ISSUE 5

DIGIPLAN USER INFORMATION

IF1 & IF2 STEPPER MOTOR INTERFACES

CONTENTS

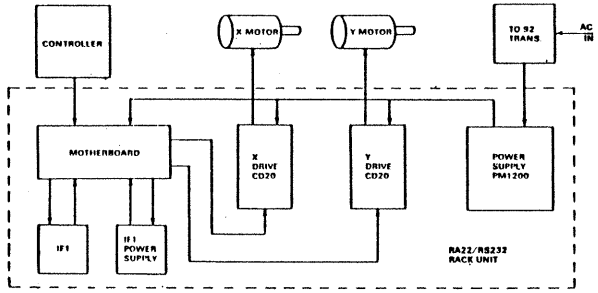
IF1 & IF2 STEPPER MOTOR INTERFACES

INTRODUCTION

The IF1 and IF2 Interface Cards permit up to three stepper motors to be controlled via an RS232C serial link. The controller may be a simple non-intelligent terminal, a microprocessor, programmable logic controller or a mainframe computer, in fact any suitable device with an RS232C port.

Information is transmitted to the interface in the form of a series of commands giving the required speed, direction, distance etc. From this information the interface generates a ramped pulse waveform suitable for feeding directly to the clock input of a stepper drive. The controller is then free to perform other tasks while the motor is running.

The diagram below shows a typical two-axis system using the IF1. The interface, stepper drives and power supplies are normally housed in a pre-wired rack with all the interconnections between units already made. The interface motherboard incorporates an RS232 socket for direct connection from the controller.



TYPICAL SYSTEM

The IF2 is functionally identical to the IF1 but it includes a battery backup facility, enabling a complete move sequence to be retained with power removed and the controller disconnected.

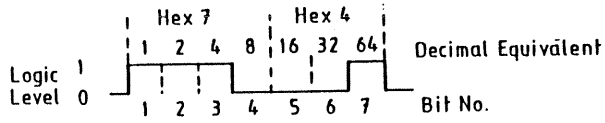
SERIAL COMMUNICATION VIA RS232C

The IF1 receives and sends data using an RS232C port connection on a suitable controller. The RS232C specification defines all the essential details of the communications link like signal levels, connector type, pin allocations and the general data format. However there are numerous variations possible within the basic framework, and both interface and controller must be configured to operate in the same way. This method of communication is referred to as "asynchronous serial". The data is transmitted one character at a time, a character comprising a number of bits sent along the same piece of wire one after the other. Start and stop bits mark the beginning and end of each character. The time interval between each

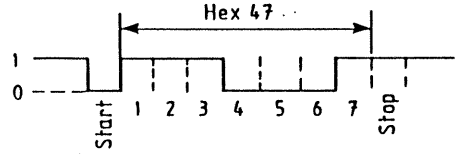
bit of the character is defined, and a number of alternative bit rates may be used. However the whole character,including start and stop bits, can be transmitted at any time with no reference to any kind of timing waveform, hence the description "asynchronous".

The serial data format.

Each transmitted character has to be represented by a string of bits which the receiver will recognise. The ASCII code is used (see Appendix 1), and all the characters we employ can be represented by 7 bits. To take an example, the run mode command "G" is represented by the hex code 47 (equivalent to decimal 71). The bit pattern will be:



The least significant bit is transmitted first. Start and stop bits are added to the character code before transmission; the start bit is at logic 0, the stop bit at logic 1. So the transmitted data becomes:



However, we have to contend with variations within this "frame" in order to match the interface to different controllers.

Number of stop bits.

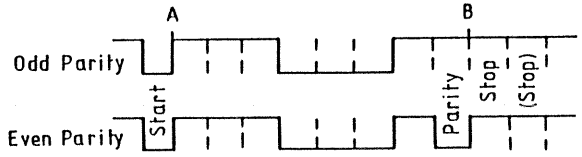
Either one or two stop bits may be used. It is quite common to use two stop bits and the interface is automatically initialised to work with two.

Number of data bits.

All the characters needed to operate the interface may be represented by 7 bits. However, some controllers send 8 bits and expect to receive 8 bits. The interface can be reconfigured to operate with 8 bits, the last bit being 0, 1 or data-dependent. The latter case is only significant in the echo-back mode or when the quick-status function is needed (see later).

Parity.

As a check for transmission errors, a parity bit can be included with the option of odd or even parity. If odd parity is used, the total number of bits at logic 1 (ignoring start and stop bits) should be odd. Therefore if the character code contains an even number of 1's, a parity bit is added to make the total odd. Similarly, with even parity the parity bit is chosen to produce an even number of 1's. Our previous example with parity added will look like this:



It can be seen that the parity bit makes the total number of bits at logic 1 between A and B odd or even as required.

Echo back.

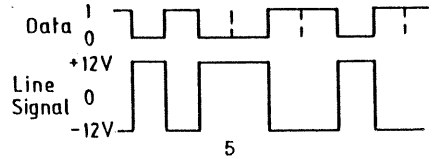
A parity check is a useful guard against transmission errors, but it's not infallible. Far greater security results from using echo-back, when each character is returned to the controller for comparison with the transmitted data. This naturally slows down communication, but the chance of an error remaining undetected becomes extremely small.

Baud rate.

All these variations on the basic format can be programmed into the interface as part of the start-up procedure. However, there's another variable parameter to mention but this one is set up automatically. Data can be sent over an RS232C link at various speeds. The options available with the IF1 are 110,300, 600, 1200, 2400, 4800 or 9600 bits per second. A speed of 1200 bits per second is described as 1200 baud; assuming 12 bits per character (8 data bits, start, parity and 2 stop bits), this is equivalent to 100 characters per second. The interface senses the baud rate automatically during transmission of a standard character from the controller. Note that speeds of 4800 and 9600 baud should only be used with echo-back.

Signal levels.

Both positive and negative voltages are used on the RS232C data lines. The IF1 uses +12v and -12v, but in practice a fairly wide range of voltage is acceptable because the receiver only has to distinguish between positive and negative. The line levels are -12v for logic 1 and +12v for logic 0, as shown below:



In the case of the control lines (DTR, CTS etc.) the positive line level corresponds to "signal present".

The RS232C control lines.

An RS232C serial link can transmit and receive data using a minimum of three lines, one to carry data in each direction and a ground. When this arrangement is used, it is essential to use echo-back for reliable communication. There are additional control lines available which are supported by the IF1, as shown in the following list. These lines enable both the transmitter and receiver to exercise greater control over the transfer of data, and ensure that both units are ready before transfer takes place.

