Vref and yet small

enough not to be affected by the connection of the A to D convertor input, which has an input impedance of approximately 10 MW.) The small capacitors are simply to remove any spurious signals and their exact value is unimportant. To make a pair of joysticks you would have to make up two copies of the circuit and connect the first to the analog

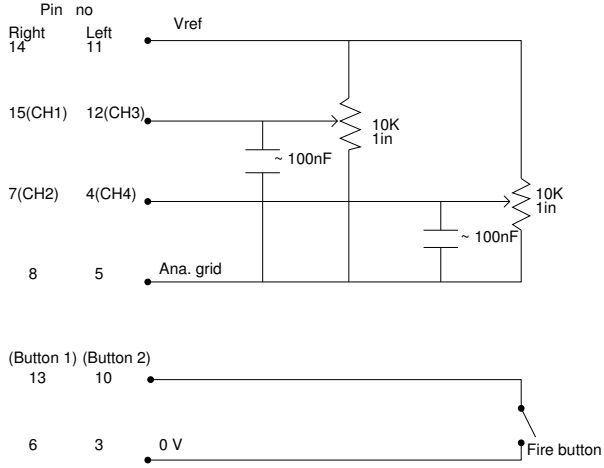

Fig. 6.2. Games paddles

input pin numbers shown under the heading ' right' and the other to the pin numbers listed under ' left' .

The major problem encountered when connecting transducers to an A to D convertor is getting the voltage range of the output of the transducer into the range that the A to D convertor requires. The problem of reducing a larger range of voltages is easily solved using a voltage divider (Figure 6.3). If a maximum voltage input is Vin then

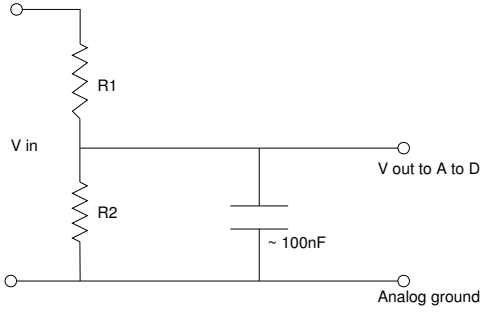

Fig. 6.3. A voltage divider.

the maximum voltage output (Vout) to the A to D is given by:

Vout=R2*Vin/(R1+R2)

Obviously, R1 and R2 should be chosen to keep the maximum input voltage to the A to D lower than Vref to avoid any damage. Also, the value of R1+R2 should be kept large to avoid taking too much current from the transducer. The definition of too much depends on the transducer, of course, but R1+R2 should be large compared to the output impedence of the transducer to avoid distortion. The opposite problem of increasing the size of small signals can be solved only by use of an amplifier, a topic which is beyond the scope of this book. For more information, see any book on practical operational amplifiers.

There are so many light sensors, either light sensitive diodes, transistors or resistors (including ones capable of detecting infra red), that is impossible to recommend any particular one for experimental purposes. You should, however, have no trouble finding one to suit your application. Other sensors that might prove interesting are the FGS7712 flammable gas detector, the RS304-431 liquid flow sensor and the 590KH temperature sensor (all available from RS Components Ltd.).

4F	A Input/Output (2)	6F
4E	Interrupt control	6E
4D	Interrupt status	6D
4C	Peripheral (handshake) control	6C
4B	Auxiliary (timer/shift reg) control	6B
4A	Shift register	6A
49	Timer 2 MSB counter	69
48	Timer 2 LSB latch/counter	68
47	Timer 1 MSB latch	67
46	Timer 1 LSB latch	66
45	Timer 1 MSB latch/counter	65
44	Timer 1 LSB latch/counter	64
43	Direction reg A	63
42	Direction reg B	62
41	A input/output (1)	61
FE40	B input/output	FE 60
VIA-A		VIA-B

Fig. 6.4. The VIA registers.

The user interface and the 6522 VIA

As explained in Chapter One, there are two 6522 VIAs inside the BBC Micro Model B. VIA-A is used for internal functions apart from the two fire button inputs on the analog connector. VIA-B is only fitted to the Model B. It provides the parallel printer interface and the user port. The 6522 VIA is a complex device that is controlled by a set of sixteen different registers (Figure 6.4). Instead of dealing with each register in turn it makes more sense to describe the three basic functions of the VIA - input/output, timing and the shift register - and give BASIC procedures to control each. If more speed is required in any application then the procedures are easy to convert into assembly language subroutines. For reference purposes, the pin connections for the user port are given below:

Table 6.1. Pin connections for the user port connector.

