

INTERFACING the **BBC** MICROCOMPUTER



Brian Bannister
Michael Whitehead

Interfacing the **BBC** Microcomputer

General Editor: Ian Birnbaum (General Adviser (Microelectronics in Education)
Education Department, Humberside County Council)

Advanced Graphics with the Acorn Electron

Ian O. Angell and Brian J. Jones

Advanced Graphics with the BBC Model B Microcomputer

Ian O. Angell and Brian J. Jones

Interfacing the BBC Microcomputer

Brian Bannister and Michael Whitehead

Assembly Language Programming for the Acorn Electron

Ian Birnbaum

Assembly Language Programming for the BBC Microcomputer (second edition)

Ian Birnbaum

Using Your Home Computer (Practical Projects for the Micro Owner)

Garth W. P. Davies

Beginning BASIC with the ZX Spectrum

Judith Miller

Using Sound and Speech on the BBC Microcomputer

Martin Phillips

Other books of related interest

Advanced Graphics with the IBM Personal Computer

Ian O. Angell

Programming in Z80 Assembly Language

Roger Hutton

Interfacing the BBC Microcomputer

Brian Bannister
Michael Whitehead

M
MACMILLAN

© Brian Bannister and Michael Whitehead 1985

All rights reserved. No reproduction, copy or transmission of this publication may be made without written permission.

No paragraph of this publication may be reproduced, copied or transmitted save with written permission or in accordance with the provisions of the Copyright Act 1956 (as amended).

Any person who does any unauthorised act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

First published 1985

Published by
Higher and Further Education Division
MACMILLAN PUBLISHERS LTD
Houndmills, Basingstoke, Hampshire RG21 2XS
and London
Companies and representatives
throughout the world

Printed in Great Britain by
Camelot Press Ltd,
Southampton

British Library Cataloguing in Publication Data
Bannister, B.R.

Interfacing the BBC Microcomputer.
(Macmillan microcomputer books)

1. Computer interfaces 2. BBC microcomputer

I. Title

001.64'04 TK7887.5

ISBN 0-333-37157-7

Contents

<i>Preface</i>	<i>vii</i>
1 Input-Output Facilities	1
2 The User Port	19
3 Analogue Signal Handling	43
4 The 1 MHz Bus	57
5 Some Applications	76
 <i>Appendix A: Transistor-Transistor Logic</i>	 <i>102</i>
<i>Appendix B: Machine Code Programming</i>	<i>107</i>
<i>Appendix C: Input-Output Memory Map</i>	<i>111</i>
<i>Appendix D: Data Sheets</i>	<i>112</i>
<i>Appendix E: Summary of Connections</i>	<i>148</i>
 <i>Index</i>	 <i>151</i>

Preface

The BBC microcomputer is a very versatile machine in its own right: it has excellent facilities for problem solving, data handling, games, and graphics and sound generation. With the addition of a printer and a disc (or cassette) unit, and perhaps other standard peripheral devices, the computer forms the core of a powerful system. However, an extremely flexible system can be created by anyone who can make use of the wide-ranging input and output facilities provided by the manufacturer, to add their own circuits, or to interact with other systems.

Initially, having acquired the computer, or at least the use of it, a user will be concerned with driving the computer itself. However, having become conversant with BASIC, and possibly machine code programming, the more imaginative user will see many possible applications, most of which will require the addition of external devices. In the widest sense, these applications involve the elements of control: taking readings from some equipment, or sensing the settings of switches or relays; controlling indicator lights or the position of a stepper motor; and so on. By bringing the two operations of input and output together, we can then automatically detect certain conditions and, perhaps, sound a warning, or even adjust the input or output settings according to pre-determined rules.

This book explains how to set about building such systems. It explains the interfacing features that are available on the BBC micro and how to make the best use of them in a wide range of applications. Suitable devices are introduced and techniques for using them are explained, with practical examples chosen to illustrate the basic ideas. These can then be extended in many ways, limited only by the reader's imagination (and pocket!).

The book does not assume a high level of technical ability, although an understanding of transistor action is desirable. Certainly staff and students in schools and colleges will find the book most useful, and it should present no difficulties to the 'advanced' hobbyist.

Components used in the circuits described are all relatively cheap and are available from recognised distributors such as Farnell Electronic Components Ltd and RS Components Ltd. Program listings with comments are included throughout, but the comments should be omitted when copying the programs for use in the computer. The logic symbols used are ANSI Mil. Spec. symbols as used by Acorn Computers Ltd.

'Acorn' is a registered trademark of Acorn Computers Ltd. Throughout the book, the interpretation of published material is the authors' own and does not imply any endorsement by Acorn Computers Ltd.

We gratefully acknowledge permission from NEC Electronics (UK) Ltd, Synertek Inc. and Texas Instruments Ltd to publish the data sheets contained in Appendix D. Finally we owe thanks to Don Whitehead for his helpful comments, and to Debbie Eaton for typing the manuscript.

The University, Hull
1984

B. R. BANNISTER
M. D. WHITEHEAD

1 Input-Output Facilities

The BBC microcomputer has many advanced features that are not normally provided on computers of this size. Three general principles seem to have been foremost in the drawing up of the design specification:

To make programming of the computer as easy and flexible as possible by providing an enhanced BASIC and the ability to make use of multiple programs and assembler language sections.

To include many of the features found only in rudimentary form on other microcomputers, such as high-resolution colour graphics, complex sound generation, programmable keys, and so on.

To provide extensive interfacing capability, allowing the computer to form part of an extended system communicating with discs, printers, second processors, Teletext and even other computers via Econet, and able to accept new devices which may be produced as the technology develops.

It is the third area, of interfacing, with which we are concerned and we shall see that there are many ways in which the user can add extra circuitry for specific purposes, so taking full advantage of the flexibility that has been built in. This first chapter summarises the input and output facilities provided on the BBC model B microcomputer, but we first set the scene by considering some of the fundamental ideas that are necessary to the understanding of the transfer operations involved.

The average user scanning the circuit diagram of the microcomputer would see a bewildering array of symbols and interconnecting lines. A more practised eye, however, will recognise that there are distinct blocks of circuitry arranged to carry out specific tasks. In each block, one or two specialised large-scale integrated circuits are used, with additional logic gates providing what the designers call 'glue logic' to hold the system together. Between the blocks run the interconnecting wires arranged as two distinct *buses*, or *highways*, known as the *data bus* and the *address bus*, and a third, less well-defined, *control bus* made up of the controlling signals needed to ensure correct timing and the general smooth running of the system.

The heart of the computer is the microprocessor, which is where the program instructions are actually carried out, and which controls, and responds to, the other sections. The processor relies heavily on the memory sections which are of two types. One bank of chips makes up the read/write memory, which is general-purpose memory to hold user programs and data, and also to store the data needed

in continually refreshing the video display. Another bank of chips is the read only memory, ROM, containing the system programs that must always be available whenever the computer is switched on. The read/write memory is normally referred to as random access memory, RAM, since any location can be addressed and selected as readily as any other. But bear in mind that ROM is equally randomly accessible, unlike cassette or disc memory where data is stored in a serial form.

Any program, whether originally in machine code or in BASIC, or indeed in any other language, is ultimately processed by the computer as a succession of instructions which are executed consecutively. The processor runs a repetitive cycle in which the next instruction is fetched from memory and decoded, and then the indicated operation is carried out.

Each instruction must provide sufficient information for the processor to carry out the operation satisfactorily. The first section, or *field*, of the instruction is one byte long, and carries a code indicating the particular operation required. The list of available operations is called the *instruction set*, or *op-code set*, and each mnemonic of the assembler language corresponds to one of the op-codes. In some cases the instruction is complete in the one byte, but most instructions require additional information which normally gives the address of one of the operands to be used. Where this is the case, the code of the first byte also indicates that the fetch operation must be extended to retrieve the address field data, which can require one or two more bytes.

In executing the instruction, the address code defining the memory location of the operand to be used at a given time is generated by the processor and is distributed on the 16-bit unidirectional address bus. The data involved is carried on the 8-bit data bus, which has driver circuits that enable it to operate bi-directionally, and the drivers must be switched to the appropriate direction by the processor. Both the address bus and the data bus are extended to link all the other blocks of the computer circuitry.

The majority of the time spent by the processor, as in any computer, is concerned with transferring data from one location to another. Data bits are moved around internally on the data bus, and each group of eight bits forms a byte which, in many cases, represents one character. The bytes are transferred between registers of various types, where a register can be thought of as a temporary store for the data bits but having a different name and characteristics dependent on the job that it is required to do. The first type, high-speed registers inside the processor, are used as working registers in carrying out the operations specified by the successive instructions of the program. The 6502 processor used in the BBC micro-computer has one *accumulator*, A, and two *index registers*, X and Y, directly available to the programmer, but also makes use of a 16-bit *program counter*, PC, an 8-bit *status register*, a *stack pointer*, and some auxiliary registers.

The second type is the random access memory, which is an array of registers designed to hold programs and data. Read only memory can also be considered as an array of registers that have been modified to retain data in a permanent, or

semi-permanent, form, so that we can easily read from any desired location, but can write in new data only with special equipment, if at all.

When we come to transferring data into or out of the computer we use a third type of register known as an *input port* or *output port*, and many of the specialised input-output (I/O) chips use other internal registers to provide appropriate control over how the ports operate at any given time.

In all cases, however, an individual register is uniquely defined by its 16-bit address which is generated by the processor and distributed on the address bus. In fact, therefore, all transfers to and from input and output ports, and their associated control registers, are treated as memory transfers, and the system is said to use *memory-mapped I/O*. The *User Guide* provides a memory map that shows the allocation of all 65 536 (64K) locations from &0000 to &FFFF (the ampersand, &, is used to denote a hexadecimal number). The input-output area of the map is from &FC00 to &FEFF, and is considered as being made up of three sections of 256 bytes each. These sections are known, in Acorn literature, as FRED, from &FC00 to &FCFF, JIM, from &FD00 to &FDFF, and SHEILA, from &FE00 to &FEFF. SHEILA is the area devoted to internal input-output devices and, within that area, individual controller chips used in the micro-computer are selected by use of 74LS139 dual 2-line-to-4-line, and 74LS138 3-line-to-8-line data selectors, figure 1.1. When a data selector chip is enabled by the G input signals, one and only one of the outputs goes low.* The 74LS138 data selector in figure 1.1, for example, is selected when input G1 is high and G2A and G2B are low. G2B is taken low when the inputs to the 74LS30 NAND gate are all high, or 1, so the 74LS138 is selected when address bit A8 is at 0 and bits A9 to A15 are at 1. This means that the data selector reacts only to addresses that begin with &FE, since &FE is a shorthand way of representing the binary value 11111110. The particular output of the selector that goes low is then determined by the values on the remaining three inputs, A, B and C, controlled by address bits A5, A6 and A7. This method of selection allocates a block of addresses for each device, but they may not all be required. The CRT controller chip, for example, has the eight locations &FE00 to &FE07 allocated, but only the first two are used.

The JIM and FRED areas are used in conjunction with the 1 MHz bus and will be dealt with when we come to consider the action of that bus.

The setting up of a port, and subsequent writing of data into or reading data from the port, is carried out under program control. Although the computer circuitry responds only to machine code, the program may be written in either assembler language or BASIC, which is then translated to the form required by the processor. In BASIC the statements are of the form

* Logic values are indicated by voltages, and are said to be 'low', or logic 0, when the voltage is approximately zero, and 'high', or logic 1, when the voltage is something between about 3.5 and 5 volts. A small circle on the logic symbol indicates that output or input is active when low.

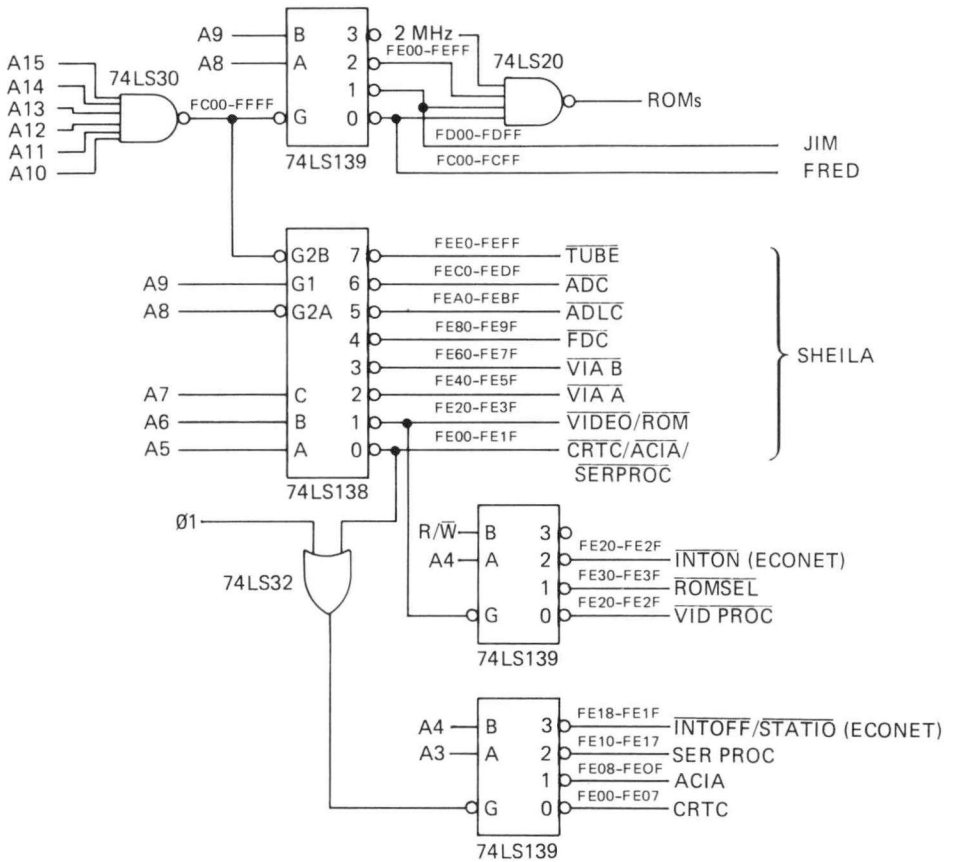


Figure 1.1 Address decoding circuitry

10 ?&FE62 = 255

meaning 'set location &FE62 to 255', or

10 A = ?&FE60

meaning 'set variable A to the value read from location &FE60'. The initial 10 is the line number in both cases. In assembler form the corresponding instructions would be

```
LDA #&FF
STA &FE62
```

for the first, and

```
LDA &FE60
```

for the second.

Assembly language is faster than BASIC and much more compact, so it is often useful in dealing with data transfers through input-output ports which would otherwise interfere with the smooth running of the main program. An understanding of assembly language (that is, machine code) programming is very desirable, and sometimes allows access to features of the computer not otherwise available. A brief introduction to machine code programming is given in appendix B.

In all but very short sections of assembly language it is convenient to make use of the standard routines that have been provided in the operating system programs, using *FX and OSBYTE calls. This is also good programming practice because it means that your programs are relocatable if and when you modify or extend your system. Full details of these calls are given in section 42 of the *User Guide*, but the method of operation is always the same.

The required routine is indicated by the number of the call, and a linking routine is necessary to point the processor to its first instruction. All the routines, including the linking routine, are held in ROM since they form part of the operating system. Being in ROM, however, means that we are unable to write the start address directly into the link routine and we must achieve the transfer indirectly. The linking routine is held in ROM, starting at &FFF4, and it makes use of a pair of bytes in RAM, located at &020A and &020B, and given the label BYTEV, which stands for Byte Vector. When an *FX or OSBYTE call is made, the number of the call is placed in the accumulator, A, and the value is used to determine the start address of the routine. The start address is written into RAM as the Byte Vector, BYTEV, and the linking routine now merely needs to be an indirect jump to the routine using the vector provided. The actual sequence is

<i>Location</i>	<i>Hex code</i>	<i>Mnemonic</i>	<i>Meaning</i>
&FFF4	6C		
&FFF5	0A	JMP I BYTEV	Jump indirect to &020A/B
&FFF6	02		

Thus, when the *FX or OSBYTE call finds the 'Jump to subroutine at &FFF4' instruction, it is told to do another jump to the start of the routine which begins at the address it will find in locations &020A and &020B, and so arrives at the correct starting point.

Any values needed by the call are first placed in the X and Y registers, and if there are any values to be returned by the call they also use the X and Y registers.

In dealing with data transfers to and from input-output devices mapped onto the memory area SHEILA, we could present our earlier examples in OSBYTE call form as

```
LDA #&97      /Prepare OSBYTE call to write to SHEILA
LDX #&62      /with offset &62
LDY #&FF      /and &FF as the value to be written.
JSR  &FFF4    /Call OSBYTE.
```



```

LDA #&96      /Prepare OSBYTE call to read from SHEILA
LDX #&60      /with offset &60.
JSR &FFF4     /Call OSBYTE.

```

It is particularly easy with the BBC computer to insert assembler sections into BASIC programs, since the assembler is included in the BASIC interpreter. The DIM P% statement, for example, provides a simple way of reserving a block of memory for assembler code as it becomes necessary. When the BASIC program is run, the DIM statement is used to set aside a block of memory locations for the machine code section of the program, which is listed later. As we have seen, the microprocessor uses the program counter to indicate where its next instruction starts, but, until the program is run, the actual program counter value, P%, corresponding to the start of the section, is unknown. We therefore refer to it as a variable, such as PROG, which we define at the start of the machine code section. The length of the block of bytes reserved can be specified, and, where the exact number of bytes required is not known, it is sensible to include some value large enough to ensure some spare bytes. For example

```
10 DIM PROG 29
```

reserves thirty bytes of memory for a machine code section starting at location PROG. The actual value of PROG is allocated by the interpreter when the program is run, and may change if the program is changed. Subsequently we set the program counter by means of a statement of the form

```
100 P% = PROG
```

P% can be set directly to an absolute value if required.

We follow with the assembler code, which must be in square brackets, thus:

```

10  DIM PROG 29      /Reserve 30 bytes at PROG
100 P% = PROG        /Set PC to allocated start value
120 [                /Start of machine code section
130 .LIGHTS          /Reference label
140 LDA #151         /Program code
150 LDX #&60
160 LDY &70
170 JSR &FFF4
180 RTS              /Return from subroutine
190 ]                /End of machine code section
200 etc.             /Continue with BASIC statements
280 CALL LIGHTS      /Call machine code section LIGHTS

```

Line 130 contains the reference label acting as the name of the machine code routine. It must be preceded by the full stop which causes the assembler to allocate the program counter value as a variable. This can be used in BASIC sections of the program as well as machine code sections. The routine is ter-

minated by the RTS, return from subroutine, instruction and the closing square bracket.

As part of its duties, the assembler must allocate specific locations to the labels defined by the program writer. The assembler scans the program twice: during the first scan, listing all labels, and during the second, matching the labels to the locations allocated. The labels are used when program jumps are called for, and during the initial stages of the assembly process any jump forward in the program will refer to a label that has not yet been defined in terms of a memory location. The assembler includes error-checking routines which, among other tasks, look for undefined labels, and, during the first pass, we must switch out the checking routines, though at all other times, of course, we wish to have them operating. This is normally achieved by use of the OPT directive to the assembler, in conjunction with a FOR statement. For example, we would add extra lines to our previous listing to give

```

10  DIM PROG 29
100  P% = PROG
110  FOR I = 0 TO 2 STEP 2  /Select OPT 0 during first
120  [OPT I                /pass, and OPT 2 during second.
    |
    |
190  ]
200  NEXT                /Close FOR loop
210  etc.
280  CALL LIGHTS

```

We are now ready to look at the input-output facilities that are provided on the computer, and to see how they can be used. If we go first to the back of the computer, we see a range of connectors, as shown in figure 1.2. Many of these give access to circuits that are designed for particular purposes, but others, such as the analogue and RS423 ports, are more general purpose.

The three sockets at the left are all concerned with the connections of the display unit so we shall look first at the video signals provided. These allow the computer to display text or graphic information held in the video memory. Unlike many other microcomputers, this one uses a bit-mapped display and, except in mode 7, does not use a character generator ROM. This means that one bit of memory is provided for every addressable spot, or *pixel*, on the screen, so that the hardware can produce high-resolution displays with graphics and text intermixed. It does, however, necessitate a large video RAM area: in fact it needs up to 20 kbytes of RAM in the highest resolution modes, to cater for 81 920 pixels, giving 32 lines of 80 characters each on an eight by eight dot matrix (mode 1) or fewer pixels with more colours (modes 2 and 3). In the Teletext mode, mode 7, a maximum of only 1 kbyte of memory is required since standard character codes are stored and the character writing information is then provided by the SAA5050 character generator.

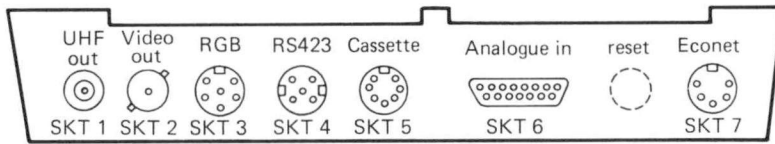


Figure 1.2 Connectors viewed from rear of microcomputer

The video circuitry consists mainly of a Motorola MC6845 CRT controller, CRTC, working in conjunction with a custom-designed video-processor chip which produces the three video signals, red, green and blue (RGB) as correctly timed sequences for a high-quality colour monitor set. The three colour signals, together with a composite synchronising pulse signal, all at TTL levels (see appendix A), are brought out to the six-way DIN connector, SK3, labelled RGB, figure 1.3. A composite video signal for use with a PAL baseband monitor set is created from R', G' and B' by a simple summing amplifier (as shown on page 504 of the *User Manual*). This is brought out to the BNC socket, SK2, labelled 'video out'. Finally, another version of the composite video signal is used to modulate a UHF signal giving an output suitable for use with a conventional domestic PAL colour television receiver, operating nominally on channel 36. The UHF signal is brought out to the Belling-Lee coaxial socket, SK1, labelled 'UHF out'.

The video circuitry must also deal with the refreshing of the memory. The memory devices used are sixteen MB8118, 16K \times 1 bit dynamic RAMs giving a total memory of 32 kbytes. Dynamic RAM is used because it is very fast in operation, but it must be refreshed regularly so as not to lose the data held in it. This is done by the CRTC. As the CRTC and the 6502 processor cannot be allowed to access the memory simultaneously, they must alternate, or interleave, their demands. The 6502 processor is therefore clocked at 2 MHz (though we shall see that it can be switched to run at 1 MHz when required) so that it requires access to the memory every 500 nanoseconds. But the memory is fast enough to allow two accesses in that time and each 500 nanosecond period gives one access for the memory and one for the CRTC.

The next two sockets provide for serial data transfers either at standard RS423 levels or in a coded form suitable for use with a domestic cassette recorder. The RS423 serial data communications standard is an updated version of the well-known RS232-C or V24 standard which is used extensively by terminals and modems. Inexpensive networking controllers, such as Clearway,* can also make use of the RS423 connection.

Data transfers on this serial port need only two lines for the data: one for incoming data, Receive Data, and the other for outgoing data, Transmit Data. A common return line is provided. In addition, two control lines are used: $\overline{\text{RTS}}$ (Not ready to send) and $\overline{\text{CTS}}$ (Not clear to send). The control signals are specified

*Produced by Realtime Developments Ltd.

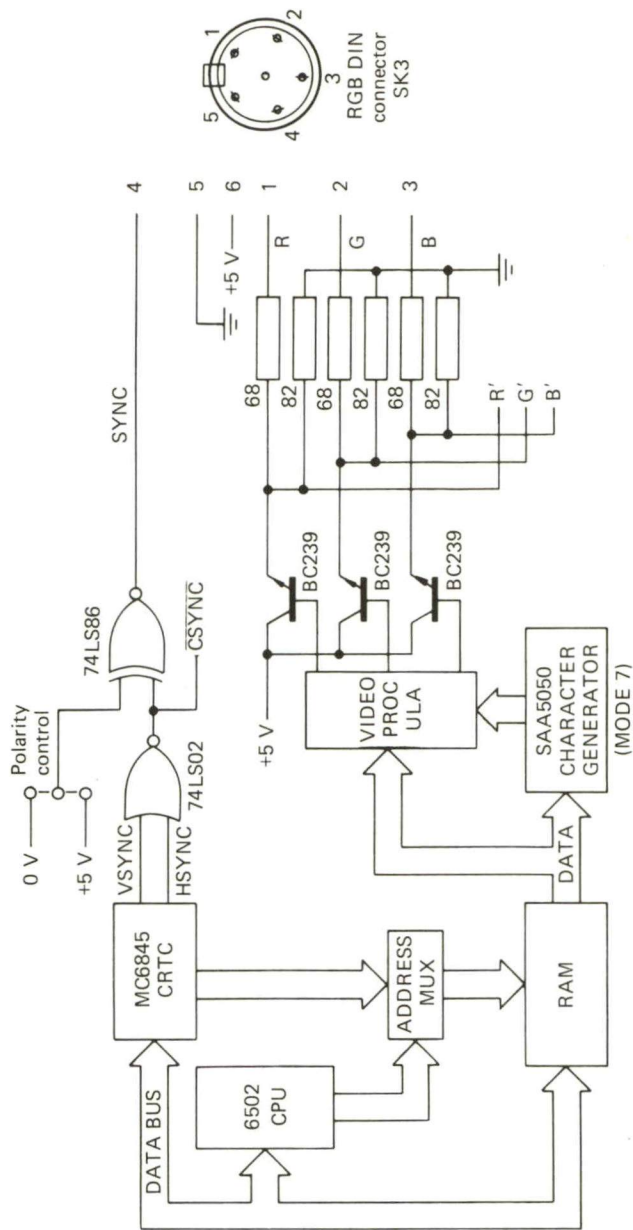


Figure 1.3 Generation of the video signals

in the negative sense because each normally sits at the higher voltage level and is taken low when the transmitter is ready to send or the receiver is ready to receive respectively. The receive and transmit directions are appropriate to the device only at one end of the connection, of course, since data in the transmit direction, for example, is received data to the device at the other end of the line. Care must therefore be taken in connecting devices to the serial port, and often a fair amount of trial and error is necessary before the connections are correct.

The conversion of an eight-bit data byte to serial form, and vice versa, is carried out by a Motorola MC6850 Asynchronous Communications Interface Adaptor, ACIA. This type of device is often referred to as a UART, standing for Universal Asynchronous Receiver/Transmitter. The ACIA arranges that the transmitted data is formatted correctly or *framed*, as in figure 1.4, with an initial START bit and a



Figure 1.4 Framing of an eight-bit data byte

final STOP bit. Similarly with received data the ACIA checks for correct formatting before extracting the required data bits. The negative-going edge of the START bit indicates to the receiver that a character is on the line and, after confirming a genuine start, the receiver clock is arranged to sample the data at about the middle of the bit period as each bit arrives. The correct timing is ensured by the receive and transmit clock signals which are generated by another special custom-designed signal processor chip. This chip contains a control register which is selected by address &FE10, and allows the clock rates to be adjusted to one of eight values indicated in terms of the baud rate, figure 1.5. The baud rate can be considered as the number of bits per second on the serial line, and ranges from 75 to a slightly optimistic 19 200 bits per second. Transmit and receive can operate at different rates if required.

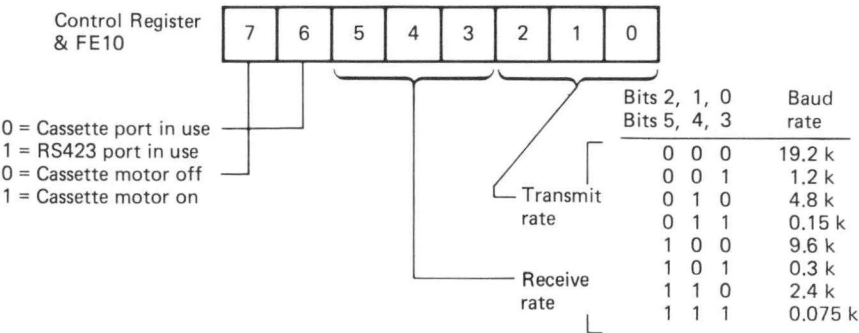


Figure 1.5 Allocation of bits in the serial processor control register

When the RS423 port is in use, the signals to and from the ACIA are passed directly through the serial processor chip and its associated buffers, and connections are made at the five-pin DIN socket, SK4, marked RS423. The cassette port can be brought into use by setting bit 6 of the register &FE10 to '1'. In this mode the ACIA is still used to convert parallel data to serial, and vice versa, but the serial processor chip now plays a much larger part.

The mode of operation of the ACIA is determined by values in two registers, control and status, and two further registers carry the data in and out. Since a control register, which accepts commands, needs only to be written to, and a status register, which reports conditions, needs only to be read, the two registers share a single address, &FE08. Similarly the data registers share one address, &FE09, and writing transfers data to the transmit register, whereas reading takes data from the receive register. The detailed allocation of bits in the registers is summarised in figure 1.6, but for most purposes these are set-up and interrogated automatically by the operating routines provided in the microcomputer.

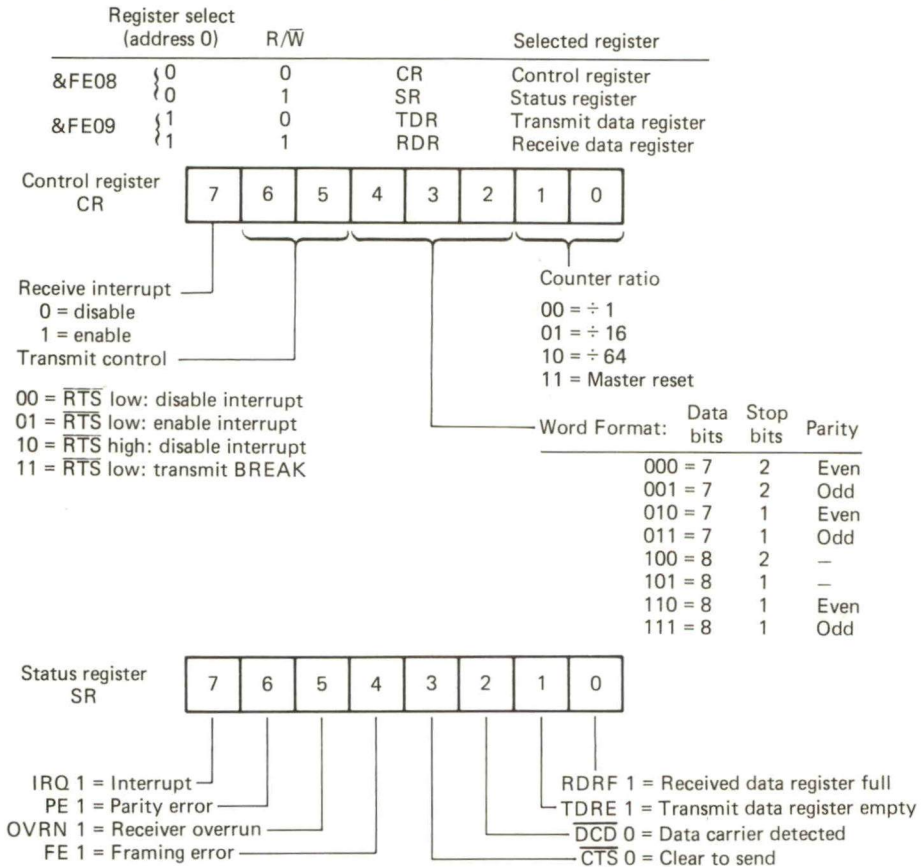


Figure 1.6 The registers of the ACIA

Continuing along the connectors we come next to the analogue inputs socket which is where we can, with suitable adjustments, connect any voltage that is not already coded digitally.

The computer operates in a digital framework, and the voltages it handles can exist only at certain levels which we designate '0' and '1', but we can cope with other voltages by means of special converter circuits. We can generate such voltages using a digital-to-analogue converter circuit, DAC, as we shall see later. The DAC gives an output voltage that is proportional to the digital value provided to it, and any change in the digital value causes a step in the resulting output voltage. We can easily set the voltage to any of the 2^n levels available from n bits in the code: for example, eight bits would give us 256 possible levels, ten bits 1024 levels, twelve bits 4096 levels, and so on. If we allocate more bits to the code representing the voltage there are more possible levels, and the step between adjacent levels becomes smaller. If we could increase the number of bits in the code to an infinitely large number, the step between levels would become infinitely small, figure 1.7. We then arrive at an analogue representation of the voltage, since an

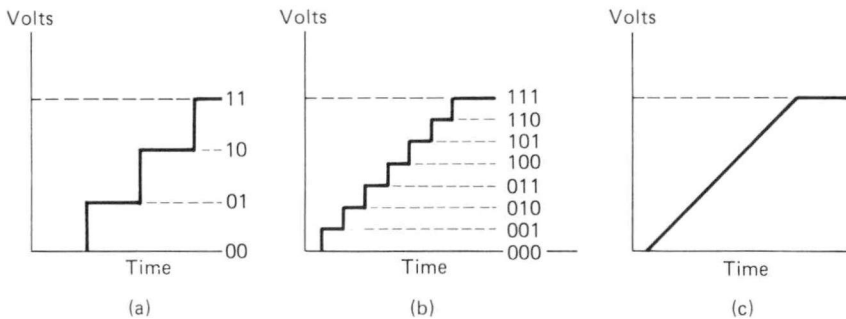


Figure 1.7 Step size decreases as number of bits in code increases

analogue voltage can take any value between the low and high limits. The voltages coming from transducers that measure variables in the 'real' world are almost invariably analogue in form. We therefore need a method of converting from the analogue to the digital form, and this is done for us by the μ PD7002 analogue-to-digital converter, ADC, provided in the computer. The μ PD7002 has four analogue inputs and these are brought out to the 15-way D-type socket, SK6, which is where the games paddles and light pen are normally connected, figure 1.8.

A games paddle is merely a variable resistor, or *potentiometer*, similar to those used as volume controls on radio and television sets. Two of the three terminals, A and B in figure 1.9a, are connected to the ends of the resistor, and the third is connected to the *wiper*, W. As the knob is turned, the ratio of R1 to R2 is varied, though $(R1 + R2)$ remains constant as the total resistance between A and B. This value should be relatively small compared with the input impedance of the converter to which it is connected, typically greater than 10 M Ω , and yet not be so

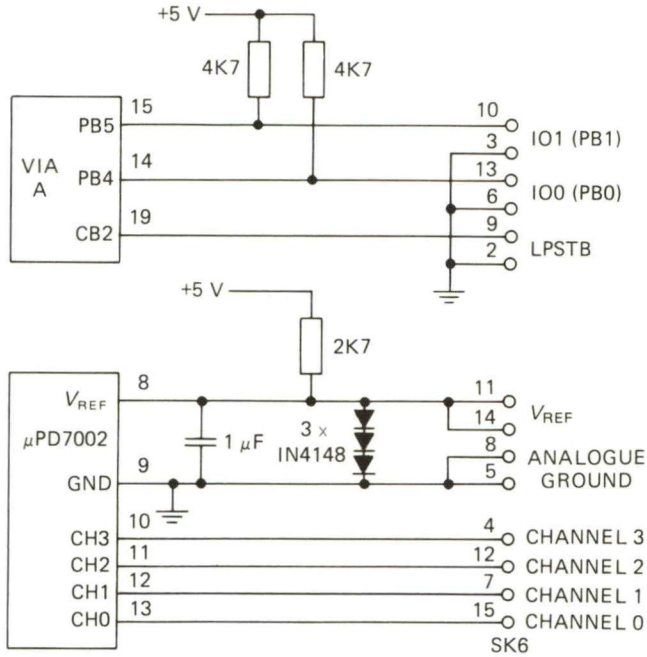
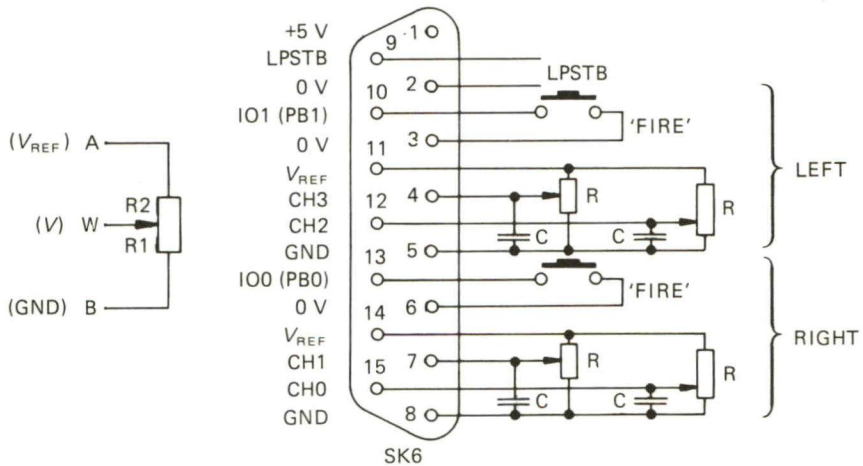


Figure 1.8 SK6 connections for the analogue-to-digital converter



(a) Voltage divider action

(b) Connections for paddles and 'fire' buttons
 $R = 10k$ linear potentiometer
 $C = 150$ nF decoupling capacitor

Figure 1.9 Potentiometer action and use in games paddles

small that it draws too large a current from the voltage reference circuitry. A value of about 10 K Ω is normally appropriate.

When the voltage V_{REF} is applied across the resistor it acts as a voltage divider to give a voltage at the wiper of $V = V_{REF} \cdot R1/(R1 + R2)$ so that V must always lie between 0 volts and $+V_{REF}$. It is important that the maximum value of V is not too great for the ADC chip to handle and, although the μ PD7002 converter is designed for a V_{REF} of 2.5 volts, the computer designers have allowed a safety margin by providing a reference voltage of 1.8 volts. This should always be used if possible, as in figure 1.9b.

A joystick controller makes use of two variable resistors, so that, as the joystick is moved, the motion is resolved into two components at right angles and the resistors are adjusted accordingly. The four channels available on the μ PD7002 means that it can handle up to two joysticks or four games paddles. The connector also carries the digital signals, which can be generated by the 'fire' buttons often provided on games paddles and joysticks. The 'fire' inputs are designated I00 and I01 and the signals are taken to bits 4 and 5 of VIA A which is the *Versatile Interface Adapter* used internally for a variety of tasks such as controlling the keyboard, sound and speech generators and so on. The state of the 'fire' buttons, '0' or '1', is written into memory from the VIA and is read by means of the $X = ADVAL(0)$ command, to become the bottom two bits of the variable X , I00 affecting bit 1 and I01 bit 0. A command of the form $X = ADVAL(0) AND 3$ selects the bottom two bits to give $X = 0, 1, 2$ or 3 dependent on whether neither, left, right, or both buttons are pressed.

The final input, light pen strobe, LPSTB, is intended for use with a light pen which generates a pulse whenever it detects the passing of the electron beam that continually scans the display unit screen to refresh the information written there. The light pen contains an optical sensor which is illuminated by the phosphor glow as the beam of the cathode ray tube passes under the pen. The pulse it generates when illuminated is routed to input CB2 of VIA A and causes the normal sequence of operations to be interrupted, so that an operating system routine can be initiated to deal with it.

The operation of the μ PD7002 converter is considered in more detail in chapter 3.

Early models of the computer included a RESET button next to the analogue inputs socket but this is not now provided though the wiring for it is still present on the board. A 'hard' reset is achieved on more recent models by pressing the CTRL and BREAK keys simultaneously.

The final connector on the back of the computer is used in interfacing the computer to the Econet. Econet is a local area network, LAN, developed by Acorn to allow up to 255 microcomputers to communicate and so share certain expensive facilities such as Winchester disc units, printers, and so on. The method of operation is very similar to Ethernet* and uses a method known as carrier

* A good description of Ethernet and other networking methods is given by K. C. E. Gee in *Introduction to Local Area Computer Networks*, Macmillan, 1983.

sensing multiple access with collision detection, CSMA-CD. All computers linked to Econet have equal status and are independent except that a common timing clock signal is taken from a designated one of the computers. A failure in one computer will not affect the operation of the network and adding more computers does not affect the inherent speed of data transfers. Inter-computer transfer rates, however, are dependent on the distance between the computers; for short distances, up to about a quarter of a mile, transfer rates up to 210 kbits per second are possible, but at a mile the maximum rate is about 100 kbits per second. Inter-connections are by 4-wire cable which gives a relatively low-cost network. The interface makes use of a Motorola 6854 Advanced Data Link Control, ADLC, chip with line driving and receiving circuits working in differential mode, through socket SK7, to provide data and clock lines, figure 1.10a. Each computer linked to the net is allocated a unique 8-bit station identity code which is set up on jumpers, designated S11, inside the computer.

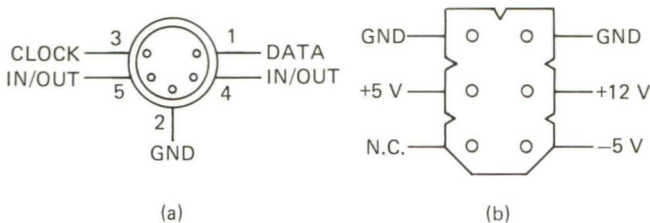


Figure 1.10 (a) Econet connector. (b) Power extension socket

If we now look under the computer, by lifting up the front edge of the keyboard, we find many more connectors, figure 1.11. Apart from the power extension socket on the far left, these all come directly from the front edge of the main printed circuit board inside the computer and make use of *insulation displacement connectors*, IDC, ranging in size from 20-way to 40-way.

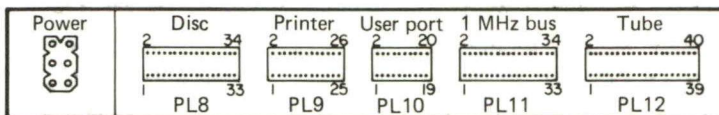


Figure 1.11 Connectors viewed from front of computer

Let us again take each connector in turn, starting, as before, with the leftmost one which, as we have noted, is the power extension socket. It is, in fact, a 6-way connector carrying +5 volts, -5 volts, +12 volts and ground returns, figure 1.10b, for use with a floppy disc unit when fitted. These supplies are not highly rated and are not intended to power other peripheral units, which should have their own supplies. However it is possible to draw a limited current at +5 volts for external use if the disc unit is not fitted.

The first of the IDC connectors is a 34-way plug, PL8, for use in interfacing to a disc unit operating to SA400 standard specification. The main bulk of the interfacing work is done by an Intel 8271 floppy disc controller, FDC, chip, which handles both the data transfers and the control signals. It can support two devices, referred to as 0 and 1, which can be either single-sided or double-sided and which use 8-inch, $5\frac{1}{4}$ or the newer 3 or $3\frac{1}{2}$ -inch diameter discs. The data rate can be switched between 125 and 250 kbits per second, as appropriate to the type of drive in use, by means of jumpers on the board. The signals involved in the control of a disc interface are complex and specialised, so it is not possible to make other use of them and we need discuss them no further here. The remaining connectors, however, provide interfaces which are more flexible to a greater or lesser extent.

The next two connectors can be taken together as they make use of a single 6522 Versatile Interface Adapter, VIA, chip. The 26-way connector, PL9, is the printer plug which is connected to one-half of the VIA and, with its additional buffer circuits, provides a standard Centronics parallel printer interface. The software routines necessary to drive the interface are provided in the machine operating system. The other half of the VIA is connected to the 20-way plug, PL10, and provides the User Input/Output Port which has eight data lines and two additional control lines for general interfacing requirements. This is the most flexible port for the attachment of external circuits or equipment to the computer and a detailed review of its operation is given in the next chapter.

Moving to the next connector, we come to another general-purpose interface but one which is more structured for specific interfacing purposes when extending our computer system. The 34-way connector, PL11, carries what the designers have called the 1 MHz bus. It is so called because one of its main features is a 1 MHz clocking signal. The computer processor runs normally at 2 MHz but this is too fast for many of the standard components used in interfacing and the processor must be slowed down whenever a transfer on the 1 MHz bus is involved. The 1 MHz clock used externally, 1MHZE, is itself derived from the 2 MHz clock used internally by the processor and, when an address lying in the areas of FRED or JIM is detected (that is, an address starting with &FC or &FD), the 2 MHz clock is stretched to make one cycle last as long as one cycle of the 1 MHz clock.

The 1 MHz bus carries the following signals: eight data lines, D0 to D7; eight address lines, A0 to A7; NPGFC; NPGFD; NRST; NIRQ; NNMI; 1MHZE; R/NW; ANALOG IN, and several ground return lines. Some of these need a bit of explanation. A full address in the computer needs sixteen bits and it is often convenient to think of it as made up of two parts: the upper half defining a *page* address with each page containing 256 bytes, and the lower half defining a particular byte within the page. In this representation, memory area FRED, which contains only addresses beginning with &FC, is then page FC, and similarly JIM is page FD. Because the decoded signals are low active, that is, they switch to logical '0' when selected, the bus signals are coded NPGFC and NPGFD, meaning 'not page FC' and 'not page FD' respectively. The bidirectional data bus drive circuits are arranged to be active only when NPGFC or NPGFD is present, and then only the

low eight address lines, A0 to A7, are required to define an address within the page. The direction in which the data bus drivers operate is controlled by the R/NW, read/not write, line with data transferred to the computer when the line is at '1', and from the computer when it is at '0'.

Page &FC, FRED, is intended for use with peripheral units requiring only limited memory space, and specific blocks have been reserved for Teletext, Prestel, IEEE-488 bus, Cambridge Ring, Winchester disc memory and test hardware. Other blocks are reserved for future use by Acorn, but one block of 63 bytes running from &FCC0 to &FCFE is allocated for general use, and it is best to restrict our usage to that area.

Page &FD, JIM, on the other hand, is intended for peripheral equipment requiring a larger memory space, and it makes available a total of up to 64 kbytes by use of an external paging register. The eight address lines, A0 to A7, available on the bus are used to define the location within a page; the eight-bit *extended page* number, which acts as the top eight bits of the address for the peripheral unit, is transferred on the data bus to the page register, which must be provided by the user and is itself addressed as location &FF on page &FC. A copy of the page register content is kept for internal reference at &EE of the zero page of the computer. The backplane card of the BBC Expansion Box, which is specifically designed for use with the 1 MHz bus, contains a page register with address recognition and signal shaping circuits, and its use is recommended wherever possible. The page register is reset to 00 by use of the NRST reset line provided on the bus. Again, a lot of the address space on page &FD is reserved by the manufacturers for specific uses. In fact all pages from &00 to &7F are reserved, which means that only the half of the pages that have a '1' as the most significant bit of the address are available to the user; that is, pages &80 to &FF.

All the signals described so far, except the reset, NRST, are high-speed signals which, when transmitted on an extended length of wire can give rise to reflections and crosstalk. It is therefore recommended that each line should not exceed about 2 feet (60 cm) in length, and should be correctly terminated by 2K2 resistors to +5 volts and ground. The load on each line should not exceed one LSTTL load. The three signals NRST, NNMI and NIRQ, however, are different and do not require termination; NNMI and NIRQ use open-collector gates (see appendix A) connected to give a wired-OR function, and the necessary pull-up resistors are provided as 3K3 resistors internally at the microprocessor. *Non-maskable interrupt*, NMI, and *Interrupt request*, IRQ, are signals that can be used to interrupt the program in the computer when external equipment requires attention. The wired-OR arrangement means that several devices can be attached to the line and any one is able to switch the line to the '0' active state. Interrupts are covered in chapter 2, when we look at the interface adaptors used in inputting and outputting.

One other feature of the 1 MHz bus is the ANALOG IN connection which provides a convenient way of coupling an external analogue signal to the internal audio circuits. The signal, which should not exceed 3 volts rms, is mixed with the output

of the sound and speech generators just before the audio amplifier which drives the loudspeaker.

The final 40-way IDC plug is that for the *tube*, PL12. The tube consists of a few control signals and a set of connections directly onto the internal address and data buses of the computer, and as such should be used very carefully. In fact, as has been emphasised earlier, it is advisable to use the tube only in the way intended so that the operation of an extended system is not impaired. It is designed to allow very high-speed communication between the internal 6502 and a high-speed *second processor*, so that the BBC computer effectively acts as a slave, dealing with routine and time-consuming operations such as scanning the keyboard, refreshing the display and dealing with other input-output operations including transfers to disc, printer and so on. Several second processors are available or planned, ranging from the 6502 running at 3 MHz clock rate and with 64 kbytes of RAM, through the Z80 running at 4 MHz, again with 64 kbytes of RAM, enabling all CP/M based programs to be run, to the powerful 16032 16-bit processor, allowing up to 16 Mbytes addressing and with 32-bit internal data operations. The connections of PL12 are shown in figure 1.12 and we see that, although all eight data bits are

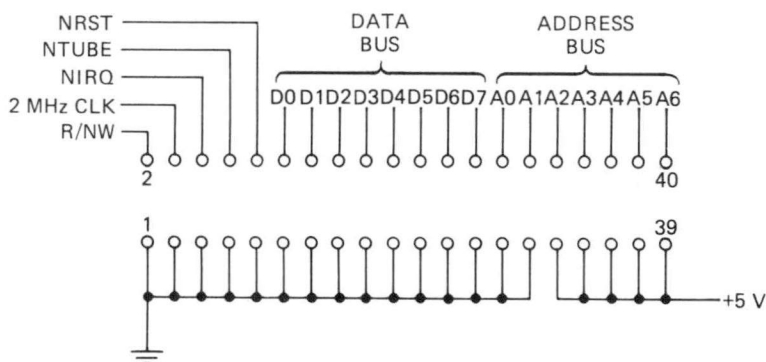


Figure 1.12 Connections to tube, PL12

brought out, only the seven lowest address lines are made available. The NTUBE select line goes low (that is, to '0') when a tube address is detected, and that occurs for any address from &FEE0 to &FEFF. These 32 locations are used in conjunction with the 7 address lines by a special tube controller chip to transfer data through the tube at the full 2 MHz clock rate.