Signal name	Description	Direction	Pin no.
DTR	Data terminal ready	O/P	20
DSR	Data set ready	I/P	6
RTS	Request to send	O/P	4
CTS	Clear to send	I/P	5
TXD	Transmit data	O/P	2
RXD	Receive data	I/P	3
0v.	Signal 0v	-	7

The line level is positive (+12v) when the signal is present, and negative (-12v) when it is absent. The function is described as "set" when positive and "reset" when negative.

Note that the data carrier detect function (DCD) is not supported by the IF1.

Before the IF1 can transmit data back to the controller, all the control lines RTS, CTS, DSR and DTR must be set. RTS and DTR are both outputs from the interface, and these will be set prior to the start of transmission. The interface will therefore not transmit data until DSR and CTS are both set. The following section gives further details on the operation of the DTR line.

Control of data transfer.

Characters sent to the IF1 are received by a UART and then loaded into a 64-character input buffer. Transfer from the UART to the input buffer takes place at approximately 1mS intervals, which means that characters can be transmitted safely at speeds up to 2400 baud. Above this speed, it is possible for characters to be lost by being overwritten before they get transferred to the buffer. It is therefore essential to use echo-back at rates above 2400 baud. If the correct character is not returned, the cancel character "# <CR>" should be sent before retransmission is attempted.

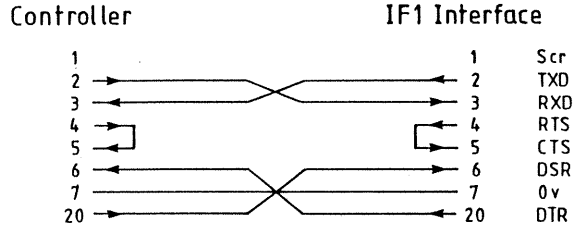
If characters are sent to the IF1 more rapidly than they can be processed, they will accumulate in the input buffer. Long strings of characters sent at high speed could therefore cause the buffer to overflow, and to guard against this the interface resets the DTR line if the buffer becomes 80% full. DTR is set again when the buffer has become less than 20% full. The controller should therefore interrupt data transmission if this signal becomes reset.

The DTR line is normally connected to the DSR input at the controller. Not all controllers support the DSR function, and in this case the DTR line may be connected to the CTS input at the controller (see sample connections in appendix).

An alternative method of controlling data transfer by software is to use the X-on, X-off mode. If the IF1 buffer is 80% full, an X-off character (13 Hex) is transmitted to the controller to terminate data transfer. When the buffer becomes 20% full, X-on (11 Hex) is transmitted to advise the controller to resume transmission. This facility is very useful when the controller features this form of handshake, since only a simple 3-wire connection is required for communication.

Alternative connection methods.

The connection method employed will depend on the facilities available in the controller. The 5-wire system should be used wherever possible as this offers the greatest security, and a typical connection scheme is shown below.

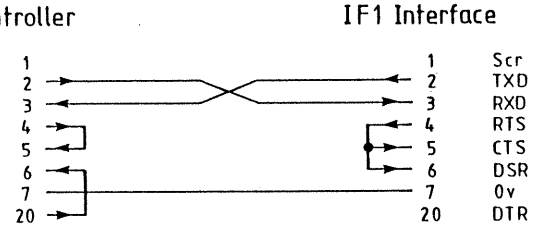


Always check the controller manual for confirmation that these connections are correct.

The 5-wire system may be used without echo-back at speeds up to 2400 baud. If strings of more than 64 characters are to be transmitted, the DTR output must be monitored to check for the "buffer full" condition. Echo-back must be implemented if higher baud rates are to be used, however this does not guarantee that the effective data transfer rate will be improved.

Note that if no connections are made to the two input control lines CTS and DSR, they will be held negative (i.e. reset) by resistors R14 and R15 on the IF1. The interface will not transmit any data under these conditions.

Where the controller cannot support the additional control lines, the 3-wire system may be used as shown below. As with the 5-wire system, echo-back should be used at speeds above 2400 baud.



The following chart summarises the requirements of the two systems.

3-WIRE SYSTEM.

Baud rate	maximum char. string	echo-back required?
up to 600	any	no
1200 or 2400	<64	no
	>64	yes
4800 or 9600	any	yes

5-WIRE SYSTEM.

Baud rate	maximum char. string	echo-back required?	need to check DTR?
up to 600	any	no	no
1200 or 2400	<64	no	no
	>64	no	yes
4800 or 9600	any	yes	no

Note that if echo-back is used it is unnecessary to monitor the DTR line since echo-back only occurs after a character has been processed. For this reason the use of echo-back will slow down communication to an extent dependant on the data being sent. In general the fastest communication will be achieved using a 5-wire system at 2400 baud without echo-back, monitoring the DTR line to control data transfer.

INSTALLATION

Power Requirements

Power required by the interface is +5V DC +/-0.1V at 900mA, +12V and -12V 10% at 30mA. The IF1 power supply card generates fully-isolated +5v, +12v and-12v rails to power the interface. It plugs into the IF1 motherboard and can derive its input power from the +24v drive logic supply or from an independent +24v supply. The power supply card also incorporates power drivers to give the five optically-isolated outputs a switching capability of 0.5A at 30v.

When the motherboard is fitted in a standard Digiplan rack, power connections will be made automatically via the 8-way jumper cables connecting the interface to the drives. To use an external +24v supply, remove links 1 and 2 from the motherboard and connect the external supply to PL3 terminal 7 (0v) and terminal 8 (+24v). If the interface is being driven from a separate 24v rail and there is no 24v supply available from the stepper drive, links 1 and 2 should be inserted. This is necessary in order to power the opto-isolators on the interface outputs.

Fault disable links.

The X and Y drive outputs include a fault connection which will halt the interface in the event of a drive fault (the auxiliary Z axis has no fault connection). The interface requires to see a logic low on the X and Y axis fault inputs even if no drive is connected. To simulate a working drive where none is fitted, fit link 3 for the X axis and link 4 for the Y axis.

Motherboard connector identification.

PL1	14-way connector for the auxiliary inputs and outputs
PL2	25-way D-connector for the RS232C port connection
PL3	8-way connector for emergency stop, limit and external +24v supply connections
PL4	X drive connector
PL5	Y drive connector
PL6	Z drive connector
PL7	expander connection (for future use)

Connections to the RS232C Port (PL2).

The alternative connection methods have already been discussed in the previous section. A full list of terminal functions appears in Appendix 2. Appendix 3 gives examples of connection schemes for specific machines, and it is recommended to use these where appropriate since they have been proven in practical tests.

Input and output signal connections.

Connectors PL1 and PL3 on the motherboard are used for external signal connections to the interface. The functions are listed below.

Connector PL1.