Pin	Function	Pin	Function
1	+5V	2	CB1
3	+5V	4	CB2
5	ground	6	PB0
7	. .	8	PB1
9	. .	10	PB2
11	. .	12	PB3
13	. .	14	PB4
15	. .	16	PB6
17	ground	18	PB7

Input/output lines

Every VIA has twenty input/output lines grouped into an A side and a B side of ten lines each. These ten lines are further divided into a group of eight data lines and two special handshake lines. Most of the complications lie in the use of the handshake lines so consideration of these is left until later. The eight data lines are called PA0 to PA7 on the A side of the port and PB0 to PB7 on the B side. In principle, any of the data lines can be set to an input or an output line using the appropriate data direction register (registers 2 and 3). However, the

BBC Micro uses the A side of VIA-B as a buffered output to drive the printer port so only the B side data lines can be used as inputs. To use a line as an output you have to store a one in the correct place in the appropriate data direction register. For example, to use PB3 as an output you must store a one in the b3 (bit 3) of the B data direction register. If you want to use a line as an output then you must store a zero in the same place. For example, the command ?&FE62=&F0 would set up PB0 to PB3 as inputs and PB4 to PB7 as outputs. The lines PA0 to PA7 are set up as outputs automatically by the MOS when the machine is switched on. The following BASIC procedures can be used to set any of the B side lines to outputs or inputs as desired.

```
100 DEF PROCIOSET(S$)
110 LOCAL I,S%
120 S%=0
130 FOR I=1 TO LEN(S$)
140 S%=S%+2^EVAL(MID$(S$,I,1))
150 NEXT I
160 ?&FE62=S%
170 ENDPROC
```

To set a line to an output, simply include its number in the string S$. For example, to set lines 3,5 and 7 to outputs, use PROCIOSET("357"). Lines 0,1,2,4 and 6 will be set to inputs by default.

To state of the output lines is controlled by writing to the A or B input/output registers. An output line can either by high i.e. 5 V, or low i.e. 0 V, and these states correspond to writing one and zero respectively to the same number bit in the input/output register. For example, ?&FE60=&01 sets PB0 high and PB7 low, because the assignment sets bit 0 of the B input/output register to one and all the other bits to zero. This method of controlling output lines is not too difficult to use but it would be easier to have a command that would set a particular line high or low without affecting any of the others. The following two BASIC procedures do just this:

```
200 DEF PROCON(N%)
210 IF N%<0 OR N%>7 THEN ENDPROC
220 N%=2^N%
230 ?&FE60=?&FE60 OR N%
240 ENDPROC

300 DEF PROCOFF(N%)
310 IF N%<0 OR N%>7 THEN ENDPROC
320 N%=2^N%
330 ?&FE60=?&FE60 AND (NOT N%)
```

```
340 ENDPROC
```

The procedure PROCON will turn any line on, i.e. high, without affecting the state of any other line. For example, PROCON(5) will set line 5 high and leave everything else as it was. PROCOFF does the opposite and sets the selected line low, again without affecting any of the other lines. The only lines in the above procedures that might need any explanation are lines 230 and 330. You will notice that the address of the input/output register occurs on both sides of the expression. This is because you can not only write to the input/output register, you can also read it to discover what the output lines are currently set to.

Using lines set to inputs is just as easy. If you read from the input/output register then the current state of any input line is reflected in the state of the bits with the same number. If the input line is high i.e. set to 5 V, then the bit will be set to one and if the line is low then the bit will be set to zero. For example, if line PB2 is set to input and it has 5 V applied to it, then bit 2 in the input/output register is a one. The only problem is what happens if the voltage on the input line is between 0 V and 5 V. The answer is that the corresponding bit may or may not be set depending on the exact value of the voltage on the line. In general, reading the input/output register is unreliable if the input lines are not set either at 0 V or 5 V. If you have some lines set to output and some set to input then you can write to the input/output register without fear of affecting the input line and you can read the input/output register to obtain the current state of the output and input lines. The function FNIN(N%) will return the state, zero or one of any line PB0 or PB7 irrespective of the line being being an input or an input.

```
400 DEF FNIN(N%)
410 LOCAL S%
420 IF N%<0 OR N%>7 THEN =-1
430 S%=?&FE60
440 S%=(S% DIV 2^N%) AND &01
450 =S%
```

For example, FNIN(3) returns the state, zero or one line PB3.