2 The User Port

In connecting external equipment to a computer, one of the most common requirements is to light lamps to indicate the condition, or *status*, of the system, and to sense the settings of switches which are used to control the equipment. The user port is designed specifically for this sort of use and is, therefore, the first of the general-purpose interfacing provisions for us to look at in some detail. We recall from chapter 1 that it makes use of part of a Versatile Interface Adaptor, VIA, which provides a wide range of facilities to interface external circuits to the computer, and in order to control the flow of signals through the ports we must have a good understanding of how the device operates.

The main function of the VIA is the provision of two registers, port A and port B, each of which allows eight bits of data to be transferred to or from the computer in any desired pattern, but it also provides a shift register for parallel-serial operation, and two internal timers. The way in which the ports, shift register and timers operate is controlled by the values held in additional registers inside the chip. These control registers are cleared to a quiescent, disabled condition when the internal reset line of the computer is taken to ground, as it is when powering-up the computer and when CONTROL/BREAK is generated. The appropriate bit patterns must therefore be loaded into the control registers during the initialisation program sequence which immediately follows the reset. Once the VIA is programmed to operate in a certain way, it will continue to do so until the bit patterns are changed. There are several different sections of the VIA, figure 2.1, with signals to and from the microprocessor on the left, and signals for external connections on the right. When we add up all the necessary registers inside the VIA we find that there are sixteen, so four inputs, RS0 to RS3, are provided to allow us to specify any one of them, and these inputs are controlled by address bits A0 to A3. In order to select, or activate, the VIA, the chip select input, CS1, must be at '1' and CS2 must be at '0'. CS1 is held permanently at '1' so selection is controlled by the VIAB signal which is connected to CS2 and is generated by the address decoding circuitry when the address range &FE60 to &FE7F is detected.

Address bits A0 to A3 indicate the individual register address and the full list of address codes is given in the table of figure 2.2. The remaining signals to the control section are used in carrying through transfers of data between the VIA and the 6502 microprocessor. These are under the control of the processor which uses the read-not write line, R/ \overline{W} , to indicate whether it is providing data (in which case the R/ \overline{W} line is at '0' indicating writing) or is expecting to receive data

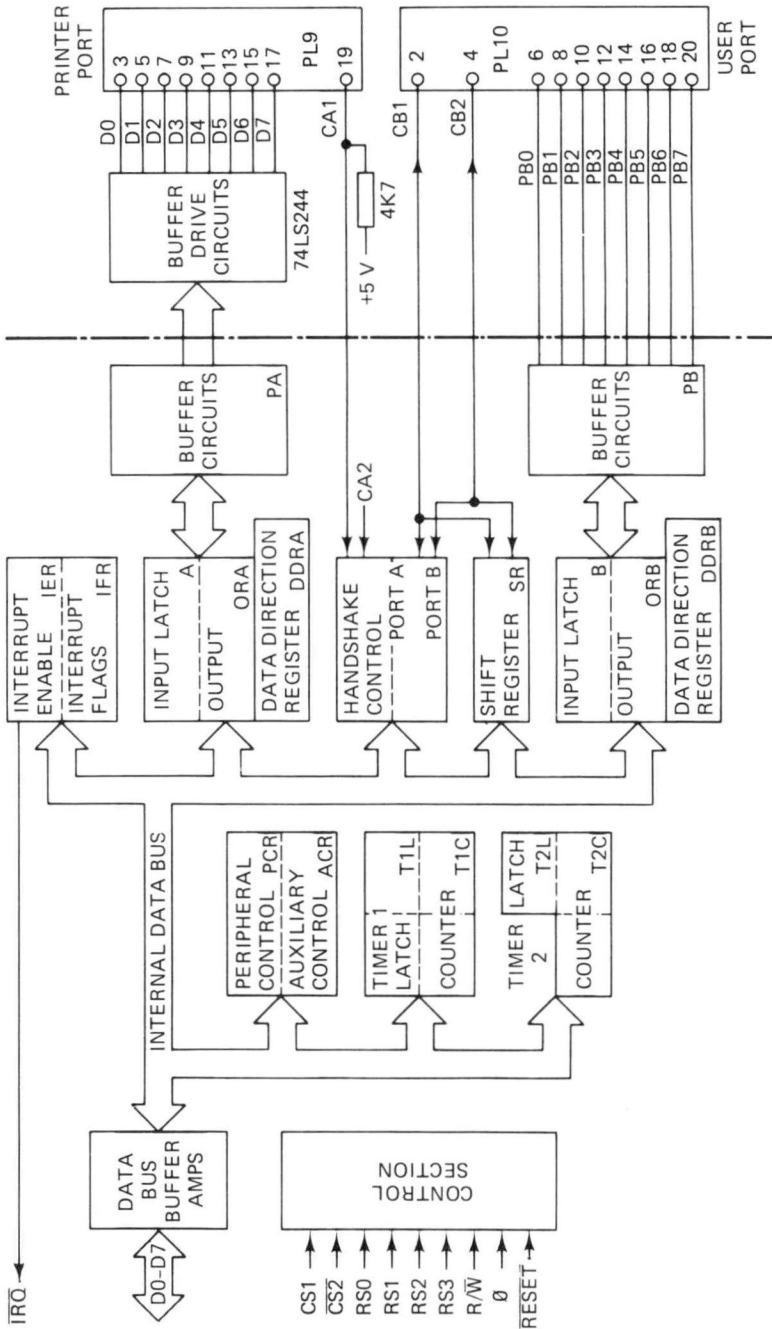


Figure 2.1 Versatile Interface Adapter schematic

	RS3	RS2	RS1	RS0	ADDRESS	REGISTER
0	0	0	0	0	&FE60	DRB Data register B
1	0	0	0	1	&FE61	DRA Data register A
2	0	0	1	0	&FE62	DDRB Data direction register B
3	0	0	1	1	&FE63	DDRA Data direction register A
4	0	1	0	0	&FE64	T1CL Timer 1: write latch, read counter
5	0	1	0	1	&FE65	T1CH initiate count
6	0	1	1	0	&FE66	T1LL load latch, low byte
7	0	1	1	1	&FE67	T1LH high byte
8	1	0	0	0	&FE68	T2CL Timer 2: write latch, read counter
9	1	0	0	1	&FE69	T2CH initiate count
10	1	0	1	0	&FE6A	SR Shift register
11	1	0	1	1	&FE6B	ACR Auxiliary control register
12	1	1	0	0	&FE6C	PCR Peripheral control register
13	1	1	0	1	&FE6D	IFR Interrupt flag register
14	1	1	1	0	&FE6E	IER Interrupt enable register
15	1	1	1	1	&FE6F	DRA Data register A, without handshake

Figure 2.2 VIA register address codes

(in which case the R/\bar{W} line is at '1' indicating reading). The 1 MHz system clock, ϕ , ensures the correct timing of the transfers.

The data registers in both peripheral ports, A and B, consist of eight data lines which can be programmed to act as inputs or outputs in any order, as defined by the settings in the corresponding bits of the associated *Data Direction Registers*, DDRA and DDRB. Setting a particular bit of a DDR to '0' ensures that the corresponding bit of the data register acts as an input, whereas a '1' ensures that the bit acts as an output. When a bit of the data register is programmed to operate as an output, the actual value of that bit is determined by the setting of the corresponding bit in the *Output Register*, OR. From figure 2.2 we see that DDRB is addressed as &FE62, so, to set up all eight bits of port B to act as outputs for example, we would load the DDR with 255. That is

```
10 ?&FE62 = 255
```

Thereafter, we output to port B merely by writing to the output register at &FE60. For example

```
40 ?&FE60 = A
```

The output register has no effect on bits of the data register that are programmed to act as inputs.

In addition to controlling the logical values on the lines of the data registers, we must also ensure that each output circuit is capable of providing the current necessary to drive whatever equipment we wish to connect. The drive capabilities of the two ports of the VIA are slightly different and anyway port A has been provided with additional buffer drive circuits since it is intended to act as the output port to a parallel printer, via plug PL9. The drive circuits are provided by a 74LS244 octal buffer-driver chip in which each output can drive up to 15 standard TTL loads (see appendix A for details of TTL operation). Because of these output

buffers, we cannot use port A as an input port but we can, of course, use it to drive equipment other than a printer if we wish, provided that we do not exceed the drive capabilities of the 74LS244 chip. Port B is connected to the output connector, PL10, directly, and each line will drive one standard TTL load. In addition, each line can provide a source current of one milliamp at 1.5 volts, allowing it to drive Darlington transistor circuits if required. A Darlington transistor, or more strictly a Darlington pair, is a compound connection of two transistors in which the collectors are connected together and the emitter current of the first transistor provides the base current of the second. The resulting current gain is the product of the two individual gains, so we get a high gain without increasing the overall base current demands, though the base-emitter voltage increases to about 1.4 volts.

In many cases an output port is used to drive a set of indicators and one of the most popular types of indicator is the light emitting diode, LED, because it will operate at voltage and current levels similar to those directly available from the logic circuitry. A typical LED will give a good illumination when passing a current of 15 to 20 milliamps. Our output port circuits, however, will sink up to 1.6 milliamps when at '0', and will source a minimum of 1 milliamp when at '1'. Neither of these levels is sufficient to drive the LED directly, but we can employ a simple transistor amplifier to boost the current level. In the circuit of figure 2.3a, the transistor turns on when the port output is at '1', turning the LED on at a current level limited to a safe value by the series resistor. If it is required that the LED switch off when the port output is at '1', the circuit of figure 2.3b can be used.

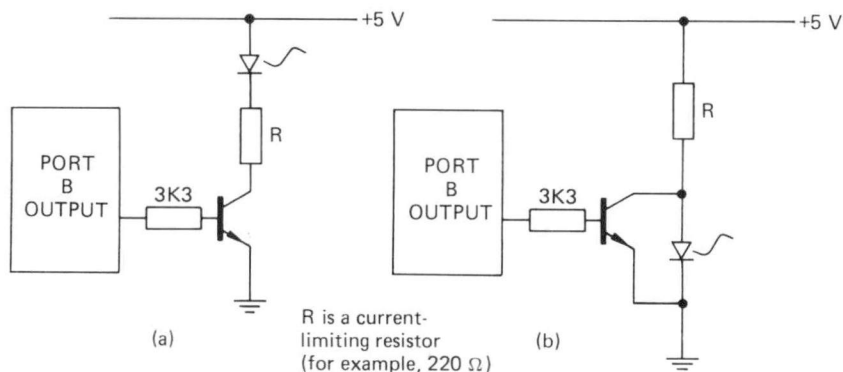


Figure 2.3 Use of transistor to drive the LED. (a) LED ON when output at '1'.
(b) LED OFF when output at '1'

A seven-segment LED display is a single-digit numeric display, which is made up of seven light emitting diodes, and in many cases an eighth diode for the decimal point. In general, all the diode anodes are connected together, although common cathode versions are also available. We can thus drive one seven-segment display from our eight-bit user port, figure 2.4a. The individual decimal digits are

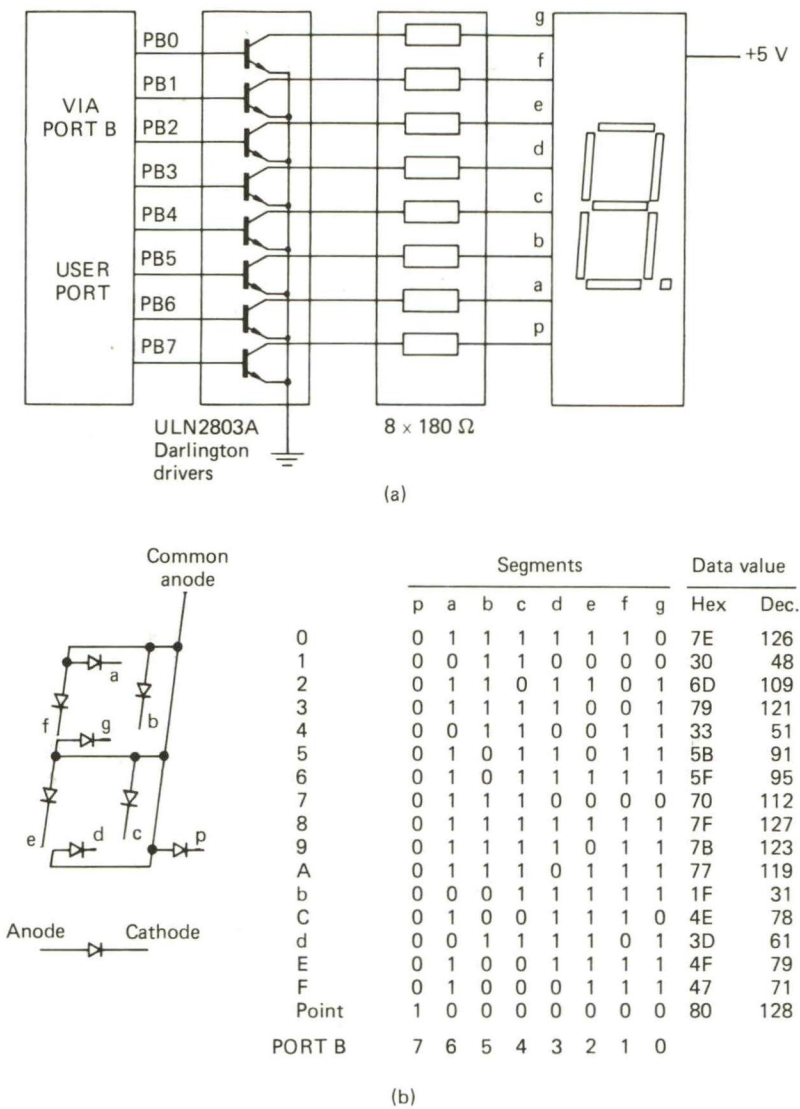


Figure 2.4 Driving a seven-segment LED display. (a) Segment connections. (b) Segment coding

formed by lighting the correct segments, and we can extend the idea to generate the full hexadecimal character set on the same display by including the six letters A to F, as long as we tolerate lower case b and d, so as not to confuse B with 8 and D with 0.

A program to output data to a single display is very simple. This one, figure 2.5, counts in decimal. Lines 60 and 70 use the internal timer, which counts in one-hundredths of a second, to define the half-second period during which each digit is displayed.

```

10 ?&FE62=255          /Setup User Port as output
20 I=0                  /Initialize integer count
30 READ A               /Read bit pattern code
40 ?&FE60=A             /Output pattern to User Port
50 I=I+1: IF I>9 THEN I=0: RESTORE /Restart count
60 TIME=0
70 REPEAT UNTIL TIME > 50 /Pause
80 GOTO 30              /Repeat
90 DATA 126,48,109,121,51,91,95,112,127,123

/This program counts repetitively
from 0 to 9.

```

Figure 2.5 Counting in decimal

A simple extension of the program, figure 2.6, allows us to display the value of any of the numeric keys on the keyboard. By pressing any key from 0 to 9 the value is displayed, and pressing the full stop illuminates the point. The point is permanently displayed until the full stop is pressed again. Further extension of the program, figure 2.7, will give us a flashing display. We choose here to flash at a visible rate, but in many cases a light emitting diode will operate more efficiently if pulsed, and if pulsed sufficiently rapidly it still appears to be permanently on. The pulsing means that we can operate the LED at much higher current levels for short periods so that the average power dissipation is still not excessive. Many

```

10 ?&FE62=255          /Setup User Port
20 DP=0                /Ensure point is off
30 RESTORE
40 A=GET               /Input a value
50 IF A=46 THEN DP=128-DP /If fullstop, invert msd
60 A=A-47              /Select data entry to match
70 IF A<1 OR A>10 THEN GOTO 30 /Key not numeric
80 FOR I=1 TO A: READ B: NEXT I /Read up to required code
90 C=B+DP              /Add point setting
100 ?&FE60=C           /Output to User Port
110 GOTO 30
120 DATA 126,48,109,121,51,91,95,112,127,123

/This program displays the value of
any numeric key pressed. Pressing
fullstop switches the decimal point
to its other condition.

```

Figure 2.6 Display numeric keys

systems use a higher drive voltage of 10 to 15 volts, with the LED pulsed on for a few milliseconds then off for a few milliseconds. We shall see later that this allows us to *multiplex*, or timeshare, the display so that we can deal with a large number of displays from a given number of ports.

```

10 ?&FE62=255 ----- /Setup User Port
20 RESTORE
30 A=GET /Input from keyboard
40 IF A=70 THEN PROCflash /Press F to flash
50 A=A-47 /Select data entry
60 IF A<1 OR A>10 THEN GOTO 20 /Key not numeric
70 FOR I=1 TO A: READ B: NEXT /Read up to required code
80 ?&FE60=B /Output to port
90 GOTO 20 -----
100 DEF PROCflash /FLASH procedure
110 ?&FE60=0 /Clear output port
120 RESTORE
130 REPEAT
140 ?&FE60=B /Output bit pattern
150 TIME=0: REPEAT UNTIL TIME >50 /Pause
160 ?&FE60=0 /Clear port
170 TIME=0: REPEAT UNTIL TIME >40 /Pause
180 A=INKEY(10) /Press any key to stop
190 UNTIL A<>-1
200 RESTORE
210 ENDPROC
220 DATA 126,48,109,121,51,91,95,112,127,123

```

/A procedure "FLASH" is defined to switch display on and off for periods specified in lines 150 and 170.

Figure 2.7 A flashing display

In some cases we may need to control larger and more powerful lamps or other loads that cannot be driven directly from the interfacing circuitry, and it is then worth considering the use of relays, or power transistors or thyristors. All these devices are merely electrically controlled switches and there are many different types in many different shapes and forms. The important point to remember in every case is that the contact rating or current rating of the selected device must be high enough to cope with the expected load current.

The simplest form of relay is the reed relay, figure 2.8a, which has usually a single normally-open contact that closes when the external coil is energised. In many cases the coil is designed to be driven directly from logic circuitry, and contact ratings up to about 0.5 amps are available. Larger, multiple contact relays are available but require driver transistors to energise the coils, figure 2.8b. As with all inductive loads, when switching the current in the relay coil, large transient voltages can be produced by the stored energy, and it is essential to connect a suitable diode across the coil in such a way that the transient voltage

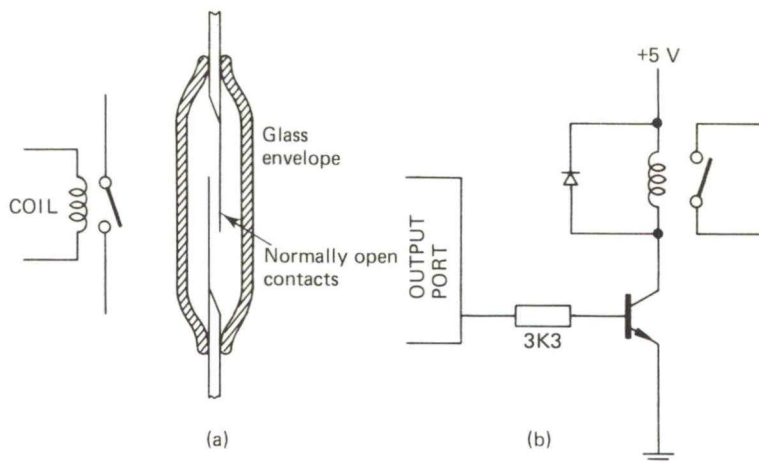


Figure 2.8 (a) Reed relay. (b) Driving a larger relay

forward-biases the diode and is short-circuited. Many relays have the protective diode built in.

A very convenient type of relay for larger loads is the *solid-state relay*, SSR, which consists of a driver circuit optically coupled to a light emitting diode controlled by the input signal. The optical coupling ensures complete isolation between input and output circuits of the relay up to a voltage of several thousand volts. Since the input circuit of the relay consists of a diode, it can be driven very easily from the output port, figure 2.9. Low power rated opto-isolated drivers may have to use a second higher-rated relay to cope with heavy loads, but the larger solid-state relays can handle several amps and make use of a triac as the output driver. They often include zero-crossing detectors to control the switching point so as to prevent spikes being generated on the supply. The efficiency is limited, so these devices run quite warm and should be mounted on a suitable heat-sink. Another point to watch is that a logic '1' at the input of the SSR switches the relay on. When a VIA is initialised then each bit of the port is set as an input, which appears to the SSR as a '1', so if it is important to ensure that any equipment does not switch on until required, and can be switched off quickly by pressing the **BREAK** key, it is safest to control the SSR with a logic '0'. This can be achieved by putting an inverter, or simple common-emitter transistor circuit between the port and the SSR.

A common extension to the controlling of switches is the requirement to sequence the operations correctly so as to indicate necessary steps or to carry out an automatic routine. This can take a multitude of forms, but probably the best-known sequence is that used for traffic lights, and it forms a convenient illustration of the method fundamental to all control sequences. Programs of any desired complexity can be built-up around the basic sequence, to include with the

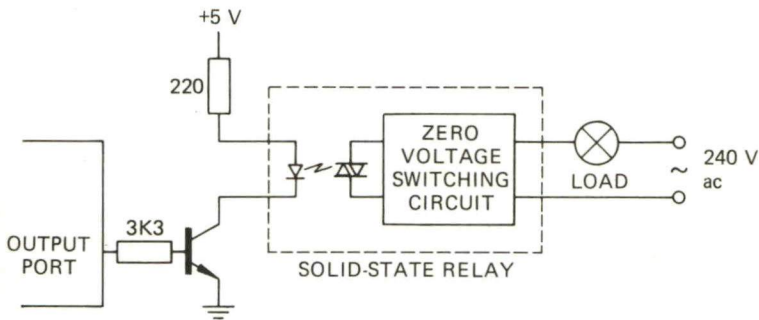


Figure 2.9 Use of a solid-state relay

traffic lights, for example, filter lights, biasing, pedestrian phases and so on. For our simple example we assume two sets of lights, which must sequence as shown in figure 2.10a, and a program to achieve this simple sequence, as given in figure 2.10b. Here we make use of a procedure, **CHANGE**, which itself makes use of a subroutine, **LIGHTS**. The subroutine involves an **OSBYTE** call to the **SHEILA** base address plus **&60**, that is, to the user port address, and the value to be sent to that address is placed in location **&70**. The procedure outputs the new pattern to the lights and then waits **N** seconds before sending out the next pattern. Merely by adjusting the output codes and time delays appropriately, this program can be used as the basis for a large range of sequencing controllers.

We have concentrated so far on outputting signals from the computer, but accepting signals is equally important, and that is what we now move on to. At this stage we will consider only those signals that are already in a digital form; that is, those that come from switches of some sort, so that they are either on or off. Simple switches and relays are obviously included here, and we can also deal with keyboards and the few direct digital output transducers, such as the optical shaft position encoder. We will deal with non-digital signals later.

We now use the **VIA** port in its input mode by setting the data direction register at **&FE62** to **&00**. This means that when we read from **&FE60** we read the input register, **IRB**, and this register can be operated in either the latching mode or the non-latching mode. Latching mode is selected by setting bit 1 of the auxiliary control register (register 11 at **&FE6B**) to '1'. If it is at '0' we get non-latching mode, and reading from port **B** reads the values present on the port pins at that time, whereas when operating in latching mode the values read are those that were present when a change of signal occurred on control line **CB1**. The sense of the change causing the latching can also be programmed, either high to low transition or low to high transition, by setting bit 4 of the peripheral control register (register 12 at **&FE6C**) to '0' if negative-going latching is required, or to '1' for a positive-going edge.

Since the port is to receive either a '1' or a '0', it might seem that the best arrangement would be to use a change-over switch, as in figure 2.11a. This will

		1			2					
		R A G			R A G					
PORT B	7	6	5	4	3	2	1	0	Hex	Time (seconds)
—	—	1	0	0	0	0	1	21	5	
—	—	1	1	0	0	1	0	32	1	
—	—	0	0	1	1	0	0	0C	5	
—	—	0	1	0	1	1	0	16	1	

(a)

```

10 DIM PROG 30      /Reserve 31 bytes for assembler at "PROG"
20 ?&FE62=255        /Set port to output mode
30 OSBYTE=&FFF4      /&FFF4 will be referred to as OSBYTE
40 FOR I=0 TO 2 STEP 2: P%=PROG  /Set assembler options and
50 [OPTI              /PC=PROG.      Start machine
60 .LIGHTS           /code section LIGHTS.
70 LDA #151          /Setup OSBYTE &97 (151) call
80 LDX #&60           /with offset &60
90 LDY &70            /and value to be written
100 JSR OSBYTE        /Call linking routine
110 RTS
120 ]                /End of LIGHTS section
130 NEXT -----     /Repeat with OPT2
140 N=500             /Setup 5 sec. period
150 ?&70=&21          /Load R1 G2 code
160 PROCchange
170 N=100             /Setup 1 sec. period
180 ?&70=&32          /Load R1A1 A2 code
190 PROCchange
200 N=500             /Setup 5 sec. period
210 ?&70=&0C          /Load G1 R2 code
220 PROCchange
230 N=100             /Setup 1 sec. period
240 ?&70=&16          /Load A1 R2A2 code
250 PROCchange
260 GOTO 140 ----- /Repeat the cycle
270 DEF PROCchange    /The CHANGE procedure
280 CALL LIGHTS       /calls the LIGHTS routine
290 TIME=0           /and waits for N/100 seconds
300 REPEAT UNTIL TIME > N
310 ENDPROC

```

/The main program runs from
line 140 through line 260.

(b)

Figure 2.10 Traffic light control. (a) Traffic light sequence. (b) Control program

work quite happily, but change-over switches are more expensive than single-throw types and the arrangement of figure 2.11b is just as effective. R is a *pull-up* resistor which ensures that the line to the port is held at logic '1' until the switch is operated.

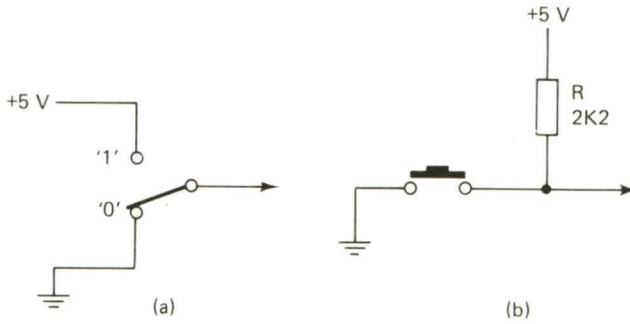


Figure 2.11 Switch inputs. (a) Change-over switch. (b) Single-throw switch

All mechanical switches suffer from *bounce* of the contacts to a greater or lesser extent, so that as the switch is operated the contacts will make but will not remain in contact at the first attempt. They bounce open again, and may bounce several times before finally resting in contact as intended. This settling period, which may extend over several milliseconds, is of no consequence in some applications, since the computer may not be programmed to check the switch value until the switch has had ample time to settle down. But the computer operates very quickly and in a few milliseconds can carry out several thousand operations, so if it is programmed to check the switch setting at short intervals it may wrongly interpret the bouncing as several independent operations of the switch. To be on the safe side we almost always *debounce* any switches, either by use of a simple flipflop or by a checking sequence in our program. The use of the flipflop is illustrated in figure 2.12 where the flipflop sets when the switch contacts first come together, and remains set until reset by the return of the switch or the application of a reset signal. The disadvantage of this method is that it requires additional circuitry, though several flipflops suitable for use in this way are packaged together for convenience, as in the SN74LS279 which contains four flipflops. As we are already making use of the computer to sense the setting of the switch,

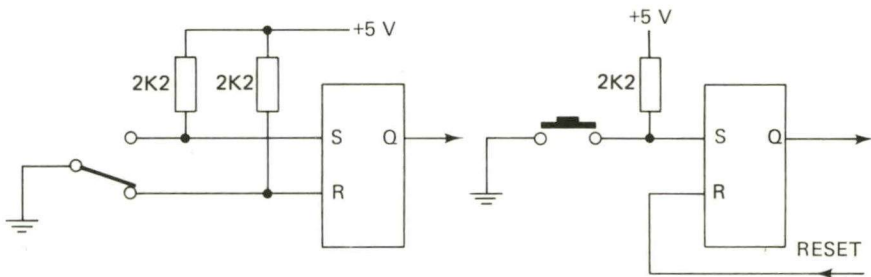


Figure 2.12 Switch debouncing by use of flipflop

```

10 DIM PROG 500           /Reserve a block of bytes for assembler
20 DIM A(10)              /Reserve array to receive switch numbers
30 T1LO=&FE64              /Define locations of VIA B Timer 1, low byte
40 T1HI=&FE65              /Timer 1, high byte
50 ACR=&FE6B               /Aux. control reg
60 FR=&FE6D                /Interrupt flg. reg
70 ?&FE62=0               /Data direction reg
80 FOR I=0 TO 2 STEP 2: P%=PROG
90   COPTI                 /Start machine code section START
100  .START
110  LDA &FE6D: STA &70     /Read input port and store in &70
120  CMP #0: BEQ START     /If zero, start again
130  LDA #0: STA ACR       /otherwise clear aux. control reg
140  LDA #&88: STA T1LO    /Setup time delay in Timer1
150  LDA #&13: STA T1HI
160  .DELAY
170  BIT FR                /Check bit 6 of flag reg. to detect end
180  BVC DELAY             /of delay and branch back until bit set
190  LDA &FE6D             /Read input port again and compare
200  CMP &70: BNE START    /with previous value; if not equal
210  RTS                  /start again.
220  J
230 NEXT                  /Exit routine with switch values in &70
240 CALL START
250 C=255-?&70           /Complement, so that an ON switch gives a "1"
260 W=0
270 FOR I=0 TO 7          /Find ON switches by ANDing each bit
280  A=2^I                /in turn (referenced as powers of two)
290  B=A AND C             /with the stored value. Write numbers
300  IF B(<) 0 THEN W=W+1: A(W)=I /of ON switches in array A
310 NEXT
320 MODE7
330 IF W=0 PRINT TAB(5,6); "SWITCHES ALL OFF" : GOTO 240
340 FOR I=1 TO W           /List ON switches
350  PRINT TAB(5,5+I); "SWITCH ";A(I);" IS ON"
360 NEXT
370 GOTO 240

```

/The main program runs from line 240 through line 370. It makes use of a machine code routine to check for any switch being set, and checks again when Timer1 interval is completed. This is shown by the Overflow flag, which is set by bit6 of the flag register when BIT test is carried out.

Figure 2.13 Debouncing program

however, we normally prefer to use the programmed checking method. The technique is to note when a switch closure is detected and then to check again a certain time later when the bounce has had time to die away. If the switch is still closed we accept that the switch has been operated. The program of figure 2.13 scans eight switches connected to the input port, and uses a time delay of eight milliseconds to allow for the bounce decay.

The outputs from an optical shaft position encoder are similar to those from mechanical switches, but being electronic switches they do not suffer from contact bounce. The encoder uses a disc, mounted on the shaft, which has *windows* laid out in a special pattern, figure 2.14. The output code is generated by opto-transistors which detect light from the sources on the other side of the disc if a window happens to lie between. Gray-coded binary, rather than the standard binary, is used to eliminate errors arising from misreading the settings when window edges happen to coincide with the detector and some ambiguous indication could result. With Gray-coded discs no two window edges coincide.

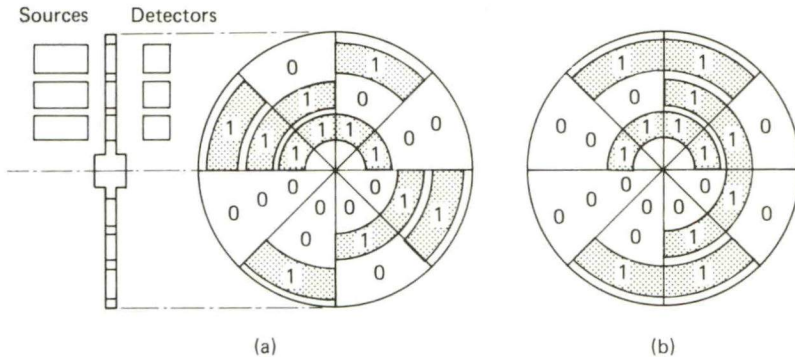


Figure 2.14 Optical shaft position encoder. (a) Binary-coded disc.
(b) Gray-coded disc

If we have a requirement to input more than eight bits of code, or to scan more than eight switches, we must either provide more input ports or must develop some external selection circuitry to allow us to check successive sets of eight bits. A useful approach is to arrange the switches in a matrix, as in figure 2.15, which is the method widely used in keyboard scanning. Here we use half the port to output a scanning pattern, and the other half to input codes from the matrix which can therefore contain up to sixteen keys. When no key is pressed, the pull-up resistors ensure that the four inputs are all at '1'. The scanning pattern is made up of four bits, only one of which is at '0'. This single '0' is shifted on each succeeding step of the complete scan so that it appears first on the A column of keys, then on the B, then on the C, and so on. After D it reverts to A and the cycle repeats. When a key is pressed, nothing happens until the scanning pattern puts the '0' on the column containing that key. The '0' is then routed through the closed contact to the input bit connected to that row and the computer detects a pressed key. The scanning pattern and inputted code together make up a 'grid reference' for the key. If key B2 is pressed, for example, the input code becomes 1101 when the output pattern is 1011 so the key B2 is defined by the 8-bit code 1101 1011. The character or value indicated by the key is found by using the code as a pointer to a *look-up* table of character codes, as in the program of figure 2.16a.

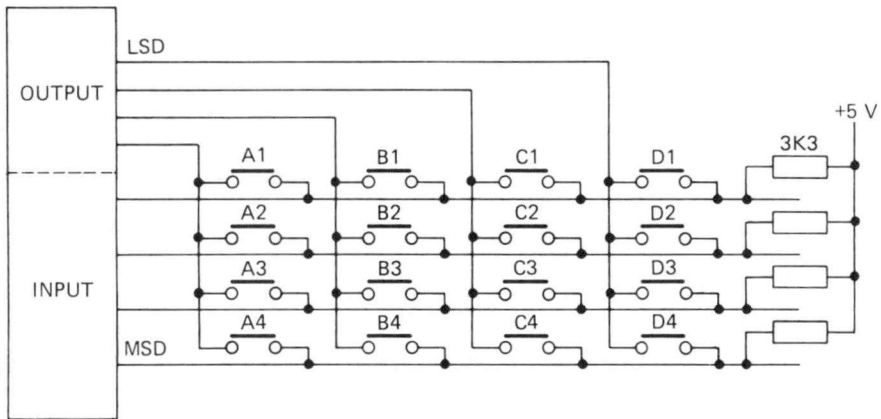


Figure 2.15 Matrix keyboard

A modified method of scanning, making use of the ability to switch the port between input and output modes, is sometimes preferred. The four column bits are first set to output 0000. If a key is pressed, say B2 again, the row input code becomes 1101. This is latched in the port register and the modes are now reversed so that the row connections become outputs, carrying the pattern 1101, and the column connections become inputs. Since switch B2 is still pressed the '0' is connected back to the column inputs to give the pattern 1011, so that the overall 'grid reference' is 1101 1011 as before. The program for this type of scanning is shown in figure 2.16b. Larger keyboard matrices can be dealt with by including extra decoder chips, though the reversing scan method just described cannot then be used. The arrangement in figure 2.17 is capable of dealing with a full 64-key QWERTY-type keyboard.

In discussing the inputting and outputting of data by the computer we have so far assumed that the program routines are entirely under the control of the computer. In other words, it is the computer that initiates all the operations. In checking the keyboard, for example, the scanning sequence is run at regular intervals by the computer, regardless of whether anyone is likely to use the keyboard or not. In simple systems, where the processor is not required to carry out many other tasks, this is perfectly acceptable. Most calculators work on this basis, with the processor idly scanning the keyboard for most of its time waiting for the next key to be pressed. But most systems can make better use of the time available, if they are notified in some way only when a key is pressed, and can ignore the keyboard until then, continuing meanwhile with some other productive program. When a key operation does occur, the running of the existing program must be interrupted until the keyboard scanning operation is completed and then the

```

10 MODE7
20 PRINT TAB(0,0);CHR$(141);"PRESS A KEY ON THE PAD"
30 PRINT TAB(0,1);CHR$(141);"PRESS A KEY ON THE PAD"
40 DIM PROG 500           /Reserve a block of bytes for assembler
50 OSBYTE=&FFF4
60 FOR I=0 TO 2 STEP 2
70   P%=PROG
80   [OPTI                /Start machine code section START
90   .START
100  LDA ##F:STA &FE62      /Set DDR to give outputs on 4 lsd
110  LDA ##97:LDX ##60:LDY #7:JSR OSBYTE      /and write 0111
120  LDA ##96:LDX ##60:JSR OSBYTE:STY &70      /Read port
130  LDA &70:CMP ##F7:BNE FIN      /Branch if key detected
140  LDA ##97:LDX ##60:LDY ##B:JSR OSBYTE/Write next pattern
150  LDA ##96:LDX ##60:JSR OSBYTE:STY &70      /1011 and repeat
160  LDA &70:CMP ##FB:BNE FIN
170  LDA ##97:LDX ##60:LDY ##D:JSR OSBYTE/Write next pattern
180  LDA ##96:LDX ##60:JSR OSBYTE:STY &70      /1101 and repeat
190  LDA &70:CMP ##FD:BNE FIN
200  LDA ##97:LDX ##60:LDY ##E:JSR OSBYTE/Write next pattern
210  LDA ##96:LDX ##60:JSR OSBYTE:STY &70      /1110 and repeat
220  LDA &70:CMP ##FE:BNE FIN
230  .FIN
240  RTS
250  J
260 NEXT I                /Exit routine with value held in &70
270 CALL START
280 IF ?&70=254 GOTO 270/Repeat scan cycle until key detected
290 N=0
300 RESTORE
310 READ A                /Look up value in DATA table
320 IF A=?&70 PROCdisplay:GOTO 270
330 N=N+1
340 IF N>15 PRINT CHR$(7):GOTO 270/Indicate multiple keypress
350 GOTO 310
360 DEF PROCdisplay
370  PRINT TAB(15,10);CHR$(141);~N /Print double-height value
380  PRINT TAB(15,11);CHR$(141);~N
390  ENDPROC
400 DATA 238,222,190,126,237,221,189,125,235,219,187,123,231
410 DATA 215,183,119

```

/The main program runs from line 270 through line 350. The machine code scanning routine is repeated until a depressed key is detected then the value in &70 is used to look up the corresponding character code in a DATA table.

(a)

Figure 2.16 (continued overleaf)

```

10 MODE7
20 PRINT TAB(0,0);CHR$(141);"PRESS A KEY ON THE PAD"
30 PRINT TAB(0,1);CHR$(141);"PRESS A KEY ON THE PAD"
40 DIM PROG 500           /Reserve a block of bytes for assembler
50 OSBYTE=&FFF4
60 FOR I=0 TO 2 STEP 2
70   P%=PROG
80   [OPTI
90   .START
100  LDA #&F0:STA &FE62           /Set DDR to give outputs on
110  LDA #&97:LDX #&60:LDY #&F:JSR OSBYTE /4 msd; write 1111
120  LDA #&96:LDX #&60:JSR OSBYTE:STY &70 /Input pattern and
130  LDA &70:AND #&F:STA &70           /store 4 lsd at &70
140  LDA #&F:STA &FE62           /Reverse DDR setting and output
150  LDA #&97:LDX #&60:LDY &70:JSR OSBYTE /returned pattern
160  LDA #&96:LDX #&60:JSR OSBYTE:STY &70 /Store final
170  RTS                           /pattern and exit
180  ]                               /routine with value
190  NEXT I                         /held in &70.
200 CALL START
210 IF ?&70=255 GOTO 200/Repeat scan cycle until key detected
220 N=0
230 RESTORE
240 READ A
250 IF A=?&70 PROCdisplay:GOTO 200
260 N=N+1
270 IF N>15 PRINT CHR$(7):GOTO 200
280 GOTO 240 -----
290 DEF PROCdisplay                /DISPLAY procedure
300 PRINT TAB(15,10);CHR$(141);~N
310 PRINT TAB(15,11);CHR$(141);~N
320 ENDPROC
330 DATA 238,222,190,126,237,221,189,125,235,219,187,123,231
340 DATA 215,183,119

```

/The main program is almost identical to that in figure 2.16a, but the machine code section uses a reversing scan method, changing the DDR setting twice each cycle.

(b)

Figure 2.16 (above and page 33) (a) Program look-up table. (b) Reversing scan pattern

program can be resumed. The same sort of reaction can be incorporated to cover any external interruption. This is exactly the way we operate in real life when we have to interrupt some job that we are doing to answer the telephone or a knock at the door.

In a computer system we think in terms of *tasks* being carried out, although, of course, the processor can run the program for only one of the tasks at a time. The main task, in most cases the user's program, is known as the *foreground task*, but many other sections of program have to be brought into operation on occasion, as and when required, and these are classed as *background tasks*. Updating of the internal elapsed time clock, for instance, is carried out under interrupt control every hundredth of a second. When a background task is to be carried out it will need to use some or all of the registers, and all the current values and settings being used by the foreground task must be preserved so that on completion of the background task we can take up the original task again exactly as when we left it. These *housekeeping* operations are organised by the interrupt servicing routine in conjunction with the interrupt hardware.

The 6502 processor has two hardwired connections which carry the signals from the interrupting circuitry. They are the *non-maskable interrupt*, NMI, and the *interrupt request*, IRQ. The non-maskable interrupt is so called because the processor cannot be programmed to ignore it, whereas the IRQ signal can be enabled or disabled under program control using special instructions. As part of the hardware operating procedure, at the end of every instruction the processor checks whether an interrupt has occurred, and, when it detects that an interrupt signal has appeared at one of the pins, it goes into a special instruction fetch routine, regardless of what the next program instruction would have been. The special instruction provided by the processor hardware is of the indirect jump to sub-routine type and makes use of an address, or *vector*, provided by the user and held at the top end of the memory. The six highest memory locations are set aside for three 16-bit vectors which point to the starts of the appropriate handling routines. In addition to the IRQ and NMI signals there are a software interrupt, using the BRK instruction, and the hardware RESET signal. The BRK instruction should not be confused with the BREAK facility on the keyboard. In fact, the BREAK key uses the RESET signal. The vector locations and their contents are shown in figure 2.18. When an interrupt occurs, the processor first inhibits any further interrupts and then carries out the indirect jump to the interrupt routine, using the vector held in the memory. In so doing it *pushes* the contents of the program counter and the status register onto the stack so that the existing values can be restored when the completion of the routine is indicated by the *return from interrupt*, RTI, instruction. The stack is a 256-byte block of the read/write memory, &0100 to &01FF, which acts as a *last in, first out*, LIFO, memory. When we push data onto the stack it is written automatically into the next location indicated by the *stack pointer*, which is then incremented by one. When we read from the stack, commonly called *pulling* or *popping*, the data is copied from the location indicated by the stack pointer after it has been decremented by one.

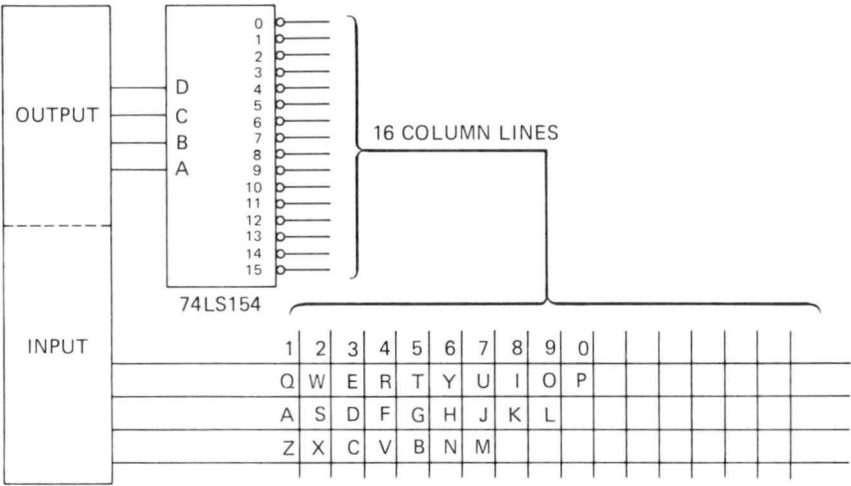


Figure 2.17 Scanning a larger matrix

We see from figure 2.18 that **IRQ** and **BRK** share the first vector, and if that vector is called it is necessary to determine which of the two caused the interrupt, since it is unlikely that they will require the same interrupt handling routine. The hardware is designed to set the **BRK** status bit, bit 4 in the status register, when a **BRK** instruction is encountered, and by reading the status register value from the stack we can check the value of the status bit. The sequence used by the BBC machine operating system, **MOS**, is given on page 466 of the *User Manual*.

There is an inherent priority in the interrupt arrangements so that simultaneous interrupts would be given precedence in the order **RESET**, **NMI**, **BRK**, and **IRQ**. The operating system of the computer is heavily dependent on interrupt action and it makes use of various facilities in different ways. Because of its hardware design we cannot choose to ignore an **NMI** interrupt signal and it is therefore used in circumstances that demand very fast response from the processor, such as the disc and Econet controllers. The start of the **NMI** service routine can be found at **&0D00**, but it is safest to consider **NMI** as being strictly reserved for system use, and to leave well alone.

The **BRK** instruction is mainly used in dealing with errors and makes use of the **BRKV** vector, which is one of several software vectors held in page 2 of RAM.

&FFFF }	High byte Start address of IRQ or BRK handling routine
&FFFE }	Low byte
&FFFD }	High byte Start address of RESET sequence
&FFFC }	Low byte
&FFFB }	High byte Start address of NMI handling routine
&FFFA }	Low byte

Figure 2.18 6502 interrupt vectors

When the BRK is detected, the vector stored at &0202 and &0203 is used to enter the error routine, having copied into &FD and &FE the address of the instruction following BRK. This is in fact a single byte instruction, &00, but it is taken to be a two-byte instruction so as to allow more flexibility in inserting BRK instructions when required. The extra byte is used to store an error code, the error message then follows the error code, and the whole thing is terminated by a zero byte.

As far as we are concerned, IRQ is the main interrupt facility and it is used extensively by the machine operating system as well as being available for the user's own purposes. There are two levels at which the system will respond, priority being given to its own requirements when dealing with the keyboard, the system VIA, the ADC chip, the serial system, sound generator, and so on. The interrupts associated with these background tasks are called *events* since they are routine signals to the operating system rather than unexpected external signals, or often catastrophic error indications. When an event is detected the event vector, EVNTV, held in &0220 and &0221, is used in conjunction with an event code in the accumulator. The event code indicates which event has occurred, as summarised in figure 2.19. The user event, code 9, can be generated during an interrupt handling routine by an OSEVEN call using address &FFBF. The value 9 should be loaded into register Y, and when the call is executed the contents of the Y register and the accumulator are exchanged.

Code	Indication
0	Output buffer, X, is empty
1	Input buffer, X is full (Y contains the character waiting to be entered)
2	A key on the keyboard has been pressed (Y contains the key code)
3	ADC conversion is complete (see chapter 3)
4	TV vertical sync. pulse has occurred
5	Internal timer has reached zero
6	ESCAPE code has been received from keyboard or RS423 channel
7	An error has been detected on the RS423 channel (Y contains the doubtful character)
8	An ECONET error has been detected
9	A user event has occurred

Figure 2.19 Event codes

If checking an IRQ interrupt indicates that it is not a BRK, program control is passed to the handling routine whose starting address is held as IRQ1V in &0204 and &0205. The routine checks first whether the interrupt is from one of the background operations. This it does by a *polling* process which looks in turn at the interrupt status of each of the known interrupt sources in the system. If it finds that the interrupt is not from one of its expected sources it passes control to a secondary routine, whose starting address is indicated by IRQ2V in &0206 and &0207. The first chip checked, and therefore the one with highest priority, if interrupts coincide, is the ACIA used in the serial communication section. Bit 7 of the status register at &FE08 is set if this chip is the source of the interrupt, and bits 0, 1 and 2 must then be checked to discover exactly which section of the ACIA generated the interrupt. If bit 7 is not set, the routine next checks the

system VIA, VIAA, which deals with the majority of the background operations. The VIA is a very complicated device from the interrupt point of view and has a much larger range of possible interrupt sources. The routine checks the *interrupt flag register* at &FE4D to discover whether bit 7 is set. If it is, it then checks which section has generated the interrupt by checking bits 0 to 6. The significance of a '1' on each bit of the flag register is listed in figure 2.20.