Terminal	Function	Circuit type (see appendix)
1	Output 5	D
2	0v for output 5	
3	Output 1	D
4	0v for output 1	
5	Output 2	D
6	0v for output 2	
7	0v for output 3	
8	Output 3	D
9	Output 4	D
10	0v for output 4	
11	External 0v connection	
12	Input 1	A
13	Input 3	A
14	Input 2	A

Connector PL3.

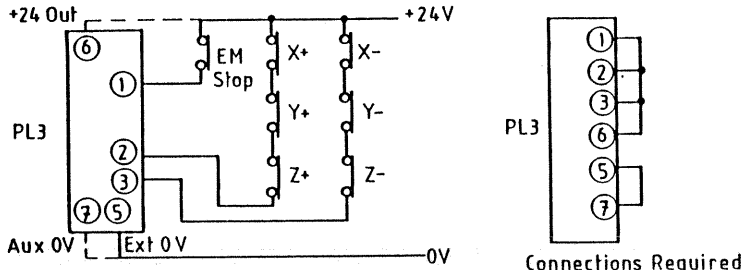
Terminal	Function	Circuit type (see appendix)
1	Emergency stop	A
2	Positive limit	A
3	Negative limit	A
4	No connection	
5	External 0v	
6	+24v out	
7	Auxiliary 0v in	
8	Auxiliary +24v in	

Emergency stop and limit inputs.

Inputs are provided for axis limit switches and an emergency stop switch. These inputs are optically isolated and commoned to the "Ext.0v" terminal. They must remain connected to +24v for the interface to run, using normally-closed switches. If the positive limit goes low or open-circuit, the interface can only be driven in the negative direction and vice versa after a status request has been made. Where more than one axis is used, connect the limit switches in series as shown. Ensure that all positive limit switches are activated by positive movement, and that negative switches are activated by negative movement.

The emergency stop input can be used to inhibit all movement and would normally be a latching-type switch. Following an emergency stop input, a Motion Status request must be made before the system can be restarted.

Emergency stop or limit inputs will light the LED (red) on the interface. It will turn off after Motion Status has been read and the input removed.

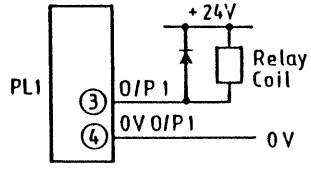


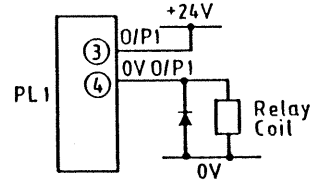
Connections Required if No Switches used

Using the optically-isolated outputs.

The five optically-isolated outputs are buffered on the power supply card by Darlington transistors which will deliver up to 0.5A with a 30v off-state voltage. These outputs are intended for driving devices such as relays or small solenoids. If using the output signals for other purposes, note that the on-state voltage may be up to 2.5v. Ensure that inductive loads are suppressed.

Two examples of circuits using these outputs are shown below.





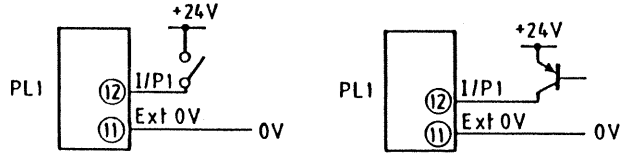
Optional fault output.

The LED (red) on the interface card is turned on when any fault, limit or emergency stop condition is present. Output 5 may be reconfigured to duplicate the function of the LED and so provide an external signal, this may be used for instance to drive an alarm and can be particularly useful when the IF2 is used in the battery-backup mode. To use the output in this way, transfer link 3 from position "a" to position "b" (nearest to capacitor C22). The output cannot be controlled by program commands with the link in this position.

Using the optically-isolated inputs.

There are three inputs which may be used to read in signals from switches etc. The inputs drive opto-isolators via built-in 3K3 series resistors, and may be driven directly from +24v. The isolators are commoned to the "Ext. 0v" terminal.

Typical connection schemes are shown below.



CONFIGURING THE INTERFACE

Initialisation.

When power is first applied, the LED comes on for approximately 1 second and during this time the interface runs through its internal initialisation routine. After this period the controller should transmit the open bracket or parenthesis character "(" at the required baud rate. The IF1 will receive this character and from it will determine the baud rate. At this point the LED flashes twice per second. The interface then transmits a "U" character at the new baud rate which the controller should receive correctly (note that all alpha characters are upper case). The "U" will be followed by a carriage return, and this will in future be represented by <CR>. If the controller misses the "U <CR>", it can transmit another open bracket "(" and the interface will respond with another "U <CR>". This may be repeated as often as necessary, however the interface only detects the baud rate on the first "(" received.

The controller must now send information to set up the data format. Two alternative formats are available to suit the requirements of different controllers. The "U" format is terminated only by carriage return <CR>, whilst the "V" format is terminated by carriage return plus line feed <CR><LF>. If the "U" format is used, the interface expects only <CR> to terminate all subsequent messages and it will similarly terminate all status and position data with <CR>. If the "V" format is selected, all following instructions must be terminated by <LF> and the interface will terminate all returned data with <CR><LF>. In the latter case, the inclusion of <CR> before <LF> is optional in messages sent from the controller.

The format control instruction will be a message of the form "U X1 X2 <CR>" or "V X1 X2 <CR> <LF>", with X1 and X2 selected from the following table:

X1 =	1	2	3	4	5	6	7	8
Number of stop bits:	1	2	1	2	1	2	1	2
Number of data bits:	7	7	8	8	8	8	8	8
Significance of bit 8:	-	-	*	*	0	0	1	1

(* = data).

X2 =	1	2	3	4	5	6	7	8	9
Parity:	None	None	Odd	Even	Odd	Even	None	Odd	Even
Echo back:	No	Yes	No	No	Yes	Yes	X-on	X-on	X-on

The characters transmitted for setting the data format MUST be sent in the ordered sequence shown above. If an error is made in transmitting any character, the whole string may be retransmitted. After the format instruction has been sent, allow a delay of at least 5mS for the UART to re-initialise.

NOTE: All future instruction examples shown in this manual will be terminated by <CR>. If the "V" format has been used, this should be interpreted as representing <CR><LF>.

Following correct receipt of the data format message, the LED will cease flashing and the interface is ready to receive move instructions. If the LED comes on again, it indicates that a fault condition has arisen which must be cleared (see later).

Inter-character delay.

If the controller operates with an inherently-slow high level language it may only be able to receive characters slowly. In this case the IF1 needs to be set up for a lower transmission rate if the data is to be received correctly.

An inter-character delay may be programmed by sending the command "D" followed by a number giving the required delay in multiples of 10mS. For example, sending "D5 <CR>" will set a minimum delay of 50mS. The maximum programmable delay is 250mS. This delay will be inserted before each transmitted character.