This completes the discussion of using the data lines PB0 to PB7 as inputs or outputs. The data lines PA0 to PA7 are permanently set as outputs to the printer port but this doesn't mean that you cannot use them as extra output lines to supplement the user port. Any of the procedures or functions given above can be rewritten to use the A input/output register. Finally, it is worth pointing out that the output lines cannot be used to provide very much in the way of power to

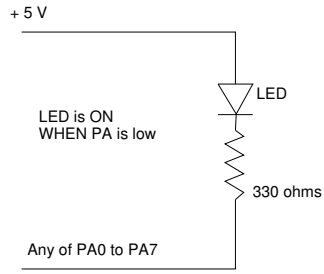

Fig. 6.6. LED output

control things. You will almost certainly have to use a transistor or a reed relay to switch any real equipment on and off. The 6522 can only sink 1.5 mA of current and provide about .5 mA and this is very little. However, you can drive an LED directly from the printer port, as shown in Figure 6.5, and this can be very useful while you are testing software or just learning about the user port. Similarly, Figure 6.6 shows a simple input circuit using a switch.

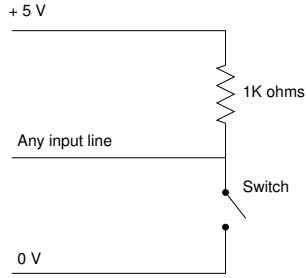

Fig. 6.6. Simple switch input

The handshake lines

As mentioned in the previous section, there are two additional lines on the VIA. On the A side these are called CA1 and CA2 and on the B side they are called CB1 and CB2. Once again the A side lines are involved in the printer interface. However, CB1 and CB2 are available for use by the user and are brought out on the user port connector. CA1 and CB2 are both inputs and CA2 and CB2 can be set as either inputs or outputs. There are so many ways that the handshake lines can be

used that it is only possible to give a summary here. In practice, the handshake lines are only required for applications where the BBC Micro is being connected to another computer or piece of equipment that is as complicated as a computer. For simple applications the handshake lines are best ignored.

The handshake lines are controlled by the peripheral control register. Which bits of the register control which handshake line can be seen in Figure 6.7. The CA1 and CB1 inputs can each set bits in

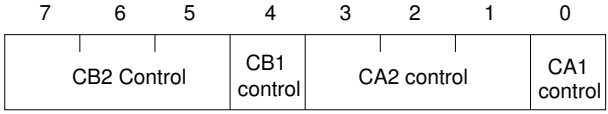

Fig. 6.7. The peripheral control register.

another VIA register - the interrupt status register. The corresponding CA1 and CB1 control bits govern when the bits are set. If the control bit is set to zero then the bit in the interrupt status register will be set to one when the voltage on the input line goes from high to low. If the control bit is a one then the bit in the interrupt status register is set when the voltage on the input line goes from low to high. Notice that his is different from the data lines in that the condition in which sets the bit isn't a voltage level but a change in voltage levels. Such inputs are known as edge triggered inputs. Notice also that there is no mention of how the bits in the interrupt status register are set back to zero. This will be discussed in connection with the interrupt status register.

The CA2 and CB2 control bits have the following effects:

Table 6.2. Effects of CA2 and CB2 control bits.

Bits 3 2 1 (CA2) 7 6 5 (CB2)	*Effect*
0 0 0	Input mode - set CA/B interrupt flag on negative transition of the input signal. The interrupt flag is cleared by a read or write of the A/B input/output register respectively.
0 0 1	Input mode - set CA/B interrupt flag on negative

transition of the input signal.
The flag is *not* cleared by reading or writing the input/output register.
The flag can only be cleared by writing a one to the interrupt status register (see later).

0 1 0 — Input mode - as for 0,0,0 but flag set on positive transitions of input signal.

0 1 1 — Input mode - as for 0,1,0 but flag set on positive transitions of input signal.

1 0 0 — Output mode - CA/B 2 is set high by an active transition of CA/B 1 input. Reset by reading or writing A/B input/output register.

1 0 1 — Output mode - CA/B 2 goes low for one cycle (of the 1 MHz clock) following a read or write of the A/B input/output register.

1 1 0 — Output mode - CA/B 2 always low.

1 1 1 — Output mode - CA/B 2 always high.