Bit	Cause of Interrupt	Meaning
0	Change of signal on pin CA2	A key has been pressed at the keyboard
1	Change of signal on pin CA1	A vertical sync. pulse has occurred
2	Completion of eight shifts in the shift register	Does not normally occur as the shift register is not used by the system. If an interrupt does occur here, it is passed to the user routine via IRQ2V
3	Change of signal on pin CB2 in input mode	A light pen interrupt; again not used by the system. If an interrupt does occur it is passed to user routine
4	Change of signal on pin CB1	The ADC has completed a conversion
5	Timer 2 interval is complete	The current word from the speech synthesizer is complete
6	Timer 1 interval is complete	One-hundredth of a second has elapsed. The internal clock must be incremented
7	Any bit, 0 to 6, is set	One of the above interrupts has occurred

Figure 2.20 Flag register indications

If, having polled the ACIA and the system VIA, the routine has still not found the cause of the interrupt, it then checks the interrupt flag register of the second VIA, VIAB, at &FE6D. However, as VIAB is almost entirely devoted to the user port transactions, only the setting of bit 1 is of importance to the operating system, since that pin is used to signify that a new character can be sent to the parallel printer channel. Any other interrupts are passed on by a call to the user's routine via IRQ2V. When the user routine is completed a *return from subroutine*, RTS, instruction should be used to rejoin the main system handling routine.

Unlike the NMI interrupts, IRQ interrupts can be selectively disabled, that is, *masked*, or enabled by the user. The 6502 processor has two instructions to control the acceptance of interrupts on IRQ, and a flag in the status register to indicate the current state of the interrupt mask. If the flag is at '1', interrupts will not be accepted at IRQ. The flag is set to '1', that is, the interrupts are masked, by use of the *set interrupt disable*, SEI, instruction, and it is cleared by the *clear interrupt disable*, CLI, instruction. Remember that all further interrupts are disabled automatically when an interrupt is accepted, and interrupts must be re-enabled by the handling routine. It is important that interrupts are not disabled for too long, not more than about 2 milliseconds, say, since all the background operations will quickly come to a halt; but equally we must be careful if we enable further interrupts too soon, as the handling routine can itself then be interrupted. Various operating system routines are accessible to enable or disable specific interrupts. An OSBYTE call with A = &E7, for instance, will enable or disable interrupts from the user VIA, and *FX14, N will enable event number N when used in assembler program sections. The corresponding disable call is *FX13, N.

Finally in this section we must look in a bit more detail at how the VIA can be programmed for different conditions, so that we can make use of the interrupt facilities provided by the processor. Looking back to figure 2.1, we see that there are two registers specifically concerned with the control of interrupts, the Interrupt Enable register and the Interrupt Flag register. They in turn make use of the Peripheral Control and Auxiliary Control registers. The Interrupt Flag register, register 13 at &FE6D, indicates with a '1' in the appropriate position if an interrupt has been generated. This can be an interrupt generated within the VIA, such as from the shift register or one of the timers, or it can be from outside using CB1 and CB2 (or CA1 and CA2, although on VIAB these are used in connections to the printer on PL9). Bit 7 of the Interrupt Flag register is a general indicator and sets to '1' if any other bit is set. The allocation of the remaining bits is the same as for VIAA, and is indicated in the column headed 'Cause of Interrupt' in figure 2.20. That column refers only to changes of signal but the direction of change of the signal that generates the interrupt can also be controlled by the setting of bits in the Peripheral Control register, register 12 at &FE6C. The user port makes use of port B, so we shall not concern ourselves with the port A codes on bits 0 to 3. Bit 4 relates to CB1: if bit 4 is set to '1' an interrupt will be indicated in bit 4 of the Interrupt Flag register by a positive-going transition on CB1; if bit 4 is at '0' the interrupt is indicated when a negative-going transition is detected. Any of these interrupt flags can be cleared automatically when the output register, ORB at &FE60, is selected, as it is almost certain to be during the handling routine that deals with the interrupt. We should note that, if the shift register within the VIA is in use, CB1 is used to carry the clock pulses controlling the shifting, but the interrupt flag will still respond as programmed. Bits 5, 6 and 7 relate to CB2 which can operate as either an input or an output. This is determined by bit 7, and bits 5 and 6 then indicate the mode of operation. This is most clearly shown in tabular form, as in figure 2.21.

The Interrupt Flag register, therefore, tells us which if any of the possible interrupts has occurred, but there is yet another register which allows us to disable and subsequently re-enable any of the seven interrupt sources. The Interrupt Enable register, register 14 at &FE6E, corresponds bit for bit with the Interrupt Flag register, except for bit 7 which is used to indicate whether the action is to be an enabling or a disabling of the interrupts. To enable interrupts, we set bit 7 to '1' and also set to '1' the bit positions corresponding to those interrupts that we wish to enable. Other interrupts are unaffected and remain enabled or disabled as before. Setting bit 7 to '0' has the effect of disabling the interrupts indicated by the '1' settings in the other bit positions. Thus, for example, storing &48 at &FE6E would disable interrupts from timer 1 (bit 6) and pin CB2 (bit 3).

We can illustrate some of these ideas by introducing a pedestrian-controlled button in our traffic light sequence of figure 2.10b. The amended program shown in figure 2.22c reacts to the interrupt generated when a push button connected to CB1 is pressed by the pedestrian. During the routine light sequence the program checks whether or not a pedestrian interrupt has occurred. If not, it continues

Peripheral control register

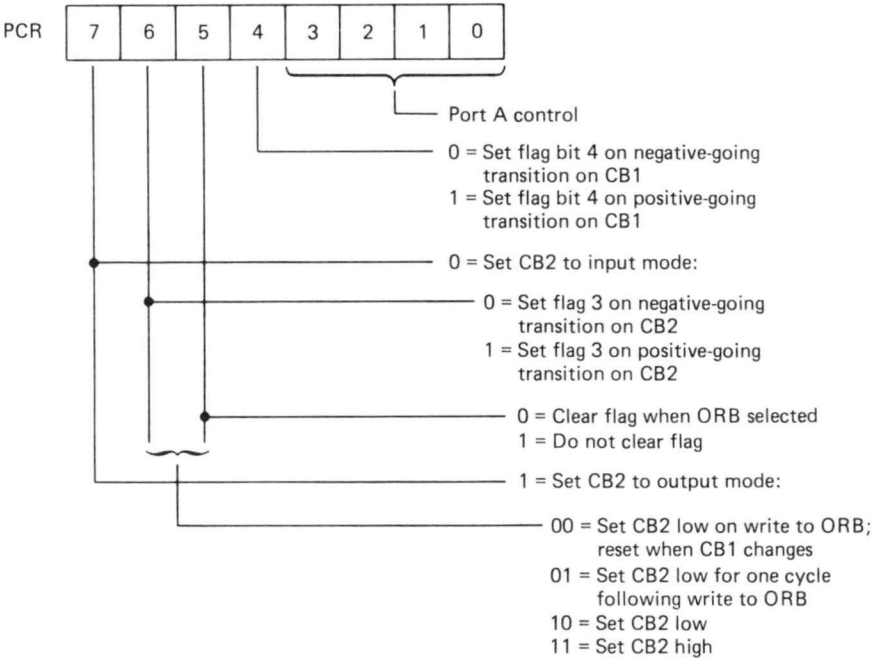
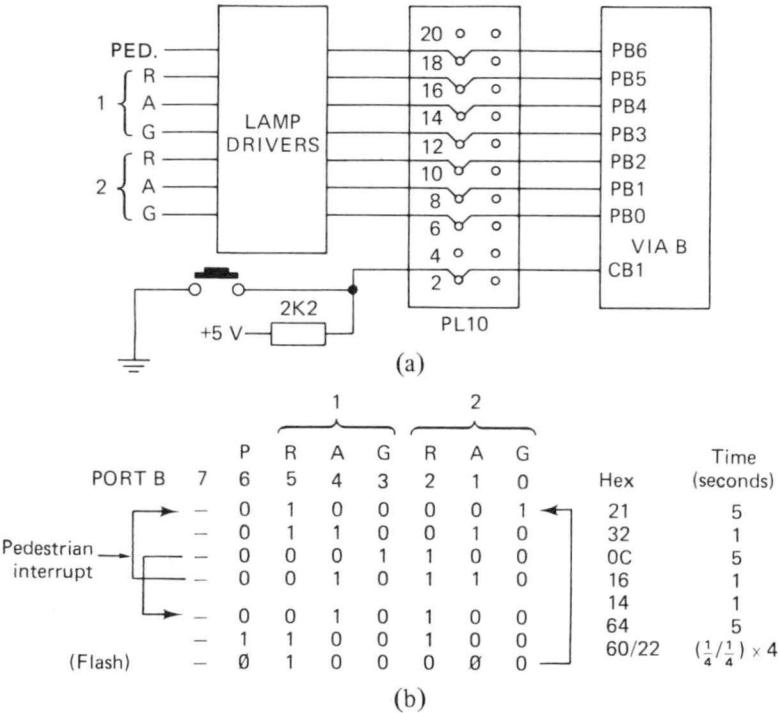


Figure 2.21 VIA peripheral control register




```

10  BUTTON=0:DELAY=0                                /Reset condition
20  ?&FE6C=0:?&FE62=255                             /Set up the VIA
30  DIM PROG 30
40  OSBYTE=&FFF4
50  P%=PROG
60  [
70    .LIGHTS
80    LDA #151
90    LDX #&60
100   LDY &70
110   JSR OSBYTE
120   RTS
130   ]
140  N=500                                           /Set up 5 sec. duration
150  ?&70=&21                                       /Load R1 G2. Change lights
160  PROCchange:PROCbutton                         /and check pedestrian button
170  N=100                                           /Set up 1 sec. duration
180  ?&70=&32                                       /Load R1A1 A2. Change lights
190  PROCchange:PROCbutton                         /and check pedestrian button
200  N=500                                           /Set up 5 sec. duration
210  ?&70=&0C                                       /Load G1 R2. Change lights
220  PROCchange:PROCbutton                         /and check again.
230  IF DELAY>0 DELAY=DELAY-1                     /While RED2 is on, deal with
240  IF BUTTON=1 AND DELAY=0 PROCpedest /pedestrian interrupt
250  N=100                                           /Set up 1 sec. duration
260  ?&70=&16                                       /Load A1 R2A2. Change lights
270  PROCchange:PROCbutton                         /and check pedestrian button
280  GOTO 140
290  DEF PROCchange                                /Define CHANGE procedure
300  CALL LIGHTS
310  TIME=0
320  REPEAT UNTIL TIME > N
330  ENDPROC
340  DEF PROCbutton                                /Define BUTTON procedure to check
350  A=?&FE6D                                       /for pedestrian interrupt on CB1
360  IF (16 AND A)=16 BUTTON=1                     /Set BUTTON if interrupt has
370  ENDPROC                                       /occurred.
380  DEFPROCpedest                                /Define PEDESTRIAN procedure
390  N=100                                           /Set up 1 sec. duration
400  ?&70=&14                                       /Load A1 R2
410  PROCchange                                   /Change lights
420  N=500                                           /Set up 5 sec. duration
430  ?&70=&64                                       /Load R1 R2 and pedestrian G0
440  PROCchange                                   /Change lights and
450  I=0                                             /flash AMBER2 four
460  REPEAT I=I+1                                  /times; 0.25 sec.
470  N=25                                           /on, and 0.25 sec.
480  ?&70=&60                                       /off.
490  PROCchange
500  N=25
510  ?&70=&22
520  PROCchange
530  UNTIL I=4
540  BUTTON=0:DELAY=2                               /Set DELAY to prevent new pedestrian
550  ENDPROC                                       /sequence for two complete light cycles

```

/The main program, lines 140 through 280, makes use of three procedures. The first, CHANGE, uses a machine code routine to output the new lights code and to time the delay specified in the main program. The second, BUTTON checks for an interrupt on CB1. The third, PEDEST, adds the pedestrian sequence.

(c)

Figure 2.22 Traffic light sequence with pedestrian interrupt. (a) Port connections. (b) Light sequences. (c) Program

with the routine light sequence. If it has, however, the program sets a variable and, when RED2 on is reached, the sequence is changed to include the three additional steps, figure 2.22b, necessary to switch both light sets to RED. After the flashing AMBER 2 and pedestrian light the main sequence is rejoined, with the re-enabling of the pedestrian interrupt being delayed for two full cycles so as not to disrupt the flow of traffic unduly.

3 *Analogue Signal Handling*

We have already noted that most values we might wish to measure with our computer are analogue in nature, in that they vary smoothly between an upper and a lower limit. But the computer itself can handle only digital values, and we therefore have to convert these variables from analogue to digital form. One of the simplest methods of conversion makes use of an accurately controlled ramping voltage, from an integrator circuit, which is fed to one input of a comparator circuit. The other input carries the analogue voltage that is to be converted. As long as the ramping voltage is smaller than the analogue voltage, clock pulses are allowed to increment a digital counter. At the start of a conversion both the ramp voltage and the counter are at zero, but the count increases as the ramp voltage grows until the comparator detects that the two voltages are equal. The output of the comparator is digital, and when it switches low the clock pulses are stopped so that the counter contains a digital value which is directly proportional to the analogue voltage. When a new conversion is required, the integrator capacitor must be discharged and the counter reset to zero. The conversion process has counted the number of regular clock pulses in a time interval that has been made exactly proportional to the analogue voltage, if the voltage ramp can be controlled accurately enough. The accuracy does rely very heavily on the stability of the clocking pulse rate and also on the linearity of the ramp voltage waveform. The linearity in turn depends on the characteristics of the capacitor used in the integrator and the stability of the reference voltage used with it. However, there are various methods that can be used to improve the stability and linearity,* and this type of converter is quite accurate enough for many applications.

A second type of analogue-to-digital converter, ADC, replaces the integrator with a digital-to-analogue converter, DAC, which generates a digital ramp voltage to compare with the analogue voltage, figure 3.1. The counter starts each conversion from zero and is incremented by each clock pulse. The DAC generates the *staircase* voltage, V_0 , proportional to the value in the counter as each pulse arrives, and when the comparator detects that the voltage V_0 just exceeds the analogue voltage it changes its output state and stops the count. The counter then holds the required digital approximation to the analogue voltage.

A third type of converter makes use of a voltage-to-frequency conversion,

* These are explained in more specialised texts, such as in G. B. Clayton, *Data Converters*, Macmillan, 1982.

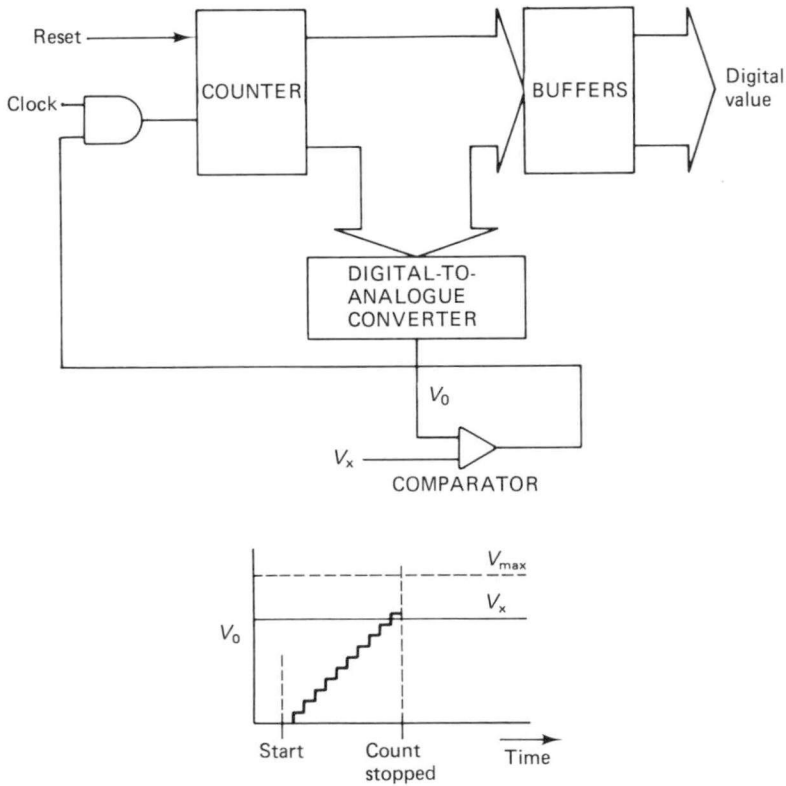


Figure 3.1 Feedback analogue-to-digital converter

rather than the voltage-to-time principle used in the previous types. Again an integrator is used to generate a ramp voltage and a comparator compares the ramp voltage against a reference voltage. When the two voltages are found to be equal, the comparator triggers a monostable circuit to generate a short pulse which operates an electronic switch and shorts out the capacitor. The ramp voltage drops to zero and, as it is now lower than the reference voltage, the cycle repeats. The result is a sawtooth waveform at the integrator, and the pulses used to short out the capacitor, figure 3.2, can also be used as the output waveform. The analogue voltage applied to the integrator circuit determines how rapidly the ramp voltage increases, so that a larger voltage will mean that the ramp voltage reaches the reference voltage more quickly, and the output pulses will be more frequent. Thus the frequency of the pulses is directly proportional to the magnitude of the analogue voltage. A considerable frequency range is available from commercial voltage to frequency converters, VFC, typically varying from about 10 Hz to 100 kHz. In order to use the converter as an ADC we simply have to count the number of pulses in a given period.

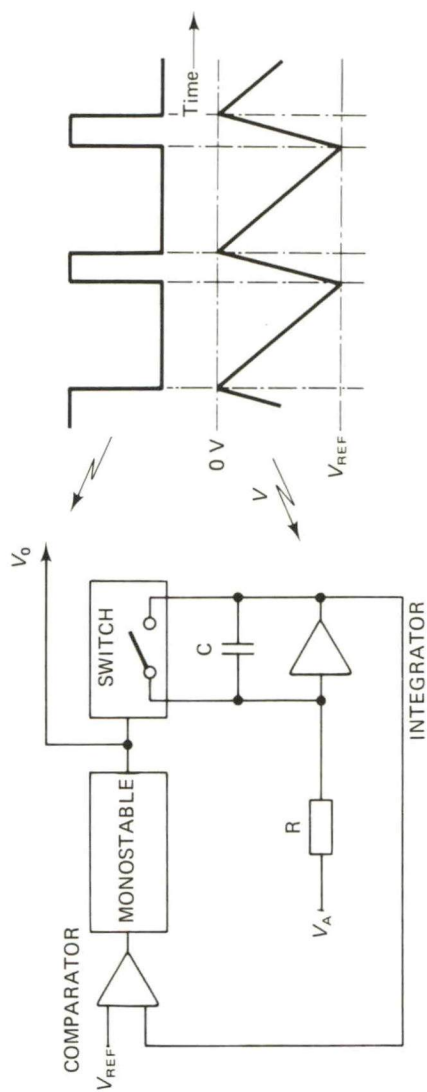


Figure 3.2 Voltage-to-frequency converter

There are many other types of ADC but these three are sufficient to allow us to explore the possibilities of use with the computer. All converters are designed to convert an analogue voltage value into an equivalent digital value, but the accuracy, speed and resolution vary widely. The *resolution* of a converter is, in general, given by the number of bits in the digital counter, and that in turn determines the frequency required of the clocking waveform, since we want to ensure that the maximum input voltage to the converter gives a maximum count on the counter. A resolution of 8 bits means that the input voltage range can be divided into 256 steps, so that each step is approximately 0.4 per cent of the full-scale value.

A 10-bit resolution gives us a step of approximately 0.1 per cent, and high-quality converters give even better resolution by using 14 and sometimes 16 bits. The *conversion time*, or sometimes its inverse the *conversion rate*, is a measure of how quickly the converter can complete the digitization process, and is clearly related to the clock rate and the resolution. Typical times vary from 50 nanoseconds in a very fast, and expensive, type, to several milliseconds. In order to ensure that the input voltage does not vary during the conversion period, and especially if we must know the exact time at which our digitized value was accurate, we sometimes make use of a *sample and hold* circuit, which, as its name indicates, samples the input voltage when strobed and holds the value until strobed again. Having achieved a digital value from our converter, with a certain resolution and conversion speed, we have to accept that it is still only an approximate representation of the analogue voltage and its accuracy is affected by many factors. Deviations from linearity in the integrator voltage ramp, variations in clock frequency, and unwanted signals (or *noise*) are often troublesome, and may mean that the reliable resolution is lower than the nominal resolution of the converter.

The BBC computer uses an ADC that is based on the first of the conversion principles described above. The other two methods have been included because they can be used very easily in conjunction with the computer, as we shall see later. In chapter 5 we shall also use a software-based successive approximation method. The NEC μ PD7002, provided in the computer, is a four-channel CMOS integrating converter using a single 5 volt supply and capable of converting to a resolution of either 8 or 10 bits. The μ PD7002 was intended originally to be a 12-bit converter, and in 10-bit mode actually gives a 12-bit value. Only one of the four input channels can be operative at any time, of course, and a multiplexer connects the selected channel to the conversion circuits. A *selection register*, figure 3.3, is used to hold the channel number and the 8-bit or 10-bit resolution indicator, and writing to this register starts a conversion sequence. On completion of the conversion the *end of conversion*, $\overline{\text{EOC}}$, line is taken low to indicate that the counter holds the digital value and can be used to provide an interrupt signal. When required, the value is read out as one or two bytes, dependent on the resolution demanded, with the lower four bits of the second byte always at '0'. The *status register* can be read at any time to discover the state of the conversion process, and figure 3.4 shows the significance of the bits. In normal usage, the

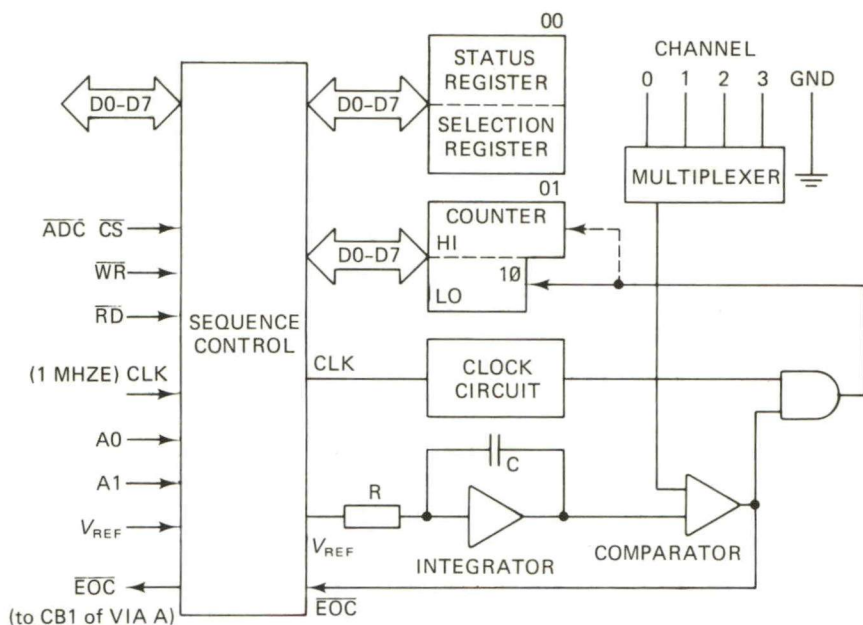


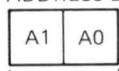
Figure 3.3 NEC μ PD7002 A/D converter block schematic

selection register is addressed by writing to $\&FECO$ and the status register is addressed by reading from $\&FECO$.

The conversion time depends on the resolution required. For 8-bit resolution and a clock frequency of 1 MHz, the time is typically 5 milliseconds, but for 10-bit resolution it doubles to about 10 milliseconds. The reference voltage in the computer is generated using three diodes in series to give approximately 1.8 volts. This defines the maximum input voltage that can be accepted for conversion, and also indicates why the highest resolution attainable in practice is limited to 10 bits rather than 12. A 10-bit resolution implies that the smallest recognisable step in voltage is $(1.8/1024)$ volts, which is approximately 1.75 millivolts. Anything smaller would not be distinguishable from noise voltage variations, though the converter will continue to present a full 12-bit value.

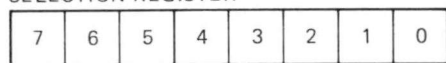
The routine control of the ADC is one of the background tasks carried out by the operating system, and, unless we instruct it otherwise, the four channels are continually scanned in turn. The converter takes 10 milliseconds to complete a channel conversion, which it indicates by generating an *end of conversion* interrupt signal to VIAA, via pin CB1. This interrupt is used to indicate an EVENT 3, with the channel number placed in the Y register. The operating system calls the event handling routine and the two-byte digital value is transferred to selected memory locations in page 2 of memory, as listed in figure 3.5. Then the REPORT location, $\&02BE$, is updated to show which channel has just completed conversion. Finally, the next consecutive channel number is written to the converter, so starting the

ADDRESS BITS



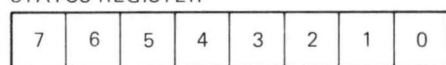
	ADDRESS	WRITING	READING
00	&FEC0	Write to selection register	Read Status register
01	&FEC1	Not selected Outputs remain in high-impedance state	Read Counter: high
10	&FEC2		Read Counter: low
11	&FEC3		Read Counter: low

SELECTION REGISTER



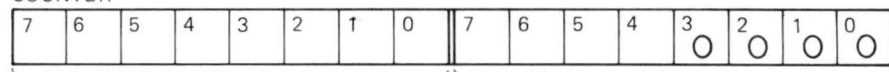
Multiplexer selection code, 0-3
 Flag input
 0 = 8 bits; 1 = 10 bits

STATUS REGISTER



Multiplexer channel code, 0-3
 Flag output
 0 = 8 bits; 1 = 10 bits
 Two most significant digits of counter
 BUSY. 0 = Conversion in progress
 EOC. 0 = Conversion completed

COUNTER



Counter: low byte
 Counter: high byte

Figure 3.4 NEC μ PD 7002 register allocations

Memory location

Channel number

&02B6	1 (0)	Low byte
&02B7	2 (1)	
&02B8	3 (2)	
&02B9	4 (3)	
&02BA	1 (0)	High byte
&02BB	2 (1)	
&02BC	3 (2)	
&02BD	4 (3)	
&02BE	Report	Number of last channel to complete

Figure 3.5 A/D converter memory locations

next conversion. We can discover the most recently measured value on any of the channels by means of the ADVAL command and a channel number, but note that the hardware recognises channel numbers 0 to 3, whereas the software refers to them as channels 1 to 4. $T = \text{ADVAL}(3)$, for instance, will set T to the value found in &02BC, high byte, and &02B8, low byte. The program of figure 3.6 will display the value on any of the four channels selected by pressing the appropriate key from 0 to 3. The channels correspond to channels 0 to 3 on the paddle connector, SKT6, connected as shown in figure 1.9b with the maximum input voltage defined by V_{REF} , which is assumed to be 1.8 volts. Then the measured voltage $V = 1.8 \times \text{ADVAL}(N)/65520$. The value used is calculated from the full 16-bit number held in memory, but we must recall that the bottom four bits of the low byte are always zero. This means that the values will always be multiples of sixteen, which is why we divide by 65 520 and not 65 536. To convert to a 12-bit equivalent numerical value we merely divide by sixteen, giving $T = \text{ADVAL}(3) \text{ DIV } 16$, for example. In most cases, however, as we have seen, the conversion is accurate only up to ten bits and it may be better to divide by 64 to give a range from 0 to 1023. If we wish to work in machine code, an OSBYTE call with $A = \&80$ has the same effect as ADVAL. The channel number must be placed in the X register before the call, and the binary value is returned to Y, high byte, and X, low byte. If an OSBYTE &80 call is used with the X register at zero, the value in location &02BE is returned to register Y and &02BE is cleared to zero. Only the three least significant digits of Y have any significance; 000 indicates no channel has completed since the value was last read, and a value from 001 to 100 indicates which was the last channel to complete. As noted in chapter 1, this call also returns to register X the status of the fire buttons on the games paddles. In this case only the bottom two bits are of significance; bit 0 indicates the state of the LEFT button, 1 = 'pressed', and bit 1 indicates the state of the RIGHT button. Using the ADVAL(0) form of command returns the value calculated from the 16-bit value formed by registers X and Y together, so to find the state of the fire buttons we must select the least significant two bits using $\text{FB} = \text{ADVAL}(0) \text{ AND } 3$; and to discover the number of the last channel to complete conversion we use $\text{CH} = \text{ADVAL}(0) \text{ DIV } 256$.

```

10 MODE7
20 PRINT CHR$(30);"WHICH CHANNEL (0-3)?";           /Home cursor
30 B=GET                                              /Input channel number
40 A=B-48                                           /and select the 4 lsd
50 IF A<0 OR A>3 GOTO 20                          /Reject if not 0 to 3
60 X=ADVAL(A+1)                                    /Read selected channel
70 VOLT=(1.8*X)/65520                               /Calculate voltage
80 VOLT=INT(VOLT*100)/100                          /Adjust to two decimal places
90 CLS
100 PRINT TAB(10,6);CHR$(141);"CHANNEL= "A        /Display value in
110 PRINT TAB(10,7);CHR$(141);"CHANNEL= "A        /double-height
120 PRINT TAB(10,10);CHR$(141);"VOLTAGE="VOLT      /characters.
130 PRINT TAB(10,11);CHR$(141);"VOLTAGE="VOLT
140 GOTO 20

```

Figure 3.6 Use of ADVAL command

<i>N</i>	<i>Effect of *FX16,N</i>
0	All channels switched off
1	Channel 1 (0) selected
2	Channels 1 (0) and 2 (1) selected
3	Channels 1 (0), 2 (1) and 3 (2) selected
4	Channels 1 (0), 2 (1), 3 (2) and 4 (3) selected

Note: Values of *N* greater than 4 are taken as 4

*Figure 3.7 Use of *FX16,N call*

The operating system, when left to itself, converts each channel in turn, taking approximately 10 milliseconds to complete each conversion. In many applications we do not need to use all four channels and call **FX16,N* is provided to allow us to switch off some, or all, of the channels. The number *N* defines how many channels are to be active, and must be placed in the *X* register beforehand. Its effect is summarised in figure 3.7. On completion of the call, the old value of *N* is returned in register *X*. When using only one channel we do not need to restrict our readings to channel 1 as directed by **FX16,N*. There is another call that can be used to initiate a single conversion on any channel. This is **FX17,N* and again *N*, with a value from 1 to 4, must be placed in the *X* register before the call. The converted value is then returned to memory in the normal way. The disadvantage, of course, is that each conversion must be triggered with the **FX17,N* call. Three other *OSBYTE* calls are available to provide information on what the analogue-to-digital converter is doing at any time: *OSBYTE &BC* returns the number of the channel currently being used in the conversion, *OSBYTE &BD* returns the number of channels being scanned, as in figure 3.7, and *OSBYTE &BE* returns the current resolution setting — *&00* or *&0C* indicates 10-bit resolution, *&08* indicates 8-bit.

When scanning only a limited number of channels, we can take readings much faster from the remaining active channels. Using only channel 1, for example, successive 12-bit conversions can be completed every 10 milliseconds, and rather faster if we specify an 8-bit resolution. If we wish to derive sufficient information from our sampled values to define in full a time-varying signal, the rate at which we must take the readings, or samples, is clearly dependent on the speed at which the signal changes. Theory shows* that it depends on the highest frequency component contained in the signal and the *sampling theorem* indicates that our readings must, in fact, be taken at a rate at least twice the highest frequency in the signal. If the fastest we can take readings on channel 1 is one every 10 milliseconds, then we are sampling at a maximum of 100 Hz and so the highest frequency component that we can cope with satisfactorily is 50 Hz. This is the

* See, for example, chapter 14 of Taub and Schilling, *Digital Integrated Electronics*, McGraw-Hill, 1977.

same as the mains frequency and, in electronic terms, is very low, but it is perfectly adequate for slowly varying signals such as those from potentiometers using dc voltages. We will come across several examples of such applications in later sections of the book.

In many applications we need to deal with signals that are varying much more rapidly than 50 Hz, and we then have to provide our own converters. As we have a versatile computer already to hand, it is sensible to make use of the computer as much as possible in the conversion process, and we can do so if we use one of the other two conversion principles that we looked at earlier in this chapter. The second method suggested uses a digital-to-analogue converter which gives a staircase ramp waveform as the input from a counter increases. A comparator then detects when the staircase voltage exceeds the unknown analogue voltage and stops the count. We can supply the count from the computer and so only need an external DAC and a comparator. The signal from the comparator can be used to generate an interrupt on pin CB1 of the user port to indicate that the count must be stopped.

The circuit of a digital-to-analogue converter is normally much simpler than for an analogue-to-digital converter, and often makes use of a resistive ladder network. This is shown in figure 3.8 where A, B and C are connected to current switches which have a very low impedance to ground in both states. They are controlled digitally to inject current into the ladder only when a '1' is present. The output voltage of the amplifier is governed by the amount of current at its input, and that is determined by where the currents are injected into the ladder. If we look at the structure of the ladder we will see that the resistances appearing at the various nodes are related in a very simple way. The resistance R_1 is clearly $2R$, so the resistance R_2 at node 1 is R_1 in parallel with $2R$, since switch C is effectively at earth. Thus $R_2 = R$ ($2R$ in parallel with $2R$), and resistance R_3 is R_2 in series with R , making $2R$ again. In fact, if we go to the corresponding point at any of the nodes we will get a resistance to the left of $2R$. A little bit of thought shows that the resistance to the right also will always be $2R$ since the ladder is

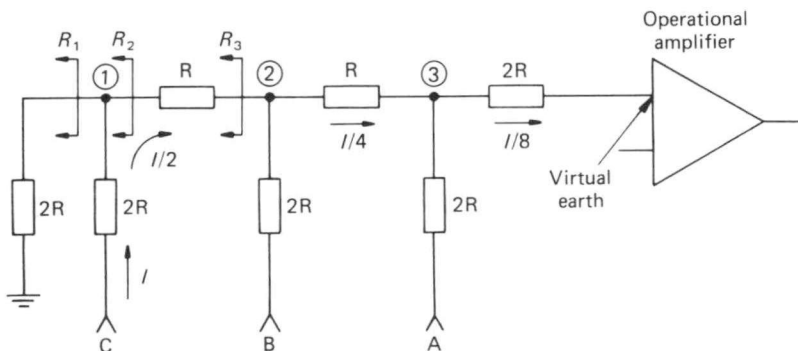


Figure 3.8 The resistive ladder network

symmetrical. And the ladder can be extended by adding $R-2R$ sections for as many inputs as we need.

Now let us see what this means for a current, I , injected into the ladder at one of the switches, switch C for example. At the first node, the resistance to the left is $2R$ and the same to the right, so the current will split equally and $I/2$ will flow towards node 2. At node 2 the resistance ahead is $2R$ with the same to the right (since switch B presents an effective earth), so the current splits equally again and $I/4$ flows towards node 3. The final split at node 3 means that $I/8$ arrives at the amplifier. If a second current, I , is injected at switch A, for example, it splits only once and $I/2$ arrives at the amplifier. The total current from these two inputs is therefore $I/8 + I/2 = 5(I/8)$, so the output is proportional to 5 which is the value indicated by the digital settings on ABC of 101. The current reaching the amplifier, and therefore its output voltage, is always proportional to the binary value on the switches. The circuit is normally manufactured in a single dual-in-line package containing the ladder network and amplifier together with the current switches and biasing network. In many cases a reference voltage is used in conjunction with the current switches, so that the voltage output is the product of the reference voltage and the digital value on the inputs. This is known as a *multiplying DAC*, MDAC. An alternative approach in making a DAC is to use a summing amplifier with each input resistor differing by a factor of 2 from the previous one. However, this necessitates a wide range of resistance values which makes severe demands on the current switch design, so the ladder network, with its two resistor values, is preferred. The resistance presented to any of the current switches is $3R$ regardless of its place on the ladder.

The third method of conversion suggested earlier approaches the problem from a different direction by using the applied voltage to control the frequency of a series of pulses. By counting the number of pulses in a given period of time we can obtain a digital value that is directly proportional to the analogue voltage applied to the voltage-to-frequency converter. Many of the relatively few direct digital transducers operate by generating pulse trains. This is particularly easy where a rotating shaft is in use, since optical or magnetic sensors can be used to detect each revolution of the shaft. The optical sensors often make use of an infra-red light emitting diode and a phototransistor housed together in one package and relying either on reflected light, figure 3.9a, or on interrupted light, figure 3.9b. Both types are readily interfaced with TTL and other logic circuits. The magnetic sensor responds to the movement of ferrous parts within a few millimetres of the pole piece at the end of the unit. This type of 'pickup' is often used to sense the teeth of a rotating gear wheel. Bear in mind that the output of a simple magnetic pickup is more a nasty looking sinewave than a recognisable series of pulses, and it must be smartened up by use of a Schmitt trigger type of circuit. Several logic gates and buffers, such as the 74LS14 and CD40106B hex Schmitt inverters, are provided with these special input circuits specifically for this purpose.

The rotating shaft idea can also be used in digital flow meters or wind velocity meters by using a propellor-driven rotor, but in applications where a rotating shaft

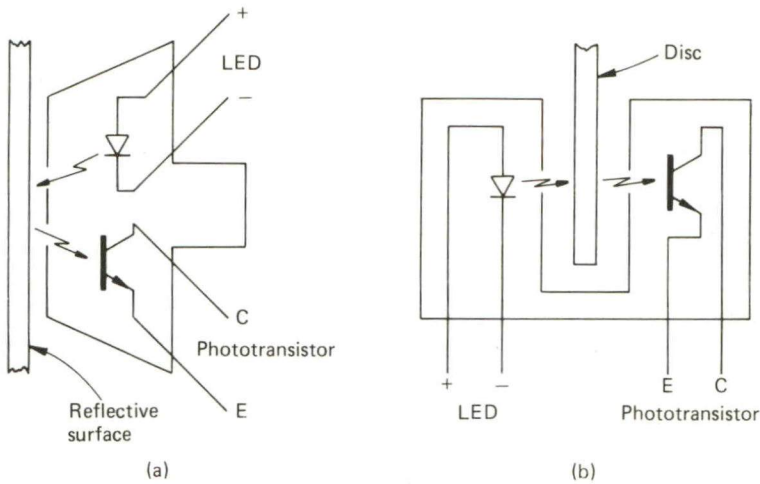


Figure 3.9 Optical sensors. (a) Reflective opto-switch. (b) Slotted opto-switch

is not part of the system, the pulse train can be generated by a *relaxation oscillator* using a *unijunction transistor*, UJT, or *programmable unijunction transistor*, PUT. The frequency of the oscillator is controlled by the value of one of the timing components. Figure 3.10, for example, shows a light-dependent resistor, ORP12, being used to control the charging rate of the capacitor. The voltage on the emitter of the UJT increases as the capacitor charges up via the variable resistance. At a certain critical voltage the UJT conducts and discharges the capacitor. The collapse of the voltage switches off the UJT and the cycle begins again. When the UJT, T1, conducts, the discharge current generates a transient voltage at the base of T2 and

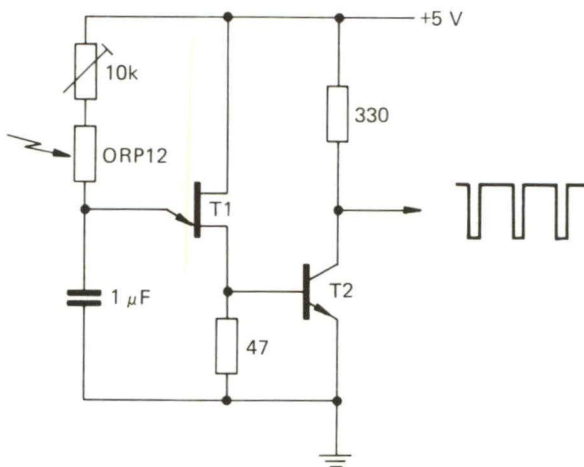


Figure 3.10 Light intensity to frequency converter

the resulting output at the collector is a negative-going pulse that is several tens of microseconds in length. The repetition rate is governed by the resistance of ORP12 which varies from several megohms to less than 2000 ohms when fully illuminated.

Regardless of how the pulses are generated, it is necessary to count them over a pre-determined period to give the final digital representation of the input variable. A simple method is to use each pulse as an interrupt signal, which indicates to the computer that the count must be incremented. We can use either CB1 or CB2 on the user port for this purpose, and can cope with positive-going or negative-going pulses by programming the peripheral control register, &FE6C, appropriately (using the details given in figure 2.21). For correct working, figure 3.11a, the interrupt must be serviced and further interrupts enabled before the next pulse appears. If the computer is heavily loaded or the pulses appear at too high a rate, we can relieve the pressure of repetitive interrupts by making use of the timer-counter provided in the VIA, as in figure 3.11b. Negative-going pulses applied to pin PB6, pin 18 of the user port, can be used to decrement the value held in the

```

10 MODE7
20 DIM PROG 100           /Reserve block of bytes for assembler
30 RB=&FE60                /Label Data Register B
40 T1CL=&FE64              / Timer1 low byte
50 T1CH=&FE65              / high byte
60 PCR=&FE6C               / Peripheral control reg
70 FR=&FE6D                / Flag register
80 IER=&FE6E               / Interrupt enable reg
90 FOR I=0 TO 2 STEP 2:P%=PROG      /Start SETUP routine
100 COPTI
110 .SETUP
120 LDA T1CL               /Clear flag bit 6 by reading T1CL
130 LDA #&90:STA IER       /Set IER to enable interrupts on CB1
140 LDA #&EF:EOR PCR:STA PCR /using negative-going edge
150 LDA #&80:STA T1CL
160 LDA #&81:STA T1CH      /Set TIMER1 to value found in &80/81
170 LDA RB                 /Read register B to clear interrupt flag
180 .LOOP
190 LDA FR:AND #&40        /Select bit 6 of flag register and
200 BEQ LOOP              /loop until bit 6 sets indicating
210 LDA #&10:STA IER       /time out, then disable interrupts
220 RTS
230 J
240 P%=&B00                /Interrupt routine is located at &B00
250 COPTI
260 PHA                   /Preserve content of accumulator on stack
270 CLC
280 LDA #&70:ADC #1:STA #&70 /Increment 16-bit counter
290 LDA #&71:ADC #0:STA #&71 /in #&70/71.
300 LDA RB                 /Clear interrupt flag
310 PLA                    /Restore accumulator content from stack
320 RTI
330 JNEXT
340 PRINT TAB(0,0);"Time period required";
350 INPUT W$               /Input required time interval and
360 !&80=EVAL(W$)          /store in &80/81.
370 !&70=0                 /Zero the counter in #&70/71
380 !&206=&0B00            /Set interrupt vector at IRQ2V

```

```

390 CALL SETUP
400 P=(?&71*256)+?&70           /Calculate number of pulses
410 PRINT TAB(0,2);"Number of pulses=";P           /and display

```

/The main program runs from line 340 through line 410.

A user supplied time value is stored and used in the SETUP routine which loads Timer1 and enables interrupts on CB1. Interrupts cause the program to vector via IRQ2V to the handling routine at &B00; the counter increments and interrupts are reenabled. When bit 6 of the flag reg is set, indicating time up on Timer1, further interrupts are disabled and the number of pulses is calculated and displayed.

(a)

```

10 MODE7
20 DIM PROG 100
30 DDRB=&FE62           /Label Data Direction register
40 T1CL=&FE64           /      Timer1 low byte
50 T1CH=&FE65           /      high byte
60 T2CL=&FE68           /      Timer2 low byte
70 T2CH=&FE69           /      high byte
80 FR=&FE6D             /      Flag register
90 ACR=&FE6B            /      Auxiliary control reg
100 FOR I=0 TO 2 STEP 2:PX=PROG           /Start SETUP routine
110 COPTI
120 .SETUP
130 LDA #0:STA DDRB           /Set DDRB for inputs
140 LDA #&20:ORA ACR:STA ACR /Set pulse counting mode(bit5)
150 LDA #&FF:STA T2CL:STA T2CH /Set Timer2 to count pulses
160 LDA &70:STA T1CL           /on input bit 6 of port B
170 LDA &71:STA T1CH           /Load and start Timer1
180 .LOOP
190 LDA #64:BIT FR           /Test bit 6 of Flag register and
200 BVC LOOP                 /loop until timeout on Timer1
210 RTS
220 J
230 NEXT _____ /Exit routine only when Timer1 has timed out
240 PRINT TAB(0,0);"Time period required";
250 INPUT W$
260 !&70=EVAL(W$)           /Set time period
270 CALL SETUP
280 T2=?T2CL+((?T2CH)*256)   /Calculate number of pulses
290 T2=&FFFF-T2              /as complement of T2 count
300 PRINT "Number of pulses=";~T2 _____ /Display value

```

/Timer2 counts down from &FFFF as pulses are received on ORB pin6. The final number of pulses is the complement of the value held in Timer2 when Timer1 times out.

(b)

Figure 3.11 Pulse counting. (a) Using interrupts on CB1. (b) Using VIA timer-counter

16-bit counter of timer 2. The counter makes use of the registers T2C-L, &FE68, and T2C-H, &FE69, and must be set to the count mode by setting bit 5 of the auxiliary control register, &FE6B, to '1'. In this mode, the counter is intended primarily to count down for a pre-determined number of pulses and then to generate an interrupt signal. However, we can read the counter registers at any time, so, if we set the initial value to its maximum, and ensure that the counting time interval is less than that needed to count to zero, we can use the counter in the normal way. Actually, even if the count reaches zero and the interrupt signal is generated, the counter continues to decrement so the value is still recoverable. The loading procedure is carried out in two parts: writing the low byte of the starting value to register &FE68 loads the value into a latch as a temporary store. When the high byte is written to register &FE69 it goes directly to the counter and the low byte is simultaneously transferred from the latch to the counter. Thus loading the high byte initiates the count — and also clears the interrupt flag.

4 *The 1 MHz Bus*

We saw in chapter 1 that, in addition to the user port, the computer has a very flexible high-speed interfacing bus operating at 1 MHz and providing buffered access to the internal address and data buses. The 1 MHz bus allows us to connect circuitry and system extensions to our computer, which are more complicated than those on the user port, but in order to make good use of its capabilities we have to understand the way in which it is structured and a few of its idiosyncracies.

The peripheral circuitry using the 1 MHz bus is intended to be housed in an expansion box external to the computer and, in common with all the interface connections except the disc, to use a separate power supply rather than drawing power from the computer. Acorn Computers Ltd, and other manufacturers,* supply standard expansion enclosures to take the Eurocard-based range of peripherals, and many of the commonly required peripheral controllers are, therefore, already available at reasonable cost. However we may still need to build our own non-standard equipment, either to mount in the expansion box or to be free-standing.

The expansion box should be connected to plug 11 of the computer by a 34-way ribbon cable of up to about 2 feet (600 mm) in length. The address, data and control signals on the bus are carried through TTL buffers at the computer end and should be buffered again at the peripheral end of the bus so as to limit the load on each line to one LS TTL (see appendix A). This restriction, together with the wired-OR arrangements on the interrupt lines, is imposed so as to allow more than one peripheral unit to be attached to the bus, and, for the same reason, 'feed through' connector arrangements are recommended, so that the additional items can be cascaded as required. The buffers used are the 74LS244 octal buffer/line driver for the address bus, and the 74LS245 octal bidirectional transceiver for the data bus. Full details of these two chips are given in appendix D.

The input-output devices using the 1 MHz bus are *memory mapped* onto pages &FC and &FD. That means that the processor refers to the registers used by the input-output equipment as though they were memory locations. The full input-output memory map for the computer is given in appendix C. OSBYTE calls are provided to initiate transfers of data on the bus, some for reading, that is, transferring data to the processor, and some for writing to the input-output registers. OSBYTE &92 (or *FX146) reads from page &FC, FRED, and OSBYTE &93

* Control Universal Ltd, for example, with their CUBE products.

(*FX147) writes to page &FC. Similarly OSBYTE &94 (*FX148) reads from page &FD, JIM, and OSBYTE &95 (*FX149) writes to page &FD. Before making the call, the location required within the page must be loaded into the X register, and the byte that is to be written must be put into register Y. If the call involves reading, the byte is routed to register Y.