SENDING BASIC MOVE INSTRUCTIONS.

There are two basic modes in which the interface can operate - index mode, in which the number of steps is pre-determined, and run mode, in which the motor will run continuously until commanded to stop. In each case it is possible to program the final speed, start/stop speed and acceleration rate, so these parameters will be covered first.

Note that in the instruction examples which follow, spaces have been included for the sake of clarity. The space characters should not be interpreted literally, but they may be transmitted to the interface if desired - they will simply be ignored.

Speed.

The interface has four speed ranges in order to cater for different translator resolutions and different applications. The following table shows the maximum speed and resolution available in each range, together with the ramp increment used during acceleration.

Range	Max. speed	Resolution	Ramp increment
1	25,000 s/s	6.25 s/s	100 s/s
2	50,000 s/s	12.5 s/s	200 s/s
3	100,000 s/s	25 s/s	400 s/s
4	4,000 s/s	1 s/s	16 s/s

When a speed above the start/stop speed is programmed, the interface accelerates in increments as shown up to the final speed within the resolution of that range. For example, programming a speed of 503 steps/sec in range 1 will give a final speed of 500 steps/sec, programming 504 steps/sec will give 506.25 steps/sec. The higher speed resolution of range 4 allows speeds to be set within 1 step/sec but with a restriction of 4000 steps/sec on the maximum speed.

To select the speed range, send the command ">" followed by the range number 1 to 4 and <CR>, e.g. ">2 <CR>" will set up speed range 2. The interface defaults to speed range 1 during initialisation.

The final speed is programmed by sending the "@" character followed by the required speed in steps/sec, e.g.

"@ 5400"

will set a speed of 5400 steps/sec. If a speed is requested which is above the maximum for that range, the interface will indicate a "range error" fault (see Report-back Functions). There is a limitation of 5 decades on the programmed speed, so the maximum programmable speed in range 3 is in fact 99,999 steps/sec. However this will result in a speed of 100,000 steps/sec due to the 25 s/s resolution.

Below the start/stop speed, the speed will also be the nearest or next higher multiple of the speed resolution. So in range 1, which has a resolution of 6.25 steps/sec, 21 steps/sec programmed would run at 18.75 steps/sec, 22 programmed would run at 25. Speeds between 1 and 8 steps/sec will run at 6.25 steps/sec.

Start/stop speed.

The start/stop speed is the rate at which the clock starts when accelerating the motor into the slew region. Below this speed, no acceleration is applied and the clock will run at a fixed frequency. The start/stop speed is programmable between 100 and 4000 steps/sec, in multiples of 100 steps/sec in speed range 1. In other speed ranges, the start/stop speed is still programmed in multiples of 100 steps/sec but the interface can only set up a rate which is a multiple of the ramp increment for that range. For example, setting a start/stop rate of 300 steps/sec would produce an actual rate of 300 s/s in range 1, 200 s/s in range 2, 400 s/s in range 3 and 288 s/s in range 4.

The command character for start/stop speed is "<", therefore:

"<12 <CR>"

will set the start/stop rate at 1200 steps/sec in speed range 1. The optimum start/stop rate depends on a number of factors, notably the mechanical inertia of the system. Setting it too high will result in the motor failing to start, but if set very low it will increase the indexing time. A useful starting point is about 400 steps/sec using a 400 step/rev translator.

Acceleration rate.

The rate at which the clock accelerates from the start/stop speed up to the programmed speed can be set in multiples of 1000 steps/sec2 up to 500,000 steps/sec2. The data must be preceded by the "^" character, e.g.:

"^7 <CR>"

will set an acceleration rate of 7000 steps/sec2. Again the optimum setting depends largely on the system inertia.

Note that the angular acceleration of the motor shaft will depend on the drive resolution, so for the same angular acceleration at 1000 steps/rev the acceleration rate in steps/sec2 must be 2.5 times higher than at 400 steps/rev. The interface will remember any programmed data unless it is overwritten, so the speed, start/stop and acceleration data only needs to be sent once unless it must be altered, e.g. a new speed is required. For example:

">1 <10 ^15 @ 4800 <CR>"

will select speed range 1, a start/stop speed of 1000 steps/sec, acceleration rate of 15,000 steps/sec2 and a speed of 4800 steps/sec. This data may be sent as part of the initialisation routine or may be included with the first move instruction.

Note that after new acceleration, start/stop or speed range data has been received the interface has to calculate a new ramp, this takes up to 1 second and it will not respond during this period. However it will still accept any characters sent to it, and will act on any commands after calculating the new ramp.

Index mode instructions.

The axis to be moved is selected by an axis code X, Y or Z. This is followed by the required move distance in motor steps, together with the initialise command "$". So:

"X4000 $ <CR>"

will run the X motor 4000 steps at the programmed speed. Similarly:

"Y2500 @ 1500 $ <CR>"

will run the Y motor 2500 steps at 1500 steps/sec. To change direction, send a "-" (minus) character after the axis code, e.g.:

"X-3000 $ <CR>"

will run the X motor 3000 steps in the negative direction. A "+" character may be used after the axis code if required, but the positive direction is assumed if no direction command is sent.

More than one axis may be selected, e.g.:

"X+ Y- $ <CR>"

will run X positively and Y negatively the previously-programmed distance. The maximum programmable distance is 9 decades, i.e. 999,999,999 steps.

Note that if the speed is above the start/stop rate, the clock will be accelerated up to the programmed speed and decelerated back to rest. The distance travelled at programmed speed will be the index length less the total acceleration and deceleration distance. If the index distance is less than the number of steps required to accelerate and decelerate, the interface will calculate a profile giving a symmetrical acceleration and deceleration with a small plateau at the top.

To cancel an index, send the cancel code "# <CR>". This will stop any axis in motion.

Run mode instructions.

In this mode the axis to be moved is selected as before, but no distance is programmed and the Go command "G" is included:

"X @ 5000 G $ <CR>"

will run the X axis continuously at 5000 steps/sec. Similarly:

"Y- G $ <CR>"

will run the Y axis in the negative direction at the previously-programmed speed.

To stop the motor, send the cancel code "# <CR>".

Note that the data used in a previous index will be lost if the run mode is used, and the interface will default to zero index length. Therefore the distance MUST be stated in a subsequent index instruction.

Backlash correction.

The interface may be instructed to finish any index with a fixed number of steps in a specified direction in order to eliminate any backlash in the system. The command to store the backlash distance is "B" followed by the required direction and distance in motor steps, up to 255 steps. For example:

"B - 20 <CR>"

will store a backlash distance of 20 steps in the negative direction. The correction will be applied at the programmed start/stop speed. If this is followed by an index command such as:

"X 100 B $ <CR>"

the motor will run 120 steps in the positive direction, then reverse and move back 20 steps in the negative direction. In this case there will be no correction applied if the index is negative.