The Timers

The 6522 VIA contains two timers. This sounds a little like over-provision - with two VIAs the Model B machine has four different timers at its disposal. However, each of the timers within a VIA has a different set of features designed to suit it to a particular application. Because of this difference between the timers it is worth dealing with them in turn. Both timers are controlled by bits within the *auxiliary control register* (see Figure 6.8).

Timer 1 consists of two main components. A sixteen bit latch and a sixteen bit counter. The latch is used to store a number that can be

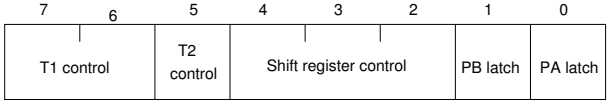

Fig. 6.8. The auxiliary control register.

loaded into the counter which is continuously being decremented at the system clock rate i.e. 1 MHz. The latches can be written to and read directly at addresses &FE66 and &FE67 (on VIA-B). However, they are also used whenever the counter is loaded at addresses &FE64 or &FE65. If the low order byte of the counter was loaded directly then its value would have changed by the time the high order byte was loaded (remember the counter is continuously decrementing). To get round this problem, data that is written to the low order byte of the counter is in fact stored in the low order byte of the latch. Writing data to the high order byte of the counter not only loads the high order byte of the latch as a side effect but also causes the low order byte of the latch to be loaded into the low order byte of the counter. Using this method, the high and low order bytes of the counter are loaded at the same time. The latch holds any value stored until it is changed.

Timer 1 has two distinct operating modes - single shot and free running. In addition, it can also cause an output on PB7 every time the counter reaches zero. The function of the bits in the auxiliary control register are as shown below:

Table 6.3. Timer 1 control bits.

Bits 7	6	Effect
0	0	Set the T1 interrupt flag after the counter has reached zero after being loaded (single shot mode).
0	1	Set T1 interrupt flag after the counter has reached zero after having been loaded. Automatically reload the counter from the latch each time it reaches zero (free running mode).
1	0	Single shot mode but PB7 goes low when the high order counter/latch is loaded and returns high after when the T1 interrupt flag is set.
1	1	Free running mode but the output of PB7 is inverted each time the T1 interrupt flag is set.

The best way to explain the working of the two modes is by example. If we want to wait for a fixed time interval in a program then we need to use T1 in its single shot mode. After setting the appropriate bits in the auxiliary register to place T1 in single shot mode, the low

order byte of the count is loaded into the latch/counter at &FE64. To start the countdown for the time interval all you have to do is load the high order byte of the count into the latch/counter at &FE65. To detect the end of the time interval, i.e. when the counter reaches zero, you can either examine bit 6 in the interrupt status register at &FE6D until it becomes a 1 or you can enable the T1 interrupt and write an interrupt service routine (as described in Chapter Three). To repeat the time interval, all you have to do is load the high byte latch/counter. The number that has to be loaded into the counter to produce any given time interval is easy to determine. If the counter is loaded with n then the flag will be set after n+1.5 ms. If the PB7 output is enabled, then you will also get a single pulse out of the PB7 line lasting for the same time interval. When the counter reaches zero it continues to decrement so the counter registers can be used by the processor to find out how long ago the counter reached zero.

The free running mode is very similar to the single shot mode except for the fact that the counter, both high and low order bytes, are reloaded from the latch each time the counter reaches zero. This allows a series of interrupts and, if the output of PB7 is enabled, a series of pulses to be produced. You can change the values stored in the latches by writing to the latch registers at &FE66 and &FE67 while the count is being decremented. The new value will be loaded into the counter the next time it reaches zero. Using this method you can produce pulses of varying length in a continuous stream.

Timer 2 operates as a timer in the single shot mode only and as a counter. As a timer it is more limited than timer 1, not only because it cannot be used in a free running mode, but because there is no way of reading or writing to the latches directly. Setting timer 2's counters follows the same pattern as for timer 1 in single shot mode. First the low order byte is loaded into the low order latch at &FE68, then the high order byte is loaded into the counter at &FE69 which also transfers the contents of the low order counter. The counter then decrements until it reaches zero when it sets the T2 interrupt flag in the interrupt status register. No output pulse can be produced with timer 2. The second mode in which timer 2 can be operated is a pulse-counting mode. In this mode a number can be loaded into the counter registers in the same way as in single shot mode. However, the counter now decrements each time a (negative going) pulse is applied to PB6. The T2 interrupt flag is set when the counter reaches zero but the counter still decrements with each input pulse. The T2 control bit (bit 5 in the

auxiliary control register) selects the operating mode of timer 2. If it is zero then the timer operates in a single shot mode. If it is one then the timer operates as a counter.