The data bus drivers are enabled only when an address lying within the FRED or JIM range is detected, but the address bus drivers are permanently enabled. The direction of the data flow on the data bus is determined by the setting of the R/W (read/not write) signal, being into the computer (that is, reading) when high, and out (or writing) when low. The address bus carries only the lower eight bits of the full 16-bit address, but the page indicators, NPGFC and NPGFD, are also provided. If the user is operating in the FRED page of memory, we have seen that the address can lie between &FC00 and &FCFF. The full address is represented by NPGFC plus the eight bit value on the address bus, as in figure 4.1. We can use any of the 256 locations on the page, but Acorn have indicated their standard allocation of various ranges, and, if it is possible that the system may be extended at a later date, it is best to restrict our operations to the address ranges allocated. From the list given in figure 4.2 we see that 63 bytes in the range &FCC0 to &FCFE have been reserved for user requirements and, in practice, this is more than adequate for the majority of applications. Where more memory space is necessary we move to the JIM page and gain access to a possible 64 kbytes of external memory. However, before we leave FRED we must look in more detail at how the transfers are organised.

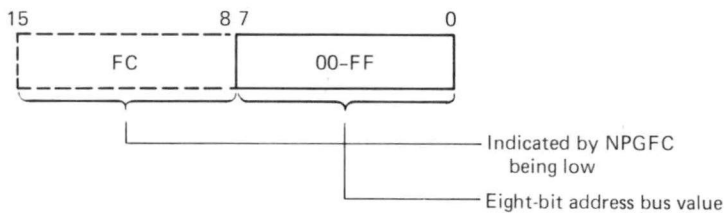


Figure 4.1 Effective address on FRED, page &FC

&FC00-&FC0F	Test hardware
&FC10-&FC13	Teletext
&FC14-&FC1F	Prestel
&FC20-&FC27	IEEE-488 interface
&FC28-&FC2F	Reserved for future Acorn use
&FC30-&FC3F	Cambridge ring interface
&FC40-&FC47	Winchester disc interface
&FC48-&FC7F	Reserved for future Acorn use
&FC80-&FC8F	Test hardware
&FC90-&FCBF	Reserved for future Acorn use
&FCC0-&FCFE	User applications
&FCFF	Paging register for use with JIM

Figure 4.2 Allocation of addresses on FRED, page &FC

It was explained in the first chapter that the clocking rate on the 1 MHz bus is half the rate of the microprocessor clock, so as to allow the use of many of the relatively slow peripheral devices which are widely available. The normal clocking arrangement with the 6502 microprocessor is that a memory operation, such as an instruction fetch or reading or writing data to memory, occupies one cycle of the clock waveform. In effect, the clock pulse acts as an indication that the address signals on the bus at the time it is high constitute a valid address. The external 1 MHz clock signal, 1MHZE, is derived from the same 2 MHz waveform that controls the microprocessor in normal operation via the 74LS32 OR gate, gate 1 on figure 4.3. However, when a 1 MHz bus address is detected (that is, &FC or &FD), the pulse-stretching circuitry is triggered and, in effect, one cycle of 1 MHz waveform is inserted via the OR gate to give the waveform shown as ϕ_{IN} in figure 4.4. Thus for the duration of the 1 MHz bus operation the microprocessor is running at 1 MHz, not 2 MHz. This approach works perfectly well as far as the microprocessor is concerned, but creates a couple of problems when we try to carry out memory transfers at the other end of the bus. In all systems we make use of the *valid memory address*, VMA, type signal, to ensure that the circuitry reacts to an address on the bus only when it is intended to be there and is stable. At other times the signals on the various address lines are changing in a haphazard way, and we may briefly detect an address that we are seeking, which in reality is merely a chance occurrence on the bus. These spurious pulses or spikes from the address detection circuits are normally of no significance because the processor clock signal is low, so indicating that the address is not valid and should be ignored. But the external equipment, which relies on the 1MHZE clock signal, will see alternate spikes as valid indicators since its clock signal is then high. We must therefore suppress the awkward spikes.

A second problem arises when a device using the 1 MHz bus is selected while the 1MHZE clock happens to be high. When NPGFC (or NPGFD) goes low, the device will be enabled immediately, as required, but, because the processor clock is stretched, it will be accessed again when the 1MHZE clock goes high the next time. This is shown in the waveforms of figure 4.4. Of course, in many cases this double enabling will not matter, but in some cases it can have awkward side-effects, such as clearing an interrupt flag prematurely.

Several methods have been suggested for overcoming these problems, by generating a 'clean' version of the selection signal; NPGFC then becomes CNPGFC, the C indicating 'clean'. A similar operation is required for NPGFD. Some of the proposed circuits, however, merely shift the problem from occurring when 1MHZE is high to when 1MHZE is low, and are not recommended. Two circuits have been found to work well. The first uses a gated set-reset flipflop formed from the gates contained in a single quad two-input NOR package, 74LS02, as shown in figure 4.5a. This circuit prevents the selection signal going low until the 1MHZE signal is low. The method is almost always satisfactory, but it does allow the spurious pulse preceding the selection signal to get through. A second circuit, figure 4.5b, is more thorough and uses a single D-type flipflop,

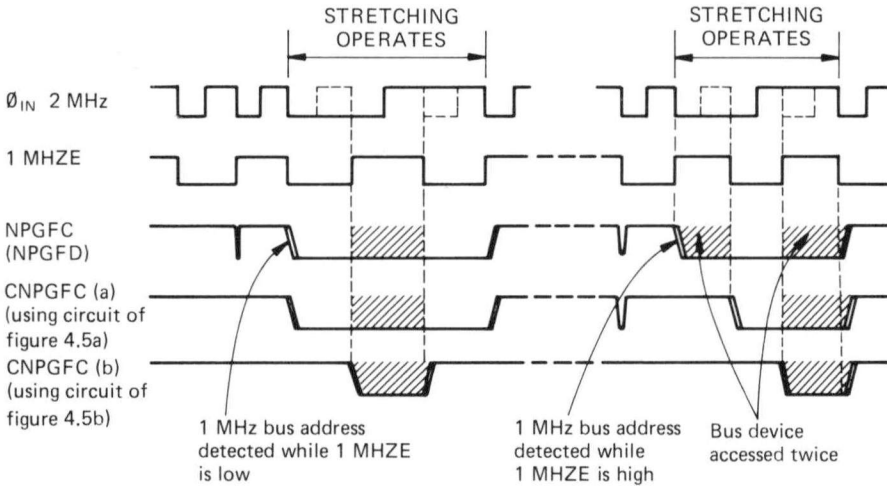


Figure 4.4 Bus timing waveforms, showing double access if 1MHzE is high when the bus address is detected

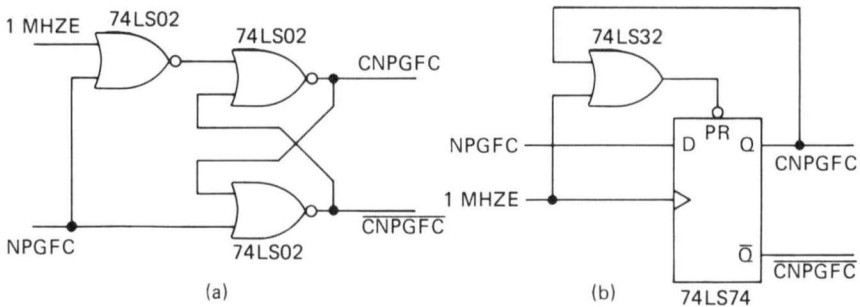


Figure 4.5 Generation of the 'clean' selection signal

74LS74, and an OR gate, 74LS32, to filter out the spikes and to generate a single width pulse under all conditions.

Where more memory than the 63 bytes available on page &FC is required, we must move to page &FD, JIM. Here the external equipment can use up to 64K locations, which still needs sixteen address bits, but, in this case, since the locations are all external, all sixteen bits have to be defined and we have only the lower eight bits available on the bus. The problem is overcome by making use of an external register which holds eight bits sent over the data bus. This *paging register* provides the upper eight bits of the address, which are known as the *extended page number*, and, together with the lower eight bits on the address bus, gives us the full 16-bit

address, as in figure 4.6. The paging register itself must have an address so that we can select it when we wish to change the external page number, and it is allocated the top location in the FRED page, that is, &FCFF.

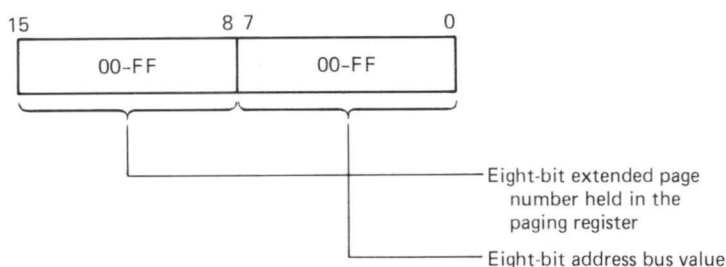


Figure 4.6 Effective address on JIM, page &FD

Commercially available expansion boxes include the paging register on the backplane of the box. It takes the form of a 74LS273 eight-bit D-type latch which is controlled by a latching signal generated by a 74LS133 NAND gate. The gate output signal, FCFF, goes low only when A0 to A7, WRITE, CNPGFC and the 1MHZE clock are all high, since this concurrence indicates a write operation to address &FCFF. The data on the data bus is then latched into the 74LS273, since the control signal goes high again when the 1MHZE clock pulse ends. The register is cleared whenever a reset, NRST, signal occurs. The overall backplane arrangements, including the paging register, are shown in schematic form in figure 4.7 but for full details the appropriate Acorn application note* should be consulted.

Two further features should be noted with respect to operation on page &FD. Read and write strobes, NRDS, NWDS, are generated from the 'clean' page select signal and the 1MHZE clock. Also a block 0, BLK0, signal is provided, which is, in effect, an NVMA (FD) signal present only when address bits A12 to A15 are zero. In the same way as we have found it useful to think of 64K locations as 256 pages of 256 bytes, it is also useful on occasions to consider *blocks* of pages, rather like chapters, and, in this context, we have sixteen blocks of sixteen pages. The block number is given by the top four address bits, so it also appears as the most significant of the four digits in the hex address. For example, &072A lies in block 0.

The addressing arrangements provided allow selection of any external location from &0000 to &FFFF, but, again, the manufacturers have reserved part of the available address space for specific purposes. User applications should be restricted to the upper half of the address range, using blocks &8 to &F. Since the paging register is a write-only memory location as far as the computer is concerned, it has

* BBC Microcomputer—Application Note No. 1 – 1 MHz Bus. Part No. 40700, Acorn Computers Ltd.

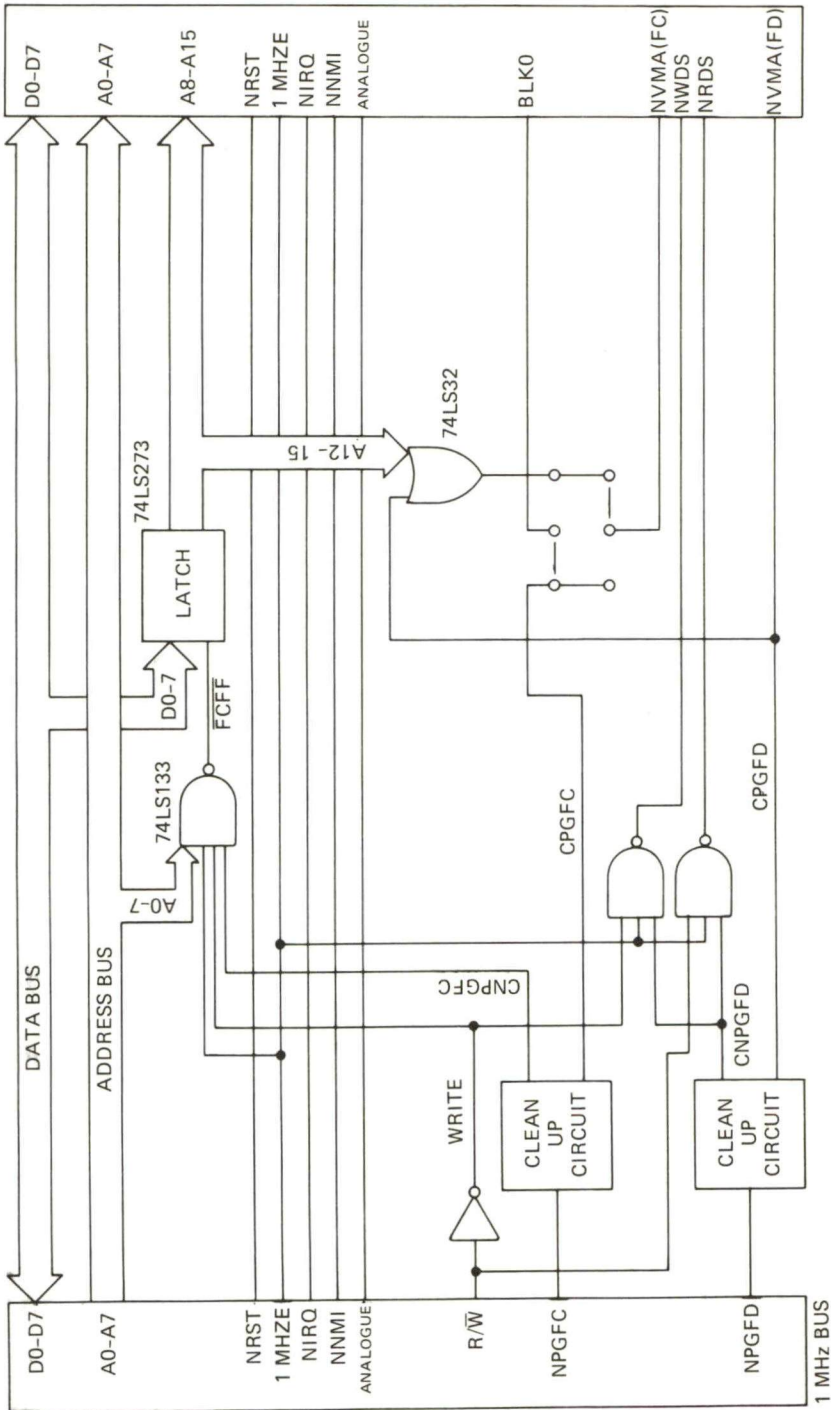


Figure 4.7 Extension box backplane logic

no way of checking what the value is at any time. When interrupts occur it would normally be arranged that the page register value would be read and pushed onto the stack, and then be rewritten on completion of the interrupt handling routine. But reading the register is not possible, so a copy, or *image*, is held at a specified location in the internal RAM, where it can be read when required. So that it is still within the input-output processor's map, even when a second processor is in use, the image register is located in page zero, at &00EE. The image must, of course, be updated every time the page register is altered, and, to prevent confusion if an interrupt occurs when one and only one of the locations has been modified, the image register at &00EE should always be changed before the actual register at &FCFF. Reversing the procedure would mean that, in such circumstances, the page register would be restored to its old value still held in the image register, when the interrupt sequence was completed.

The potential uses of the 1 MHz bus are pretty well unlimited: FRED page gives us 255 directly usable locations, and the JIM page extends the available memory mapping space by another 64K through use of the paging register. Most of the locations on FRED are reserved for use with specific system extensions such as a Teletext decoder or a Winchester disc unit. But one of the extension units, the IEEE-488 bus interface, is unlike most of the others in that it provides a gateway into any cluster of other instruments interconnected by the IEEE-488 bus, figure 4.8. This bus was developed originally by the Hewlett-Packard Corporation as their HP-IB (interface bus) to allow standardised interconnection of the increasing number of 'intelligent' instruments used in laboratories, control systems and automatic test equipment. Instruments with widely varying operating speeds were able to exchange data using a standard procedure, and the idea was quickly taken up by other manufacturers, being known as the GP-IB, General Purpose Interface Bus, by those who did not wish to publicise Hewlett-Packard.

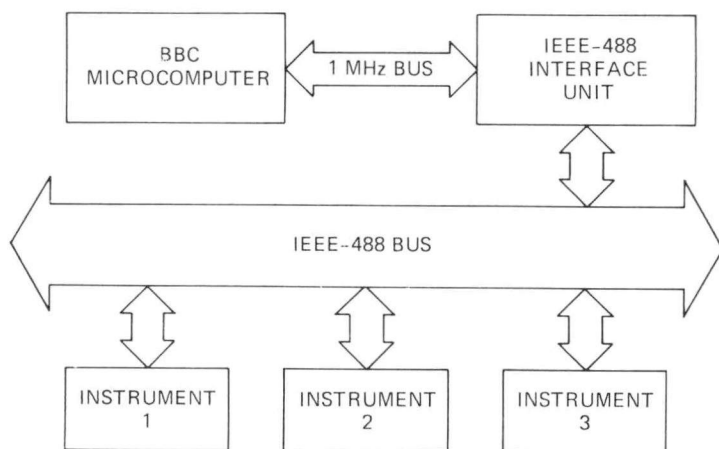


Figure 4.8 IEEE-488 interface arrangements

It was eventually standardised as ANSI/IEEE Std 488-1975 and has had only minor modifications since. A slightly different form is known in Europe as standard IEC-625-1.

When in use, the units connected to the bus are classified as the bus controller, talkers and listeners. Some instruments, such as printers, are always listeners, some are always talkers, but some, such as video display units with a keyboard and display, can be either a listener or a talker, depending on the transaction required. Only one talker may be active at any time, but the number of listeners is restricted only by the addressing arrangements on the bus. The original standard provided for a maximum of fifteen device addresses, but the current arrangement contains an extended addressing facility to handle a very much larger number.

In attempting to understand the working of the bus, we must remember that there are two levels of operation, which we shall refer to as the protocol level and the physical level. The *physical level* covers the operation of the bus signals themselves in actually transferring messages along the cables between units. The *protocol level* consists of the set of rules governing how the transactions are set up and managed. For instance, suppose we have a system in which the IEEE bus connects several instruments, including perhaps a digital voltmeter, a printer, a programmable power supply and the computer, which acts as the system controller. When we switch on, the controller initialises the system by sending an *interface clear*, IFC, signal to all interface units, followed by a coded message, *device clear*, DCL, to set the units to a known quiescent state, rather like a reset on the computer. The controller then has to set-up, or program, the units by sending a sequence of messages and data to each unit in turn. For each unit we need a *listen address* message to alert the unit, then the necessary programming data word and finally an *unlisten address* message to tell it to stand down again. At some stage in an experiment, we may wish to send data from the digital voltmeter to the printer, so the transaction must be set-up by the controller. The first thing is for the controller to send the listen address message to the voltmeter, followed by the data word needed to activate its digital output circuitry, and then the unlisten command. Next it sends the listen address message for the printer, and follows that with the talk address message to the voltmeter. The 'route' is now set up and the voltmeter feeds a succession of data words onto the bus and the printer accepts them from the bus. An 'end of data string' signal is fed onto the bus when the last byte is transmitted, to indicate that the transaction is completed.

The protocol sequences are specified in terms of bus signals, such as the interface clear, IFC, signal, and bus messages, such as the listen address, talk address, data word, unlisten command, and so on. The messages make use of the data lines on the bus together with one of the control lines. The complete set of signal lines on the bus comprises eight data lines, five bus management lines and three data transfer control lines. There are also eight ground return lines. Specific pins on a standard 24-way connector are allocated as shown in figure 4.9. The standard connector is stackable to give the necessary feed-through facility, and is held in place by captive locking screws which are often finished in black to indicate the standard metric thread since some early instruments used non-metric threads. The

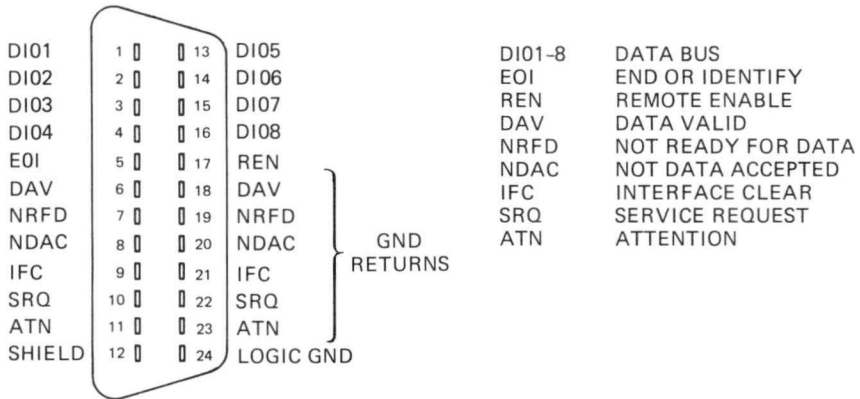


Figure 4.9 Standard pin connections for IEEE-488 bus signals

signals on the data lines are interpreted as an 8-bit byte of data unless the ATN line is at logical '1', in which case they are intended as a 7-bit command. In general, the commands are of four classes indicated by bits 6 and 7* of the byte:

- 00 defines a universal bus management command
- 01 is a listener address
- 10 is a talker address
- and 11 is a secondary address or command.

When an address is indicated, the actual value is held in bits 1 to 5, but the address with all ones is reserved as a universal unlisten or untalk command. We can, therefore address up to 31 units directly, but, by using the secondary addressing facility, each primary address can contain up to 32 secondary addresses. Figure 4.10 shows the sequence to carry through the transfer of binary-coded data from the volt-meter to the printer.

The remaining bus management lines, REN, SRQ, and EOI, are used in rather more specialised operations. Remote enable, REN, can disable any local controls on an instrument, so putting it under bus control. Service request, SRQ, is an interrupt type signal, and end or identify, EOI, indicates the last byte in a multiple-byte data transfer, but is also used by the controller when identifying interrupting units.

The electrical signals on the bus are specified in terms of TTL levels, using negative logic. Thus a signal between 2 and 5 volts is the high or '0' state, and a signal between 0 and 0.8 volts is the low or '1' state. The active-low convention is used to allow wired-OR operation, and open-collector drivers are used in most cases, though three-state drivers can be used on the data lines and some of the control lines. The total bus length is restricted to 20 metres, and each signal line

* The data bits are specified as 1 to 8, not 0 to 7.

	ATN	EOI	IFC	DATA BITS								
				8	7	6	5	4	3	2	1	
IFC	0	0	1	0	0	0	0	0	0	0	0	Reset
DCL	1	0	0	0	0	0	1	0	1	0	0	Device clear
LA (Voltmeter)	1	0	0	0	0	1	0	0	1	0	1	Listen address
DAB	0	0	0	1	1	0	0	0	0	0	0	Voltmeter set-up code
UNL	1	0	0	0	0	1	1	1	1	1	1	Unlisten
LA (Printer)	1	0	0	0	0	1	0	0	1	1	0	Listen address
TA (Voltmeter)	1	0	0	0	1	0	0	0	1	0	1	Talk address
DAB	0	0	0	0	0	1	1	1	0	0	0	Data byte
DAB	0	0	0	0	0	1	1	0	1	0	1	Data byte
DAB	0	0	0	0	0	1	1	0	1	1	1	Data byte
⋮												
DAB/EOI	0	1	0	0	0	1	1	0	0	1	0	Last data byte
UNT	1	0	0	0	1	0	1	1	1	1	1	Untalk
UNL	1	0	0	0	0	1	1	1	1	1	1	Unlisten

Figure 4.10 IEEE bus transactions for data transfer

must be correctly terminated by a resistive load, with a diode clamp to prevent negative voltage excursions. By using a terminated bus, the data transfer rate can be taken up to 2 Mbytes per second, but the actual overall transfer rate is governed by the speed at which the slowest unit involved in the transfer can provide or absorb the data. The matching of widely differing transfer capabilities is achieved by means of a *handshake* arrangement involving the three data transfer control signals, DAV, NRFD and NDAC. The main principle of any handshaking transfer is that each side involved must respond to the last action of its partner before the transfer can proceed. Let us assume, by way of illustration, that we have one talker and one listener already addressed by the controller, as with our voltmeter and printer example a little earlier. The talker having outputted the current data bits, and sensing that the listener is ready for data, because NRFD is high and NDAC is low, sets DAV low. The listener detects the change on the data valid, DAV, line and so pulls NRFD low. When the data word has been absorbed into the buffer, NDAC is allowed to go high. The talker holds the data steady on the bus until it senses that NDAC has gone high, then sets DAV high and replaces the data with the next word. The cycle now repeats. The procedure appears somewhat complicated at first, but is in fact simple in concept and is widely used, not just with the IEEE-488 bus, since it removes the problems of differing delays on long interconnection paths. Data is accepted by the listener only when the negative-going edge of DAV arrives at the listener, and is changed only when the talker in turn receives the response from the listener. The interdependencies are shown diagrammatically in figure 4.11. The active-low wired-OR arrangement ensures that several listeners can be active at once, since the control line will remain low until the last listener is ready to let it go high. One unfortunate feature of the handshake method is that a transfer can ‘hang-up’ if a response is not forthcoming – we might have forgotten to switch the printer on! – and a *watchdog*

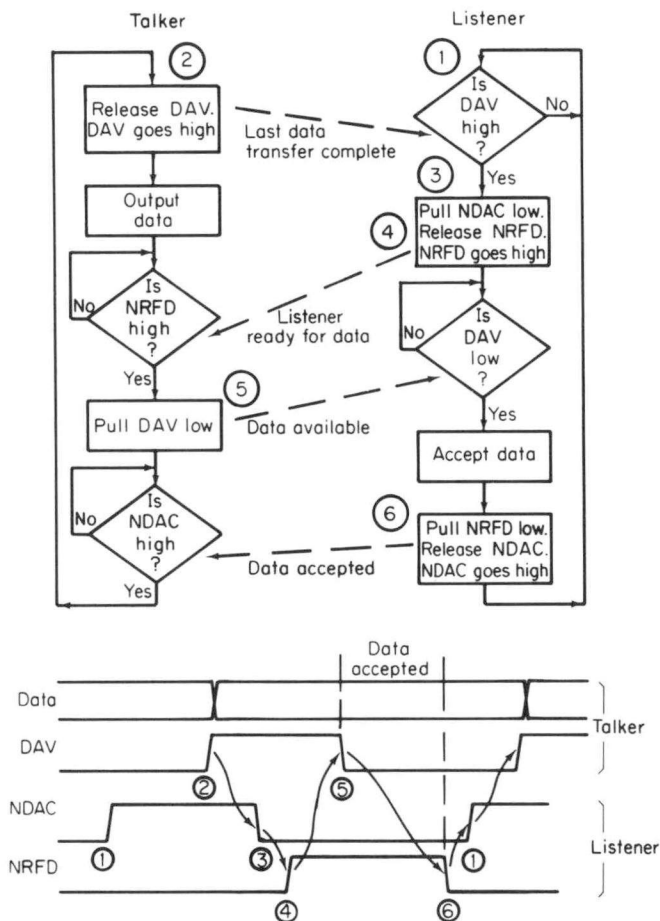


Figure 4.11 Flowchart and timing waveforms of the IEEE-488 standard communication procedure, showing the 'handshake' principle

timer must be built in to prevent an endless wait. As each signal is generated, the timer is restarted and, if it 'times out' before a response is received, then the whole transfer is aborted.

The IEEE-488 interface is available from Acorn as a ready-to-use unit. It comes complete with a ROM, which contains the filing system and will handle up to 14 pieces of equipment. In many other applications, however, we may wish to use the 1 MHz bus with our own non-standard interface, so let us now look at how to control data transfers across the bus, by making use of the 'cleaned up' selection signals that we derived earlier. In order to illustrate the principles, we will assume that the requirements are very simple: the external circuitry requires only one input port and one output port. We could use our old friend the Versatile Inter-

face Adaptor here since it provides two ports, but we have already discussed the VIA in some detail, so it is perhaps a convenient place to introduce another similar device, the Motorola MC6821, which is known as the *Peripheral Interface Adapter*, PIA. This was actually the forerunner of the VIA and is worth considering since, although not provided with so many features, it is only about two-thirds the price of the VIA. It also provides two peripheral ports, A and B, and each has associated with it a data direction register, DDR, and a control register, figure 4.12. Although the internal data buses are shown as bidirectional for clarity, there are in fact two unidirectional buses. The DDR is used to set up each bit of the port as either input or output, '0' for an input. The A and B ports differ slightly, so that, although they can both be used as either inputs or outputs, port A is provided with internal pull-up resistors to allow its use as an input port without the need for external components, and port B has three-state drivers allowing its use directly as an output port with a current-sinking capability of 10 milliamps, and a current-sourcing capability of 1 milliamp at 1.5 volts. The handling of interrupt signals is also easier if the ports are used in this way. Two control signals are associated with each port, for use with external circuitry, and they are controlled in turn through the various bits of the control register.

The PIA has six internal registers but it is mounted in a standard 40-pin dual-in-line package and, when all the data bus and port connections and the necessary control pins have been allocated, there are only two pins available for internal register selection. The problem is overcome by carrying out the selection in two parts, using the two pins, RS1 and RS0, and one of the bits of the control register. RS1 selects either the A or the B side of the PIA, figure 4.13, and, if RS0 is at '1', the control register is uniquely selected. However, if RS0 is at '0', it indicates either the data direction register or the peripheral register, and the particular one intended is determined by the setting of bit 2 of the appropriate control register, CRA2 or CRB2. Thus in programming the PIA to operate in a certain way, we must address the first word to the control register and make sure that bit 2 is at '0'. We then address the data direction register and set up the port bits as inputs or outputs. Finally we go back to the control register to set bit 2 to '1' so that future accesses will be to the peripheral register. Note that a low on the $\overline{\text{RESET}}$ input has the effect of setting all internal registers to zero, so bit 2 of the control register is then already at '0'. Similarly the data direction register will already contain '0's, indicating an input port, and need not be re-addressed if that is what is required.

The remaining bits of the control register are devoted to controlling and responding to signals on the control lines, CA1 and CA2 (or CB1 and CB2). Lines CA1 and CB1 are input-only lines and each sets bit 7 of its control register when it receives the appropriate signal, which is indicated by the code in bits 0 and 1 of the control register. CA1, for example, can be programmed to respond to either a negative-going edge or a positive-going edge, as shown in figure 4.14. By setting bit 0 to '1' we can enable the $\overline{\text{IRQ}}$ output to provide an interrupt signal to the computer, but, if we prefer, we can disable the $\overline{\text{IRQ}}$ signal and monitor

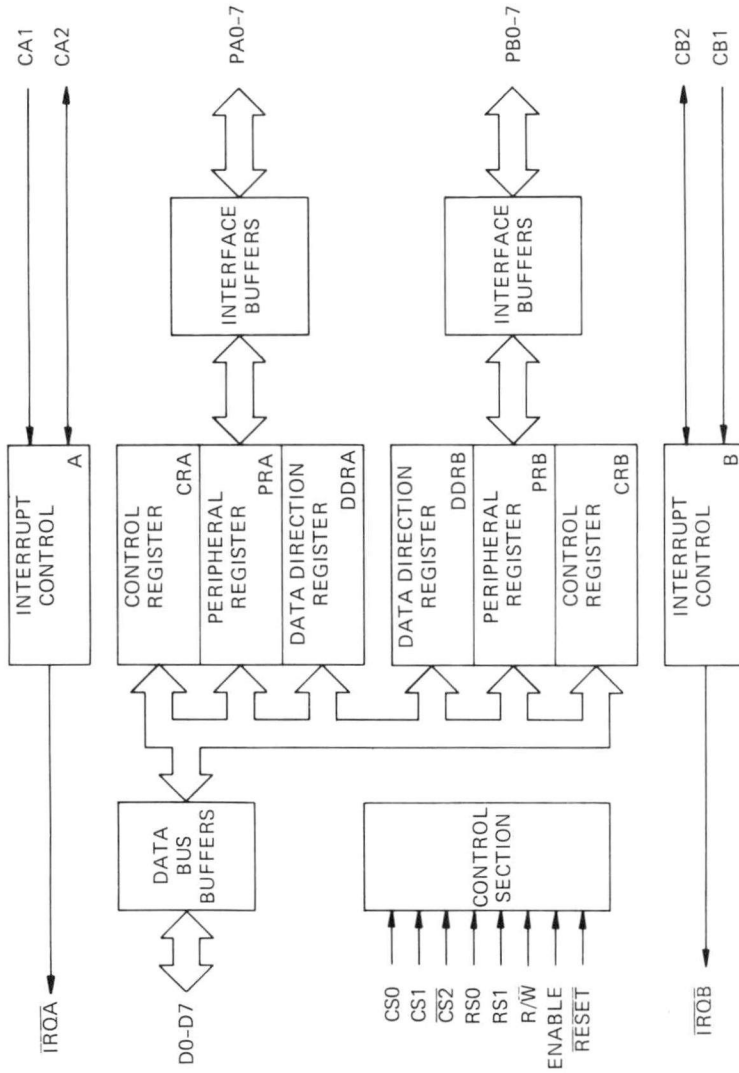


Figure 4.12 PIA schematic

RS1	RS0	CRA2		RS1	RS0	CRB2	
0	0	0	DDRA	1	0	0	DDRB
0	0	1	PRA	1	0	1	PRB
0	1	0	CRA	1	1	0	CRB

Figure 4.13 PIA register selection

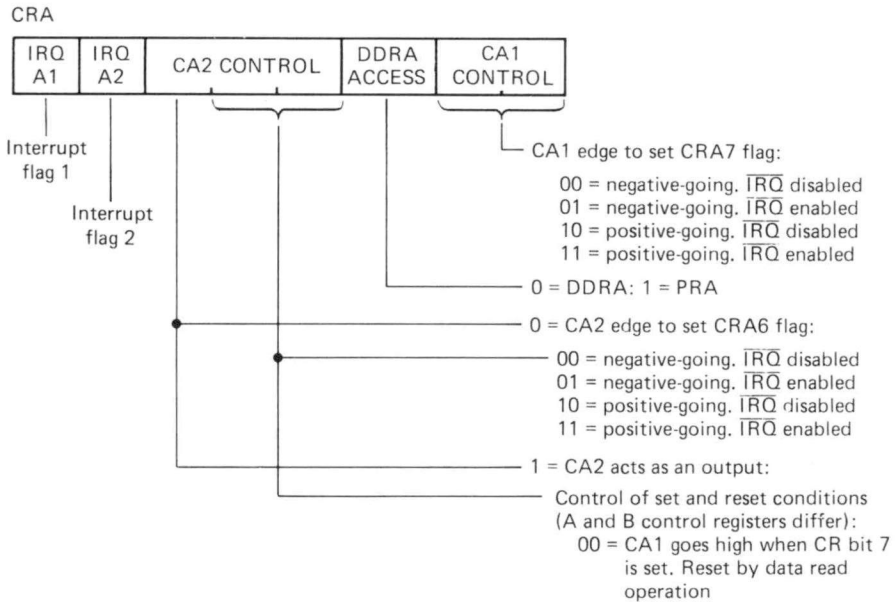


Figure 4.14 PIA control register, CRA

the operation of input CA1 under program control by checking the flag bit, CRA7. A similar arrangement holds for the B side. CA2 can act either as an input or an output, controlled by bit 5 of the control register. When set as an input it operates in the same way as CA1 but uses CRA6 as the flag bit. CB2 operates in the same way. The flag bits are cleared automatically during a read operation of the peripheral register, PR. When set as outputs, CA2 and CB2 are intended to be used in controlling peripheral data transfers, and they operate in slightly different ways. The possible operating modes are too numerous to be summarised effectively here, and the data sheets* should be consulted when the complete specification is required.

Having decided to use a PIA in our simple application, we must ensure that the device is correctly connected to the bus and that an initialisation procedure is included in our program to set up the ports as we need them. The registers of the

* Motorola MC6821. *Microcomputer Components*, Motorola Inc., 1979.

PIA are to be mapped onto page &FC and we should use locations within the allocated block which begins at &FCC0. The individual registers could then be selected according to the values on address bits 0 and 1 by connecting them to RS0 and RS1 respectively. A convenient arrangement would be

- &FCC0 selects DDRA/PRA
- &FCC1 selects CRA
- &FCC2 selects DDRB/PRB
- &FCC3 selects CRB.

The memory space that we are using lies at the top of the page, so in most cases it is sufficient to detect when an address on page &FC is &C0 or higher. We can do that by using address bits 6 and 7 as chip select signals, since any address in which they are at '1' will be &C0 or higher. However, if page &FD is likely to be used in our system, we must also exclude from our chip selection range the address of the paging register, &FCFF. An address with a '0' on any address line from A2 to A5 will indicate an address other than &FCFF, and, since we need only a few locations, we can afford to use a very restricted code detection, and look for a '0' on only one of the address lines. The circuit of figure 4.15 makes use of CS2 on the PIA to look for '0' on A5, so with A6 and A7 connected to CS0 and CS1, the device responds only to addresses on page &FC that start with &C or &D, and not &E or &F. The inverse of CNPGFC is connected to the Enable input to control the timing of the transfers. The timing waveforms involved are shown in figure 4.16 and, as with any bus organised system, it is important that the timing requirements are satisfied.

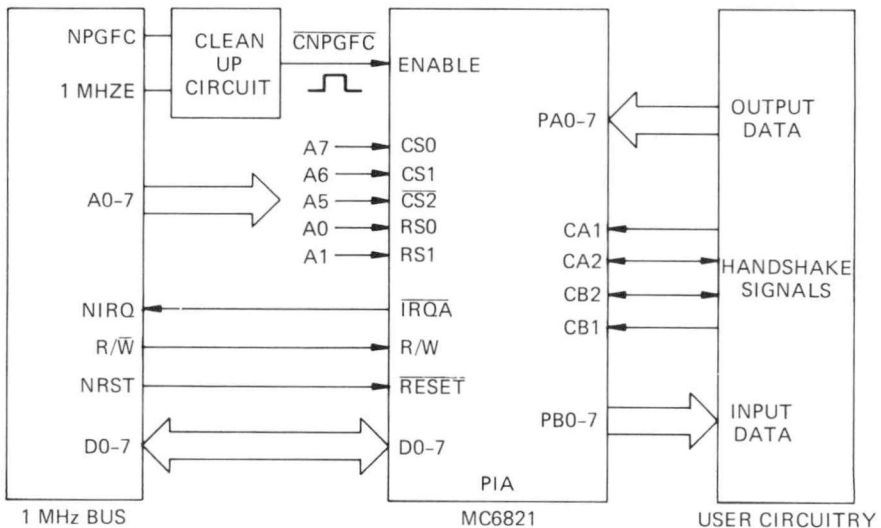


Figure 4.15 PIA connections to the 1 MHz bus

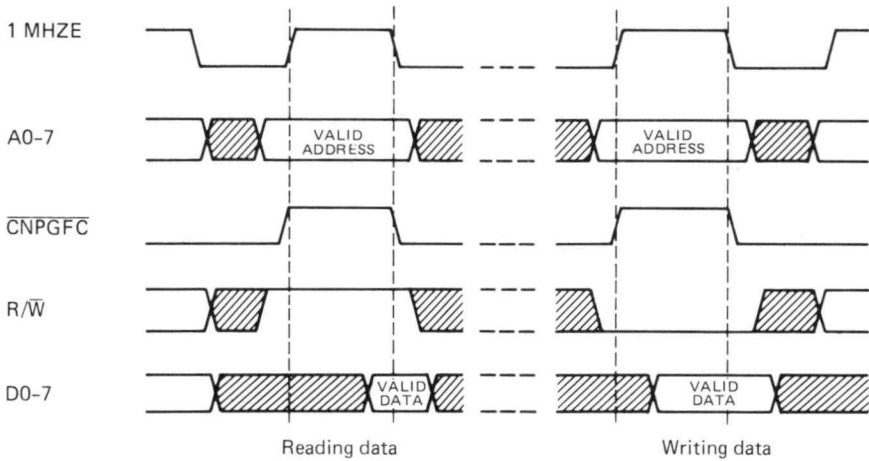


Figure 4.16 Bus transfers timing waveforms

Finally, the initialisation procedure must set up port A as an input port and port B as an output port, and leave each control register set for the interrupt mode required, ensuring that bit 2 is at '1' in both cases. The sequence can be summarised as follows:

- At &FCC1 load binary 00000000 Bit 2 of CRA = 0
- At &FCC0 load binary 00000000 DDRA set to indicate 8-bit input port
- At &FCC1 load binary 00000100 Bit 2 = 1: other bits are set as required for interrupt operations
- At &FCC3 load binary 00000000 Bit 2 of CRB = 0
- At &FCC2 load binary 11111111 DDRB set to indicate 8-bit output port
- At &FCC3 load binary 00000100 Bit 2 = 1: other bits are set as required for interrupt operations.

This can be achieved as shown in the program section of figure 4.17, though, as previously explained, some of the steps can be omitted if a reset will always precede the initialisation.

The circuitry using the signals from the PIA can vary widely but must, of course, provide and accept data in the correct form, and handle the handshaking requirements. For the sake of convenience, the designer will, in many cases, make use of one of the standard interfacing chips. We have looked at the VIA and the PIA, but other common variants, such as the programmable IO chips by Intel and Zilog, are similar in operation.

```

10 DDRA=&FCCC                                /Label Data Direction register A
20 CRA=&FCC1                                  /      Control register A
30 DDRB=&FCC2                                /      Data Direction register B
40 CRB=&FCC3                                  /      Control register B
50 P%=&70                                     /Program starts at &70
60 [
70   LDA #0:STA CRA                          /Clear CRA; bit2=0 selects DDR
80   STA DDRA                                /Set port A for inputs
90   LDA #4:STA CRA/Define CA1,CA2 edges;bit2=1 selects port
110  LDA #0:STA CRB                          /Clear CRB; bit2=0 selects DDR
110  LDA #&FF:STA DDRB                       /Set port B for outputs
120  LDA #4:STA CRB/Define CB1,CB2 edges;bit2=1 selects port
130  RTS
140 ]
150 CALL &70
    etc.

```

Figure 4.17 Initialisation procedure for PIA

An example of a typical requirement would be to use our computer to accept data from a VELA which is acting as a data logger in some experiment. VELA, which stands for Versatile Laboratory Aid,* is a self-contained, 6802-based instrument which is programmed to operate as a wide range of laboratory instruments. The VELA programs are held in ROM, and program 15 of ROM 1 controls the transfer of any 1024-byte block of data from the internal RAM to an external computer. The interface chip is a Motorola PIA, located at &E000 in the VELA memory map. CA2 of the PIA is programmed to provide a positive pulse, DAV, indicating valid data on PA0 to PA7, and CA1 is programmed as an input to receive the data acknowledge signal, which completes the transfer. In order to match the PIA to the VELA PIA, we program PA0 to PA7 as inputs, as before, to accept the data. CA1 is programmed as an input, reacting to the positive-going DAV signal and setting the flag of bit 7 of the control register, CRA, and CA2 is programmed to go high when CRA bit 7 is set, so acknowledging the receipt of data. The flag is reset automatically when the data on PA0 to PA7 is read by the computer. Thus CRA must be set to 00100110, &26, and line 90 of figure 4.17 should become

```
LDA #&26: STA CRA
```

The VELA data transfer program is listed on pages 180 and 181 of the *Software Reference Book*** and all we need in the computer is a short linking program to check bit 7 of the control register, so that, when the flag sets, the next of the 1024 bytes of data can be read and transferred to the computer memory. CA2 generates the acknowledge signal automatically for us when the flag sets, and it is removed automatically when the processor reads the data in.

* VELA is supplied by Educational Electronics, Leighton Buzzard, Bedfordshire, UK.

** The *VELA Software Reference Book*, by A. R. Clarke, is distributed by Instrumentation Software Ltd, Leeds, UK.

The linking program would take the form

```
300 FOR N = 1 TO 1024
310   REPEAT
320     F = ?&FCC1           /Check for bit 7 set
330   UNTIL (F AND 128) = 128
340   D = ?&FCC0           /Read data byte
350   M = &2200 + N        /Adjust pointer
360   ?M = D               /Store byte
370 NEXT
```

5 *Some Applications*

Many devices that we may wish to control from our computer require only to be switched on and off at appropriate times. This type of device is digitally controlled and will respond to the value '0' or '1' in the form of a voltage on a particular connection. Other, apparently non-digital, devices can also be controlled by switching on and off, so that the final value of some variable is governed by how long the on and off intervals are. For example, the heating element in an oven can be switched on and off at intervals to give any required temperature between a minimum and a maximum, as in a conventional thermostatically controlled oven. The water temperature control section of an automatic washing machine also works in this way; the hot water supply is turned fully on, and the cold water supply is pulsed on and off at a rate that gives the correct temperature of the water mixture.

This method is often referred to as *bang-bang control*, since it involves switching between the two extreme limits of fully on and fully off, by means of pulses. The generation of a train of pulses is therefore a common requirement, sometimes varying the pulse repetition frequency, PRF, while maintaining a constant mark-to-space ratio, and sometimes holding the frequency constant but varying the ratio. The *mark-to-space* ratio is the ratio of the time in each cycle during which the voltage is high, the mark, to the time during which it is low, the space, and is illustrated in figure 5.1. The timer-counters provided in the user port VIA can be programmed to generate a simple pulse sequence very easily. When bits 6 and 7 of the auxiliary control register, &FE6B, are both set to '1', the top bit of the port, PB7, provides a square wave output at a frequency determined by the value fed into timer-counter 1, and with a one-to-one mark-to-space ratio. The frequency is defined by the value, N, held in the latches T1L-L and T1L-H, associated with the counters, since each half of the output cycle is generated by counting N down to zero under the control of the clock signal. At the end of the count, the polarity of the output signal on PB7 is reversed and the counter is automatically restarted. In practice, the change-over operation involves taking the counter through the all ones condition as well as the zero, so the actual period is $(N + 2)$ clock periods. In our case, that is $(N + 2) \mu\text{s}$ since the VIA uses the 1 MHz clock. If we are wanting to time operations very accurately we must note a little quirk of the VIA in that the first, but only the first, low period of the output waveform, after loading the high byte to the counter, is $(N + 1.5) \mu\text{s}$ long, not $(N + 2) \mu\text{s}$. The interrupt request output, IRQ, can be enabled, if required, by setting bit 6 of the interrupt enable

register, &FE6C. It has no effect on the signal at PB7 but can be used as an indication of the end of each counting period. The IRQ signal will stay low until reset either by writing to the high byte of the counter, &FE65, or by reading the low byte at &FE64. The pulse generator program of figure 5.2 illustrates many of these ideas and can be used to generate square wave signals over quite a wide frequency range.

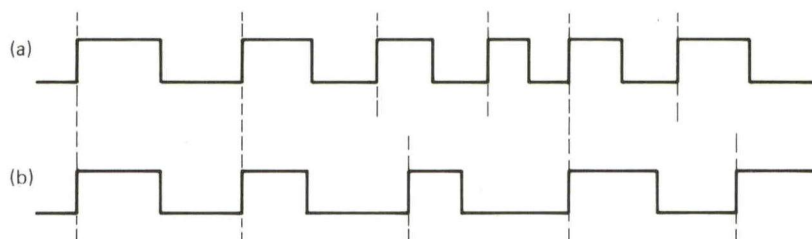


Figure 5.1 Pulse trains. (a) Varying frequency, constant 1:1 mark-to-space ratio.
(b) Constant frequency, varying mark-to-space ratio

```

10 MODE7
20 DIM PROG 100
30 T1LL=&FE66 /Label Timer1 latch low byte
40 T1CH=&FE65 / counter high byte
50 ACR=&FE6B / Auxiliary control reg
60 IER=&FE6E / Interrupt enable reg
70 FOR I=0 TO 2 STEP 2:P%=PROG /Start WAVE routine
80 GOPTI
90 .WAVE
100 LDA #&40:STA IER /Enable interrupts on Timer1
110 LDA ACR:DRA #&C0:STA ACR /Set bits 6 and 7 of aux.reg
120 LDX &70:STX T1LL /for squarewave output at PB7
130 LDX &71:STX T1CH /Start Timer1
140 RTS
150 ]
160 NEXT _____/Exit routine with pulse generator free-running
170 REPEAT
180 INPUT "TIME LENGTH FOR PULSE";T$
190 N=EVAL(T$)
200 !&70=N /Set length of pulse
210 CALL WAVE
220 UNTIL N=0 /Pulse generator continues until N is set to 0
230 END _____

```

/This program, lines 170 through 230, takes a user supplied pulse length, N, and uses it in the routine WAVE so that Timer1 is automatically reloaded with it each time the count reaches zero. On each zero, the output level on PB7 is inverted giving a 1:1 mark:space waveform of period 2N.

Figure 5.2 Pulse generator program

One common application of this type of waveform is in the control of the speed of a small stepper motor. Stepper motors are now not very expensive and their use can often result in more accurate timing or metering than with a dc motor. When run at a constant speed, a stepper motor is, in effect, a synchronous motor running on square waves.