Note that the backlash direction must ALWAYS be specified, even if it is positive. So to set a positive backlash correction of 50 steps, send "B + 50 <CR>".

If more than one axis is moved at a time with backlash correction, the directions must all be the same otherwise a position error will be introduced.

A pause of at least 3mS will occur before the motor reverses, this may be extended if the interface is being interrogated at the time.

OUTPUT CONTROL.

Output switching commands.

The 5 optically-isolated outputs may be set high or reset low under program control in order to activate external circuits. The buffer transistor is turned on by the set command "S" and turned off by the reset command "R", followed by the output number and the "$" character. For example:

"S2 $ <CR>"

turns on output 2.

"S13 R2 $ <CR>"

turns outputs 1 and 3 on, and turns output 2 off. When combining set and reset commands in this way, always send the set command first.

The outputs may also be driven in a pulsed fashion with a programmable delay. The command used is "P" followed by the output number, comma and pulse length in increments of 10mS. The maximum pulse length is 990mS. For example:

"P4, 20 $ <CR>"

will generate a pulse 200mS long on output 4. The phase of the output pulse will depend on the prior state of the output, so if output 4 was previously reset it will turn on for 200mS.

Combining inputs and outputs with move instructions.

Move instructions can be combined with output commands so that one or more outputs will change state at the end of the move. In a similar way, the inputs may be interrogated so that the move is initiated by an input transition. Take the following simple example:

"H1 X 1000 @ 200 S3 $ <CR>"

In this case, the X-axis move of 1000 steps at 200 steps/sec will begin when input 1 goes high. At the end of the move, output 3 is set (i.e. turned on).

When commands are combined in this way, the following format must be followed:

1. Read inputs for high levels.
2. Read inputs for low levels.
3. Send index and speed data.
4. Set outputs (i.e. turn on).
5. Reset outputs (i.e. turn off).

Stages 4 and 5 may be replaced by an output pulse command, but an output cannot be pulsed at the same time as other outputs are set or reset (see Sequence Mode for this type of operation).

Not all the stages shown above need be included, but the remainder MUST be sent in the order shown. For instance, it is not permissible to reset one output before another output is set.

Input commands can only be used in combination with move commands. The commands "H" and "L" are used to specify a high or low state respectively, as shown in the following examples:

"H12 X 1000 @ 200 $ <CR>"

This move will commence when inputs 1 and 2 are high.

"H1 L3 X 1000 @ 200 $ <CR>"

In this case the move starts when input 1 goes high followed by a low on input 3.

A high-to-low transition can be specified as follows:

" H2 L2 X 1000 @ 200 $ <CR>"

This move will be performed following a high-to-low transition on input 2. Note that a low-to-high transition is not allowed within the specified format, but the function can be performed using the Sequence Mode. Two further examples illustrate the combination of inputs and outputs with move commands:

"L13 X 1000 @ 200 S23 R5 $ <CR>"

"H1 L23 X 1000 @ 200 P4, 20 $ <CR>"

In the first example, a low on inputs 1 and 3 will start the move. At the end of the move, outputs 2 and 3 will be turned on and output 5 will be turned off. The second move will begin when input 1 is high followed by a low on inputs 2 and 3; following the move, output 4 is pulsed for 200mS.

SEQUENCE MODE.

Programming a sequence.

The IF1 may be programmed to perform a sequence of up to 63 separate operations, each of which can be anything from a simple command to a complex combination as in the last two examples. The IF2 has a battery backup facility which enables the sequence to be retained when power is removed (this feature is described later).

The stages in a sequence are denoted by the ":" symbol. Here is a simple example:

": X 1000 @ 200 : Y : X- : Y- : $ <CR>"

This sequence comprises four moves which cause an X-Y system to describe a square. The sequence starts as soon as the "<CR>" is received, and the whole sequence can be repeated by simply sending ":$<CR>". Note that parameters such as start/stop rate, speed range and acceleration rate cannot be placed within a sequence and must be set up before the sequence is programmed.

The next example illustrates a wider variety of instructions:

": S2 : H13 L2 Z 100 @ 40 S4 R1 : P2, 50 : Y @ 200 : $ <CR>"

This sequence will turn on output 2; when inputs 1 and 3 are high followed by a low on input 2, the Z motor will run for 100 steps at 40 steps/sec, after which output 4 is turned on and ouput 1 is turned off. Output 2 is then turned off for 500mS, and finally the Y motor runs for 100 steps at 200 steps/sec. The format used in specifying complex sequence-mode instructions is the same as in direct mode.

If a mistake is made when entering a sequence, terminate the incorrect sequence by sending ": $ # <CR>". The sequence may then be re-entered.

Repeating parts of a sequence.

It is possible to repeat parts of a sequence, or indeed the whole sequence, without sending all the data again. The repeat command "=" is used for this purpose, followed by the number of repeats required and the step number from which the repeat is to commence. Here is a simple example:

": S1 : X 400 @ 100 : Y : = 5, 2 : R1 : $ <CR>"

After output 1 has been set, X runs 400 steps at 100 steps/sec followed by the same move in Y. The sequence is then repeated five times from step 2, i.e. the X move, giving a total of six X moves and six Y moves. Finally output 1 is reset. Up to 255 repeats can be specified, so the same move or group of moves can be performed 256 times.

Subsequent parts of the sequence can also be repeated, or a more complex routine can be programmed by nesting repeats within the sequence. Look at the following example:

": S1 : X400 @ 100 : Y : = 5, 2 : R1 : Z50 : = 24, 1 : $ <CR>"

The initial part of this sequence is the same as the previous example, so will perform six X moves and six Y moves as before. After output 1 has been reset, the Z motor runs for 50 steps and then the entire sequence is repeated from step 1 a further 24 times. The repeat command can be combined with input commands, provided the following format is followed:

1. Read inputs for high levels.
2. Read inputs for low levels.
3. Send repeat data.

The instruction ": H2 = 10, 4 :" would repeat the sequence ten times from step 4, with each repetition waiting for a high level on input 2.

A sequence can be made to repeat indefinitely by programming zero for the number of repeats, as in this example:

": X 1000 @ 400 : Y : X- : Y- : = 0, 1 : $ <CR>"

Here is our X-Y system tracing out a square again, and it will continue round until the cancel code "# <CR>" is sent. This

terminates the sequence, and it can be restarted from step 1 by sending ":$ <CR>". Once a sequence has been terminated, it can only be restarted from the beginning.