As a practical example of using the timers requires assembly language, it is postponed until Chapter Eight.

The shift register

The 6522 VIA contains an eight bit re-cycling shift register. It can be used either in an input or output mode. In an input mode bits are shifted into the register from CB2. In an output mode, bits are shifted out to the same CB2 line. Apart from this choice of input or output mode the only other variation in the way that the shift register works is the selection of the source or speed of the clock that causes a bit to be shifted into or out of the register. When the shift clock is derived internally it is available on the CB1 line and when it is derived externally it is supplied through CB1. The shift register' soperating mode is controlled by bits 2 to 4 in the auxiliary control register as follows:

Table 6.4. Shift register control bits.

Bits 4	3	2	*Effects*
0	0	0	Shift register disabled.
0	0	1	Shift in controlled by timer 2.
0	1	0	Shift in controlled by system clock (1 MHz).
0	1	1	Shift in controlled by external pulses on CB1.
1	0	0	Free running output controlled by timer 2.
1	0	1	Shift out controlled by timer 2.
1	1	0	Shift out controlled by the system clock (1 MHz).
1	1	1	Shift out controlled by external pulses on CB1.

In each of the modes apart from the free running mode the shift out or in is initiated by writing or reading the shift register and after eight shift pulses the shift register interrupt flag is set to one in the interrupt register. If the shift clock is derived internally this stops after eight pulses. However, in free running mode the shift clock is applied continuously. As the shift register recycles, the same eight bits are sent

out over CB2 repeatedly.

The shift register is a fairly specialised device in that is generally used to transfer information from one computer to another. However, the free-running mode can be used to generate short irregular pulse trains by loading the shift register with the correct pattern of ones and zeros.

The interrupt control and status register

While discussing the various functions of the VIA, the setting of a bit in the interrupt register has often been used as a way of indicating that something has happened. In fact, the interrupt status register works as a pair with the interrupt control register (see Figure 6.9). The meaning of all of the bits in the interrupt status register has already been explained apart from bit 7 or IRQ. The 6522 VIA can cause an IRQ interrupt (see Chapter Three) if so desired when any selected bits in the interrupt status register are first set to one. Bit 7 is one if the VIA has caused an

7	6	5	4	3	2	1	0	
IRQ	T1	T2	CB1	CB2	SR	CA1	CA2	Interrupt flags
Set/clear	T1	T2	CB1	CB2	SR	CA1	CA2	Interrupt control

Fig. 6.9 Interrupt status/control registers.

IRQ interrupt and zero otherwise. Whether the setting of a bit in the interrupt status register actually causes an interrupt or not depends on the setting of the corresponding bit in the interrupt control register. For example, if T2 is 1 in the interrupt control register when the VIA will cause an IRQ interrupt as soon as the T2 bit in the interrupt status register is set to one. The bits in the interrupt register can be cleared either indirectly by the methods described in the previous sections (for example, bit T2 is cleared by reading the T2 low order counter) or directly by writing to the register with the corresponding bit set to one. For example, to clear bit 3, i.e. CB2, but leaves all the others unaffected you would write &08 to the interrupt status register. Setting a bit to one or zero in the interrupt control register also uses an odd method. To alter a given bit you must write to the interrupt control register a value with a one in the same bit position. If bit 7 of the value

is a 1 then the selected bit (or bits) will be set to one. If bit 7 of the value is 0 then the selected bit (or bits) will be set to zero. In other words, the selected bits are set to the same state as bit 7. An interrupt flag that is not enabled will not cause an IRQ interrupt but can still be set and cleared in the same way as normal.

Data latching

There are two bits left in the auxiliary control register that haven't been discussed so far - the PB and PA latch bits. If either of these bits are set to one then the input data to the A and B sides will be latched when the CA1 and CB1 interrupt flags are set to one respectively. In this mode reading the input/output registers returns the data values most recently latched rather than the current state of the input lines. The advantage of this is that fast changing data can be held automatically for the processor to read at a later date.

Conclusion

This chapter began with a warning that the 6522 VIA was a complicated device. By the end of this chapter you should be convinced that it is indeed complicated but you should be beginning to see how versatile it really is - living up to its name of Versatile Interface Adaptor. The only way you can become familiar with any area of interfacing is to make use of the information in a practical problem. Using the BBC Micro's A to D convertor and its VIA is no exception to this rule.