Most stepper motors are of the permanent magnet type, though other forms of construction are available. The stator surrounds the rotor and is made up of two cup-like housings with staggered pole pieces, which are magnetised by the coils, figure 5.3. The rotor is a toothed permanent magnet with as many teeth, or poles, as there are on each stator cup. It is the interaction of the rotor magnetic field with the field generated by the coils that causes the rotor to step, and the stator pole positions ensure that there are well-defined stable equilibrium points at which the rotor will settle. By correctly phasing the supply to the separate stator coils, the rotor can be made to step forwards or backwards as required, and, because the rotor is a permanent magnet, there is a restoring torque developed to hold the rotor at the stable point until intentionally stepped on again. The essential elements of the stepper motor are shown diagrammatically in figure 5.3a, but the action of the fields is clearer if the motor is 'opened out' as in figure 5.3b. As can be seen from the figure, the pole pairs of the stator are mechanically displaced by half a pole pitch, and there is a quarter pole pitch displacement between the poles of stator section A and stator section B. Each step of the rotor takes it through one-quarter of a pole pitch, so the shaft angle is determined by the number of pole pairs in the stator section. Commonly available motors have step angles and corresponding steps per revolution of

1.8°	200 steps
7.5°	48 steps
15°	24 steps
18°	20 steps

An important feature of stepper motors is that, although the error on the single step angle may be up to ± 6.5 per cent, errors on multiple steps are non-cumulative. In fact they average out to zero within the four-step sequence needed to move the rotor one pole pitch, and maximum accuracy is obtained by working in multiples of the four steps.

The simplest motor drive circuits are possible with unipolar windings of the field coils, so that each stator section has two windings as shown in the diagram, one for magnetising in a positive sense and the other for magnetising in a negative sense. The normal four-step switching sequence is achieved by energising the positive and negative windings in the following order:

	<i>Section A</i>	<i>Section B</i>
<i>Step 1</i>	+	+
<i>Step 2</i>	+	—
<i>Step 3</i>	—	—
<i>Step 4</i>	—	+

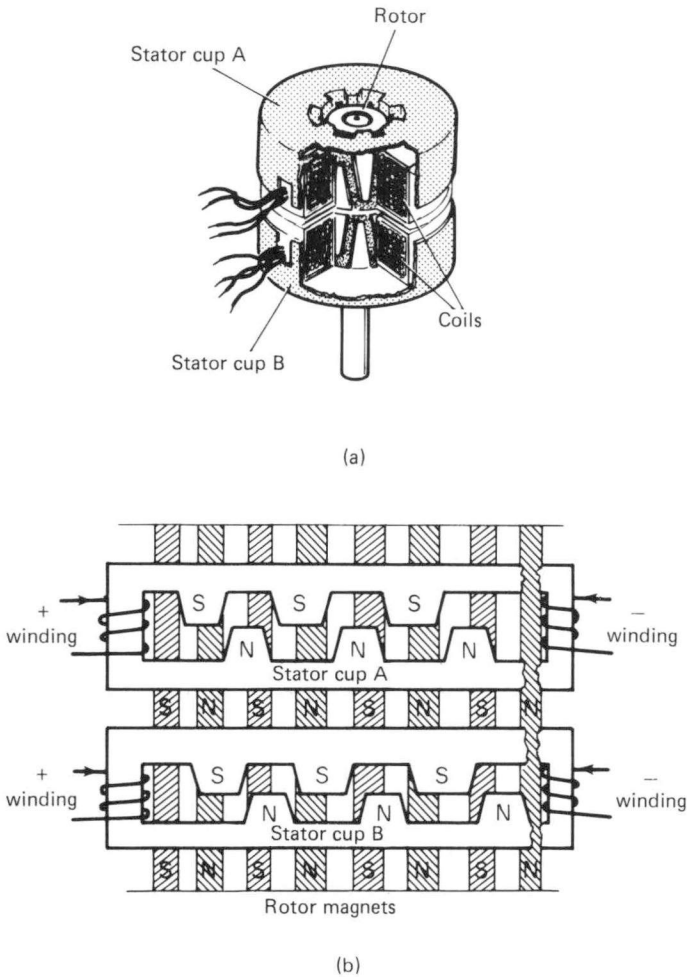


Figure 5.3 The stepper motor. (a) Construction. (b) Magnetic arrangements

In terms of the pulses needed from the driving circuitry, the waveforms are as shown in figure 5.4a, and these could be generated on four pins of the user port with both the timing and the phasing of the signals being calculated internally. However, it is often preferable to use a standard drive chip, such as the SAA1027, to provide the correct phasing, so that only the timing information need be provided by the computer, and this is where our pulse waveform comes in. The chip requires a train of pulses at the trigger input, T on figure 5.4b, and each step is triggered by a positive-going edge. The repetition rate of the pulses determines the stepping rate, and the number of pulses determines the number of steps. The signal

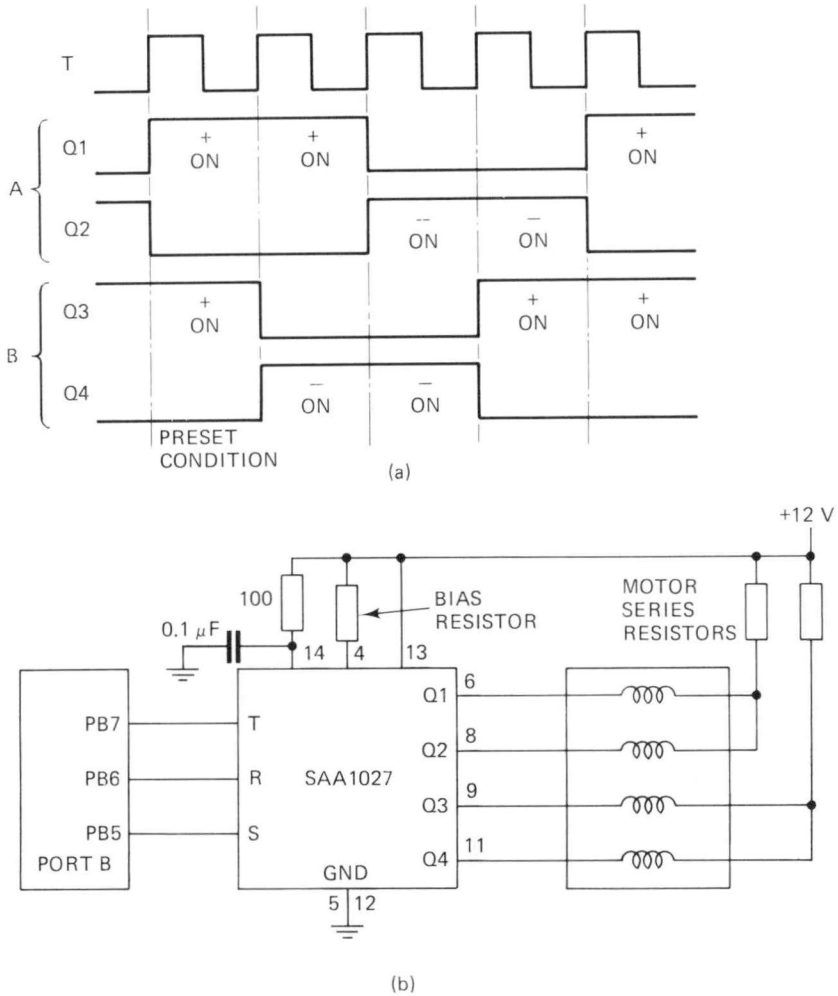


Figure 5.4 Stepper motor drive arrangements. (a) Drive waveforms for clockwise rotation. (b) Circuit

level at the rotation input, R, indicates the direction of rotation required, with a low level giving clockwise rotations. The third input, S, is a preset input, allowing us to set the outputs to a known condition as shown on the waveform figure. The values required for the bias and motor series resistors are governed by the actual motor used, and are quoted in the manufacturer's data.

The program of figure 5.2 allows us to generate a square wave sequence and to change its frequency, but it is always a 1:1 mark-to-space ratio. However, the VIA timer has an alternative mode of operation which will give a much more complex output, in that we can control the length of each high period and each low

period of the waveform. In this mode of working we make use of the fact that the counter is reloaded, at the end of each count, from the latches, and the latches can be loaded with new values while the counter is running. The procedure is to start the sequence by loading the initial values into the counter, T1C-L at &FE64 and T1C-H at &FE65. The count begins as soon as T1C-H is loaded so we immediately load our second value into the latches, T1L-L at &FE66 and T1L-H at &FE67. At the end of the count, bit 6 of the interrupt flag register, &FE6D, is set, and the new count value is transferred from the latches to the counter. By testing for bit 6 being set we know when we can load the next value into the latches, and we then reset bit 6 by reading T1C-L, &FE64. Each time bit 6 of the flag register is set we can load a new value into the latches. This method of operation can be used to vary the pulse width within a constant frequency waveform, as in figure 5.1b, and gives a form of *pulse width modulation*, PWM. The program of figure 5.5 modulates the width of the pulse at PB7 on the user port, by loading first the required pulse length, defined by the current value of W%, to give the correct high-level duration, then loading the complement of the value for the low-level duration, so that the total cycle length is constant. We have shown the variable as a linear function but it could be any signal, such as the digitized audio signal from a microphone, for example.

The pulse width modulation idea can also be used in the sort of bang-bang control application mentioned at the beginning of the chapter, but now we need some feedback of information, so that we know what the value of our controlled variable actually is and, therefore, how much it differs from what we want it to be. The difference is called the *error*, and knowing its magnitude we can generate the appropriate signals to adjust the variable in such a way that the error reduces towards zero. The schematic diagram of figure 5.6 shows this *closed loop* approach in general terms, though it must be recognised that our explanation is very much simplified: a large body of control theory is necessary to ensure that a complex system works satisfactorily, with good speed of response and accuracy of control. But on a simple system such as an oven controller we can get reasonable results, because the changes occur very slowly. The sensor in this case, figure 5.7, is a temperature-sensitive current source which increases its current output by $1\ \mu\text{A}$ for each 1°C rise in temperature, and the voltage developed by the current is measured on channel 0 of the analogue-to-digital convertor. The control element is a resistor connected to the mains through a solid-state relay switch, which includes a zero-level detecting switch circuit to reduce interference problems. Pulses to switch the mains on and off are supplied, as before, from PB7 on the user port. The times involved in this application, as already noted, are very long, being of the order of seconds rather than microseconds, so from a computer point of view we are talking in terms of changing the level on PB7 at intervals, rather than pulsing, but the principle is the same. The temperature is read by means of an ADVAL(1) command, and is subtracted from the value corresponding to the required setting. If the temperature is too low a positive error results and the heater is switched on, or left on, until a negative error occurs which indicates that the temperature is now

```

10 MODE7
20 DIM PROG 200
30 T1CL=&FE64           /Label Timer1 counter   low byte
40 T1CH=&FE65           high byte
50 T1LL=&FE66           Timer1 latch   low byte
60 T1LH=&FE67           high byte
70 ACR=&FE6B           Auxiliary control reg
80 FR=&FE6D           Flag register
90 IER=&FE6E           Interrupt enable register
100 FOR I=0 TO 2 STEP 2:PX=PROG
110  [OPTI
120  .START
130  LDY #0           /Clear index register
140  LDA T1CL         /Clear flag by reading Timer1 count
150  LDA #&40:STA IER  /Enable interrupts on Timer1
160  LDA ACR:ORA #&C0:STA ACR  /Set bits 6 and 7 in ACR
170  LDA &B00,Y:STA T1CL
180  LDA &C00,Y:STA T1CH  /Start counter with first values
190  .LOOP
200  LDA &B00,Y:EOR #&FF:STA T1LL  /Load latches with comp.
210  LDA &C00,Y:EOR #&FF:STA T1LH /of values for next count
220  .TEST1
230  BIT FR:BVC TEST1  /Wait until bit 6 of flag reg sets
240  LDA T1CL         /indicating end of count.Clear flag
250  INY             /Increment index
260  LDA &B00,Y:STA T1LL  /Load latches with values
270  LDA &C00,Y:STA T1LH  /for next count.
280  .TEST2
290  BIT FR:BVC TEST2  /Wait until end of count
300  LDA T1CL         /Clear flag register
310  JMP LOOP
320  JNEXT  -----
330  FOR I=0 TO 255           /Calculate and store 256 values
340  W%=65280*I/255         /in locations &B00/C00 to &BFF/CFF
350  I?&B00=W% MOD 256
360  I?&C00=W% DIV 256
370  NEXT
380  CALL START_-----

```

/The main program consists of lines 330 through 380, in which 256 values of W% are calculated and stored, and the routine START is called. START uses the first pair of values in the counter of Timer1 and loads the complement into the latches to make up the next counter value. The end of count, which is detected by bit 6 of the flag reg, starts the next count automatically, then loads the latches with the next pair of values or the complement. By using Y as an index register, the cycle repeats every 256 values.

Figure 5.5 Pulse width modulation program

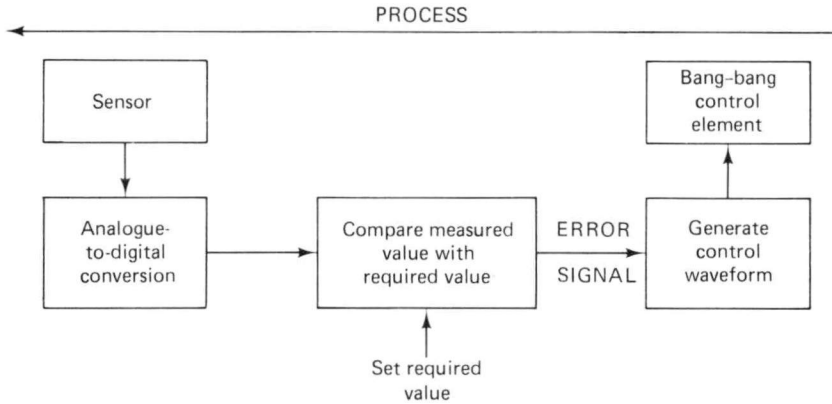


Figure 5.6 Closed loop control

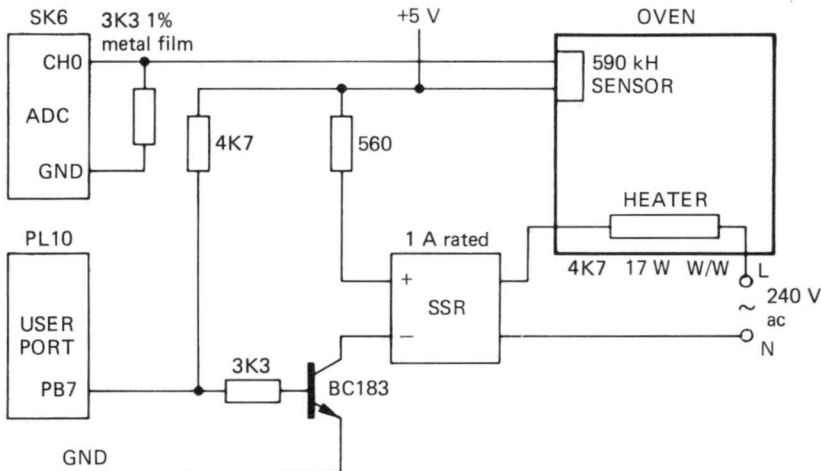


Figure 5.7 An oven controller

too high. The heater is then switched off until the oven cools down and a positive error is indicated again. The values corresponding to specific temperatures are governed by many factors in the make-up of the system and must be found by means of a calibration exercise across the temperature range.

The pulse mode of operation is also of use in driving multiple LED displays. When compared with the arrangement in which the segments of several different seven-segment displays are continuously driven, the number of drive circuits is greatly reduced because we time-share the drivers between the displays. Far from reducing the effectiveness of the displays, our eyes actually perceive a higher brightness level from pulsed displays than from those that are continuously driven.

The procedure is to send the code for the segments to be lighted to all displays at the same time, but to provide a return path only for the one that is to display the digit. One way of achieving this is shown in the circuit of figure 5.8 which has four seven-segment displays. We have the seven drivers for the segments as usual, but these feed all the displays, and in addition we merely need four drivers for the digits, which are to be selected one at a time. We must find some way to provide the extra drive signals, since we have only eight lines available on the port, and one way is to generate the digit select signals in a four-stage ring counter. A ring counter is a special type of counter in which only one output is at '1' at any time, and the '1' steps on each time that a pulse is applied to the counter. It is in fact a shift register, with only one stage set to '1', and the last stage is connected back to the first to form the ring. A 74LS194A shift register can be used for the ring counter as shown in figure 5.9, making use of the handshake signal from CB2 of the VIA as the clocking signal. Since only seven segment lines are required, PB7 can be used to control the initial setting of the shift register, to ensure that it acts as a ring counter. CB2 must be programmed to pulse mode by setting bits 5, 6 and 7 of the VIA peripheral control register, &FE6C, to 101. Then, each time data is written to the output port, a negative-going data ready pulse is generated on CB2, and the ring counter steps on at the end of the pulse. If PB7 is set to '1', the data ready pulse will load the values pre-wired on the shift register parallel inputs, so ensuring that it contains the single one in stage D. Next we set PB7 to '0' and PBO to PB6 to the code needed for the segments of display 1. Subsequent values on the output port relate to displays 2, 3, 4, back to 1, and so on. Note that the segment drivers switch on when a '0' is presented by the output port; PB7 must be retained at '0' throughout.

We have already seen that pulse mode operation is of use in generating special irregular waveforms of square waves, and we now move on to the generation of some non-square waves. One such waveform is the staircase which is a sort of digital sawtooth waveform in which the voltage increases, or decreases, in steps, normally at regular intervals. We previously came across this type of waveform in one of the analogue-to-digital converters, and another of its uses is in the control of the spot position on the face of a cathode ray tube when characters are to be drawn. It is then providing a digital timebase signal. The staircase is simply the output we get from a digital-to-analogue converter when we feed the inputs from a counter, as in the feedback ADC shown in chapter 3 (figure 3.1). The counter is easily provided by the computer, and we can connect the DAC directly to the user port. Line 130 of the program of figure 5.10 provides the counter action by incrementing the value in the output port and, for each value of the count, timer 1 is triggered to generate a delay. Lines 170 to 190 detect when the time delay is completed. Although BIT FR is primarily an instruction that compares the contents of the accumulator with the interrupt flag register, we make use of its secondary feature of transferring bits 6 and 7 of the interrupt flag register to the overflow and sign flags respectively. We then wait for the completion of the time delay by checking the completion flag in bit 6 of the interrupt flag register, and jumping back to

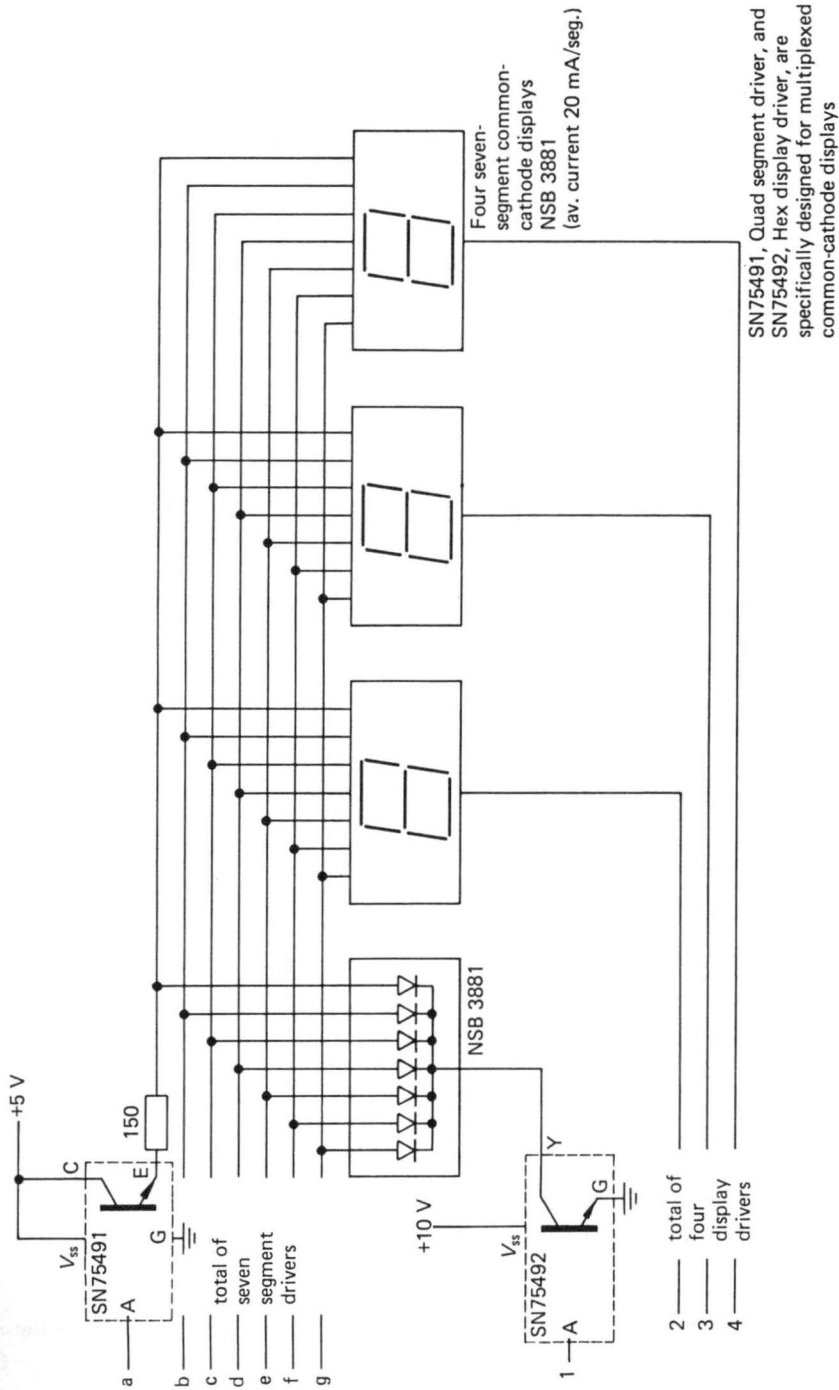


Figure 5.8 Multiplexed LED display

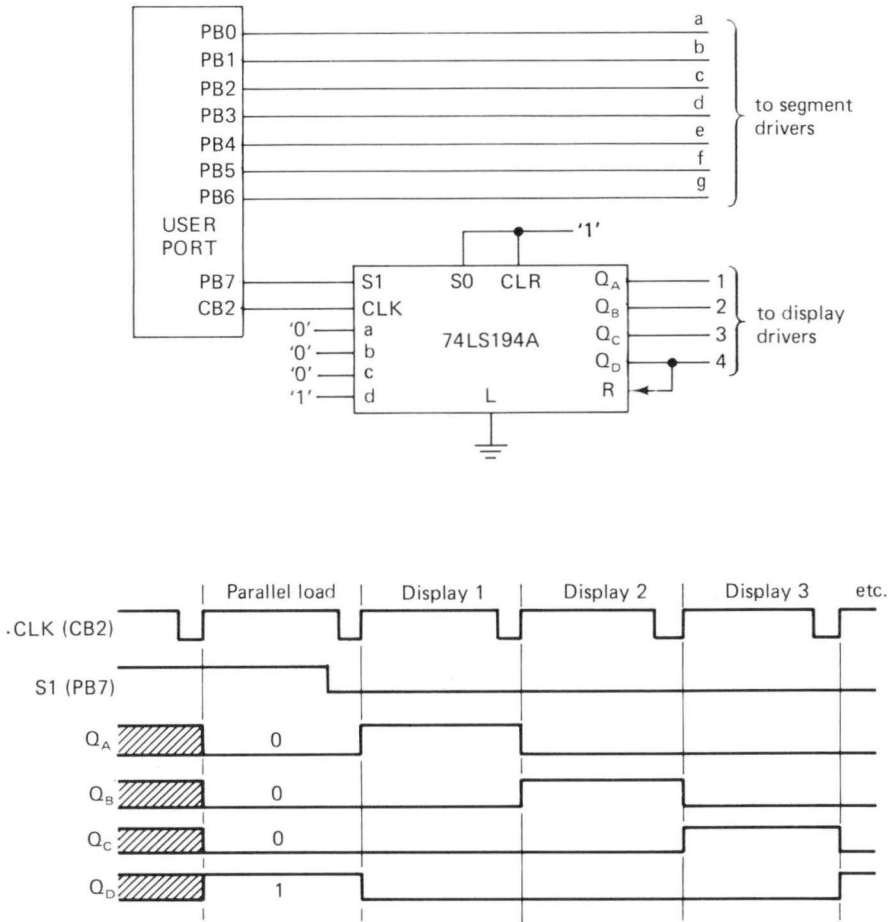


Figure 5.9 Generation of display selection signals

.DELAY until it is set. The program continues until restarted by pressing **BREAK**.

The size of the steps in the staircase can be varied by altering the counter increment value. We can also vary the step size as the amplitude of the voltage increases, as is done in some *componder*, compressor-expander, circuits used in digital communication systems. But if we wish, we can do without the counter altogether and feed values to the DAC which will give us an output waveform of any complexity within the limits of the converter performance. The program of figure 5.11 gives us a sinewave by calculating the amplitudes of the components of one cycle of the sinewave at the frequency selected at the keyboard, storing them at locations &2000 upwards, and then outputting them repetitively using the subroutine

```

10 *KEY10 OLDIMRUNIM           /Program BREAK key to restart
20 ?&FE62=255                  /Set output port
30 T1CL=&FE64                   /Label Timer1 low byte
40 T1CH=&FE65                   /          high byte
50 ACR=&FE6B                    /      Aux.control reg
60 FR=&FE6D                    /      Flag register
70 DIM PROG 300
80 FOR I=0 TO 2 STEP 2:P%=PROG
90  [OPTI
100  .SETUP
110  LDA #0:STA &FE60           /Output zero
120  .CYCLE
130  INC &FE60                 /Increment output value
140  LDA #0:STA ACR             /Set ACR
150  LDA &70:STA T1CL
160  LDA &71:STA T1CH          /Load and start counter
170  .DELAY
180  BIT FR                   /Check for flag set when
190  BVC DELAY                /count complete.
200  JMP CYCLE                /Repeat cycle
210  ]
220 NEXT -----
230 MODE7
240 PRINT TAB(7,2);CHR$(141);"DIGITAL STAIRCASE"
250 PRINT TAB(7,3);CHR$(141);"DIGITAL STAIRCASE"
260 PRINT TAB(3,10);"DELAY LENGTH BETWEEN PULSES=";
270 INPUT A$
280 !&70=EVAL(A$)             /Load user defined delay in &70/71
290 CALL SETUP_____

```

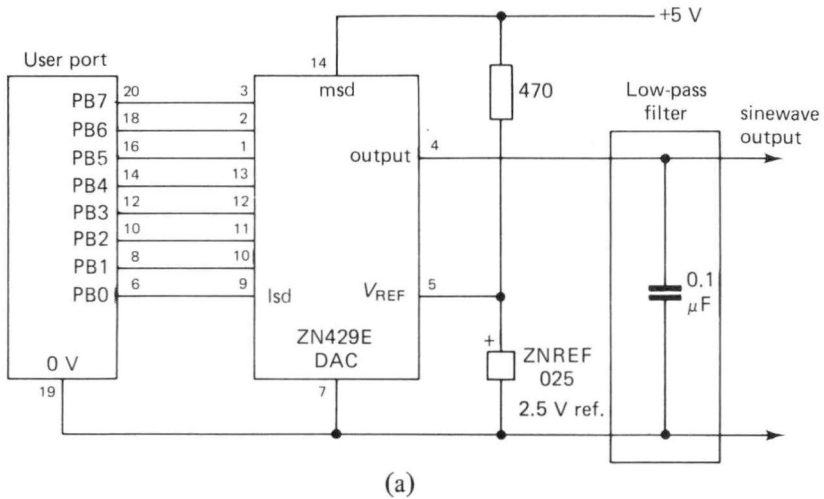
/The main program starts at line 230. It calls
SETUP which increments the step counter and
uses the defined delay in Timer1 to control
the duration of each step

Figure 5.10 Digital staircase generator

.CYCLE. The calculation of the values is carried out in lines 200 to 280. For component n , the value stored is

$$C\% = 127.5 \left[\sin \left(\frac{n \cdot 2\pi \cdot \text{FRE}}{52945} \right) + 1 \right] \times \text{AMP}$$

where FRE is the frequency defined by the input at line 420, and AMP is the percentage amplitude defined at line 400. The number of components stored for a complete cycle is governed by the rate at which the values can be dealt with by the circuitry, and, since each output takes about $18 \mu\text{s}$, the number of components varies from a maximum of about 130 at 400 Hz to a minimum of less than four at 16 kHz. In fact, at the higher frequencies the irregularities become so pronounced that the waveform cannot be said to be sinusoidal, since the number of components is too low to give the smoothness required. At all but the highest frequencies, a reasonable sinewave can be obtained by including a $0.1 \mu\text{F}$ capacitor across the DAC output, to act as a simple low pass filter.



```

10 HIMEM=&2000           /Reserve memory above &2000 to prevent
20 DIM PROG 200          /overwriting of calculated data.
30 ?&FE62=255            /Set output port
40 FOR I=0 TO 2 STEP 2:P%=PROG
50   OPTI
60   .WAVE
70   LDA #0:STA &70:LDA #&20:STA &71 /Load store pointer to
80   .CYCLE               /&2000
90   LDY #0               /Clear index register
100  LDA (&70),Y          /Load next value
110  BEQ WAVE              /If zero, jump to WAVE
120  STA &FE60             /otherwise, output value
130  CLC:LDA &70:ADC #1:STA &70
140  LDA &71:ADC #0:STA &71 /Increment pointer
150  JMP CYCLE            /and repeat the cycle
160  ]
170 NEXT
180 MODE7:PROGtitles      /Ask user for amplitude and frequency
190 I=0
200 REPEAT                /Calculate values of components of
210   A=A+F2              /one cycle and store from &2000 onwards
220   B=SIN(A)+1
230   C=B*127.5
240   C%=(C*Amplitude)/100
250   IF C%=0 C%=1
260   ?(&2000+I)=C%
270   I=I+1
280 UNTIL A>2*PI
290 PRINT TAB(10,20);"WE'RE OFF";SPC(20)
300 ?(&2000+I)=0          /Set last entry to zero
310 CALL WAVE
320 DEF PROGtitles
330  *KEY10 OLDIRUNIM      /Program BREAK key to restart
340  PRINT TAB(8,1);CHR$(141);"SINE WAVE GENERATOR"
350  PRINT TAB(8,2);CHR$(141);"SINE WAVE GENERATOR"

```



```

360 PRINT TAB(2,5);"AMPLITUDE PERCENTAGE ="
370 PRINT TAB(2,6);"FREQUENCY HZ(400-1600) ="
380 PRINT TAB(10,20);"PRESS BREAK TO EXIT"
390 PRINT TAB(27,5);
400 INPUT Amp
410 IF Amp<1 OR Amp>100 GOTO 390
420 PRINT TAB(27,6);:INPUT Fre
430 IF Fre<1 GOTO 410
440 DIG=52945/Fre
450 F2=2*PI/DIG /Calculate frequency F2
460 PRINT TAB(10,20);"Please wait a moment"
470 ENDPROC

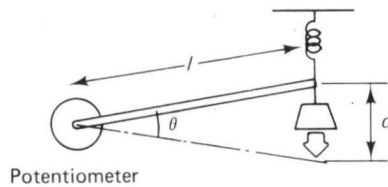
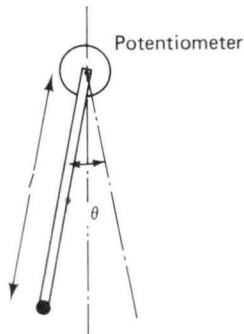
```

/The main section starts at line 180 and runs through line 310. The TITLES procedure requests an amplitude (as a percentage) and a frequency between 400 and 1600Hz. The component values of one cycle of the waveform are calculated and stored at &2000 onwards. Routine WAVE successively outputs the components and repeats the cycle until BREAK is pressed.

(b)

Figure 5.11 Sinewave generator. (a) Circuit. (b) Program

When we come to dealing with input signals we can often work in pulse mode, as we have with the outputs. It is very easy to count pulses and even easier to detect levels, such as we get from switches and other digital transducers. But the majority of input variables are analogue and the values must be converted to digital form using the voltage-to-frequency converters or other methods of analogue-to-digital conversion. In all cases the input signal amplitude must lie within the limits acceptable to the converter, and the range is often narrow — 0 to 1.8 volts in the case of the internal ADC. If the signal lies outside the acceptable range, it must be either attenuated or amplified to suit. Reducing the amplitude is straightforward, and in most cases we use the potentiometer effect of two resistors, as described in the first chapter. The circuit using a resistor ratio with a variable voltage is often replaced by a variable resistor ratio with a fixed voltage, as in the games paddles, for instance. The voltage then indicates the angular position of the spindle of the potentiometer. Several simple pieces of equipment are available, or can be made very cheaply, making use of this principle to convert rotational, or linear, movement to an analogue voltage which is then converted to digital form. Figure 5.12a shows an arm which can be used to provide data on spring extensions and simple harmonic oscillations, by means of the rotation of the horizontal spindle of the potentiometer, as the weight moves up and down. If the displacement is relatively small, we can express it in terms of the rotation angle in radians and the length of the arm, so that $d = l\theta$. Also, the speed at which the analogue values have to be converted is then within the capability of the internal converter, and the simple



(a)

```

10 MODED
20 PROCinitialize      /Select single ADC channel and adjust
30 MODED              /scale factor
40 PROCscreen          /Draw axes and prompt user
50 PROCreading         /Take first reading
60 X%=10:MOVE X%,512
70 REPEAT              /Plot readings
80  PROCreading
90  X%=X%+1
100  DRAW X%,Y%
110  UNTIL X%=1280
120  A$=GET$           /Press any key to restart
130  GOTO 30
140 DEF PROCscreen
150  MOVE 10,1024:DRAW 10,0:MOVE 9,1024:DRAW 9,0
160  MOVE 10,512:DRAW 1280,512      /Draw axes
170  PRINT TAB(2,0);"Press any key to start";
180  A$=GET$:PRINT TAB(2,0);SPC(24); /Read key and
190  ENDPROC                       /clear prompt
200 DEF PROCreading              /Take a reading and scale it
210  V=ADVAL(1)
220  Y%=(V*SCALE/64)
230  IF Y% 1023:Y%=1023          /Limit maximum
240  ENDPROC
250 DEF PROCinitialize           /Adjust scale factor
260  *FX16,1                     /Use single ADC channel
270  PRINT TAB(0,0);"Set to maximum swing then press a key"
280  REPEAT
290    Vmax=ADVAL(1)             /Set maximum voltage
300  UNTIL INKEY(1) <> -1
310  IF Vmax<500 PRINT"Reading too low;try other side"
320  IF Vmax<500 GOTO 280
330  SCALE=65520/Vmax           /Calculate scale factor
340  ENDPROC

```

/The procedure INITIALIZE selects a single ADC channel and calculates the voltage scale factor. Procedure SCREEN draws the axes and prompts the user to start a plot. Procedure READING is used to take readings which are scaled and drawn, as X% is incremented to 1280. A new plot is started by pressing any key.

(b)

Figure 5.12 Study of simple harmonic motion. (a) Arm arrangement as pendulum and for spring extensions. (b) Program

program of figure 5.12b is sufficient to plot the oscillations. The voltage from the potentiometer is proportional to the angle and is used with the scaling factor to give the value to be plotted. The initialize procedure is included to allow the user to adjust the scale of the graph plotted on the screen. Bear in mind that the damping effect of the friction in the potentiometer spindle can be large, so we must use a long arm (which also helps the accuracy) and as slack a spindle as possible.

It is not essential for a potentiometer to have a rotating spindle of course; the contact that picks off the voltage along the standard wire of a Wheatstone bridge is also a potentiometer, in which the voltage at the contact is proportional to the distance along the wire. This feature can be used in a simple digitizer, with which we can enter coordinates of different points and shapes drawn on a sheet of paper. Two wires are required, one for the X direction, AB, and one for the Y , CD, as shown in figure 5.13a. The slider on each wire carries a strip of Perspex or a ruler, and the voltages V_x and V_y give the coordinates of the point, P, at which the strips coincide. The program of figure 5.13b reads the X potentiometer value on ADVAL channel 1, and the Y value on channel 2. The display is presented initially as the normal white foreground, colour 7, but can be changed to any colour value 0 to 9 by pressing the appropriate key. In addition

- Key C clears the screen
- Key S saves the display on tape or disc
- Key L loads a previous display from tape or disc
- Key Q stops the program.

The two slider form of digitizer is not particularly easy to use, and commercial versions use the radius arm principle. Two arms, of length l , are used; the first controls potentiometer A and carries potentiometer B at its free end. The second arm then controls potentiometer B as the pointer, P, is moved, as in figure 5.14. Using simple trigonometric theory, the coordinates of a point, P, are given by $X = r \cos \theta$ and $Y = r \sin \theta$. Unfortunately, we do not know the values of r and θ directly, and we must develop expressions for them in terms of the length, l , and the angles, A and B, which we do know. Firstly, from figure 5.14 we see that $r/2 = l \sin(B/2)$ so

$$r = 2l \sin(B/2)$$

Secondly, $\theta = A - \{(\pi/2) - (B/2)\}$ giving

$$\cos \theta = \sin \{A + (B/2)\} \text{ and } \sin \theta = -\cos \{A + (B/2)\}$$

Using these relationships we can devise an extension of the previous program to cope with this form of digitizer, figure 5.14b. For those who wish to take advantage of the work that has already been done, a full description of a radius arm digitizer, with constructional details and controlling program, is given in the June 1983 issue of *The Micro User*.*

* Mike Cook, 'Arms Stretch with the Graphics Digitizer', *The Micro User*, Volume 1, No. 4, June 1983, pages 80-86.


```

400  Xold=X%:Yold=Y%
410  ENDPROC
420  DEF PROCdrawline
430  MOVE X%,Y%                                /Position cursor
440  DRAW X%,Y%                                /Draw line
450  ENDPROC
460  DEF PROCinitialize
470  PRINT TAB(0,0);"Rotate both arms to max. position"
480  PRINT TAB(5,1);"and press any key"
490  REPEAT
500    A=ADVAL(1)
510    B=ADVAL(2)
520  UNTIL INKEY(1) <> -1
530  Scale1=65520/A                            /Scale factors on ADC so as to give
540  Scale2=65520/B                            /a maximum reading equal to 2*pi.
550  Xscale=310                                /Scale factors to control
560  Yscale=512                                /screen window size.
570  ENDPROC

```

/Speed of operation is not important, so the main program, lines 10 through 100, uses procedures rather than assembler routines. INITIALIZE sets up scale factors; CALC takes readings and calculates the co-ordinates X,Y; CONTROLS reacts to control key operation, and PLOTPOINTS plots the new point. Two other procedures, DRAWLINE and MOVE, are used in plotting the output as either lines or points.

(b)

Figure 5.14 (above and page 93) Radius arm digitizer. (a) Construction.
(b) Program

The analogue voltage input levels from the digitizers are well within the required levels because we make use of the reference voltage provided. In many other cases, however, the analogue voltage presented to the system is so small that it falls below the minimum voltage accepted by the converter, and we must amplify the signal. This is true of the outputs of many transducers, and the pre-amplification is said to be *conditioning* of the signal. This has to be done with far more care than when we attenuated an over-large signal, because in amplifying the signal we also emphasise any errors and effects due to noise and other unwanted signals. The usual approach is to use an operational amplifier with negative feedback, but before we see how to use the op-amp in a specific circuit (in this case dealing with the signals from a microphone), we shall look at the basic ideas on which it is based.

An op-amp is a difference amplifier which amplifies the voltage existing between its two inputs, and takes its name from the circuits used for arithmetic operations in analogue computers. When the intrinsic gain of the amplifier is large enough, the performance of the amplifier with feedback can be controlled almost entirely by the feedback components. The open-loop voltage gain of a typical op-amp is of the order of 20 000 (106 dB) but, when used with negative feedback, the gain is something nearer 100 (40 dB). The input impedance of the op-amp is also very high,

being typically in the megohm range, so the amplifier itself takes negligible input current. When the negative feedback is applied, the input voltage approaches zero, and the input acts as a virtual ground.

The circuit of figure 5.15a shows the simple arrangement for an inverting amplifier. The overall voltage gain is $-R_F/R_1$ and the effective input resistance is R_1 . R_F is usually between 10 k Ω and 100 k Ω and R_1 is 1 k Ω to 10 k Ω . If we apply the input to the other terminal we get a non-inverting amplifier with a gain of $(R_F + R_1)/R_1$ and a very much higher input resistance. Regardless of the connection, the action of the circuit is always to change the output voltage in such a way that the difference voltage at the inputs is reduced towards zero. If, for instance, the input to the circuit of figure 5.15b is increased, the output increases in phase with it and, by potentiometer action, a proportionate voltage increase occurs at the negative input, so preventing the difference voltage from increasing. Additional components may be necessary to provide compensation for currents and voltages arising from asymmetries inside the op-amp, but many of the popular types contain their own compensating networks.

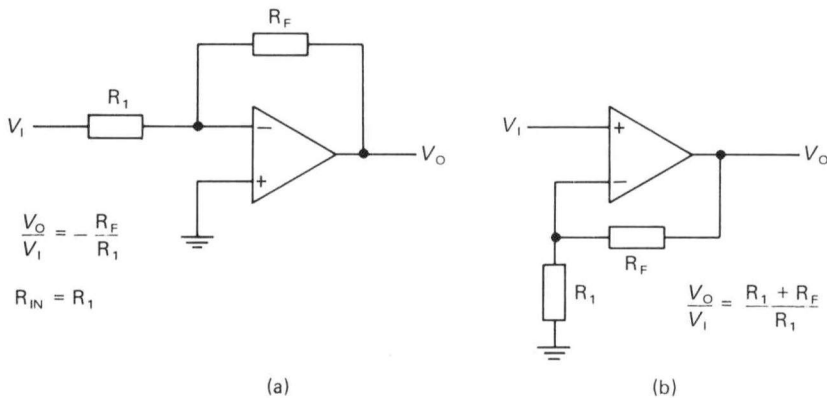
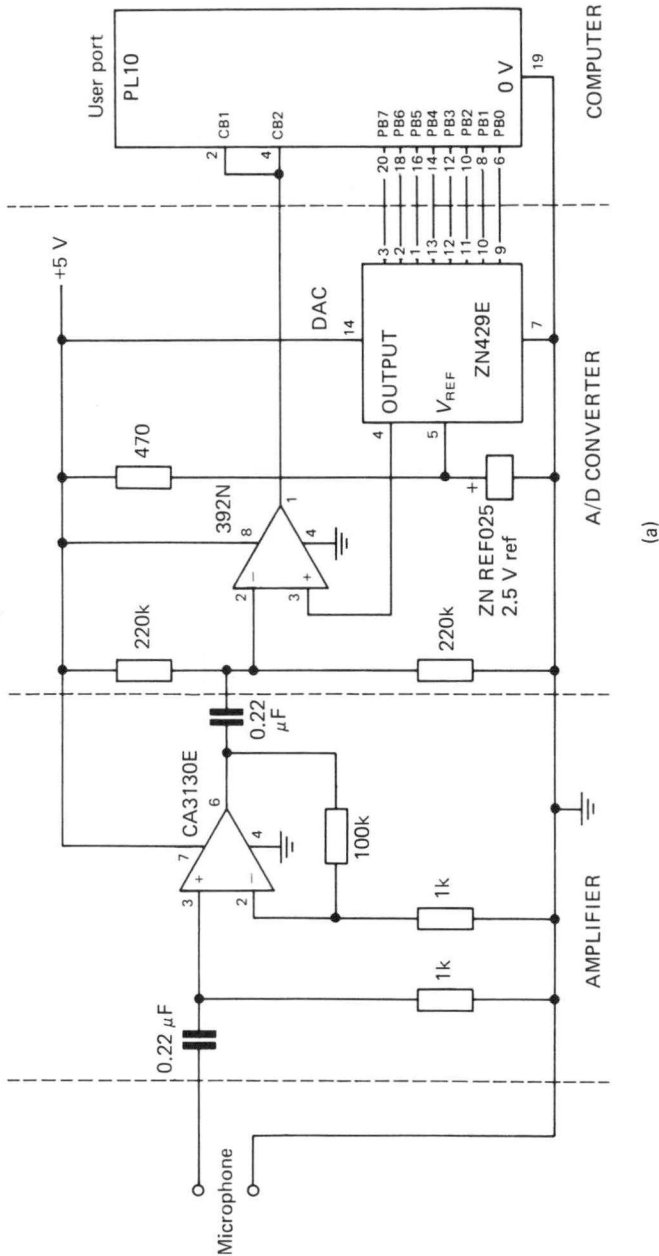


Figure 5.15 Operational amplifier with feedback components. (a) Inverting amplifier. (b) Non-inverting amplifier

The power supply voltages to the op-amp are normally between 3 and 22 volts, positive and negative. The two polarities are provided to enable the output signal to vary about zero volts, as is usually required of an ac signal, and the amplitude of the output signal has a maximum within about 1.5 volts of the supply voltage. In our applications, however, we require the output voltage to vary between zero and the positive reference voltage. The op-amp can be operated with the negative supply terminal at ground, but then the lowest achievable output level is about 1.5 volts above ground. In order to balance the signal, we use ac coupling and a bias network at the input so that the quiescent, or no-input, value at the output lies midway between ground and the reference voltage. Figure 5.16a shows the



(a)

```

10 MODE4
20 Vref=2.5
30 ?&FE6C=64
40 ?&FE62=255
50 !&72=&2200
60 DIM PROG 500
70 FOR I=0 TO 2 STEP 2:P%=PROG
80   [OPTI
90     .SETUP
100    LDA #0:STA &FE60
110    LDA #&80:STA &70:STA &FE60
120    LDA &FE6D:AND #8:BNE NEGTRAN
130    .POSTRAN
140    CLC:ROR &70
150    LDA &FE60:CLC:ADC &70
160    LDY #0:STY &FE60
170    STA &FE60:LDA &FE6D:AND #8:BNE NEGTRAN
180    LDA &70:BNE POSTRAN
190    JMP STORE
200    .NEGTRAN
210    CLC:ROR &70
220    LDA &FE60:SEC:SBC &70
230    LDY #&FF:STY &FE60
240    STA &FE60:LDA &FE6D:AND #16:BNE POSTRAN
250    LDA &70:BNE NEGTRAN
260    .STORE
270    LDA &FE60:LDY #0:STA (&72),Y
280    CLC:LDA &72:ADC #1:STA &72
290    LDA &73:ADC #0:STA &73
300    CMP #&56:BNE SETUP
310    RTS
320    ]
330 NEXT -----
340 TIME=0:CALL SETUP:T1=TIME
350 PROCscalefactors
360 Tstart=0:Tend=T:Loc=0
370 REPEAT
380   PROCscreen
390   C=0:X1%=0
400   X=10-(Scale1*Scale2%)
410   REPEAT
420     C=C+Scale2%
430     X=X+(Scale1*Scale2%)
440     Loc=Loc+Scale2%
450     IF INT(X)=X1% GOTO 500
460     Y1=? (LOC+&2200)
470     Y2%=(Y1*3.5)+40
480     X1%=X
490     DRAW X1%,Y2%
500   UNTIL C>Point% OR (Loc+Scale2%)>&33FF
510   *FX21,0
520   PRINT TAB (15,0)"Press a key";:A$=GET$
530 UNTIL (Loc+Scale2%)>&33FF
540 END -----

```

Figure 5.16 (above and opposite) (continued overleaf)

```

550 DEF PROCscreen                      /Draw and label display axes
560 CLS:@%=&20109                      /Adjust print field
570 PRINT TAB(0,0);Tstart;"ms";SPC(26);Tend;"ms"
580 MOVE 1280,40:DRAW 10,40:DRAW 10,940
590 PRINT TAB(0,1);Vref;" volts"
600 PRINT TAB(0,31);"0";" volts";SPC(20);"time";
610 Tstart=Tend:Tend=Tend+T
620 ENDPROC
630 DEF PROCscalefactors
640 CLS:@%=10
650 PRINT"Time scale for each screen (max ";T1")ms"
660 INPUT T                          /Adjust timebase to required value
670 IF T<1 OR T>T1 GOTO 640
680 Point%=&3400*T/T1
690 Scale1=1270/Point%
700 Scale2%=T/20:IF Scale2%<1 Scale2%=1
710 ENDPROC

```

/The main program runs from line
340 through line 540. It uses four
assembler routines, (60-330), and
two procedures, (550-710).

(b)

Figure 5.16 (above and pages 96, 97) A simple oscilloscope. (a) Circuit.
(b) Program

circuit arrangement for handling the signal from a microphone and feeding to the analogue-to-digital converter for display on the computer screen. This allows us to use the computer display as a simple oscilloscope.

In order to speed-up the operation of the analogue-to-digital converter we have this time used the *successive approximation* method and approximately 13 000 samples of the input waveforms are converted and stored in about 3 seconds. The successive approximation method checks the value of each bit of the digital value in turn, starting with the most significant bit, and therefore converts an eight-bit number in a constant eight comparisons, regardless of the size of the value being converted. Each conversion begins by setting bit 7 to '1' and all other bits to '0' — representing half the maximum value possible. The comparator output indicates whether the incoming sample is greater than half of the maximum value or less than half. If it is greater than the set value, that bit is retained at '1' and the next most significant bit is also set to '1', ready for the next comparison. But if it is less than the set value, that bit is cleared to '0' before the next most significant bit is set to '1'. This process continues through all the bits and leads to the alternative name for this conversion method of *put and take*. When all eight bits have been checked, the port holds the digital value of the sample.