An alternative way of stopping the sequence is to use one of the inputs. If we take the previous example and add ": H1" to the beginning, the sequence will only continue to repeat if input 1 is high. We can therefore use a switch to turn the sequence on and off, and in this case it will always stop at the origin.

Aborting a run-mode move in a sequence.

A run-mode move may be included within a sequence, followed by the abort command "A" combined with an input command:

": X @ 1000 G : H1 A : $ <CR>"

This sequence will start the X motor running at 1000 steps/sec, and will stop it again when a high level is seen on input 1. Again the format is:

1. Read inputs for high levels.
2. Read inputs for low levels.
3. Send abort command.

The repeat command can be combined with an abort instruction, as in this example:

": H3 A = 2, 1 :"

When a high level is seen on input 3, the run will be aborted and the sequence will repeat twice from step 1. The format to be used is:

1. Read inputs for high levels.
2. Read inputs for low levels.
3. Send abort command.
4. Send repeat data.

Controller intervention in a sequence.

The controller can intervene in a sequence by including the command character "%". When this command is seen the interface will transmit "% <CR>" back to the controller and then execute any move instruction it receives. The sequence will resume when another "% <CR>" command is received. Take the following example:

": Y 400 @ 200 : % : Y- : $ <CR>"

Following the Y move of 400 steps, the interface transmits "% <CR>" to the controller and another move instruction is returned. The sequence resumes with the negative Y move when "% <CR>" is sent back to the interface.

The instruction sent from the controller may be a move combined with input and output commands, as well as acceleration rate and start/stop speed data. This feature is useful where data-dependent moves must be included in a preset sequence.

Backlash correction in sequence mode.

A command to reset the backlash distance may be included within a sequence, and of course this is useful if different axes require different backlash correction. Here is an example of part of a sequence:

": B - 20 : X 1000 B : B + 50 : Y - 400 B : $ <CR>"

The backlash correction is initially set at 20 steps negative for the X move, and is then changed to 50 steps positive for the Y move. Note that the backlash direction must always be specified.

Adding time delays.

A delay of up to 990mS can be incorporated in a sequence by using the pulse output command "P" and specifying output 0, as in this example:

": X 1000 P0, 50:"

The interface will wait for 500mS after the X move before continuing the sequence. Delays longer than 990mS can be obtained by repeating the pulse command, or of course controller intervention can be used and the delay generated by the controller.

LINEAR INTERPOLATION MODE.

Linear interpolation can be performed using the IF1. The linear interpolation function operates with any two axes at one time. The data is presented as a long index (the major axis) and a short index (the minor axis) together with the speed at which the major axis is to run. The minor axis speed is calculated by the interface such that the two axes will be in proportion to their respective index lengths.

Programming in the linear interpolation mode.

Linear interpolation can be performed between any two axes. The move made by either the mojor or minor axis may be duplicated by the third axis if required. The major axis move is given first, as follows:

" X 2000 / Y 500 @ 1000 $ <CR> "

The above instruction will cause the X axis to run for 2000 steps at 1000 steps/sec, and the Y axis to run for 500 steps at 250 steps/sec (i.e speed ratio 4:1).

The "/" (02F hex) character denotes the linear interpolation mode. This character always precedes the minor axis. As with normal indexing, inputs and outputs can be combined with move instructions. For example:

"H1 L2 X2500 / Y100 @ 6000 S135 R24 $ <CR>"

Duplicating the major or minor axis move on the third axis.

This is illustrated in the following examples:

```
1.      " X 2000 / YZ 500 % 1000 $ <CR> "
2.      " XY 1000 / Z 50 % 800 $ <CR>
3.      " YZ / X $ <CR>
```

Instruction (1) will cause the X axis to move 2000 steps at a speed of 1000 steps/sec, and both Y and Z axes to run 500 steps at 250 steps/sec.

Instruction (2) will cause both the X and Y axes to run 1000 steps at 800 steps/sec, and the Z axis to run 50 steps at 40 steps/sec.

Instruction (3) duplicates the previous move with X and Z axes interchanged.

Constraints in linear interpolation mode.

Maximum velocity:	10000 steps/sec
Minimum acceleration:	5000 steps/sec^2
Maximum index on major axis:	65535 steps
Speed range:	Range 1 only

If any of these parameters do not comply with the above constraints, an out-of-range flag is set in the status command (e.g. k <CR> = 128, F <CR> = 16), and all instructions are ignored.

Linear interpolation in sequence mode.

Linear interpolation can be incorporated in a sequence operation in a similar format to normal index instruction. For example:

```
"H1 X1000:L1X / Y200 % 6000 P1,20 : Y1000:=3,1: $ <CR>"
```

If parameters are chosen which are outside the constraints for linear interpolation mode, the interface will fault out. A status request must be made to reset the fault bit, and the appropriate parameter changed before the sequence can be run.

COMMUNICATION LOCKOUT COMMANDS.

RS232 communication can be inhibited by using the lock command "[<CR>". This will prevent the interface from receiving any further instructions until the unlock command "] <CR>" is received, although it will respond to status request commands. This facility is useful if the controller must be disconnected or switched off as it prevents the interface from receiving spurious data. Note that the unlock command must be sent at exactly the same baud rate, number of bits etc. as the lock command, otherwise the interface will not unlock. However, it is possible to re-initialise the interface with a different data format, this is explained under the Battery Backup feature.

REPORT-BACK FUNCTIONS.

The interface can be interrogated to establish its status at any time, except for a period of about 1 second following receipt of new acceleration, start/stop or speed range data. An obvious requirement for status is in checking that motion is complete before sending another move instruction. Status information is obtained by sending the appropriate command and then waiting for a response terminated by <CR>.

Motion status.

The command for motion status is "K" and the typical response could be "41<CR>". The following table gives a list of information obtainable from the status number.

Value	Function
1	In Motion
2	At Programmed Speed
4	Accelerating
8	Decelerating
16	Drive Fault
32	Emergency Stop
64	Limit
128	Communication Fault

In the previous example 41 would indicate the interface has had an emergency stop and is decelerating.

If the "communication fault" bit is present, then this indicates that there is a communication or data problem. For further information on the nature of the fault, the command "F <CR>" should be sent. The response would then be typically "8 <CR>".

The following table gives information on data faults:-

Value	Function
1	Parity Error
2	Overrun Error
4	Framing Error
8	Data Fault or Buffer Overflow
16	Out-of-range Error
32	In Battery-Backup Mode

PARITY ERROR - this is set when a parity error is detected.

OVERRUN ERROR - this is set when the first character has not been read before the next one becomes available.

FRAMING ERROR - this is set when a valid stop bit is not detected at the end of every character.

DATA FAULT - this means that the correct common serial format has not been set up on initialisation or the interface is not receiving the correct format. It may also indicate that the input buffer is 95% full, in which case all the characters in the buffer will be lost.