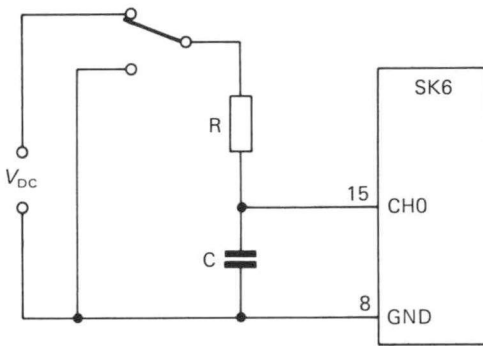
As with all programs, we can understand how the program of figure 5.16b operates if we break it down into sections. The first section, lines 70 to 330, consists of four machine code routines, .SETUP, .POSTRAN, .NEGTRAN, and .STROBE. The comparator output gives a positive transition if the digital

equivalent input exceeds the analogue input, and a negative transition when it falls below the analogue input. The comparator output is therefore connected to both CB1 and CB2, on the user port, and these are programmed in the peripheral control register (PCR at &FE6C) to detect a negative transition on CB1 and a positive transition on CB2 (line 30). Routine .SETUP sets the most significant bit to '1' and checks bit 3 of the interrupt flag register (IFR at &FE6D) for an indication on CB2. Location &70 is used as an indicator to show which bit of the word is being dealt with at each stage, and after eight successive rotates right indicates that the conversion is complete. Routine .POSTRAN rotates the pointer bit in &70 right, and adds the next bit to the existing value in the output port. In other words, we are 'putting' the next bit of the word to '1'. Routine .NEGTRAN rotates the pointer bit right, as before, but now subtracts it from the existing value in the output port, so we are 'taking' the previous bit away as we set the next bit to '1'. Routine .POSTRAN checks the setting of the CB2 flag, bit 3 of &FE6D, and routine .NEGTRAN checks the setting of the CB1 flag, bit 4 of &FE6D, to determine whether to 'put' or 'take' the next bit. On completion of the conversion, the routine .STORE writes the final port value into the data store at the next location indicated by the pointer held in &72 and &73. This pointer is set initially to &2200 at line 50, and can run up to &5600.

The section of program from line 410 to line 500 plots the curve using the values assembled in the data store. The pointer LOC gives the address of the current value to be plotted, and C is a counter to indicate when the required number of points has been reached. This section is part of the bigger section, from line 340 to line 540, which makes use of two procedures. The first, PROC screen, is defined in lines 550 to 620, to draw and label the axes for the display. The second, PROC scale factors, is defined in lines 630 to 710, and sets up various factors used in drawing the graph. Variable T defines the timebase value, Point% is the number of sample values per time period, Scale 1 is the size of the increment on the *x*-axis for each value plotted, and Scale 2% is used in calculating the resolution of the display, in terms of the number of points to be plotted.

As a final example, figure 5.17 shows a simple circuit for charging and discharging a capacitor via a resistor. The rate at which the capacitor charges and discharges is governed by the magnitude of the resistance and capacitance, and the method of analogue-to-digital conversion must be chosen so that it is fast enough to cope with the changing voltages. This form of circuit is mainly of interest in showing the charge and discharge curves, so we can choose large enough values for *C* and *R* to allow us to use the built-in converter this time.

The main program, in figure 5.17b, runs from line 10 to line 180. Line 30 calls the procedure PROC setup, which is defined in lines 190 to 370, to set-up and label the axes, and to ask the user to indicate whether the capacitor is to be charged or discharged. Having received an appropriate response, and the user having operated the change-over switch, lines 40 to 70 are used to detect when the voltage begins to change. Line 80 notes the initial voltage, line 160 notes the final voltage, and lines 100 to 150, in between, plot the graph. The second procedure,



Typical values:
 $C = 1000 \mu\text{F}$
 $R = 18\text{k}$
 Note that capacitors have wide tolerance ranges, so actual time constants may differ from estimated values

(a)

```

10 *FX16,1                               /Select single ADC channel
20 MODE0
30 PROCsetup
40 REPEAT                                /Wait for change in voltage
50  V1%=ADVAL(1)/400
60  V2%=ADVAL(1)/400
70  UNTIL V1%(<)V2%
80  Vs%=V2%*400:X%=70
90  MOVE 70,40
100 REPEAT
110  A=ADVAL(1)
120  B%=(A/84)+40
130  X%=X%+1
140  DRAW X%,B%
150  UNTIL X%=1280
160  Vf%=A
170 PROCcalculate
180 GOTO 10 -----
190 DEF PROCsetup
200  MOVE 70,940:DRAW 70,40:DRAW 1280,40
210  J=32:@%=&20202
220  FOR I=0 TO 2 STEP 0.15               /Adjust print field
230    J=J-2
240    PRINT TAB(0,J);I
250  NEXT
260  J=4:@%=10
270  FOR I=0 TO 18 STEP 4
280    PRINT TAB(J,31);I;
290    J=J+17
300  NEXT
310  PRINT TAB(27,31);"time (s)";
320  PRINT TAB(1,2);"voltage (v)"
330  PRINT TAB(15,1);
340  PRINT"Press C(charging) or D(discharging) when ready";
350  A$=GET$
360  IF A$(<)"C" AND A$(<)"D" GOTO 330
370  PRINT TAB(15,1);"Capacitor now ";
380  IF A$="C" PRINT "charging.";SPC(28)
390  ELSE PRINT "discharging";SPC(28)

```



```

400  ENDPROC
410  DEF PROCcalculate
420  IF A$="D" Y2%=Vs%/EXP(1)
430  IF A$="C" Y2%=Vf%-(Vf%/EXP(1))
440  X=75:Y2%=(Y2%/84)+40
460  REPEAT
470    X=X+1
480  UNTIL POINT(X,Y2%) <> 0 OR X=1280
490  IF X <> 1280 GOTO 530
500  PRINT TAB(15,1);"Try again - capacitor not fully ";
510  IF A$="C" PRINT "charged":GOTO 570
520  ELSE PRINT "discharged":GOTO 570
530  MOVE 70,Y2%:DRAW X,Y2%:DRAW X,40
540  TC=(X-70)*17.55/1210
550  @%=&20309
560  PRINT TAB(15,1);"TIME CONSTANT= ";TC;" seconds"
570  A$=GET$ /Press any key to continue
580  ENDPROC

```

(b)

Figure 5.17 Charge and discharge of a capacitor. (a) Circuit. (b) Program

PROC calculate, is defined in lines 380 to 530. This procedure calculates and displays the time constant of the circuit, using the initial and final values of the voltage previously measured.

And that is where we shall have to stop. The space available in this book has allowed us to give only a broad introduction to the virtually limitless possibilities in interfacing with your BBC microcomputer. The examples that we have given have been chosen to illustrate the basic methods applicable to all interfacing requirements in one way or another, and to show the interdependency of hardware and software. We have given you pointers to what is possible, but interfacing is all about doing, and the greatest satisfaction will come by trying out your own ideas. Here's wishing you success!

Appendix A: Transistor-Transistor Logic

Transistor-transistor logic, TTL, became the dominant logic family in the mid-1960s as the first steps towards integrating circuits became possible. Although metal-oxide-silicon, MOS, logic has since become at least as important, and integration is several orders of magnitude greater, many of the performance criteria of logic families in general are still specified in terms of TTL circuits. The industry standard TTL is the 74 series, in which the allocated number defines the logic operation and pin layout, regardless of manufacturer. The basic TTL circuit, as, for example, in the 7400 two-input NAND gate of figure A.1a, uses a multi-emitter transistor, T1, at the input, and a push-pull circuit, T3 and T4, at the output. This is also known as a *totem-pole* circuit. Transistor T2 acts as a phase-splitter to provide the output stage with the necessary antiphase signals. When either input A or B, or both, is taken to a voltage, V_{IL} , lower than about half a volt (but not negative), the corresponding emitter conducts and transistor T2 cuts off. The emitter current, I_{IL} , has a maximum value of 1.6 milliamps. If both input emitters are taken to a voltage between 2 and 5 volts, V_{IH} , neither emitter conducts and T2 switches on. The control of the gate can best be seen in terms of the current in the 4k resistor: the current is diverted away from T2 through the emitter if the input voltage is low, and flows to T2 if neither input is low. In the latter case, the current at the base of T2 switches it on, so that T4 is also switched on and T3 is switched off. The output voltage then falls to V_{OL} at about 0.2 volts, and the circuit draws in, or *sinks*, up to 16 milliamps of current. When the current is diverted away from T2, it switches off, also switching off T4 and switching on T3. The output voltage now rises to V_{OH} , and the circuit can provide, or *source*, up to 400 microamps. The 130 Ω resistor is included to protect the output transistors, by limiting the current during switching, but also has the effect of reducing the value of V_{OH} . The additional voltage drops across the diode D3, included to ensure correct switching levels, and across T3, mean that the typical value of V_{OH} is only 3.4 volts. The action of the circuit is such that any low voltage, V_{IL} , on an input, sets the gate output high, V_{OH} , and the gate output only goes low, V_{OL} , if there are no low voltages at the inputs. If we think of the high-level voltage as representing logic '1', the circuit acts as a NAND gate. By including inverters at inputs and output, as appropriate, other forms of gate such as AND, OR and NOR can be generated. The same form of circuit is also used in building flipflops, and more complex systems.

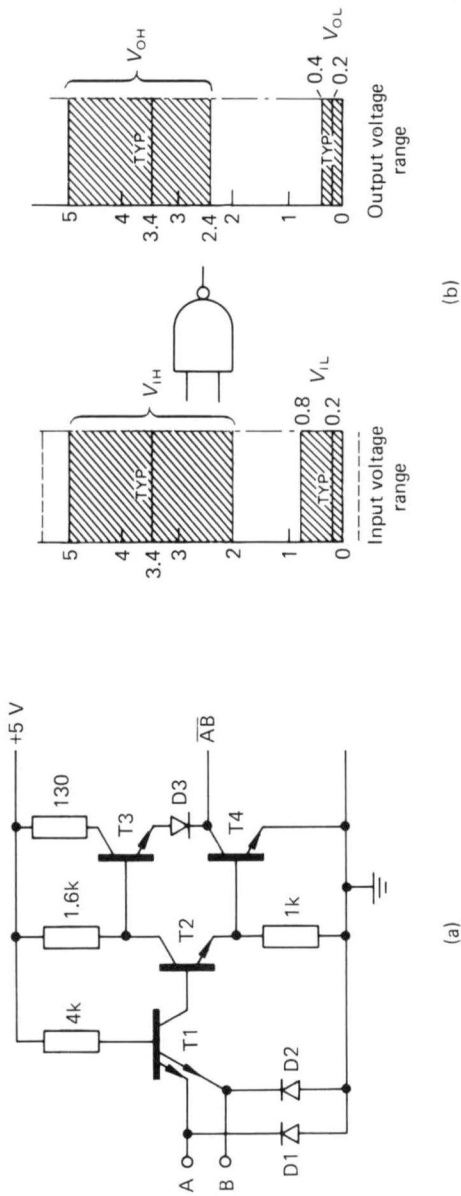


Figure A.1 Standard TTL gate. (a) Circuit. (b) Voltage levels

V_{IH} and V_{IL} are the voltage levels at the gate inputs needed to cause the gate to operate correctly, so generating the output levels, V_{OH} and V_{OL} , which are dependent on the logic conditions. The standard TTL levels have quite a wide tolerance, as shown in figure A.1b, and are specified as follows:

$$\begin{array}{ll} V_{IH} = 1.0 \text{ V minimum} & V_{OH} = 2.4 \text{ V minimum (typically 3.4 V)} \\ V_{IL} = 0.8 \text{ V maximum} & V_{OL} = 0.4 \text{ V maximum (typically 0.2 V)} \end{array}$$

The corresponding current levels are

$$\begin{array}{ll} I_{IH} = 40 \text{ } \mu\text{A maximum} & I_{OH} = -400 \text{ } \mu\text{A maximum} \\ I_{IL} = -1.6 \text{ mA maximum} & I_{OL} = 16 \text{ mA maximum} \end{array}$$

The negative sign indicates that current is flowing out of the gate terminal.

Since each gate can sink 16 milliamps at its output, and each emitter input that it drives requires a current of 1.6 milliamps, when its input voltage is low, the one gate can drive up to ten similar gates. It is said to have a *fanout* of 10. When the driven emitters are taken to the high level, each takes 40 microamps and the driving gate can provide 400 microamps, so again the fanout is 10.

Improvements in fabrication techniques over the years have led to extensive developments in TTL circuitry, and modern TTL is usually the *low-power Schottky* version, 74LS, or, increasingly, the *advanced low-power Schottky*, 74ALS (sometimes designated 74F). The majority of LS TTL circuits do not now use the multi-emitter input transistor, but make use of Schottky diodes, since these give a shorter switching delay. Schottky diodes do not suffer the charge-storage problems of conventional diodes (and transistors) and so switch between states much faster. They are also used as clamping diodes on the transistors to prevent charge-storage. With the help of the Schottky diodes, and by increasing the circuit resistance values somewhat, 74LS circuitry can operate at about twice the speed of standard TTL, but at only one-fifth of the power dissipation. Some of the input and output levels are slightly modified when compared with standard TTL:

$$\begin{array}{ll} I_{IH} = 20 \text{ } \mu\text{A maximum} & V_{OH} = 2.7 \text{ V minimum (typically 2.4 V)} \\ I_{IL} = -0.36 \text{ mA maximum} & V_{OL} = 0.5 \text{ V maximum (typically 0.35 V)} \end{array}$$

The newer 74ALS versions make use of die shrinking techniques recently developed, which, together with modified isolation methods, gives another doubling of the speed and a further halving of the power dissipation.

In many cases the input loading requirements and the output driving capabilities of different versions are quoted in terms of *unit loads* (UL) normalised to the standard TTL figures. Thus, in the high state, one unit load, UL, is 40 microamps, and in the low state, one unit load is 1.6 milliamps. The input load factor for 74LS TTL is then $0.36 \text{ mA}/1.6 \text{ mA} = 0.225 \text{ UL}$ (normally rounded to 0.25 UL) in the low state, and $20 \text{ } \mu\text{A}/40 \text{ } \mu\text{A} = 0.5 \text{ UL}$ in the high state. The output current I_{OL} is 8 milliamps, so the low-level drive factor is $8.0 \text{ mA}/1.6 \text{ mA} = 5 \text{ UL}$, and the high-level drive factor is $400 \text{ } \mu\text{A}/40 \text{ } \mu\text{A} = 10 \text{ UL}$.

Special drive circuits such as the 74LS245 are designed to operate at much higher current levels:

$$\begin{array}{ll} I_{IH} = 30 \mu\text{A maximum} & I_{OH} = -15 \text{ mA maximum} \\ I_{IL} = -0.2 \text{ mA maximum} & I_{OL} = 24 \text{ mA maximum} \end{array}$$

giving load factors of $20 \mu\text{A}/40 \mu\text{A} = 0.5$ UL (high state) and $0.2 \text{ mA}/1.6 \text{ mA} = 0.125$ UL (low state), and drive factors of $15 \text{ mA}/40 \mu\text{A} = 375$ UL (high state) and $24 \text{ mA}/1.6 \text{ mA} = 15$ UL (low state).

Many large-scale integrated circuits use NMOS technology, which is a voltage-controlled form of logic, rather than the current control used in TTL. However, in most cases, the outputs are again standardised in terms of TTL levels. The port B connections of the VIA on the user port, for example, are quoted as having current levels of

$$\begin{array}{ll} I_{IH} = 100 \mu\text{A maximum} & I_{OH} = -1 \text{ mA maximum} \\ I_{IL} = -1.6 \text{ mA maximum} & I_{OL} = 1.6 \text{ mA maximum} \end{array}$$

In terms of unit loads, therefore, the input load factors are $100 \mu\text{A}/40 \mu\text{A} = 2.5$ UL in the high state and $1.6 \text{ mA}/1.6 \text{ mA} = 1$ UL in the low state. The output drive factors are $1 \text{ mA}/40 \mu\text{A} = 25$ UL in the high state and $1.6 \text{ mA}/1.6 \text{ mA} = 1$ UL in the low state. Thus, as long as the factors are correctly matched, NMOS and TTL can be directly connected together. The same applies with CMOS and TTL, except that when driving CMOS from TTL the CMOS, operating at 5 volts, requires a V_{IH} value of 4.3 volts minimum, whereas LS TTL output voltage, V_{OH} , is typically 3.4 volts and can be as low as 2.7 volts. The solution is to include a $10\text{k}\Omega$ pull-up resistor to +5 volts at the output of the TTL gate. Standard CMOS can drive two 74LS inputs directly.

Under certain circumstances it is desirable to connect the outputs of two or more gates together, but a little thought will show that the TTL totem-pole circuit of figure A.1a would not last long if connected to a similar circuit. Very soon, one circuit would be set with its upper transistor, T3, on, while the other would be set with its lower transistor, T4, on, and one of the transistors (normally the upper) would act as a fuse, as a large current flowed from +5 volts to ground!

One occasion for outputs to be connected together is in wired-OR logic, when the common point is to be taken low if gate A, or gate B, or any other gate output goes low. In such circumstances we must use *open-collector* versions of the TTL gates. These are identical to normal gates except that T3, D3 and the 130Ω resistor are omitted. It is then necessary to provide an external resistor at the commoned outputs to act as the collector load. The resistor value depends on the number of gates involved and suitable values are given for different conditions in the manufacturer's literature.

A second requirement for commoned outputs arises in bus-organised systems, such as our microcomputer. Any one of many registers may be required to feed data onto the bus, but only one register is selected at any time. Here we make use of *three-state circuits* which have additional enabling circuitry built in.

When the gate is enabled it acts as a conventional TTL circuit with totem-pole output. But when it is disabled, both transistors in the totem-pole circuit are switched off, so that the circuit presents a high impedance to the bus, and is effectively disconnected from it.

The use of these logic circuits, and the design of logic systems in general, is a fascinating area of study. Anyone wishing to develop the ideas that we have briefly outlined here should consult more specialised books such as *Fundamentals of Modern Digital Systems* by B. R. Bannister and D. G. Whitehead, published by Macmillan, 1983.

Appendix B: Machine Code Programming

Whenever the computer is running it is working methodically through a sequence of instructions which is known as the program. The range of operations from which the instructions can be selected is limited and each operation is very simple, but the processing circuitry can deal with about half a million instructions each second, which makes it very powerful.

The instructions are coded in binary, which is the only form that the circuitry can accept, and it is this form that is called *machine code*. Users, however, find it difficult and very time-consuming to work in binary, and the need to concentrate on getting the coding right tends to restrict the range of programming ideas that we can employ. The obvious thing to do is to construct programs in a form that is more meaningful to the user, and pass the tedious work of conversion into machine code back to the computer. The more meaningful version of machine code is called *assembler language*, and the program that the computer uses to translate the assembler code to the form it requires is the *assembler*. Thus when the computer is running the assembler, it is using the user's assembler language program as data to be translated to another form of data. The user's program in assembler language is the *source code* which is translated to *object code* in binary, and the object code can then be run as a program in its own right. Since the assembler relates closely to the machine code of the particular microprocessor, it is also processor-specific, and it is often preferable to work in a higher-level language which, with minor modifications, could run on different computers if a suitable translating program is provided. Many high-level languages have been designed, but by far the most popular for personal computers is BASIC which is an acronym for Beginners' All-purpose Symbolic Instruction Code.

Programs written in BASIC, or any other high-level language, use pseudo-English statements which are much more understandable to the user, but are meaningless strings of characters to the circuitry that will actually carry out the operations defined. Again, each BASIC statement has to be translated, and a special program is provided which operates on the user's program and converts it to a form similar to assembler. It is then further converted to machine code form which is acceptable to the computer circuitry. Most high-level languages employing this method of translation make use of a type of program known as a *compiler*; the complete user program is translated to machine code form, so that, once the compilation is completed, the program can be run and rerun as often as required, without further translation. BASIC, however, is normally dealt with in a different way. Each

BASIC statement is interpreted into machine code form, and is immediately carried out. No copy is kept of the machine code form and the *interpreter* program must re-interpret each statement if it returns to it. This makes BASIC much slower than compiled languages, especially where repetitive loops are concerned.

The operation of the different levels of program can conveniently be thought of as a series of software *shells*, built around the processing circuitry of the computer. The hardware forms the kernel, and the assembler language makes the innermost shell. BASIC, or some other high-level language, forms another shell, and so on. By designing different additional shells, we can effectively change the characteristics of the computer, as far as a user is concerned, without having to change any of the hardware. Applications packages, such as word-processing programs, for example, can be added as extra shells, and users need no knowledge of how the computer is operating internally. However, there are advantages in working in assembler language, especially when we are interfacing other equipment to the computer; programs are shorter and operate much faster and they also make much more efficient use of memory. Furthermore, knowing in more detail how the processor actually works allows us to exploit some of its special features, which are not fully used by the high-level languages. There are, of course, times when it is easier to use BASIC, and a very important feature of the BASIC interpreter held in the ROMs in the BBC microcomputer is the ability to mix machine code sections with BASIC statements, to get the best of both worlds. In effect, we can operate in two shells at the same time.

The processor used in the BBC computer is the 6502, which is one of the processors in the 6500 series. The microprocessor operates on data in eight-bit bytes, and in common with all microprocessors it has a range of registers which are used in carrying out specific operations. Many of the registers are only for internal use — what we call *housekeeping* operations — and, as far as the programmer is concerned, the 6502 has three general registers, the accumulator and index registers X and Y. In addition, three of the special registers are important: the program counter, PC, the stack pointer, SP, and the processor status register.

The program counter indicates the location in program memory where the next instruction is to be found. It is a 16-bit counter since any part of the 65 536 location memory space may be used for program purposes. At the beginning of each instruction cycle the instruction must be fetched from memory. The complete information necessary to specify an instruction may require one, two or three bytes, but in all cases the first byte fetched is the *op-code* byte. This byte is decoded to indicate the type of operation involved, and it also indicates how many more bytes, if any, are needed to complete the instruction. Each op-code byte is associated with a mnemonic from the instruction set listing, such as LDA for *load accumulator*, STX for *store content of X register*, and so on. The complete set of mnemonics is given in appendix D, section 1 together with the total number of bytes needed in each case. The additional bytes are necessary to specify the operands that are to be used in carrying out the instruction. LDA, for example, means load accumulator with a data byte, and we need to specify the

location in memory, or elsewhere, that is to be used to provide the data. There are several different ways in which the operands can be specified, and these *addressing modes* are explained in appendix D, section 1. In building-up the instruction, each time a byte is taken from memory the program counter is incremented, so that, when all the necessary bytes have been fetched, the program counter points to the start of the next instruction, ready for the next fetch sequence to begin. The fetch sequence is complete when various housekeeping operations have been carried out, and the execution phase follows immediately. During this phase the operations indicated by the instruction are carried out, and, where appropriate, the *condition flags* are set. The condition flags are individual flipflops which indicate the status of the processor after each operation, and together they form the *condition code* which is held in the *processor status register*.

The flags are

carry, C	set to '1' if a carry occurred at the most-significant bit of the accumulator
zero, Z	set to '1' if accumulator value became zero
overflow, V	set to '1' if the number range of the accumulator was exceeded
negative, N	set to '1' if the msb of the accumulator became '1', so indicating a negative value
decimal mode, D	set to '1' if arithmetic operations are to be in binary-coded decimal
interrupt disable, I	set to '1' if external interrupts are disabled
break command, B	set to '1' if the instruction was BRK.

The number of mnemonics in the instruction set is 56, though, when we take all the addressing variations into account, the number of instruction types available is almost three times that. The instructions can be split into four main groups:

- Those involved with transfers of data between registers and to and from memory.
- Arithmetic and logic operations, such as add, AND, compare, decrement, exclusive-OR, increment, rotate, etc.
- Flag operations, such as set or clear the carry flag, set or clear the decimal flag, etc.

The majority of the instructions in these first three groups affect the settings of at least some of the flags in the processor status register, to reflect the new conditions.

- Branch instructions. This group includes jump, JMP, jump to subroutine, JSR, return from subroutine, RTS, and several conditional jumps, such as branch if carry flag set, BCS, branch if minus, BMI (that is, branch if negative flag set), and so on. The branch instructions react to the flag settings but do not modify them.

The effect of all jump instructions is to change the program counter value, so that the next instruction is taken not from the next sequential location but from the indicated new location. Most jumps and branches are non-return, and subsequent instructions are taken sequentially from the new position. Jump to subroutine, JSR, however, retains the original value of the program counter, so that when a return instruction, RTS, is encountered, the original value of the program counter can be restored. The value is stored until required in a section of memory known as the *stack*, which is exactly the same as all the other read/write random access memory, except that it is addressed via a counter known as the *stack pointer*, SP, which is automatically incremented or decremented when used. Special instructions allow us to write to and read from the stack, and these are known as *push* and *pull* respectively. The stack, in the BBC computer, is located in page 1 of memory and runs from &0100 to &01FF.

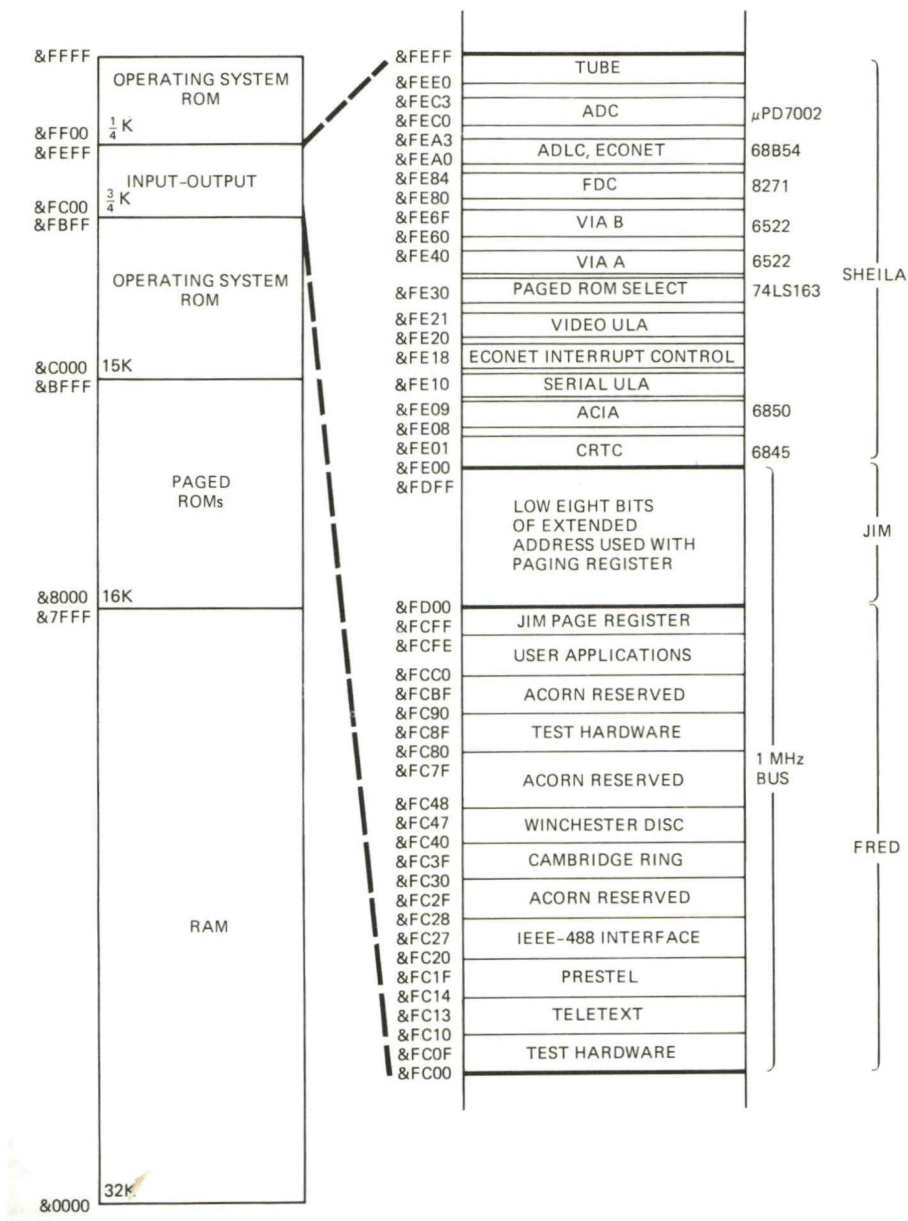
The assembler provided in the computer is a conventional two-pass assembler, which allows the use of labels to indicate locations. A small section of program, for example, could be

```
.START  
LDA &FE60 : STA &70  
CMP #0 : BEQ START  
LDA #0 : STA ACR
```

During assembly, the label .START is allocated the value of the address at which the op-code LDA is stored, and, whenever the label is encountered later on, the address is substituted. Individual instructions are terminated either by the *return* at the end of a line, or by the colon. This small section of machine code loads the accumulator from location &FE60, stores that value at location &70, then compares the value with zero and, if equal to zero, jumps back to the location labelled START. Otherwise it continues to the next instruction following, and loads the accumulator with zero; it finally stores the zero in the register location labelled ACR.

This has of necessity been only a brief introduction to assembly language programming. Section 43 of the *User Guide* contains a full account of the methods used to link the assembler sections, such as we have discussed, into a BASIC program. A much more thorough coverage of the whole field of machine code programming will be found in *Assembly Language Programming for the BBC Microcomputer* (second edition) by Ian Birnbaum, published by Macmillan, 1984.

Appendix C: Input-Output Memory Map



Appendix D: Data Sheets

1. SY6500 8-bit Microprocessor Family
Reproduced by courtesy of Synertek Inc.,
Honeywell House, Charles Square, Bracknell, Berkshire, UK
2. SY6522 Versatile Interface Adapter, VIA
Reproduced by courtesy of Synertek Inc.,
Honeywell House, Charles Square, Bracknell, Berkshire, UK
3. μ PD7002 12-bit Binary A/D Converter
Reproduced by courtesy of NEC Electronics (UK) Ltd,
Devonshire House, Bank Street, Lutterworth, Leicestershire, UK
4. SN74LS244 Octal Buffers and Line Drivers with 3-state Outputs
Reproduced by courtesy of Texas Instruments Ltd,
Manton Lane, Bedford, UK
5. SN74LS245 Octal Bus Transceivers with 3-state Outputs
Reproduced by courtesy of Texas Instruments Ltd,
Manton Lane, Bedford, UK

The manufacturers reserve the right to make changes at any time in order to improve design and to supply the best product possible. Therefore, they cannot assume any responsibility for the information given or represent that it is free from patent infringement.

Synertek

SY6500

8-Bit Microprocessor Family

Features

- Single 5 V $\pm 5\%$ power supply
- N channel, silicon gate, depletion load technology
- Eight bit parallel processing
- 56 Instructions
- Decimal and binary arithmetic
- Thirteen addressing modes
- True indexing capability
- Programmable stack pointer
- Variable length stack
- Interrupt capability
- Non-maskable interrupt
- Use with any type or speed memory
- Bi-directional Data Bus
- Instruction decoding and control
- Addressable memory range of up to 65 K bytes
- "Ready" input
- Direct memory access capability
- Bus compatible with MC6800
- Choice of external or on-board clocks
- 1 MHz, 2 MHz, 3 MHz and 4 MHz operation
- On-chip clock options
 - * External single clock input
 - * Crystal time base input
- 40 and 28 pin package versions
- Pipeline architecture

Description

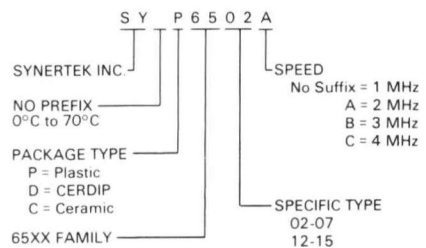
The SY6500 Series Microprocessors represent the first totally software compatible microprocessor family. This family of products includes a range of software compatible microprocessors which provide a selection of addressable memory range, interrupt input options and on-chip clock oscillators and drivers. All of the microprocessors in the SY6500 family are software compatible within the group and are bus compatible with the MC6800 product offering.

The family includes six microprocessors with on-board clock oscillators and drivers and four microprocessors driven by external clocks. The on-chip clock versions are aimed at high performance, low cost applications where single phase inputs or crystals provide the time base. The external clock versions are geared for the multi-processor system applications where maximum timing control is mandatory. All versions of the microprocessors are available in 1 MHz, 2 MHz, 3 MHz and 4 MHz maximum operating frequencies.

Members of the Family

PART NUMBERS	CLOCKS	PINS	IRQ	NMI	RYD	ADDRESSING
SY6502	On-Chip	40	✓	✓	✓	64 K
SY6503	"	28	✓	✓		4 K
SY6504	"	28	✓			8 K
SY6505	"	28	✓		✓	4 K
SY6506	"	28	✓			4 K
SY6507	"	28			✓	8 K
SY6512	External	40	✓	✓	✓	64 K
SY6513	"	28	✓	✓		4 K
SY6514	"	28	✓			8 K
SY6515	"	28	✓		✓	4 K

Ordering Information



Only 6502 and 6512 are available in 3 and 4 MHz

Synertek.

SY6500

Pin Functions

Clocks (Φ_1, Φ_2)

The SY651X requires a two phase non-overlapping clock that runs at the V_{CC} voltage level.

The SY650X clocks are supplied with an internal clock generator. The frequency of these clocks is externally controlled. Clock generator circuits are shown elsewhere in this data sheet.

Address Bus (A_0-A_{15}) (See sections on each micro for respective address lines on those devices.)

These outputs are TTL compatible, capable of driving one standard TTL load and 130 pF.

Data Bus (DB_0-DB_7)

Eight pins are used for the data bus. This is a bi-directional bus, transferring data to and from the device and peripherals. The outputs are three-state buffers, capable of driving one standard TTL load and 130 pF.

Data Bus Enable (DBE)

This TTL compatible input allows external control of the three-state data output buffers and will enable the microprocessor bus driver when in the high state. In normal operation DBE would be driven by the phase two (Φ_2) clock, thus allowing data output from microprocessor only during Φ_2 . During the read cycle, the data bus drivers are internally disabled, becoming essentially an open circuit. To disable data bus drivers externally, DBE should be held low. This signal is available on the SY6512, only.

Ready (RDY)

This input signal allows the user to halt the microprocessor on all cycles except write cycles. A negative transition to the low state during or coincident with phase one (Φ_1) will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (Φ_2) in which the Ready signal is low. This feature allows microprocessor interfacing with low speed PROMS as well as fast (max. 2 cycle) Direct Memory Access (DMA). If ready is low during a write cycle, it is ignored until the following read operation. Ready transitions must not be permitted during Φ_2 time.

Interrupt Request (\overline{IRQ})

This TTL level input requests that an interrupt sequence begin within the microprocessor. The microprocessor will complete the current instruction being executed before recognizing the request. At that time, the interrupt mask bit in the Status Code Register will be examined. If the interrupt mask flag is not set, the microprocessor will begin an interrupt sequence. The Program Counter and Processor Status Register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further interrupts may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, therefore transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A $3K\Omega$ external resistor should be used for proper wire-OR operation.

Non-Maskable Interrupt (\overline{NMI})

A negative going transition on this input requests that a non-maskable interrupt sequence be generated within the microprocessor.

NMI is an unconditional interrupt. Following completion of the current instruction, the sequence of operations defined for \overline{IRQ} will be performed, regardless of the state interrupt mask flag. The vector address loaded into the program counter, low and high, are locations FFFA and FFFB respectively, thereby transferring program control to the memory vector located at these addresses. The instructions loaded at these locations cause the microprocessor to branch to a non-maskable interrupt routine in memory.

NMI also requires an external $3K\Omega$ resistor to V_{CC} for proper wire-OR operations.

Inputs \overline{IRQ} and \overline{NMI} are hardware interrupts lines that are sampled during Φ_2 (phase 2) and will begin the appropriate interrupt routine on the Φ_1 (phase 1) following the completion of the current instruction.

Set Overflow Flag (S.O.)

A NEGATIVE going edge on this input sets the overflow bit in the Status Code Register. This signal is sampled on the trailing edge of Φ_1 .

SYNC

This output line is provided to identify those cycles in which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during Φ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the Φ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

Reset (\overline{RES})

This input is used to reset or start the microprocessor from a power down condition. During the time that this line is held low, writing to or from the microprocessor is inhibited. When a positive edge is detected on the input, the microprocessor will immediately begin the reset sequence.

After a system initialization time of six clock cycles, the mask interrupt flag will be set and the microprocessor will load the program counter from the memory vector locations FFFC and FFFD. This is the start location for program control.

After V_{CC} reaches 4.75 volts in a power up routine, reset must be held low for at least two clock cycles. At this time the R/\overline{W} and SYNC signal will become valid.

When the reset signal goes high following these two clock cycles, the microprocessor will proceed with the normal reset procedure detailed above.

Read/Write (R/\overline{W})

This output signal is used to control the direction of data transfers between the processor and other circuits on the data bus. A high level on R/\overline{W} signifies data into the processor; a low is for data transfer out of the processor.

Programming Characteristics

INSTRUCTION SET — ALPHABETIC SEQUENCE

ADC Add Memory to Accumulator with Carry	LDA Load Accumulator with Memory
AND "AND" Memory with Accumulator	LDX Load Index X with Memory
ASL Shift left One Bit (Memory or Accumulator)	LDY Load Index Y with Memory
BCC Branch on Carry Clear	LSR Shift One Bit Right (Memory or Accumulator)
BCS Branch on Carry Set	NOP No Operation
BEQ Branch on Result Zero	ORA "OR" Memory with Accumulator
BIT Test Bits in Memory with Accumulator	PHA Push Accumulator on Stack
BMI Branch on Result Minus	PHP Push Processor Status on Stack
BNE Branch on Result not Zero	PLA Pull Accumulator from Stack
BPL Branch on Result Plus	PLP Pull Processor Status from Stack
BRK Force Break	ROL Rotate One Bit Left (Memory or Accumulator)
BVC Branch on Overflow Clear	ROR Rotate One Bit Right (Memory or Accumulator)
BVS Branch on Overflow Set	RTI Return from Interrupt
CLC Clear Carry Flag	RTS Return from Subroutine
CLD Clear Decimal Mode	SBC Subtract Memory from Accumulator with Borrow
CLI Clear Interrupt Disable Bit	SEC Set Carry Flag
CLV Clear Overflow Flag	SED Set Decimal Mode
CMP Compare Memory and Accumulator	SEI Set Interrupt Disable Status
CPX Compare Memory and Index X	STA Store Accumulator in Memory
CPY Compare Memory and Index Y	STX Store Index X in Memory
DEC Decrement Memory by One	STY Store Index Y in Memory
DEX Decrement Index X by One	TAX Transfer Accumulator to Index X
DEY Decrement Index Y by One	TAY Transfer Accumulator to Index Y
EOR "Exclusive-or" Memory with Accumulator	TSX Transfer Stack Pointer to Index X
INC Increment Memory by One	TXA Transfer Index X to Accumulator
INX Increment Index X by One	TXS Transfer Index X to Stack Pointer
INY Increment Index Y by One	TYA Transfer Index Y to Accumulator
JMP Jump to New Location	
JSR Jump to New Location Saving Return Address	

ADDRESSING MODES

Accumulator Addressing

This form of addressing is represented with a one byte instruction, implying an operation on the accumulator.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction, with no further memory addressing required.

Absolute Addressing

In absolute addressing, the second byte of the instruction specifies the eight low order bits of the effective address while the third byte specifies the eight high order bits. Thus, the absolute addressing mode allows access to the entire 65K bytes of addressable memory.

Zero page Addressing

The zero page instructions allow for shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. Careful use of the zero page can result in significant increase in code efficiency.

Indexed Zero Page Addressing — (X, Y indexing)

This form of addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero

Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally due to the "Zero Page" addressing nature of this mode, no carry is added to the high order 8 bits of memory and crossing of page boundaries does not occur.

Indexed Absolute Addressing — (X, Y indexing)

This form of addressing is used in conjunction with X and Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields resulting in reduced coding and execution time.

Implied Addressing

In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Synertek.

SY6500

Relative Addressing

Relative addressing is used only with branch instructions and establishes a destination for the conditional branch.

The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the lower eight bits of the program counter when the counter is set at the next instruction. The range of the offset is -128 to +127 bytes from the next instruction.

Indexed Indirect Addressing

In indexed indirect addressing (referred to as (Indirect,X)), the second byte of the instruction is added to the contents of the X index register, discarding the carry. The result of this addition points to a memory location on page zero whose contents is the low order eight bits of the effective address. The next memory location in page zero contains the high order eight bits of the effective address. Both memory locations specifying the high and low order bytes of the effective address must be in page zero.

Indirect Indexed Addressing

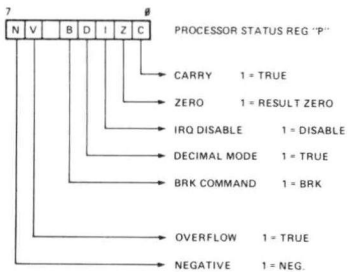
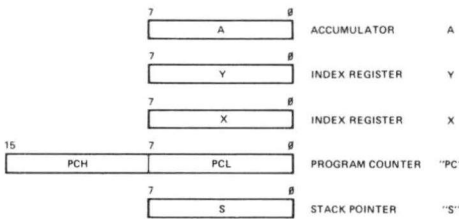
In indirect indexed addressing (referred to as (Indirect,Y)), the second byte of the instruction points to a memory location in page zero. The contents of this memory location is added to the contents of the Y index register, the result being the low order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high order eight bits of the effective address.

Absolute Indirect

The second byte of the instruction contains the low order eight bits of a memory location. The high order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low order byte of the effective address. The next memory location contains the high order byte of the effective address which is loaded into the sixteen bits of the program counter.

Programming Characteristics

PROGRAMMING MODEL






Synertek.

SY6500

INSTRUCTION SET — OP CODES, EXECUTION TIME, MEMORY REQUIREMENTS

INSTRUCTIONS	OPERATION	IMMEDIATE		ABSOLUTE		ZERO PAGE		ACCUM		IMPLICIT		IND. X		IND. Y		Z PAGE 1		A8 X		A8 Y		RELATIVE		INDIRECT		Z PAGE 2		CONDITION CODES					
		OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	N	Z	C	I	D	V
ADC	A ← M + C ← A	(4) (11)	69	2	2	6D	4	3	65	3	2			61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3					
AND	A ← M ← A	(11)	29	2	2	2D	4	3	75	3	2			21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3					
ASL	C ← 7 ← 0 ← 0					9E	6	3	9E	5	2	9A	2	1						16	6	2	1E	7	3								
BCC	BRANCH ON C=0	(2)																															
BCS	BRANCH ON C=1	(2)																															
BEQ	BRANCH ON Z=1	(2)																															
BIT	A ← M					2C	4	3	24	3	2																						
BMI	BRANCH ON N=1	(2)																															
BNE	BRANCH ON Z=0	(2)																															
BPL	BRANCH ON N=0	(2)																															
BRK	(See Fig. 1)													90	7	1																	
BVC	BRANCH ON V=0	(2)																															
BVS	BRANCH ON V=1	(2)																															
CLC	0 ← C													18	2	1																	
CLD	0 ← D													D8	2	1																	
CLI	0 ← I													58	2	1																	
CLV	0 ← V													88	2	1																	
CMP	A ← M	(11)	C9	2	2	CD	4	3	C5	3	2			C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3					
CPX	X ← M					E8	2	2	EC	4	3	E4	3	2																			
CPY	Y ← M					C8	2	2	CC	4	3	C4	3	2																			
DEC	M ← M - 1					CE	6	3	C6	5	2																						
DEX	X ← X - 1													CA	2	1																	
DEY	Y ← Y - 1													8B	2	1																	
EOR	A ← M ← A	(11)	49	2	2	4D	4	3	45	3	2			41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3					
INC	M ← M + 1					EE	6	3	E6	5	2																						
INX	X ← X + 1													EB	2	1																	
INY	Y ← Y + 1													CB	2	1																	
JMP	JUMP TO NEW LOC					4C	3	3																									
JSR	(See Fig. 2) JUMP SUB					20	6	3																									
LDA	M ← A	(11)	A9	2	2	AD	4	3	A5	3	2			A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3					

INSTRUC	OPERATION	IMMEDIATE		ABSOLUTE		ZERO PAGE		ACCUM		IMPLICIT		IND. X1		IND. Y		Z PAGE 1		A8 X		A8 Y		RELATIVE		INDIRECT		Z PAGE 2		CONDITION CODES					
		OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	OP N	#	N	Z	C	I	D	V
LDX	M ← X	(11)	A2	2	AC	4	3	A6	3	2																							
LDY	M ← Y	(11)	A8	2	2	AE	4	3	AA	3	2																						
LSR	 C					4E	6	3	4E	5	2	4A	2	1																			
NOP	NO OPERATION													EA	2	1																	
ORA	A ← M ← A					99	2	2	9D	4	3	95	3	2																			
PHA	A ← M ₁ S ← S													48	3	1																	
PHP	P ← M ₁ S ← S													86	3	1																	
PLA	S ← M ₁ ← S M ₁ ← A													68	4	1																	
PLP	S ← M ₁ ← S M ₁ ← P													28	4	1																	
ROR	 C					2E	6	3	2E	5	2	2A	2	1																			
ROR	 C					6E	6	3	6E	5	2	6A	2	1																			
RTI	(See Fig. 1) RTRN INT													48	6	1																	
RTS	(See Fig. 2) RTRN SUB													68	6	1																	
SBC	A ← M ← A	(11)	E9	2	2	ED	4	3	E5	3	2			E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3					
SEC	1 ← C													38	2	1																	
SED	1 ← D													F8	2	1																	
SEI	1 ← I													78	2	1																	
STA	A ← M					8D	4	3	85	3	2			B1	6	2	91	6	2	95	4	2	9D	5	3	99	5	3					
STX	X ← M					8E	4	3	86	3	2																						
STY	Y ← M					8C	4	3	84	3	2																						
TAX	A ← X													AA	2	1																	
TAY	A ← Y													AB	2	1																	
TSX	S ← X													BA	2	1																	
TXA	X ← A													8A	2	1																	
TXS	X ← S													9A	2	1																	
TYA	Y ← A													98	2	1																	

(1) ADD 1 TO "N" IF PAGE BOUNDARY IS CROSSED

(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE

(3) CARRY NOT > BELOW

(4) IF IN DECIMAL MODE Z FLAG IS INVALID

ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X

Y INDEX Y

A ACCUMULATOR

M MEMORY PER EFFECTIVE ADDRESS

M₁ MEMORY PER STACK POINTER

+ ADD

- SUBTRACT

^ AND

V OR

✓ EXCLUSIVE OR

✓ MODIFIED

- NOT MODIFIED

M₁ MEMORY BIT 7M₂ MEMORY BIT 6

N NO CYCLES

0 NO BYTES

Synertek.

SY6522/SY6522A

Versatile Interface Adapter (VIA)

Features

- Two 8-Bit Bidirectional I/O Ports
- Two 16-Bit Programmable Timer/Counters
- Serial Data Port
- Single +5V Power Supply
- TTL Compatible
- CMOS Compatible Peripheral Port A lines
- Expanded "Handshake" Capability Allows Positive Control of Data Transfers Between Processor and Peripheral Devices
- Latched Output and Input Registers
- 1 MHz and 2 MHz Operation

Description

The SY6522 Versatile Interface Adapter (VIA) is a very flexible I/O control device. In addition, this device contains a pair of very powerful 16-bit interval timers, a serial-to-parallel/parallel-to-serial shift register and input data latching on the peripheral ports. Expanded handshaking capability allows control of bi-directional data transfers between VIA's in multiple processor systems.

Control of peripheral devices is handled primarily through two 8-bit bi-directional ports. Each line can

be programmed as either an input or an output. Several peripheral I/O lines can be controlled directly from the interval timers for generating programmable frequency square waves or for counting externally generated pulses. To facilitate control of the many powerful features of this chip, an interrupt flag register, an interrupt enable register and a pair of function control registers are provided.

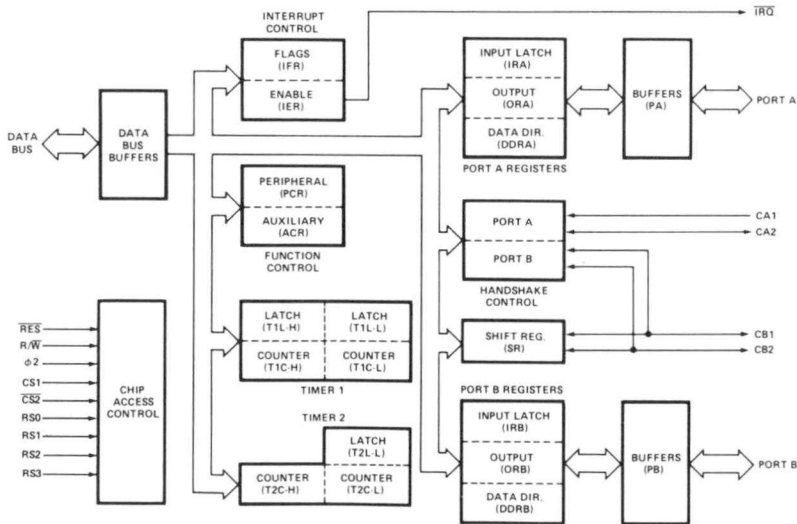


Figure 1. SY6522 Block Diagram

Synertek

SY6522/SY6522A**Absolute Maximum Ratings***

Rating	Symbol	Value	Unit
Supply Voltage	V _{CC}	-0.3 to +7.0	V
Input Voltage	V _{IN}	-0.3 to +7.0	V
Operating Temperature Range	T _A	0 to +70	°C
Storage Temperature Range	T _{stg}	-55 to +150	°C

Comment*

This device contains circuitry to protect the inputs against damage due to high static voltages. However, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages.

Electrical Characteristics(V_{CC} = 5.0V ± 5%, T_A = 0-70°C unless otherwise noted)

Symbol	Characteristic	Min.	Max.	Unit
V _{IH}	Input High Voltage (all except $\phi 2$)	2.4	V _{CC}	V
V _{CH}	Clock High Voltage	2.4	V _{CC}	V
V _{IL}	Input Low Voltage	-0.3	0.4	V
I _{IN}	Input Leakage Current — V _{IN} = 0 to 5 Vdc R/W, RES, RS0, RS1, RS2, RS3, CS1, CS2, CA1, $\phi 2$	—	±2.5	μA
I _{TSI}	Off-state Input Current — V _{IN} = .4 to 2.4V V _{CC} = Max, D0 to D7	—	±10	μA
I _{IH}	Input High Current — V _{IH} = 2.4V PA0-PA7, CA2, PB0-PB7, CB1, CB2	-100	—	μA
I _{IL}	Input Low Current — V _{IL} = 0.4 Vdc PA0-PA7, CA2, PB0-PB7, CB1, CB2	—	-1.6	mA
V _{OH}	Output High Voltage V _{CC} = min, I _{load} = -100 μAdc PA0-PA7, CA2, PB0-PB7, CB1, CB2	2.4	—	V
V _{OL}	Output Low Voltage V _{CC} = min, I _{load} = 1.6 mAdc	—	0.4	V
I _{OH}	Output High Current (Sourcing) V _{OH} = 2.4V V _{OH} = 1.5V (PB0-PB7)	-100 -1.0	— —	μA mA
I _{OL}	Output Low Current (Sinking) V _{OL} = 0.4 Vdc	1.6	—	mA
I _{OFF}	Output Leakage Current (Off state) IRQ	—	10	μA
C _{IN}	Input Capacitance — T _A = 25°C, f = 1 MHz (R/W, RES, RS0, RS1, RS2, RS3, CS1, CS2, D0-D7, PA0-PA7, CA1, CA2, PB0-PB7) (CB1, CB2) ($\phi 2$ Input)	— — —	7.0 10 20	pF pF pF
C _{OUT}	Output Capacitance — T _A = 25°C, f = 1 MHz	—	10	pF
P _D	Power Dissipation (V _{CC} = 5.25V)	—	700	mW

Synertek

SY6522/SY6522A

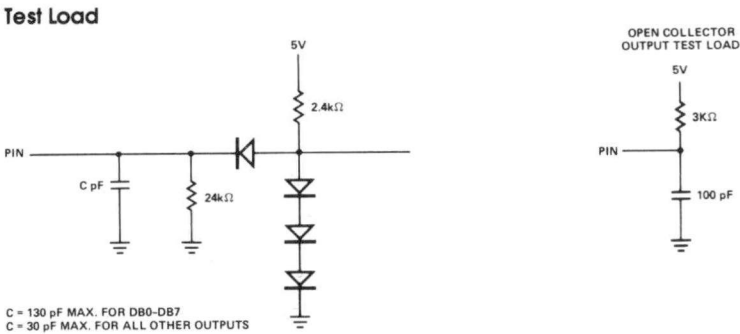


Figure 2. Test Load (for all Dynamic Parameters)

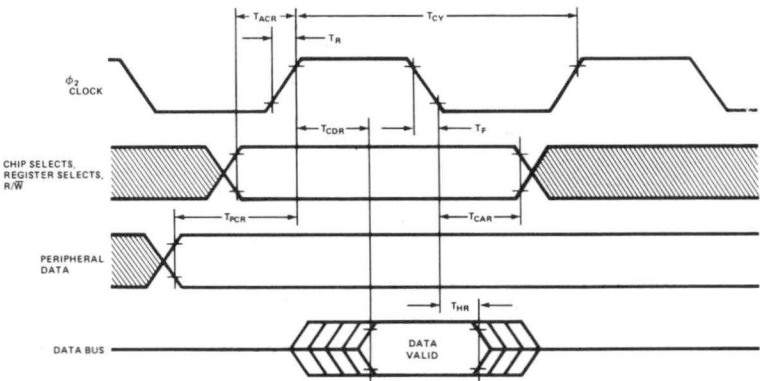


Figure 3. Read Timing Characteristics

Read Timing Characteristics (Figure 3)

Symbol	Parameter	SY6522		SY6522A		Unit
		Min.	Max.	Min.	Max.	
T _{CY}	Cycle Time	1	50	0.5	50	μs
T _{ACR}	Address Set-Up Time	180	—	90	—	ns
T _{CAR}	Address Hold Time	0	—	0	—	ns
T _{PCR}	Peripheral Data Set-Up Time	300	—	300	—	ns
T _{CDR}	Data Bus Delay Time	—	340	—	200	ns
T _{HR}	Data Bus Hold Time	10	—	10	—	ns

NOTE: tr, tf = 10 to 30ns.

Synertek

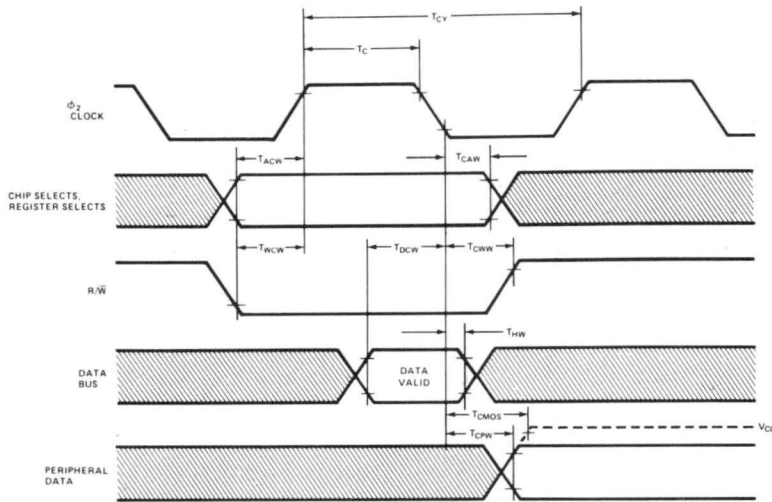
SY6522/SY6522A

Figure 4. Write Timing Characteristics

Write Timing Characteristics (Figure 4)

Symbol	Parameter	SY6522		SY6522A		Unit
		Min.	Max.	Min.	Max.	
T_{CY}	Cycle Time	1	50	0.50	50	μs
T_C	$\phi 2$ Pulse Width	0.44	25	0.22	25	μs
T_{ACW}	Address Set-Up Time	180	—	90	—	ns
T_{CAW}	Address Hold Time	0	—	0	—	ns
T_{WCW}	R/\bar{W} Set-Up Time	180	—	90	—	ns
T_{CWW}	R/\bar{W} Hold Time	0	—	0	—	ns
T_{DCW}	Data Bus Set-Up Time	300	—	150	—	ns
T_{HW}	Data Bus Hold Time	10	—	10	—	ns
T_{CPW}	Peripheral Data Delay Time	—	1.0	—	1.0	μs
T_{CMOS}	Peripheral Data Delay Time to CMOS Levels	—	2.0	—	2.0	μs

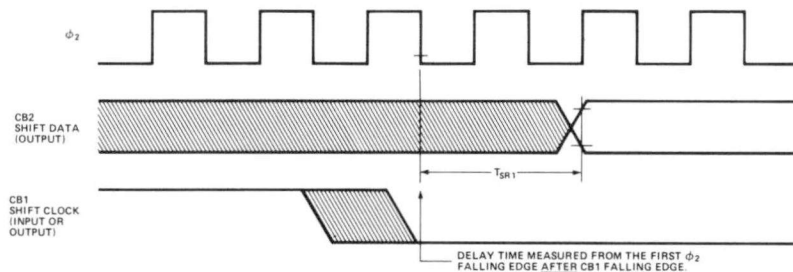
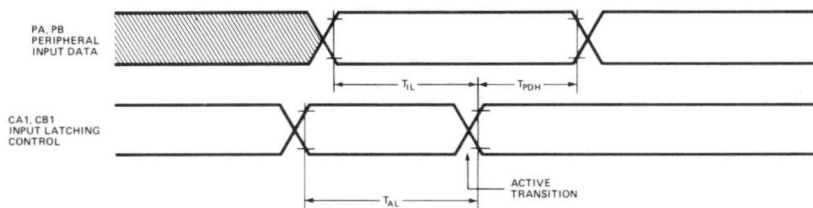
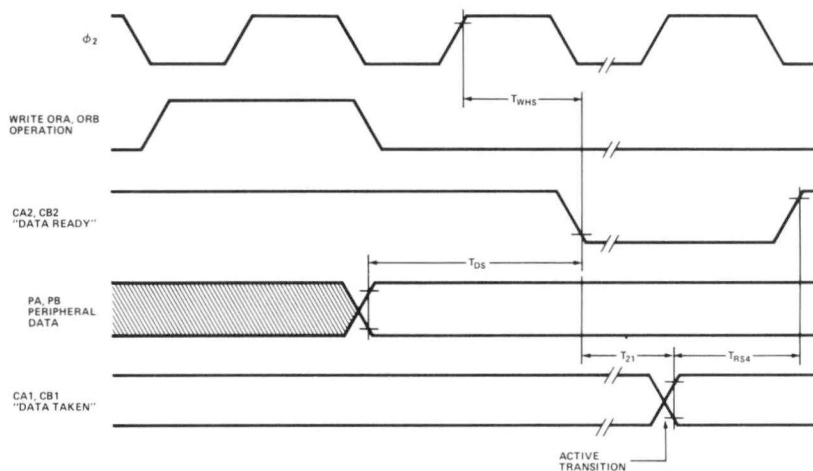
NOTE: t_r , t_f = 10 to 30ns.

Synertek.

SY6522/SY6522A

Peripheral Interface Characteristics

Symbol	Characteristic	Min.	Max.	Typ.	Unit	Figure
t_r, t_f	Rise and Fall Time for CA1, CB1, CA2, and CB2 Input Signals	—	1.0		μ s	—
T_{CA2}	Delay Time, Clock Negative Transition to CA2 Negative Transition (read handshake or pulse mode)	—	1.0		μ s	5a, 5b
T_{RS}	Delay Time, Clock Negative Transition to CA2 Positive Transition (pulse mode)	—	1.0		μ s	5a
T_{RS2}	Delay Time, CA1 Active Transition to CA2 Positive Transition (handshake mode)	—	2.0		μ s	5b
T_{WHS}	Delay Time, Clock Positive Transition to CA2 or CB2 Negative Transition (write handshake)	0.05	1.0		μ s	5c, 5d
T_{DS}	Delay Time, Peripheral Data Valid to CB2 Negative Transition	0.20	1.5		μ s	5c, 5d
T_{RS3}	Delay Time, Clock Transition to CA2 or CB2 Positive Transition (pulse mode)	—	1.0		μ s	5c
T_{RS4}	Delay Time, CA1 or CB1 Active Transition to CA2 or CB2 Positive Transition (handshake mode)	—	2.0		μ s	5d
T_{21}	Delay Time Required from CA2 Output to CA1 Active Transition (handshake mode)	400	—		ns	5d
T_{IL}	Set-up Time, Peripheral Data Valid to CA1 or CB1 Active Transition (input latching)	300	—		ns	5e
T_{SR1}	Shift-Out Delay Time — Time from ϕ_2 Falling Edge to CB2 Data Out	—	300		ns	5f
T_{SR2}	Shift-In Setup Time — Time from CB2 Data in to ϕ_2 Rising Edge	300	—		ns	5g
T_{SR3}	External Shift Clock (CB1) Setup Time Relative to ϕ_2 Trailing Edge	100	T_{CY}		ns	5g
T_{IPW}	Pulse Width — PB6 Input Pulse	$2 \times T_{CY}$	—			5i
T_{ICW}	Pulse Width — CB1 Input Clock	$2 \times T_{CY}$	—			5h
T_{IPS}	Pulse Spacing — PB6 Input Pulse	$2 \times T_{CY}$	—			5i
T_{ICS}	Pulse Spacing — CB1 Input Pulse	$2 \times T_{CY}$	—			5h
T_{AL}	CA1, CB1 Set Up Prior to Transition to Arm Latch	$T_C + 50$	—		ns	5e
T_{PDH}	Peripheral Data Hold After CA1, CB1 Transition	150	—		ns	5e
T_{PWI}	Set Up Required on CA1, CB1, CA2 or CB2 Prior to Triggering Edge	$T_C + 50$	—		ns	5j
T_{DPR}	Shift Register Clock — Delay from ϕ_2 to CB1 Rising Edge			200	ns	5k
T_{DPL}	to CB1 Falling Edge			125	ns	5k



Synertek

SY6522/SY6522A

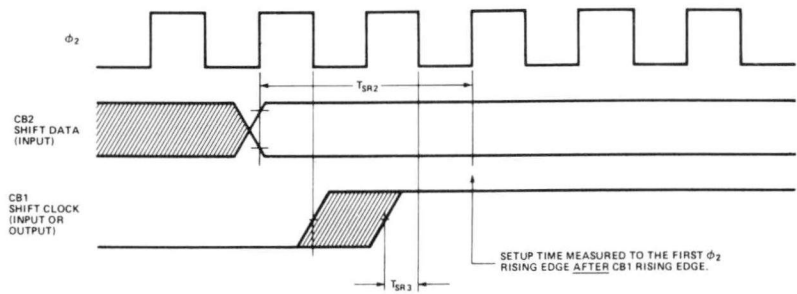


Figure 5g. Timing for Shift In with Internal or External Shift Clocking

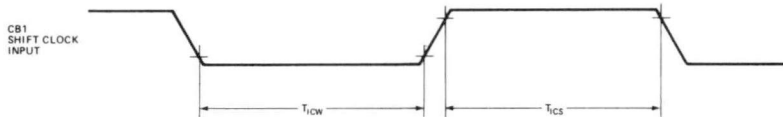


Figure 5h. External Shift Clock Timing

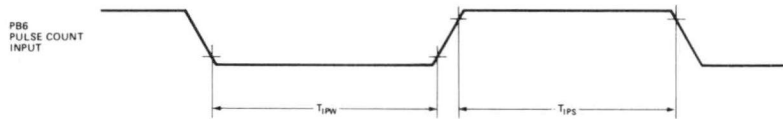


Figure 5i. Pulse Count Input Timing

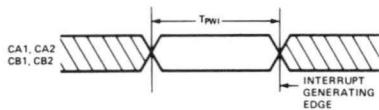


Figure 5j. Setup Time to Triggering Edge

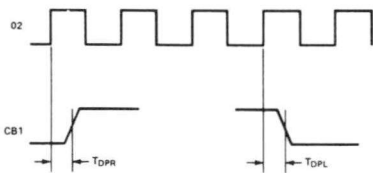


Figure 5k. Shift-in/out with Internal Clock Delay CD2 to CB1 Edge

Pin Descriptions

RES (Reset)

The reset input clears all internal registers to logic 0 (except T1 and T2 latches and counters and the Shift Register). This places all peripheral interface lines in the input state, disables the timers, shift register, etc. and disables interrupting from the chip.

φ2 (Input Clock)

The input clock is the system φ2 clock and is used to trigger all data transfers between the system processor and the SY6522.

R/W (Read/Write)

The direction of the data transfers between the SY6522 and the system processor is controlled by the R/W line. If R/W is low, data will be transferred out of the processor into the selected SY6522 register (write operation). If R/W is high and the chip is selected, data will be transferred out of the SY6522 (read operation).

DB0-DB7 (Data Bus)

The eight bi-directional data bus lines are used to transfer data between the SY6522 and the system processor. During read cycles, the contents of the selected SY6522 register are placed on the data bus lines and transferred into the processor. During write cycles, these lines are high-impedance inputs and data is transferred from the processor into the selected register. When the SY6522 is unselected, the data bus lines are high-impedance.

CS1, CS2 (Chip Selects)

The two chip select inputs are normally connected to processor address lines either directly or through decoding. The selected SY6522 register will be accessed when CS1 is high and CS2 is low.

RS0-RS3 (Register Selects)

The four Register Select inputs permit the system processor to select one of the 16 internal registers of the SY6522, as shown in Figure 6.

Register Number	RS Coding				Register Desig.	Description	
	RS3	RS2	RS1	RS0		Write	Read
0	0	0	0	0	ORB/IRB	Output Register "B"	Input Register "B"
1	0	0	0	1	ORA/IRA	Output Register "A"	Input Register "A"
2	0	0	1	0	DDRB	Data Direction Register "B"	
3	0	0	1	1	DDRA	Data Direction Register "A"	
4	0	1	0	0	T1C-L	T1 Low-Order Latches	T1 Low-Order Counter
5	0	1	0	1	T1C-H	T1 High-Order Counter	
6	0	1	1	0	T1L-L	T1 Low-Order Latches	
7	0	1	1	1	T1L-H	T1 High-Order Latches	
8	1	0	0	0	T2C-L	T2 Low-Order Latches	T2 Low-Order Counter
9	1	0	0	1	T2C-H	T2 High-Order Counter	
10	1	0	1	0	SR	Shift Register	
11	1	0	1	1	ACR	Auxiliary Control Register	
12	1	1	0	0	PCR	Peripheral Control Register	
13	1	1	0	1	IFR	Interrupt Flag Register	
14	1	1	1	0	IER	Interrupt Enable Register	
15	1	1	1	1	ORA/IRA	Same as Reg 1 Except No "Handshake"	

Figure 6. SY6522 Internal Register Summary

Synertek**SY6522/SY6522A****IRQ (Interrupt Request)**

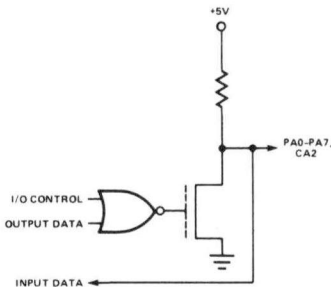
The Interrupt Request output goes low whenever an internal interrupt flag is set and the corresponding interrupt enable bit is a logic 1. This output is "open-drain" to allow the interrupt request signal to be "wire-or-ed" with other equivalent signals in the system.

PA0-PA7 (Peripheral A Port)

The Peripheral A port consists of 8 lines which can be individually programmed to act as inputs or outputs under control of a Data Direction Register. The polarity of output pins is controlled by an Output Register and input data may be latched into an internal register under control of the CA1 line. All of these modes of operation are controlled by the system processor through the internal control registers. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. Figure 7 illustrates the output circuit.

CA1, CA2 (Peripheral A Control Lines)

The two Peripheral A control lines act as interrupt inputs or as handshake outputs. Each line controls an internal interrupt flag with a corresponding interrupt enable bit. In addition, CA1 controls the latching of data on Peripheral A port input lines. CA1 is a high-impedance input only; while CA2 represents one standard TTL load in the input mode. CA2 will drive one standard TTL load in the output mode.

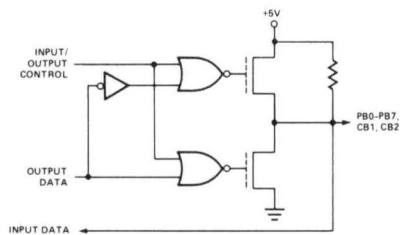
**Figure 7. Peripheral A Port Output Circuit****PB0-PB7 (Peripheral B Port)**

The Peripheral B port consists of eight bi-directional lines which are controlled by an output register and a data direction register in much the same manner as the

PA port. In addition, the PB7 output signal can be controlled by one of the interval timers while the second timer can be programmed to count pulses on the PB6 pin. Peripheral B lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. In addition, they are capable of sourcing 1.0mA at 1.5VDC in the output mode to allow the outputs to directly drive Darlington transistor circuits. Figure 8 is the circuit schematic.

CB1, CB2 (Peripheral B Control Lines)

The Peripheral B control lines act as interrupt inputs or as handshake outputs. As with CA1 and CA2, each line controls an interrupt flag with a corresponding interrupt enable bit. In addition, these lines act as a serial port under control of the Shift Register. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. Unlike PB0-PB7, CB1 and CB2 cannot drive Darlington transistor circuits.

**Figure 8. Peripheral B Port Output Circuit****FUNCTIONAL DESCRIPTION****Port A and Port B Operation**

Each 8-bit peripheral port has a Data Direction Register (DDRA, DDRB) for specifying whether the peripheral pins are to act as inputs or outputs. A 0 in a bit of the Data Direction Register causes the corresponding peripheral pin to act as an input. A 1 causes the pin to act as an output.

When programmed as an output each peripheral pin is also controlled by a corresponding bit in the Output Register (ORA, ORB). A 1 in the Output Register causes the output to go high, and a "0" causes the output to go low. Data may be written into Output Register bits corresponding to pins which are pro-

Synertek.

SY6522/SY6522A

grammed as inputs. In this case, however, the output signal is unaffected.

Reading a peripheral port causes the contents of the Input Register (IRA, IRB) to be transferred onto the Data Bus. With input latching disabled, IRA will always reflect the levels on the PA pins. With input latching enabled and the selected active transition on CA1 having occurred, IRA will contain the data present on the PA lines at the time of the transition. Once IRA is read, however, it will appear transparent, reflecting the current state of the PA lines until the next "latching" transition.

The IRB register operates similar to the IRA register. However, for pins programmed as outputs there is a difference. When reading IRA, the level on the pin determines whether a 0 or a 1 is sensed. When reading IRB, however, the bit stored in the output register, ORB, is the bit sensed. Thus, for outputs which have large loading effects and which pull an output "1" down or which pull an output "0" up, reading IRA may result in reading a "0" when a "1" was actually programmed, and reading a "1" when a "0" was programmed. Reading IRB, on the other hand, will read the "1" or "0" level actually programmed, no matter what the loading on the pin.

Figures 9, 10, and 11 illustrate the formats of the port registers. In addition, the input latching modes are selected by the Auxiliary Control Register (Figure 16.)

Handshake Control of Data Transfers

The SY6522 allows positive control of data transfers between the system processor and peripheral devices

REG 0 — ORB/IRB

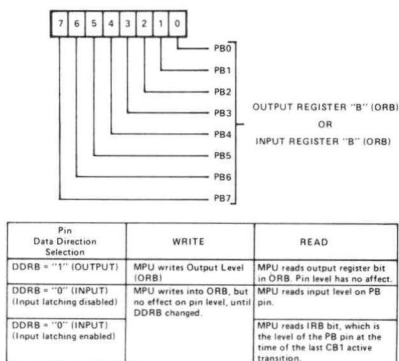


Figure 9. Output Register B (ORB), Input Register B (IRB)

REG 1 — ORA/IRA

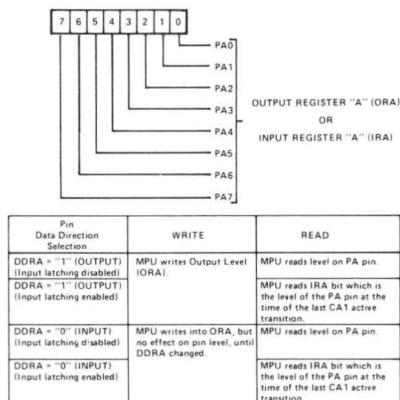


Figure 10. Output Register A (ORA), Input Register A (IRA)

REG 2 (DDRB) AND REG 3 (DDRA)

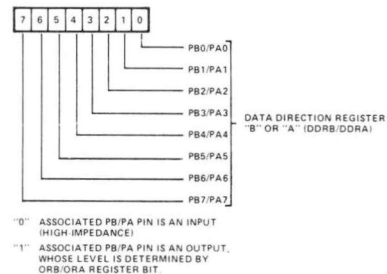


Figure 11. Data Direction Registers (DDRB, DDRA)

through the operation of "handshake" lines. Port A lines (CA1, CA2) handshake data on both a read and a write operation while the Port B lines (CB1, CB2) handshake on a write operation only.

Read Handshake

Positive control of data transfers from peripheral devices into the system processor can be accomplished very effectively using Read Handshaking. In this case, the peripheral device must generate the equivalent of a "Data Ready" signal to the processor signifying that valid data is present on the peripheral port. This signal normally interrupts the processor, which then reads the data, causing generation of a "Data Taken" signal. The peripheral device responds by making new data available. This process continues until the data transfer is complete.

Synertek.

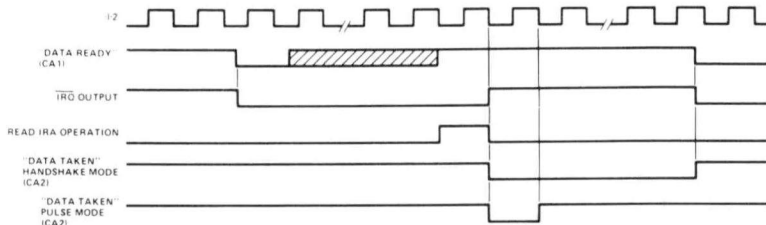
SY6522/SY6522A

Figure 12. Read Handshake Timing (Port A, Only)

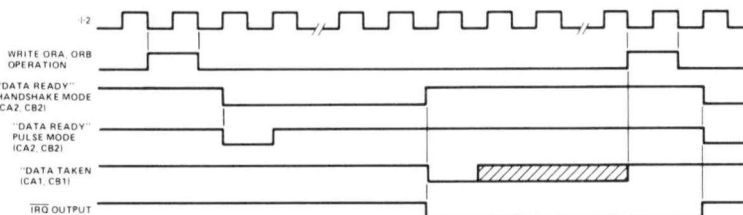


Figure 13. Write Handshake Timing

In the SY6522, automatic "Read" Handshaking is possible on the Peripheral A port only. The CA1 interrupt input pin accepts the "Data Ready" signal and CA2 generates the "Data Taken" signal. The "Data Ready" signal will set an internal flag which may interrupt the processor or which may be polled under program control. The "Data Taken" signal can either be a pulse or a level which is set low by the system processor and is cleared by the "Data Ready" signal. These options are shown in Figure 12 which illustrates the normal Read Handshaking sequence.

Write Handshake

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described for Read Handshaking. However, for Write Handshaking, the SY6522 generates the "Data Ready" signal and the peripheral device must respond with the "Data Taken" signal. This can be accomplished on both the PA port and the PB port on the SY6522. CA2 or CB2 act as a "Data Ready" output in either the handshake mode or pulse mode and CA1 or CB1 accept the "Data Taken" signal from the peripheral device, setting the interrupt flag and cleaning the "Data Ready" output. This sequence is shown in Figure 13.

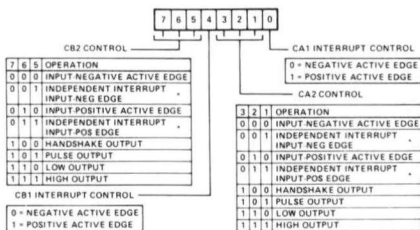
Selection of operating modes for CA1, CA2, CB1, and CB2 is accomplished by the Peripheral Control Register (Figure 14).

Timer Operation

Interval Timer T1 consists of two 8-bit latches and a 16-bit counter. The latches are used to store data which is to be loaded into the counter. After loading, the counter decrements at $\phi 2$ clock rate. Upon reaching zero, an interrupt flag will be set, and \overline{TRQ} will go low if the interrupt is enabled. The timer will then disable any further interrupts, or (when programmed to) will automatically transfer the contents of the latches into the counter and begin to decrement again. In addition, the timer may be programmed to invert the output signal on a peripheral pin each time it "times-out". Each of these modes is discussed separately below.

The T1 counter is depicted in Figure 15 and the latches in Figure 16.

REG 12 - PERIPHERAL CONTROL REGISTER



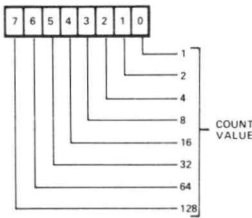
*SEE NOTE ACCOMPANYING FIGURE 25

Figure 14. CA1, CA2, CB1, CB2 Control

Two bits are provided in the Auxiliary Control Register (bits 6 and 7) to allow selection of the T1 oper-

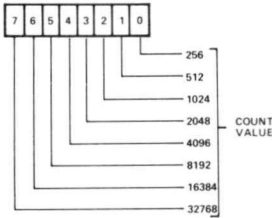
ating modes. The four possible modes are depicted in Figure 17.

REG 4 - TIMER 1 LOW-ORDER COUNTER



WRITE - 8 BITS LOADED INTO T1 LOW ORDER LATCHES. LATCH CONTENTS ARE TRANSFERRED INTO LOW-ORDER COUNTER AT THE TIME THE HIGH-ORDER COUNTER IS LOADED (REG 5).
READ - 8 BITS FROM T1 LOW ORDER COUNTER TRANSFERRED TO MPU. IN ADDITION, T1 INTERRUPT FLAG IS RESET (BIT 6 IN INTERRUPT FLAG REGISTER).

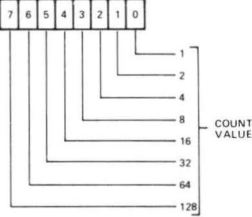
REG 5 - TIMER 1 HIGH-ORDER COUNTER



WRITE - 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. ALSO, AT THIS TIME BOTH HIGH AND LOW-ORDER LATCHES TRANSFERRED INTO T1 COUNTER, AND INITIATES COUNTDOWN. T1 INTERRUPT FLAG ALSO IS RESET.
READ - 8 BITS FROM T1 HIGH-ORDER COUNTER TRANSFERRED TO MPU.

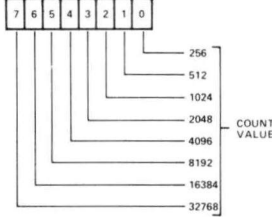
Figure 15. T1 Counter Registers

REG 6 - TIMER 1 LOW-ORDER LATCHES



WRITE - 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. THIS OPERATION IS NO DIFFERENT THAT A WRITE INTO REG 4.
READ - 8 BITS FROM T1 LOW ORDER LATCHES TRANSFERRED TO MPU. UNLIKE REG 4 OPERATION, THIS DOES NOT CAUSE RESET OF T1 INTERRUPT FLAG.

REG 7 - TIMER 1 HIGH-ORDER LATCHES



WRITE - 8 BITS LOADED INTO T1 HIGH ORDER LATCHES. UNLIKE REG 4 OPERATION NO LATCH-TO-COUNTER TRANSFERS TAKE PLACE.
READ - 8 BITS FROM T1 HIGH-ORDER LATCHES TRANSFERRED TO MPU.

Figure 16. T1 Latch Registers

REG 11 - AUXILIARY CONTROL REGISTER

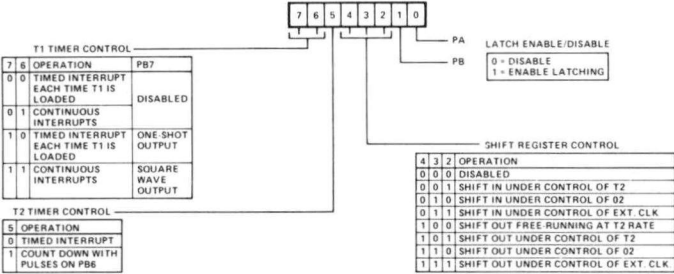


Figure 17. Auxiliary Control Register

Note: The processor does not write directly into the low order counter (T1C-L). Instead, this half of the counter is loaded automatically from the low order latch when the processor writes into the high order counter. In fact, it may not be necessary to write to the low order counter in some applications since the timing operation is triggered by writing to the high order counter.

Synertek

SY6522/SY6522A

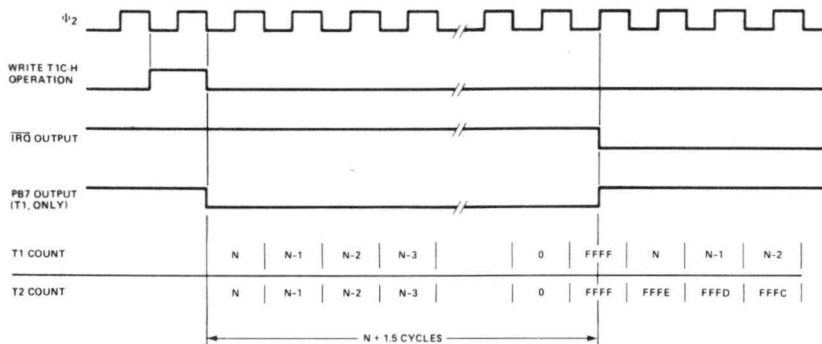


Figure 18. Timer 1 and Timer 2 One-Shot Mode Timing

Timer 1 One-Shot Mode

The interval timer one-shot mode allows generation of a single interrupt for each Timer load operation. In addition, Timer 1 can be programmed to produce a single negative pulse on PB7.

To generate a single interrupt ACR bits 6 and 7 must be 0 then either TIL-L or TIC-L must be written with the low-order count value. (A write to TIC-L is effectively a Write to TIL-L). Next the high-order count value is written to TIC-H, (the value is simultaneously written into TIL-H), and TIL-L is transferred to TIC-L. Countdown begins on the ϕ_2 following the write TIC-H and decrements at the ϕ_2 rate. T1 interrupt occurs when the counters reach 0. Generation of a negative pulse on PB7 is done in the same manner except ACR bit 7 must be a one. PB7 will go low after a Write TIC-H and go high again when the counters reach 0.

The T1 interrupt flag is reset by either writing TIC-H (starting a new count) or by reading TIC-L.

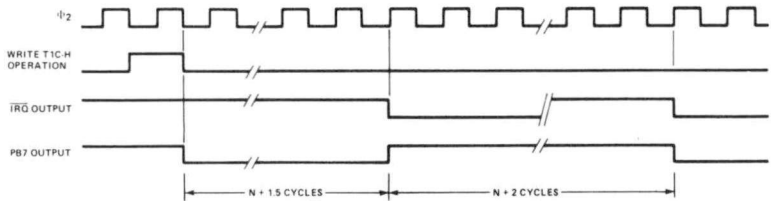
Timing for the one-shot mode is illustrated in Figure 18.

Timer 1 Free-Run Mode

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and the ability to produce a square wave on PB7 whose frequency is not affected by variations in the processor interrupt response time. This is accomplished in the "free-running" mode.

In the free-running mode, the interrupt flag is set and the signal on PB7 is inverted each time the counter reaches zero. However, instead of continuing to decrement from zero after a time-out, the timer automatically transfers the contents of the latch into the counter (16 bits) and continues to decrement from there. It is not necessary to rewrite the timer to enable setting the interrupt flag on the next time-out. The interrupt flag can be cleared by reading TIC-L, by writing directly into the flag as described later, or if a new count value is desired by a write to TIC-H.

All interval timers in the SY6522 are "re-triggerable". Rewriting the counter will always re-initialize the time-out period. In fact, the time-out can be prevented completely if the processor continues to rewrite the timer before it reaches zero. Timer 1 will operate in this manner if the processor writes into the high order counter (T1C-H). However, by loading the latches only, the processor can access the timer during each down-counting operation without affecting the time-out in process. Instead, the data loaded into the latches will determine the length of the next time-out period. This capability is particularly valuable in the free-running mode with the output enabled. In this mode, the signal on PB7 is inverted and the interrupt flag is set with each time-out. By responding to the interrupts with new data for the latches, the processor can determine the period of the next half cycle during each half cycle of the output signal on PB7. In this manner, very complex waveforms can be generated. Timing for the free-running mode is shown in Figure 19.



Note: A precaution to take in the use of PB7 as the timer output concerns the Data Direction Register contents for PB7. Both DDRB bit 7 and ACR bit 7 must be 1 for PB7 to function as the timer output. If either is a 0, then PB7 functions as a normal output pin, controlled by ORB bit 7.

Figure 19. Timer 1 Free-Run Mode Timing

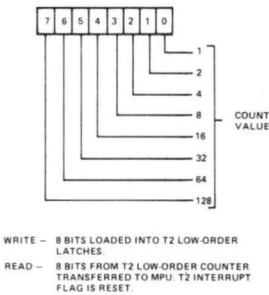
Timer 2 Operation

Timer 2 operates as an interval timer (in the "one-shot" mode only), or as a counter for counting negative pulses on the PB6 peripheral pin. A single control bit is provided in the Auxiliary Control Register to select between these two modes. This timer is comprised of a "write-only" low-order latch (T2L-L), a "read-only" low-order counter and a read/write high order counter. The counter registers act as a 16-bit counter which decrements at $\Phi 2$ rate. Figure 20 illustrates the T2 Counter Registers.

Timer 2 One-Shot Mode

As an interval timer, T2 operates in the "one-shot" mode similar to Timer 1. In this mode, T2 provides a single interrupt for each "write T2C-H" operation. After timing out, (reading 0) the counters "roll-over" to all 1's (FFFF₁₆) and continue decrementing, allowing the user to read them and determine how long T2 interrupt has been set. However, setting of the interrupt flag will be disabled after initial time-out so that it will not be set by the counter continuing to decrement through zero. The processor must rewrite T2C-H to enable setting of the interrupt flag. The interrupt flag is cleared by reading T2C-L or by writing T2C-H. Timing for this operation is shown in Figure 18.

REG 8 - TIMER 2 LOW-ORDER COUNTER



REG 9 - TIMER 2 HIGH-ORDER COUNTER

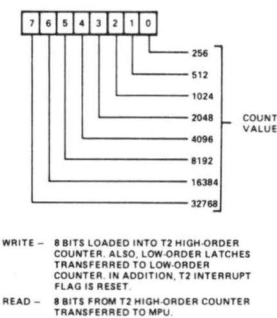


Figure 20. T2 Counter Registers

Synertek.

SY6522/SY6522ATimer 2 Pulse Counting Mode

In the pulse counting mode, T2 serves primarily to count a predetermined number of negative-going pulses on PB6. This is accomplished by first loading a number into T2. Writing into T2C-H clears the interrupt flag and allows the counter to decrement each time a pulse is applied to PB6. The interrupt flag will be set when T2 reaches zero. At this time the counter will continue to decrement with each pulse on PB6. However, it is necessary to rewrite T2C-H to allow the interrupt flag to set on subsequent down-counting operations. Timing for this mode is shown in Figure 21. The pulse must be low on the leading edge of $\Phi 2$.

Shift Register Operation

The Shift Register (SR) performs serial data transfers into and out of the CB2 pin under control of an internal modulo-8 counter. Shift pulses can be applied to the CB1 pin from an external source or, with the proper mode selection, shift pulses generated internally will appear on the CB1 pin for controlling external devices.

The control bits which select the various shift register operating modes are located in the Auxiliary Control Register. Figure 22 illustrates the configuration of the SR data bits and the SR control bits of the ACR.

Figures 23 and 24 illustrate the operation of the various shift register modes.

Interrupt Operation

Controlling interrupts within the SY6522 involves three principal operations. These are flagging the interrupts, enabling interrupts and signaling to the processor that an active interrupt exists within the chip. Interrupt flags are set by interrupting conditions which exist within the chip or on inputs to the chip. These flags normally remain set until the interrupt has been serviced. To determine the source of an interrupt, the microprocessor must examine these flags in order from highest to lowest priority. This is accomplished by reading the flag register into the processor accumulator, shifting this register either right or left and then using conditional branch instructions to detect an active interrupt.

Associated with each interrupt flag is an interrupt enable bit. This can be set or cleared by the processor to enable interrupting the processor from the corresponding interrupt flag. If an interrupt flag is set to a logic 1 by an interrupting condition, and the corresponding interrupt enable bit is set to a 1, the Interrupt Request Output (IRQ) will go low. $\overline{\text{IRQ}}$ is an "open-collector" output which can be "wire-or'ed" with other devices in the system to interrupt the processor.

In the SY6522, all the interrupt flags are contained in one register. In addition, bit 7 of this register will be read as a logic 1 when an interrupt exists within the chip. This allows very convenient polling of several devices within a system to locate the source of an interrupt.

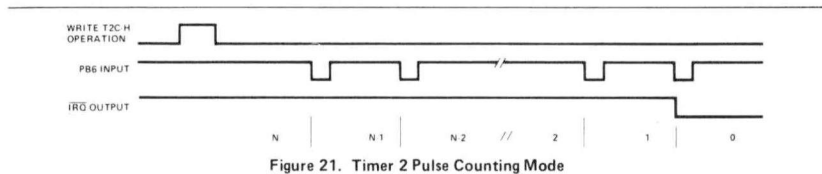
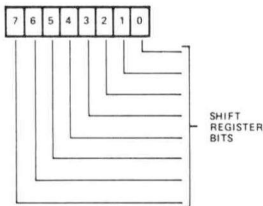
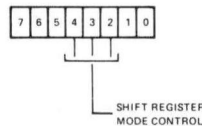


Figure 21. Timer 2 Pulse Counting Mode

REG 10 - SHIFT REGISTERNOTES:

1. WHEN SHIFTING OUT, BIT 7 IS THE FIRST BIT OUT AND SIMULTANEOUSLY IS ROTATED BACK INTO BIT 0.
2. WHEN SHIFTING IN, BITS INITIALLY ENTER BIT 0 AND ARE SHIFTED TOWARDS BIT 7.

REG 11 - AUXILIARY CONTROL REGISTER

4	3	2	OPERATION
0	0	0	DISABLED
0	0	1	SHIFT IN UNDER CONTROL OF T2
0	1	0	SHIFT IN UNDER CONTROL OF Φ_2
0	1	1	SHIFT IN UNDER CONTROL OF EXT CLK
1	0	0	SHIFT OUT FREE RUNNING AT T2 RATE
1	0	1	SHIFT OUT UNDER CONTROL OF T2
1	1	0	SHIFT OUT UNDER CONTROL OF Φ_2
1	1	1	SHIFT OUT UNDER CONTROL OF EXT CLK

Figure 22. SR and ACR Control Bits

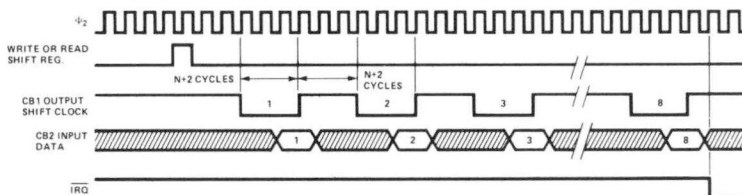
SR Disabled (000)

The 000 mode is used to disable the Shift Register. In this mode the microprocessor can write or read the SR, but the shifting operation is disabled and operation of CB1 and CB2 is controlled by the appropriate bits in the Peripheral Control Register (PCR). In this mode the SR Interrupt Flag is disabled (held to a logic 0).

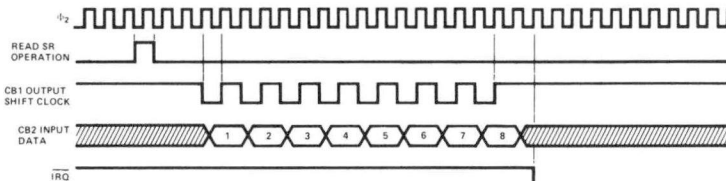
Shift in Under Control of T2 (001)

In the 001 mode the shifting rate is controlled by the low order 8 bits of T2. Shift pulses are generated on the CB1 pin to control shifting in external devices. The time between transitions of this output clock is a function of the system clock period and the contents of the low order T2 latch (N).

The shifting operation is triggered by writing or reading the shift register. Data is shifted first into the low order bit of SR and is then shifted into the next higher order bit of the shift register on the negative-going edge of each clock pulse. The input data should change before the positive-going edge of the CB1 clock pulse. This data is shifted into the shift register during the ϕ_2 clock cycle following the positive-going edge of the CB1 clock pulse. After 8 CB1 clock pulses, the shift register interrupt flag will be set and $\overline{\text{IRQ}}$ will go low.

**Shift in Under Control of ϕ_2 (010)**

In mode 010 the shift rate is a direct function of the system clock frequency. CB1 becomes an output which generates shift pulses for controlling external devices. Timer 2 operates as an independent interval timer and has no effect on SR. The shifting operation is triggered by reading or writing the Shift Register. Data is shifted first into bit 0 and is then shifted into the next higher order bit of the shift register on the trailing edge of each ϕ_2 clock pulse. After 8 clock pulses, the shift register interrupt flag will be set, and the output clock pulses on CB1 will stop.

**Shift in Under Control of External CB1 Clock (011)**

In mode 011 CB1 becomes an input. This allows an external device to load the shift register at its own pace. The shift register counter will interrupt the processor each time 8 bits have been shifted in. However, the shift register counter does not stop the shifting operation; it acts simply as a pulse counter. Reading or writing the Shift Register resets the Interrupt flag and initializes the SR counter to count another 8 pulses.

Note that the data is shifted during the first system clock cycle following the positive-going edge of the CB1 shift pulse. For this reason, data must be held stable during the first full cycle following CB1 going high.

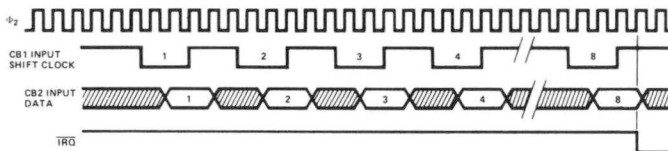
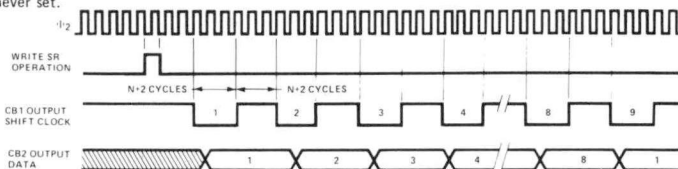


Figure 23. Shift Register Input Modes

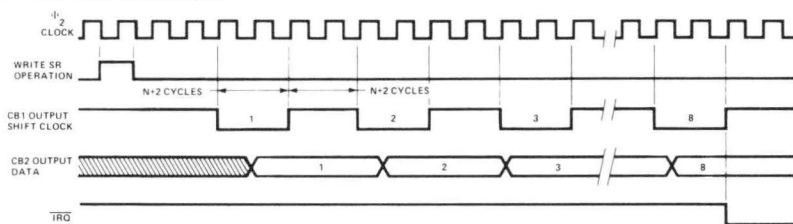
Synertek

SY6522/SY6522A**Shift Out Free-Running at T2 Rate (100)**

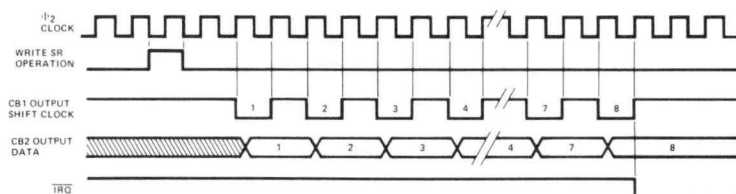
Mode 100 is very similar to mode 101 in which the shifting rate is set by T2. However, in mode 100 the SR Counter does not stop the shifting operation. Since the Shift Register bit 7 (SR7) is recirculated back into bit 0, the 8 bits loaded into the shift register will be clocked onto CR2 repetitively. In this mode the shift register counter is disabled, and $\overline{\text{IRQ}}$ is never set.

**Shift Out Under Control of T2 (101)**

In mode 101 the shift rate is controlled by T2 (as in the previous mode). However, with each read or write of the shift register the SR Counter is reset and 8 bits are shifted onto CB2. At the same time, 8 shift pulses are generated on CB1 to control shifting in external devices. After the 8 shift pulses, the shifting is disabled, the SR Interrupt Flag is set and CB2 remains at the last data level.

**Shift Out Under Control of ϕ_2 (110)**

In mode 110, the shift rate is controlled by the ϕ_2 system clock.

**Shift Out Under Control of External CB1 Clock (111)**

In mode 111 shifting is controlled by pulses applied to the CB1 pin by an external device. The SR counter sets the SR Interrupt flag each time it counts 8 pulses but it does not disable the shifting function. Each time the microprocessor writes or reads the shift register, the SR Interrupt flag is reset and the SR counter is initialized to begin counting the next 8 shift pulses on pin CB1. After 8 shift pulses, the interrupt flag is set. The microprocessor can then load the shift register with the next byte of data.