OUT-OF-RANGE ERROR - this fault occurs if the maximum start/stop rate, acceleration rate or speed is exceeded in a particular range.

IN BATTERY-BACKUP MODE - this is not in fact a fault condition but simply indicates that the move or sequence data has been locked.

Note that any communication fault bits are only cleared when the "F" command is sent, so this should always be done when "K128" is received (see "Clearing Fault Conditions"). Failure to do this may result in incorrect fault data being received at a later time.

Input Status.

This can be obtained in a similar way to motion status. The command is "I" and will read all inputs to the interface. The following table gives their values and names.

Value	Name	Input voltage (w.r.t 0v) when value returned
1	Emergency Stop	Low
2	X Fault	High
4	Y Fault	High
8	I/P 1	Low
16	+ Limit	Low
32	- Limit	Low
64	I/P 2	Low
128	I/P 3	Low

Note that all inputs return their corresponding value when there is NO current in the diode of the opto-isolator. The two drive fault inputs are pulled up to +24v (see pages 36 & 37), and the "no current" state therefore corresponds to a high input voltage. If an emergency stop input is seen, a motion status request must be made to reset the system.

Sequence status.

The command "O" (letter O, not the number zero) is used to obtain sequence status. Sending "O <CR>" will cause the interface to return a number between 1 and 63 giving the current step in the sequence.

Quick Status.

The three status commands previously described will be fairly slow in their transmission, especially if a slow baud rate is used. To enable quick detection of motion status a special command "Q" is available. The interface will respond by transmitting a single 8 bit byte which will have the value 0 to FF Hex. This only applies in 8-bit mode with bit 8 = 1 or data.

Each bit has a significance as in the table below :-

Bit	Decimal significance	Function
0	1	In Motion
1	2	At Programmed Speed
2	4	Accelerating
3	8	Decelerating
4	16	Drive Fault
5	32	Emergency Stop
6	64	Limit
7	128	Communication Fault

For example, sending "Q <CR>" we could get the response of a single byte of value 73 decimal or 49 hex. This would indicate that a limit input has appeared whilst an axis is in motion and decelerating.

Note: not all controllers can use this mode as the 8 bit byte may be equivalent to an ASCII code used in the high level software, e.g. if all axes are stationary and there are no faults present, the equivalent ASCII code is 00 (hex) or NUL. Similarly if a drive fault should occur whilst an axis is in motion at programmed speed, the "X-off" character (13 hex) is returned.

Busy Status.

Another quick method of determining if the interface is occupied is by sending the command "E". The interface will then respond by transmitting "E <CR>" if it is busy and "C <CR>" if it is clear for new data or commands. It will send back "[<CR>" if it is in the battery backup mode, and "F <CR>" if there is a fault condition. Note that the busy code "E <CR>" also encompasses "motionless" activities such as waiting for inputs and pulsing outputs. In this respect it is different from the "in motion" status returned by "K <CR>".

Position report-back.

If an index or run command is terminated, either from the controller or by an external signal, the interface may be interrogated to find the number of clock pulses sent to the motor. The command used is "N" and this will cause the interface to return the number of steps produced from the start of the index or run. As an example:

"N <CR>" sent by controller
" 527 <CR>" returned by interface.

This indicates that 527 steps were sent to the motor by the time it came to rest. The maximum distance which can be returned is approximately 4 x 10^9 steps (FFFF.FFFF hex), beyond this point the position counter returns to zero and starts again. If motion is stopped as a result of a drive fault, no deceleration occurs on that axis and due to the nature of this fault the position returned is not meaningful.

A position request may be made whilst an axis is in motion, but the value returned will be of limited accuracy. Above the start/stop speed, there will be an error of up to 256 steps. Below the start/stop speed, the error will generally be small but it will depend on motor speed and delays in RS232 communication.

Position report-back in linear interpolation mode.

In the run mode, the maximum number of steps returned is 65535. If a distance greater than 65535 steps has been covered, the value 65537 needs to be added to the number of steps reported back after the initial "rollover" of 65536 steps. If the speed is below the start/stop rate, then the position returned has a maximum value of approximately 4×10^9.

CLEARING FAULT CONDITIONS.

If a fault, emergency stop or limit condition arises, any axis in motion will stop and the LED on the interface will come on. Under these conditions further movement is inhibited, apart from motion away from a limit switch.

A status request must be made in order to establish the cause of the problem, using either the Motion Status ("K") or Quick Status ("Q") commands. Either of these commands will reset the relevant fault bit provided the fault is no longer present. The LED will go out when the fault bit has been reset.

In some instances it may be necessary to make a status request twice, for example in the case of an emergency stop input. The first status request will indicate an emergency stop, i.e. the interface returns "32 <CR>". However this will not reset the fault bit if the emergency stop is still present. Once emergency stop has been removed, a further status request will again give "32 <CR>" but this time the fault bit will be cleared and operation can resume. If a status request is made a third time, the interface will return "0 <CR>".

If the interface returns "128 <CR>", indicating a communication or data problem, the controller should always respond by sending "F <CR>". This not only serves to analyse the nature of the fault, but also resets the corresponding fault bit. If the "F" command is not sent, movement will not be inhibited but subsequent use of this command may give false information.

In the case of the limit inputs, a status request will be necessary to establish the fault and this will serve to reset the fault bit. The system can be driven off the limit in the opposite direction and no further status request need be made.

Note that Output 5 can be configured to serve as a Fault output and to duplicate the function of the LED (see "Installation").

DEFAULT VALUES.

When the power is applied to the interface, it sets up sensible default values for speed and acceleration rate etc. The following table gives these parameters.

Speed range	-	1 (0 - 25,000 steps/sec)
Speed	-	400 steps/second
Index Length	-	0 steps
Backlash correction	-	0 steps
Acceleration	-	10,000 steps/sec2
Start/Stop	-	400 steps/sec
All Direction O/P	-	High (open collector)
Outputs 1-5	-	Off (reset)
Pulse output	-	10 (100mS)
Inter-char. delay	-	0 mS
RS232 data format	-	8 bits, 2 stop bits with bit 8 = 0; no parity, no echo-back

USING THE BATTERY BACKUP (IF2 only).

The battery backup feature enables any data or sequence sent to the interface to be stored when power is removed. The interface then becomes a stand-alone pre-programmed controller.

Storing a sequence in the IF2.

Sequence data is sent to the IF2 in a similar way to the IF1 (see page 19). Having loaded the interface with the sequence statement, the lock command "[<CR>" must then be sent before power-down, otherwise the sequence data will be lost. It is recommended that the sequence data begins with an input command such as ":H1:". This command prevents the interface from executing the sequence at power-up until the appropriate input signal is seen. The IF2 can now operate as a stand-alone controller when power is restored.