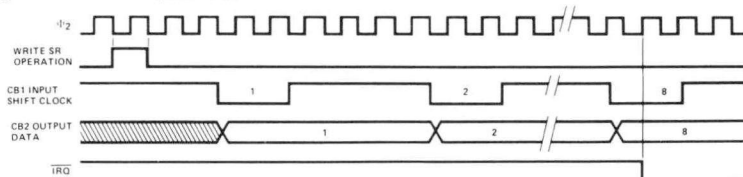


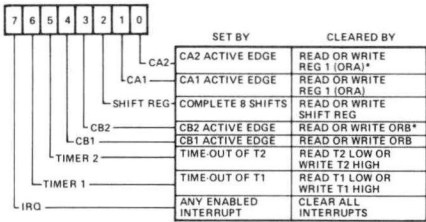
Figure 24. Shift Register Output Modes

The Interrupt Flag Register (IFR) and Interrupt Enable Register (IER) are depicted in Figures 25 and 26, respectively.

The IFR may be read directly by the processor. In addition, individual flag bits may be cleared by writing a "1" into the appropriate bit of the IFR. When the proper chip select and register signals are applied to the chip, the contents of this register are placed on the data bus. Bit 7 indicates the status of the IRQ output. This bit corresponds to the logic function: $IRQ = IFR6 \times IER6 + IFR5 \times IER5 + IFR4 \times IER4 + IFR3 \times IER3 + IFR2 \times IER2 + IFR1 \times IER1 + IFR0 \times IER0$. Note: X = logic AND, + = Logic OR.

The IFR bit 7 is not a flag. Therefore, this bit is not directly cleared by writing a logic 1 into it. It can only be cleared by clearing all the flags in the register or by disabling all the active interrupts as discussed in the next section.

REG 13 - INTERRUPT FLAG REGISTER



* IF THE CA2/CB2 CONTROL IN THE PCR IS SELECTED AS "INDEPENDENT" INTERRUPT INPUT, THEN READING OR WRITING THE OUTPUT REGISTER ORA/ORB WILL NOT CLEAR THE FLAG BIT. INSTEAD, THE BIT MUST BE CLEARED BY WRITING INTO THE IFR, AS DESCRIBED PREVIOUSLY.

Figure 25. Interrupt Flag Register (IFR)

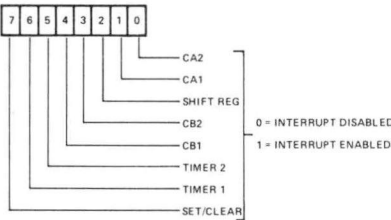
For each interrupt flag in IFR, there is a corresponding bit in the Interrupt Enable Register. The system processor can set or clear selected bits in this register to facilitate controlling individual interrupts without affecting others. This is accomplished

by writing to address 1110 (IER address). If bit 7 of the data placed on the system data bus during this write operation is a 0, each 1 in bits 6 through 0 clears the corresponding bit in the Interrupt Enable Register. For each zero in bits 6 through 0, the corresponding bit is unaffected.

Setting selected bits in the Interrupt Enable Register is accomplished by writing to the same address with bit 7 in the data word set to a logic 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each zero, the corresponding bit will be unaffected. This individual control of the setting and clearing operations allows very convenient control of the interrupts during system operation.

In addition to setting and clearing IER bits, the processor can read the contents of this register by placing the proper address on the register select and chip select inputs with the R/W line high. Bit 7 will be read as a logic 1.

REG 14 - INTERRUPT ENABLE REGISTER



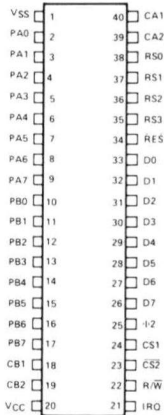
- NOTES:
1. IF BIT 7 IS A "0", THEN EACH "1" IN BITS 0 - 6 DISABLES THE CORRESPONDING INTERRUPT.
 2. IF BIT 7 IS A "1", THEN EACH "1" IN BITS 0 - 6 ENABLES THE CORRESPONDING INTERRUPT.
 3. IF A READ OF THIS REGISTER IS DONE, BIT 7 WILL BE "1" AND ALL OTHER BITS WILL REFLECT THEIR ENABLE/DISABLE STATE.

Figure 26. Interrupt Enable Register (IER)

Synertek

SY6522/SY6522A

Pin Configuration



Package Availability 40 Pin Plastic
40 Pin Ceramic

Ordering Information

Order Number	Package Type	Frequency Option
SYP 6522	Plastic	1 MHz
SYP 6522A	Plastic	2 MHz
SYC 6522	Ceramic	1 MHz
SYC 6522A	Ceramic	2 MHz

NEC Microcomputers, Inc.

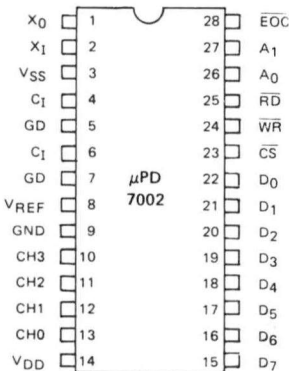
NEC
μ PD7002

12-BIT BINARY A/D CONVERTER

DESCRIPTION The μPD7002 is a high performance, low power, monolithic CMOS A/D converter designed for microprocessor applications. The analog input voltage is applied to one of the four analog inputs. By loading the input register with the multiplexer channel and the desired resolution (8 or 10 bits) the integrating A/D conversion sequence is started. At the end of conversion EOC signal goes low and if connected to the interrupt line of microprocessor it will cause an interrupt. At this point the digital data can be read in two bytes from the output registers. The μPD7002 also features a status register that can be read at any time.

- FEATURES**
- Single Chip CMOS LSI
 - Resolution: 8 or 10 Bits
 - 4 Channel Analog Multiplexer
 - Auto-Zeroscale and Auto-Fullscale Corrections without any External Components
 - High Input Impedance: 1000MΩ
 - Readout of Internal Status Register Through Data Bus
 - Single +5V Power Supply
 - Interfaces to Most 8-Bit Microprocessors
 - Conversion Speed: 5 ms
 - Power Consumption: 20 mW
 - Available in a 28 Pin Plastic Package

PIN CONFIGURATION

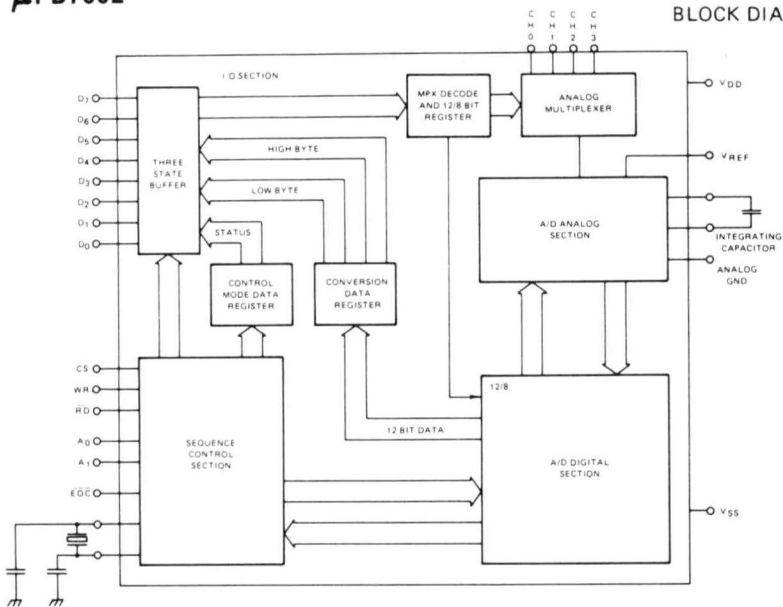


PIN NAMES

X ₀ ,X _I	External Clock Input
V _{SS}	TTL Ground
C _I	Integrating Capacitor
GD	Guard
V _{REF}	Reference Voltage Input
GND	Analog Ground
CH3	Analog Channel 3
CH2	Analog Channel 2
CH1	Analog Channel 1
CH0	Analog Channel 0
V _{DD}	TTL Voltage (+5V)
D ₀ -D ₇	Data Bus
CS	Chip Select
WR, RD	Control Bus
A ₀ ,A ₁	Address Bus
EOC	End of Conversion Interrupt

μPD7002

BLOCK DIAGRAM



$T_a = 25 \pm 2^\circ\text{C}$, $V_{DD} = +5 \pm 0.25\text{V}$, $V_{REF} = +2.50\text{V}$, $f_{CK} = 1\text{MHz}$

DC CHARACTERISTICS

PARAMETER	SYMBOL	LIMITS			UNIT	TEST CONDITIONS
		MIN	TYP	MAX		
Resolution			12		Bits	$V_{DD} = 5\text{V}$, $V_{REF} = 2.5 \pm 0.25\text{V}$
Non Linearity			0.05	0.08	%FSR	$V_{DD} = 5\text{V}$, $V_{REF} = 2.5 \pm 0.25\text{V}$
Fullscale Error			0.05	0.08	%FSR	$V_{DD} = 5\text{V}$, $V_{REF} = 2.5 \pm 0.25\text{V}$
Zeroscale Error			0.05	0.08	%FSR	$V_{DD} = 5\text{V}$, $V_{REF} = 2.5 \pm 0.25\text{V}$
Fullscale Temperature Coefficient			10		PPM/ $^\circ\text{C}$	$V_{DD} = 5\text{V}$
Zeroscale Temperature Coefficient			10		PPM/ $^\circ\text{C}$	$V_{DD} = 5\text{V}$
Analog Input Voltage Range	V_{IA}	0		V_{REF}	V	
Analog Input Resistance	R_{IA}		1000		M Ω	$V_{IA} = V_{SS}$ to V_{DD}
Total Unadjusted Error 1	T.U.E. 1		0.05	0.08	%FSR	$V_{REF} = 2.25$ to 2.75V , $V_{DD} = 5\text{V}$
Total Unadjusted Error 2	T.U.E. 2		0.05	0.08	%FSR	$V_{REF} = 2.5\text{V}$, $V_{DD} = 4.75$ to 5.25V
Clock Input Current	I_{XI}		5	50	μA	
Clock Input High Level	V_{XIH}	$V_{DD}-1.4$			V	
Clock Input Low Level	V_{XIL}			$V_{SS}+1.4$	V	
High Level Input Voltage	V_{IH}	2.2			V	$T_a = -20^\circ\text{C}$ to $+70^\circ\text{C}$
Low Level Input Voltage	V_{IL}		0.8		V	$T_a = -20^\circ\text{C}$ to $+70^\circ\text{C}$
High Level Output Voltage	V_{OH}	3.5			V	$I_O = -1.6\text{mA}$, $T_a = -20^\circ\text{C}$ to $+70^\circ\text{C}$
Low Level Output Voltage	V_{OL}		0.4		V	$I_O = +1.6\text{mA}$, $T_a = -20^\circ\text{C}$ to $+70^\circ\text{C}$
Digital Input Leakage Current	I_I		1	10	μA	$V_I = V_{SS}$ to V_{DD}
High-Z Output Leakage Current	I_{Leak}		1	10	μA	$V_O = V_{SS}$ to V_{DD}
Power Dissipation	P_d		15	25	mW	$f_{CK} < 1\text{MHz}$

μPD7002

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature -20°C to +70°C
Storage Temperature -65°C to +125°C
All Input Voltages -0.3 to V_{DD} + 0.3 Volts
Power Supply -0.3 to +7 Volts
Power Dissipation 300 mW
Analog GND Voltage V_{SS} ± 0.3 Volts
COMMENT: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

*T_a = 25°C

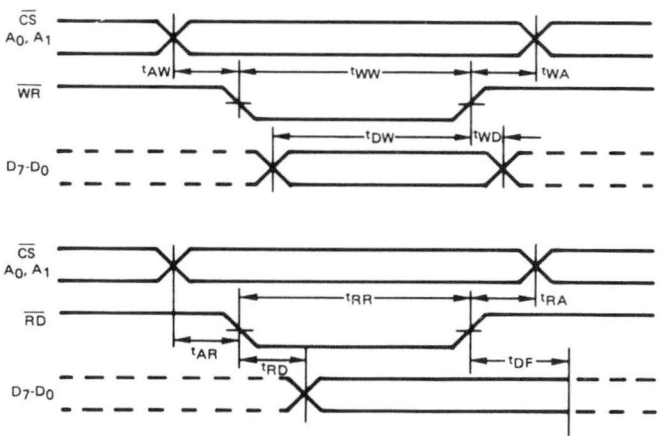
AC CHARACTERISTICS

T_a = 25° ± 2°C; V_{DD} = +5 ± 0.25V; V_{REF} = 2.5V; f_{CK} = 1 MHz; C_{INT} = 0.033 μF

PARAMETER	SYMBOL	LIMITS			UNIT	TEST CONDITIONS
		MIN	TYP	MAX		
Conversion Speed (12 bit)	t _{CONV}	8.5	10	15	ms	f _{CK} = 1 MHz
Conversion Speed (8 bit)	t _{CONV}	2.4	4	5	ms	f _{CK} = 1 MHz
Clock Frequency Range	f _{CK}	0.1	1	3	MHz	
Integrating Capacitor Value	C _{INT} *	0.029			μF	V _{REF} = 2.50V, f _{CK} = 1 MHz
Address Setup Time CS, A ₀ , A ₁ , to WR	t _{AW}	50			ns	
Address Setup Time CS, A ₀ , A ₁ , to RD	t _{AR}	50			ns	
Address Hold Time WR to CS, A ₀ , A ₁	t _{WA}	50			ns	
Address Hold Time RD to CS, A ₀ , A ₁	t _{RA}	50			ns	
Low Level WR Pulse Width	t _{WW}	400			ns	
Low Level RD Pulse Width	t _{RR}	400			ns	
Data Setup Time Input Data to WR	t _{DW}	300			ns	
Data Hold Time WR to Input Data	t _{WD}	50			ns	
Output Delay Time RD to Output Data	t _{RD}			300	ns	1TTL + 100 pF
Delay Time to High Z Output RD to Floating Output	t _{DF}			150	ns	

*C_{INT} (μF) (Min) = 0.029 / f_{CK} (MHz)

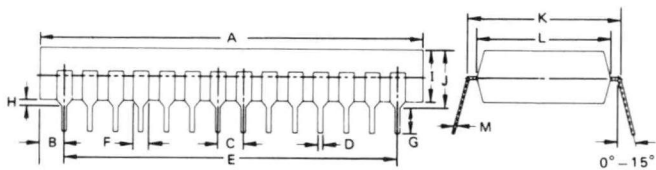
TIMING WAVEFORMS



μPD7002

CONTROL TERMINALS					MODE	INTERNAL FUNCTION	DATA INPUT-OUTPUT TERMINALS
CS	RD	WR	A ₁	A ₀			
H	x	x	x	x	Not selected	—	High impedance
L	H	H	x	x	Not selected	—	High impedance
L	H	L	L	L	Write mode	Data latch A/D start	Input status, D ₁ , D ₀ = MPX address D ₃ = 8 bit/12 bit conversion designation. ① D ₂ = Flag Input
L	H	L	L	H	Not selected	—	High impedance
L	H	L	H	L	Not selected	—	High impedance
L	H	L	H	H	Test mode	Test status	Input status ②
L	L	H	L	L	Read mode	Internal status	D ₇ = EOC, D ₆ = BUSY, D ₅ = MSB, D ₄ = 2nd MSB, D ₃ = 8/12, D ₂ = Flag Output D ₁ = MPX, D ₀ = MPX
L	L	H	L	H	Read mode	High data byte	D ₇ -D ₀ = MSB — 8th bit
L	L	H	H	L	Read mode	Low data byte	D ₇ -D ₄ = 9th — 12th bit, D ₃ -D ₀ = L
L	L	H	H	H	Read mode	Low data byte	

- Notes: ① Designation of number of conversion bits: 8 bit = L; 12 bit = H
- ② Test Mode: Used for inspecting the device. The data input-output terminals assume an input state and are connected to the A/D counter. Therefore, the A/D conversion data read out after this is meaningless.



CONTROL TERMINAL FUNCTIONS

PACKAGE OUTLINE
μPD7002C

(PLASTIC)

ITEM	MILLIMETERS	INCHES
A	38.0 MAX.	1.496 MAX.
B	2.49	0.098
C	2.54	0.10
D	0.5 ± 0.1	0.02 ± 0.004
E	33.02	1.3
F	1.5	0.059
G	2.54 MIN.	0.10 MIN.
H	0.5 MIN.	0.02 MIN.
I	5.22 MAX.	0.205 MAX.
J	5.72 MAX.	0.225 MAX.
K	15.24	0.6
L	13.2	0.52
M	0.25 +0.10 -0.05	0.01 +0.004 -0.002

TYPES SN54LS240, SN54LS241, SN54LS244, SN54S240, SN54S241, SN74LS240, SN74LS241, SN74LS244, SN74S240, SN74S241 **OCTAL BUFFERS AND LINE DRIVERS WITH 3-STATE OUTPUTS**

	Typical IOL (Sink Current)	Typical IOH (Source Current)	Typical Propagation Delay Times		Typical Enable/ Disable Times	Typical Power Dissipation (Enabled)	
			Inverting	Noninverting		Inverting	Noninverting
SN54LS'	12 mA	-12 mA	10.5 ns	12 ns	18 ns	130 mW	135 mW
SN74LS'	24 mA	-15 mA	10.5 ns	12 ns	18 ns	130 mW	135 mW
SN54S'	48 mA	-12 mA	4.5 ns	6 ns	9 ns	450 mW	538 mW
SN74S'	64 mA	-15 mA	4.5 ns	6 ns	9 ns	450 mW	538 mW

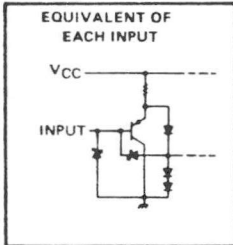
- 3-State Outputs Drive Bus Lines or Buffer Memory Address Registers
- P-N-P Inputs Reduce D-C Loading
- Hysteresis at Inputs Improves Noise Margins

description

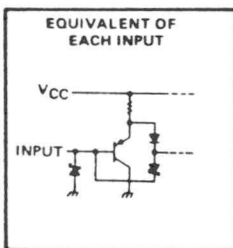
These octal buffers and line drivers are designed specifically to improve both the performance and density of three-state memory address drivers, clock drivers, and bus-oriented receivers and transmitters. The designer has a choice of selected combinations of inverting and noninverting outputs, symmetrical \bar{G} (active-low output control) inputs, and complementary G and \bar{G} inputs. These devices feature high fan-out, improved fan-in, and 400-mV noise-margin. The SN74LS' and SN74S' can be used to drive terminated lines down to 133 ohms.

schematics of inputs and outputs

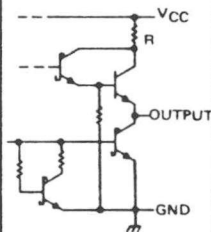
'LS240, 'LS241, 'LS244



'S240, 'S241

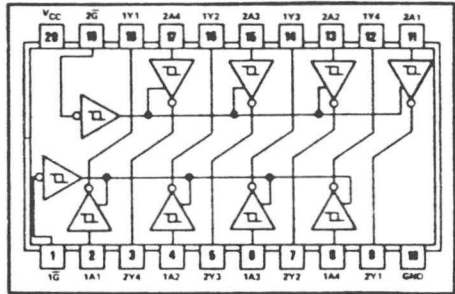


TYPICAL OF ALL OUTPUTS

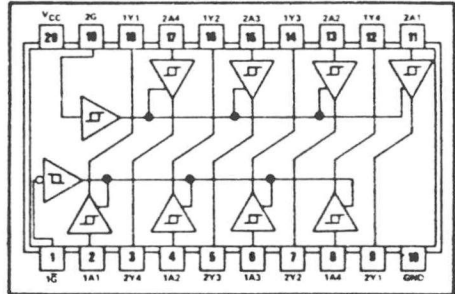


'LS240, 'LS241, 'LS244;
R = 50 Ω NOM
'S240, 'S241;
R = 25 Ω NOM

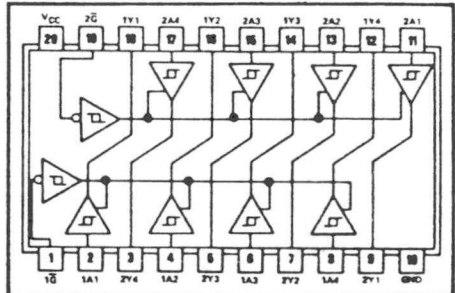
SN54LS240, SN54S240 ... J
SN74LS240, SN74S240 ... J OR N
(TOP VIEW)



SN54LS241, SN54S241 ... J
SN74LS241, SN74S241 ... J OR N
(TOP VIEW)



SN54LS244 ... J
SN74LS244 ... J OR N
(TOP VIEW)



TYPES SN64LS240, SN54LS241, SN64LS244, SN74LS240, SN74LS241, SN74LS244 **BUFFERS AND LINE DRIVERS WITH 3-STATE OUTPUTS**

recommended operating conditions

PARAMETER	SN64LS*			SN74LS*			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC} (see Note 1)	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			-12			-15	mA
Low-level output current, I_{OL}			12			24	mA
Operating free-air temperature, T_A	-55		125	0		70	°C

NOTE 1: Voltage values are with respect to network ground terminal.

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†	SN64LS*			SN74LS*			UNIT
		MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH} High-level input voltage		2			2			V
V_{IL} Low-level input voltage				0.7			0.8	V
V_{IK} Input clamp voltage	$V_{CC} = \text{MIN}$, $I_I = -18 \text{ mA}$			-1.5			-1.5	V
Hysteresis ($V_{T+} - V_{T-}$)	$V_{CC} = \text{MIN}$	0.2	0.4		0.2	0.4		V
V_{OH} High-level output voltage	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = V_{IL \text{ max}}$, $I_{OH} = -3 \text{ mA}$	2.4	3.4		2.4	3.4		V
	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.5 \text{ V}$, $I_{OH} = \text{MAX}$	2			2			
V_{OL} Low-level output voltage	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = V_{IL \text{ max}}$, $I_{OL} = 12 \text{ mA}$			0.4			0.4	V
	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = V_{IL \text{ max}}$, $I_{OL} = 24 \text{ mA}$						0.5	
I_{OZH} Off-state output current, high-level voltage applied	$V_{CC} = \text{MAX}$, $V_{OH} = 2.7 \text{ V}$			20			20	μA
I_{OZL} Off-state output current, low-level voltage applied	$V_{CC} = \text{MAX}$, $V_{OL} = 0.4 \text{ V}$			-20			-20	
I_I Input current at maximum input voltage	$V_{CC} = \text{MAX}$, $V_I = 7 \text{ V}$			0.1			0.1	mA
I_{IH} High-level input current, any input	$V_{CC} = \text{MAX}$, $V_I = 2.7 \text{ V}$			20			20	μA
I_{IL} Low-level input current	$V_{CC} = \text{MAX}$, $V_{IL} = 0.4 \text{ V}$			-0.2			-0.2	mA
I_{OS} Short-circuit output current‡	$V_{CC} = \text{MAX}$	-40		-225	-40		-225	mA
I_{CC} Supply current	Outputs high	$V_{CC} = \text{MAX}$	All	17	27	17	27	mA
	Outputs low		'LS240	26	44	26	44	
	Outputs open		'LS241, 'LS244	27	46	27	46	
	All outputs disabled		'LS240	29	50	29	50	
			'LS241, 'LS244	32	54	32	54	

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ\text{C}$.

* Not more than one output should be shorted at a time, and duration of the short circuit should not exceed one second.

switching characteristics, $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ\text{C}$

PARAMETER	TEST CONDITIONS	'LS240			'LS241, 'LS244			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	
t_{PLH} Propagation delay time, low-to-high-level output	$C_L = 45 \text{ pF}$, $R_L = 667 \Omega$		9	14		12	18	ns
t_{PHL} Propagation delay time, high-to-low-level output			12	18		12	18	ns
t_{PZL} Output enable time to low level			20	30		20	30	ns
t_{PZH} Output enable time to high level	$C_L = 5 \text{ pF}$, $R_L = 667 \Omega$		15	23		15	23	ns
t_{PLZ} Output disable time from low level			15	25		15	25	ns
t_{PHZ} Output disable time from high level			10	18		10	18	ns

TYPES SN54S240, SN54S241, SN74S240, SN74S241

BUFFERS/LINE DRIVERS/LINE RECEIVERS WITH 3-STATE OUTPUTS

recommended operating conditions

PARAMETER	SN54S*			SN74S*			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC} (see Note 1)	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			-12			-15	mA
Low-level output current, I_{OL}			48			64	mA
Operating free-air temperature, T_A (see Note 3)	-65		125	0		70	°C

NOTES: 1. Voltage values are with respect to network ground terminal.

3. An SN54S241J operating at free-air temperature above 118°C requires a heat sink that provides a thermal resistance from case to free air, $R_{\theta CA}$, of not more than 40°C/W.

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†	*S240			*S241			UNIT	
			MIN	TYP‡	MAX	MIN	TYP‡	MAX		
V_{IH}	High-level input voltage		2			2			V	
V_{IL}	Low-level input voltage		0.8			0.8			V	
V_{IK}	Input clamp voltage	$V_{CC} = \text{MIN}$, $I_I = -18 \text{ mA}$	-1.2			-1.2			V	
Hysteresis ($V_{T+} - V_{T-}$)		$V_{CC} = \text{MIN}$	0.2	0.4		0.2	0.4		V	
V_{OH}	High-level output voltage	SN74S*	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.8 \text{ V}$, $I_{OH} = -1 \text{ mA}$			2.7	2.7		V	
		SN54S* and SN74S*	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.8 \text{ V}$, $I_{OH} = -3 \text{ mA}$			2.4	3.4	2.4		3.4
		SN54S* and SN74S*	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.5 \text{ V}$, $I_{OH} = \text{MAX}$			2	2		V	
			$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.8 \text{ V}$, $I_{OH} = -3 \text{ mA}$			2.4	3.4	2.4		3.4
			$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.5 \text{ V}$, $I_{OH} = \text{MAX}$			2	2			
V_{OL}	Low-level output voltage	$V_{CC} = \text{MIN}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.8 \text{ V}$, $I_{OL} = \text{MAX}$	0.55			0.55			V	
I_{OZH}	Off-state output current, high-level voltage applied	$V_{CC} = \text{MAX}$, $V_{IH} = 2 \text{ V}$, $V_{IL} = 0.8 \text{ V}$	$V_O = 2.4 \text{ V}$			50	50		μA	
I_{OZL}	Off-state output current, low-level voltage applied		$V_O = 0.5 \text{ V}$			-50	-50			
I_I	Input current at maximum input voltage	$V_{CC} = \text{MAX}$, $V_I = 5.5 \text{ V}$	1			1			mA	
I_{IH}	High-level input current, any input	$V_{CC} = \text{MAX}$, $V_I = 2.7 \text{ V}$	50			50			μA	
I_{IL}	Low-level input current	Any A	-400			-400			μA	
		Any G	-2			-2			mA	
I_{OS}	Short-circuit output current*	$V_{CC} = \text{MAX}$	-50	-225		-50	-225		mA	
I_{CC}	Supply current	Outputs high	$V_{CC} = \text{MAX}$, Outputs open	SN54S*	80	123	95	147	mA	
				SN74S*	80	135	96	160		
		Outputs low		SN54S*	100	145	120	170		
				SN74S*	100	150	120	180		
		Outputs disabled		SN54S*	100	145	120	170		
				SN74S*	100	150	120	180		

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

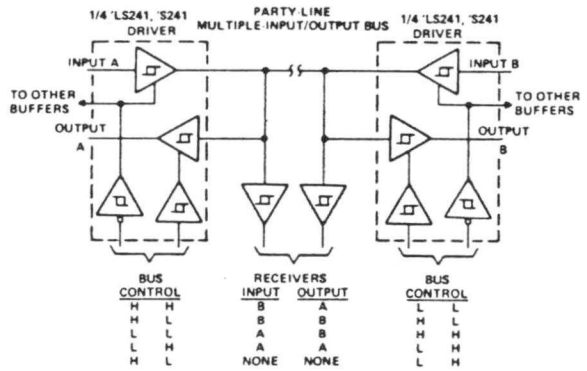
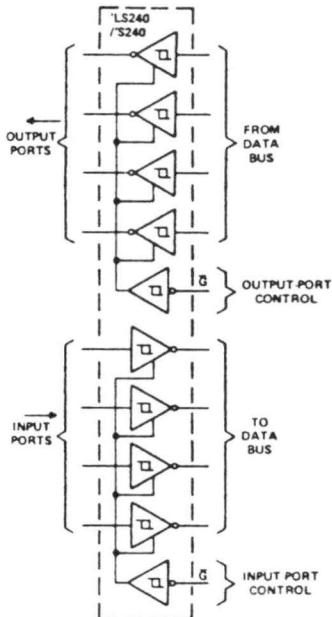
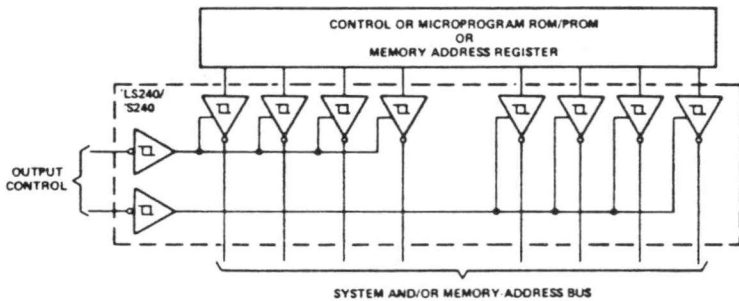
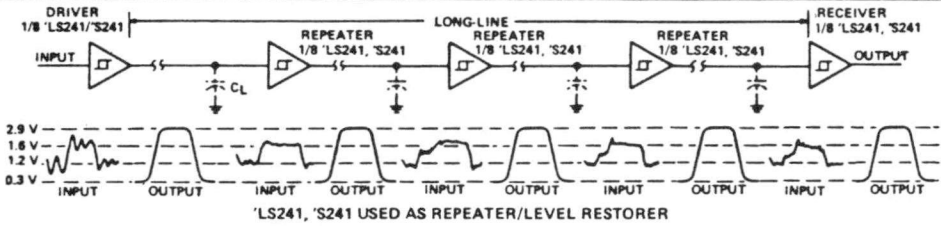
‡ All typical values are at $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ\text{C}$.

* Not more than one output should be shorted at a time, and duration of the short-circuit should not exceed one second.

switching characteristics, $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ\text{C}$

PARAMETER		TEST CONDITIONS	*S240			*S241			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low-to-high-level output	$C_L = 50 \text{ pF}$, $R_L = 90 \Omega$	4.5	7		6	9		ns
t_{PHL}	Propagation delay time, high-to-low-level output		4.5	7		6	9		ns
t_{PZL}	Output enable time to low level		10	15		10	15		ns
t_{PZH}	Output enable time to high level		6.5	10		8	12		ns
t_{PLZ}	Output disable time from low level	$C_L = 5 \text{ pF}$, $R_L = 90 \Omega$	10	15		10	15		ns
t_{PHZ}	Output disable time from high level		6	9		6	9		ns

**TYPES SN54LS240, SN54LS241,
SN54LS244, SN54S240, SN54S241, SN74LS240,
SN74LS241, SN74LS244, SN74S240, SN74S241**
OCTAL BUFFERS AND LINE DRIVERS WITH 3-STATE OUTPUTS



**PARTY-LINE BUS SYSTEM
WITH MULTIPLE INPUTS, OUTPUTS, AND RECEIVERS**
External resistance between any input of the 'S240 or 'S241 and ground or V_{CC} must not exceed 40 k Ω .

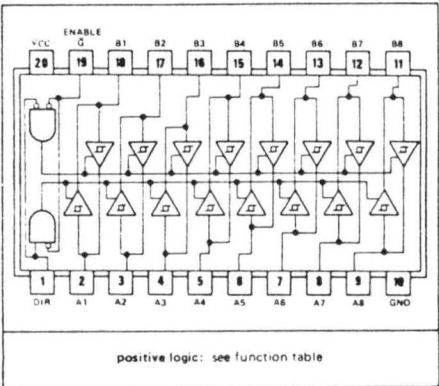
TYPES SN54LS245, SN74LS245
OCTAL BUS TRANSCEIVERS WITH 3-STATE OUTPUTS

BULLETIN NO. DL-S 7612471, OCTOBER 1976

- Bi-directional Bus Transceiver in a High-Density 20-Pin Package
- 3-State Outputs Drive Bus Lines Directly
- P-N-P Inputs Reduce D-C Loading on Bus Lines
- Hysteresis at Bus Inputs Improve Noise Margins
- Typical Propagation Delay Times, Port-to-Port . . . 12 ns
- Typical Enable/Disable Times . . . 17 ns

TYPE	I _{OL} (SINK CURRENT)	I _{OH} (SOURCE CURRENT)
SN54LS245	12 mA	-12 mA
SN74LS245	24 mA	-15 mA

SN54LS245 . . . J PACKAGE
SN74LS245 . . . J OR N PACKAGE
(TOP VIEW)



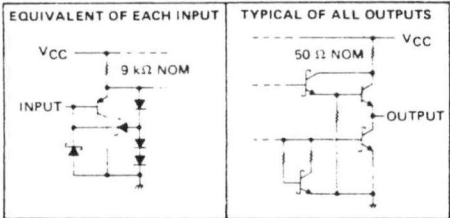
description

These octal bus transceivers are designed for asynchronous two-way communication between data buses. The control function implementation minimizes external timing requirements.

The device allows data transmission from the A bus to the B bus or from the B bus to the A bus depending upon the logic level at the direction control (DIR) input. The enable input (\bar{G}) can be used to disable the device so that the buses are effectively isolated.

The SN54LS245 is characterized for operation over the full military temperature range of -55°C to 125°C. The SN74LS245 is characterized for operation from 0°C to 70°C.

schematics of inputs and outputs



FUNCTION TABLE

ENABLE \bar{G}	DIRECTION CONTROL DIR	OPERATION
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = high level, L = low level, X = irrelevant

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V _{CC} (see Note 1)	7 V
Input voltage	7 V
Operating free-air temperature range: SN54LS245	-55°C to 125°C
SN74LS245	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

TYPES SN54LS245, SN74LS245 **OCTAL BUS TRANSCEIVERS WITH 3-STATE OUTPUTS**

recommended operating conditions

PARAMETER	SN54LS245			SN74LS245			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			-12			-15	mA
Low-level output current, I_{OL}			12			24	mA
Operating free-air temperature, T_A	-55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†	SN54LS245			SN74LS245			UNIT
		MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH} High-level input voltage		2			2			V
V_{IL} Low-level input voltage				0.7			0.8	V
V_{IK} Input clamp voltage	$V_{CC} = \text{MIN}, I_I = -18 \text{ mA}$			-1.5			-1.5	V
Hysteresis ($V_{T+} - V_{T-}$) A or B input	$V_{CC} = \text{MIN}$	0.2	0.4		0.2	0.4		V
V_{OH} High-level output voltage	$V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, I_{OH} = -3 \text{ mA}$	2.4	3.4		2.4	3.4		V
	$V_{IL} = V_{IL \text{ max}}, I_{OH} = \text{MAX}$	2			2			
V_{OL} Low-level output voltage	$V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, I_{OL} = 12 \text{ mA}$			0.4			0.4	V
	$V_{IL} = V_{IL \text{ max}}, I_{OL} = 24 \text{ mA}$						0.5	
I_{OZH} Off-state output current, high-level voltage applied	$V_{CC} = \text{MAX}, \bar{G} \text{ at } 2 \text{ V}, V_O = 2.7 \text{ V}$			20			20	μA
I_{OZL} Off-state output current, low-level voltage applied	$V_O = 0.4 \text{ V}$			-20			-20	
I_I Input current at maximum input voltage	$V_{CC} = \text{MAX}, V_I = 7 \text{ V}$			0.1			0.1	mA
I_{IH} High-level input current	$V_{CC} = \text{MAX}, V_{IH} = 2.7 \text{ V}$			20			20	μA
I_{IL} Low-level input current	$V_{CC} = \text{MAX}, V_{IL} = 0.4 \text{ V}$			-0.2			-0.2	mA
I_{OS} Short-circuit output current‡	$V_{CC} = \text{MAX}$	-40		-225	-40		-225	mA
I_{CC} Supply current	Total, outputs high		25	46		25	46	mA
	Total, outputs low		58	100		58	100	
	Outputs at Hi-Z		54	105		54	105	

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at $V_{CC} = 5 \text{ V}, T_A = 25^\circ\text{C}$.

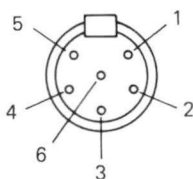
§ Not more than one output should be shorted at a time, and duration of the short-circuit should not exceed one second.

switching characteristics, $V_{CC} = 5 \text{ V}, T_A = 25^\circ\text{C}$

PARAMETER	TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH} Propagation delay time, low-to-high-level output	$C_L = 45 \text{ pF}, R_L = 667 \Omega$		12	18		ns
t_{PHL} Propagation delay time, high-to-low-level output			12	18		ns
t_{PZL} Output enable time to low level			20	30		ns
t_{PZH} Output enable time to high level			15	25		ns
t_{PLZ} Output disable time from low level	$C_L = 5 \text{ pF}, R_L = 667 \Omega$		15	25		ns
t_{PHZ} Output disable time from high level			10	18		ns

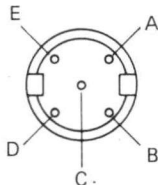
Appendix E: Summary of Connections

SK1 UHF out
SK2 Video out
SK3 RGB



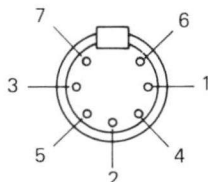
- | | |
|----------|----------|
| 1. Red | 4. Sync. |
| 2. Green | 5. 0 V |
| 3. Blue | 6. +5 V |

SK4 RS423



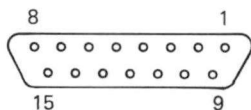
- | | |
|-------------|-----------------------|
| A. Data in | D. Clear to send, CTS |
| B. Data out | E. Ready to send, RTS |
| C. 0 V | |

SK5 Cassette port



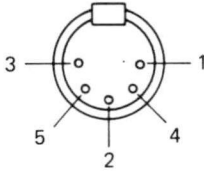
- | | |
|-----------|------------------|
| 1. Output | 5. N.C. |
| 2. 0 V | 6. Motor control |
| 3. Input | 7. Motor control |
| 4. Output | |

SK6 Analogue in



- | | |
|--------------------|----------------------------|
| 1. +5 V | 9. Light pen strobe, LPSTB |
| 2. 0 V | 10. IO1 (PB1) |
| 3. 0 V | 11. V_{REF} |
| 4. Channel 3 | 12. Channel 2 |
| 5. Analogue ground | 13. IO0 (PB0) |
| 6. 0 V | 14. V_{REF} |
| 7. Channel 1 | 15. Channel 0 |
| 8. Analogue ground | |

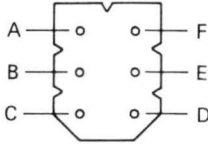
SK7 Econet



1. Data
2. 0 V
3. Clock

4. Data
5. Clock

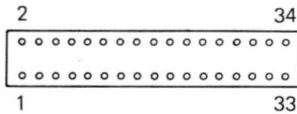
Disc power extension socket



- A. 0 V
- B. +5 V, 1.25 A
- C. N.C.

- D. -5 V, 75 μ A
- E. +12 V, 1.25 A
- F. 0 V

PL8 Disc interface

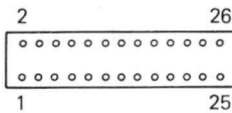


2. Side select
4. Index
6. N.C.
8. Index
10. Drive select 0
12. Drive select 1
14. N.C.
16. Load head
18. Direction

20. Seek step
22. Write data
24. Write enable
26. Track 0
28. Write protect
30. Read data
32. Side select
34. N.C.

All odd numbered pins are connected to 0 V

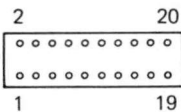
PL9 Printer connector



1. STB
3. DATA 0
5. DATA 1
7. DATA 2
9. DATA 3
11. DATA 4
13. DATA 5
15. DATA 6
17. DATA 7
19. CA1
21. N.C.
23. N.C.
25. N.C.
26. N.C.

All even numbered pins (except 26) are connected to 0 V

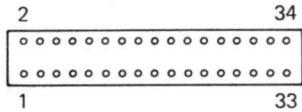
PL10 User port



1. +5 V
2. CB1
3. +5 V
4. CB2
6. PB0
8. PB1
10. PB2
12. PB3
14. PB4
16. PB5
18. PB6
20. PB7

All odd numbered pins (except 1 and 3) are connected to 0 V

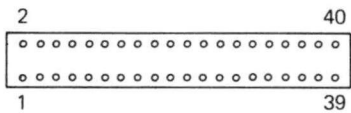
PL11 1 MHz expansion bus



- | | |
|---|---------|
| 2. R/NW ($\overline{R/\overline{W}}$) | 23. D5 |
| 4. 1MHZE | 24. D6 |
| 6. NNMI (\overline{NMI}) | 25. D7 |
| 8. NIRQ (\overline{IRQ}) | 26. 0 V |
| 10. NPGFC (FRED) | 27. A0 |
| 12. NPGFD (JIM) | 28. A1 |
| 14. NRST (\overline{RST}) | 29. A2 |
| 16. ANALOGUE IN | 30. A3 |
| 18. D0 | 31. A4 |
| 19. D1 | 32. A5 |
| 20. D2 | 33. A6 |
| 21. D3 | 34. A7 |
| 22. D4 | |

All odd numbered pins 1 to 17 are connected to 0 V

PL12 The tube



- | | |
|---|--------|
| 2. R/NW ($\overline{R/\overline{W}}$) | 22. D5 |
| 4. 2MHZE | 24. D6 |
| 6. NIRQ (\overline{IRQ}) | 26. D7 |
| 8. NTUBE (\overline{TUBE}) | 28. A0 |
| 10. NRST (\overline{RST}) | 30. A1 |
| 12. D0 | 32. A2 |
| 14. D1 | 34. A3 |
| 16. D2 | 36. A4 |
| 18. D3 | 38. A5 |
| 20. D4 | 40. A6 |

Odd numbered pins 1 to 29 are connected to 0 V;
31 to 39 are connected to +5 V

Index

- Accumulator 2
- Address bus 1
- ADVAL command 14, 48
- Advanced Data Link Control, ADLC 15
- Advanced low-power Schottky TTL 104
- ANALOG IN 17
- Analogue-digital converter, ADC 12, 43
 - conversion rate 46
 - conversion time 46
 - resolution 46
 - successive approximation method 98
- Analogue inputs 12, 43–56
- ANSI/IEEE Std 488–1975 65
- Assembler 2, 5, 7, 107
- Asynchronous Communications Interface Adapter, ACIA 10
- Auxiliary Control Register, of VIA 39

- Background task 35
- Bang-bang control 76
- Baud rate 10
- Block numbers 62
- BRK instruction 36
- Bus
 - address 1
 - control 1
 - data 1
 - 1 MHz 3, 16, 57–75
- BYTEV 5

- Capacitor, charge/discharge display 99
- Cassette port 8, 11
- Clean page selection signals 59
- Clear Interrupt Disable, CLI 38
- Clock stretching circuitry 59
- Closed loop control 80

- Compiler 107
- Condition flags 109
- Contact bounce 29
- Control bus 1
- Conversion rate, of ADC 46
- Conversion time, of ADC 46
- CRT controller 8

- Darlington transistor 22
- Data bus 1
- Data Direction Register, DDR 21, 69
- Debouncing, of switch 29
- Digital-analogue converter, DAC 12, 51
- Digitizer
 - radius arm 91
 - two sliders 91
- DIM statement 6

- Econet 14
- End of conversion signal 46
- Ethernet 14
- Events 37
- Expansion box 17, 57
- Extended page number 17, 61
- External paging register 17

- Fanout 104
- Field, of instruction 2
- Fire buttons 14, 49
- Floppy Disc Controller, FDC 16
- FOR statement 7
- Foreground task 35
- Frame 10
- FRED 3, 57
- FX calls 5

- Games paddles 14
- General Purpose Interface Bus, GP-IB 64
- Gray code 31

- Handshake transfers 67
- Highway 1
- Housekeeping operations 35, 108
- HP-IB 64
- IEC-625-1 bus standard 65
- IEEE-488 bus 64
- Image, of page register 64
- Index registers 2
- Initialisation of PIA 73
- Input port 3
- Instruction 2
- Instruction set 2
- Instruction types 109
- Insulation Displacement Connectors, IDC 15
- Interpreter 108
- Interrupt 32
 - vectors 36
- Interrupt Enable Register, of VIA 39
- Interrupt Flag Register, of VIA 38
- Interrupt Request, IRQ 17, 35, 37
- JIM 3, 57, 61
- Joystick controller 14
- Keyboard scanning 27
- Last in, first out memory, LIFO 35
- Light emitting diode, LED 22
- Light pen strobe, LPSTB 14
- Light-dependent resistor 53
- Listener, on IEEE-488 bus 65
- Local area network, LAN 14
- Look-up table 31
- Low-power Schottky TTL 104
- Machine code programming 107-10
- Mark-to-space ratio 76
- Masking, of interrupts 38
- Memory
 - LIFO 35
 - RAM 1
 - ROM 2
 - stack 35
- Memory-mapped I/O 3, 57, 111
- Mode 7 7
- Multiplexing, of displays 25, 83
- Multiplying digital-analogue converter, MDAC 52
- Non-maskable Interrupt, NMI 17, 35
- Object code 107
- Op-code byte 108
- Op-code set 2
- Open-collector TTL 105
- Operational amplifier 94
- OPT command 7
- Optical shaft encoder 31
- Opto-switch 53
- OSBYTE calls 5, 49, 57
- Oscilloscope 95
- Output port 3
- Output Register, OR 21
- Oven controller 81
- Page 16
- Page register 17, 61
- Pedestrian control of traffic lights 39
- Peripheral Control Register, of VIA 39
- Peripheral Interface Adapter, PIA 69
- Pixel 7
- Pop, from stack 35, 110
- Port 3
- Potentiometer 12
- Printer interface 16
- Program counter, PC 2
- Programmable unijunction transistor 53
- Pull, from stack 35, 110
- Pull-up resistor 28
- Pulse width modulation 81
- Push, to stack 35, 110
- Put-and-take ADC 98
- RAM, random access memory 1
- Read only memory 2
- Read/write memory 2
- Reed relay 25
- Relaxation oscillator 53
- Relay 25
 - solid-state, SSR 26
- Reset button 14
- Reset vector 35
- Resolution, of ADC 46
- Return from Interrupt, RTI 35
- Return from Subroutine, RTS 7, 38
- Reversing scan method 32
- Ring counter 84
- ROM, read only memory 2
- RS232-C standard 8
- RS423 standard 8

- Sample and hold circuit 46
- Sampling theorem 50
- Sawtooth waveform 44
- Schottky diode 104
- Second processor 17
- Selection register, of ADC 46
- Set Interrupt Disable, SEI 38
- Seven-segment display 22
- SHEILA 3
- Signal conditioning 94
- Simple harmonic motion 89
- Sinewave, generation of 86
- Solid-state relay, SSR 26
- Source code 107
- Stack 35
- Stack pointer, SP 2, 35, 110
- Staircase generator 87
- Staircase waveform 43
- Start bit 10
- Status 19
- Status register
 - of ADC 46
 - of processor 2
- Stepper motor 78
- Stop bit 10
- Successive approximation method 98
- Talker, on IEEE-488 bus 65
- Task 35
- Teletext mode 7
- Three-state TTL 105
- Totem-pole circuit 102
- Traffic light sequence 26, 39
- Transistor-transistor logic, TTL
 - 102-6
- Tube 17
- UART 10
- Unijunction transistor, UJT 53
- Unit load, of TTL 104
- Universal Asynchronous Receiver/
 - Transmitter, UART 10
- User port 16, 19-42
- V24 interface 8
- Valid memory address, VMA 59
- Vector 35
 - break, BRKV 36
 - byte, BYTEV 5
 - event, EVNTV 37
 - IRQ1V 37
 - IRQ2V 37
- VELA 74
- Versatile Interface Adapter, VIA 14,
 - 19, 118-37
 - Auxiliary Control Register,
 - ACR 39
 - Interrupt Enable Register, IER
 - 39
 - Interrupt Flag Register, IFR
 - 38
 - Peripheral Control Register,
 - PCR 39
- Video signals 7
- Voltage to frequency converter, VFC
 - 44
- Watchdog timer 67
- Wiper, of potentiometer 12
- Wired-OR 17, 57, 66, 105
- Zero page 17
- *FX calls 5
- 1MHz bus 3, 16, 57-75
- μ PD7002 converter 12, 138-41