Running the stored sequence.

A switch input may be used to start the stored sequence at power-up. Whilst the interface is locked up, full communication cannot be established. However, it is possible to communicate via the RS232 cable to obtain status information if the following conditions are met:

(1) The DSR line is inactive, i.e. low. In a 3-wire system this can be achieved by linking DSR to CTS and RTS.

(2) The controlling device is configured to transmit at the same baud rate and data format as defined by the "U" and "V" commands issued by the original programming device.

Note also that in this mode, communication is restricted to status request codes only, and any other command instructions will be ignored. No communication error checking is carried out whilst the interface is locked up. This is necessary because the lock command may also be used to ensure that the interface does not respond to spurious data. Error checking becomes operative again as soon as the unlock command has been received.

If an emergency stop input is received when the interface is operating in the battery-backup mode, power must be temporarily removed in order to reset the system.

Changing the sequence data.

To change the stored sequence data, it is first necessary to unlock the interface. The unlock command "] <CR>" should be sent at same baud rate and data format as the previous lock command, otherwise the interface will not unlock. The stored sequence can now be reprogrammed (as outlined under storing of sequence data), by sending the cancel command "# <CR>" followed by the new sequence data.

The baud rate and data format can be reprogrammed even if the interface is in the locked-up mode. In this case the DSR line must be held active, i.e. high. Again in a 3-wire system, this can be achieved by linking it to DTR. The open bracket "(" must then be sent followed by the new baud rate and data format, to establish the communication protocol (see page 13). The IF2 will start to execute the previously-stored sequence as soon as the "U" or "V" command has been received.

Breaking into a locked interface.

If for some reason the normal command "] <CR>" cannot unlock the system, a simple way to overcome this problem is to issue the open bracket "(" followed by the erase command "*". The system becomes operative immediately the erase command is acknowledged. The response to the erase command is "U<CR>". This method may also be used to erase stored sequence statements in the RAM when the EPROM issue is up-dated.

Using a computer to program the interface.

If a long sequence needs to be programmed, it may be preferable to use a small computer to prepare the sequence statements. A typical program which will perform this function is shown in sample 6. In this program, the state of the DSR is software-controlled via the Basic program. The user is prompted to switch on the interface as part of the initialisation routine. In order to prevent spurious characters being detected during the power-up procedure, the program has been designed such that the first instruction to the IF2 is discarded. If DSR cannot be controlled by software, it may be necessary to make a dedicated cable with DSR connected to DTR.

APPENDIX 1. FULL COMMAND LIST.

All commands and data are sent as ASCII codes. Characters marked with a "phi" (¢) sign vary on certain keyboards, see next page. Note that all alpha characters are upper case.

ASCII	HEX	DEC	Function
<LF>	0A	10	Line feed from controller
<CR>	0D	13	Initialise direct commands
<SP>	20	32	Space Character
#	23	35	Cancel (¢)
$	24	36	Index, run and sequence initialise (¢)
%	25	37	Wait for control
(	28	40	Baud rate detect
*	2A	42	Erase RAM (battery backup)
+	2B	43	Dir port high
,	2C	44	Delimiter
-	2D	45	Dir port low
/	2F	47	Linear interpolation
:	3A	58	Sequence divider
<	3C	60	Start/stop Speed (¢)
=	3D	61	Repeat
>	3E	62	Select Speed range (¢)
@	40	64	Speed data (¢)
A	41	65	Abort in Sequence Mode
B	42	66	Backlash
D	44	68	Inter character delay
E	45	69	Busy
F	46	70	Communication status
G	47	71	Run mode
H	48	72	Input high
I	49	73	Input status
K	4B	75	Motion Status
L	4C	76	Input low
N	4E	78	Number of steps run or indexed
O	4F	79	Sequence status
P	50	80	Pulse output
Q	51	81	Quick status byte
R	52	82	Reset output low
S	53	83	Set output high
U	55	85	RS232 format set (termination on <CR>
V	56	86	RS232 format set (termination on <LF>
X	58	88	X clock
Y	59	89	Y clock
Z	5A	90	Z clock
[	5B	91	Communication lock (¢)
]	5D	93	Communication unlock (¢)
^	5E	94	Acceleration rate data (¢)

APPENDIX 1 (CONT.)

Certain control characters are represented differently on non-USA keyboards. The table below shows these variations.

Dec	Hex	U.S.A.	France	Germany	U.K.	Denmark	Sweden	Italy	Spain	Norway	Nether lands	Switzer land	Iceland	Spanish America	Portugal	Turkey	Greece	Africa	Anglo Universal
35	23	#	#	#	£	#	#	#	₧	#	£	£	£	#	£	₺	#	£	#
36	24	$	$	$	$	$	¤	$	$	¤	$	$	$	$	$	é	$	$	$
60	3C	<	<	<	<	<	<	<	<	<	<	é	<	<	<	ı	<	<	<
62	3E	>	>	>	>	>	>	>	>	>	>	è	>	>	>	ŧ	>	>	>
64	40	@	à	§	@	@	É	@	@	É	@	§	Ð	@	§	Ç	@	@	@
91	5B	[	°	Ä	[	Æ	Ä	°	¡	Æ	ƒ	[	Þ	£	Ç	ğ	§	ħ	[
92	5C	\	ç	ö	\	Ø	ö	\	Ñ	Ø	\	ç	¨	Ñ	¨	ö	ç	ł	¥
93	5D	]	§	ü	]	Å	Å	é	¿	Å	ij	]	Æ	ª	º	ş	^	°	]
94	5E	^	^	^	^	^	ü	^	^	ü	^	^	ö	¿	^	ü	ш	^	¢

APPENDIX 2. EDGE CONNECTIONS TO IF1 CARD.

Row A		Row C	
1	+5v	1	+5v
2	0v O/P 5	2	O/P 5
3	X Direction	3	Y Direction
4	Z Direction	4	-
5	0v O/P 1	5	O/P 1
6	O/P 2	6	0v O/P 2
7	0v O/P 3	7	O/P 3
8	O/P 4	8	0v O/P 4
9	0v from drive	9	0v from drive
10	+24v from drive	10	+24v from drive
11	External 0v	11	External 0v
12	Input 1	12	Y Fault
13	X Fault	13	Em. Stop
14	RM Clock In	14	RM Clock Out
15	-	15	-
16	Input 3	16	Input 2
17	- Limit	17	+ Limit
18	-	18	-
19	-	19	Z Clock
20	-	20	X Clock
21	-	21	Y Clock
22	-	22	-
23	TTL Out	23	-
24		24	-
25	-	25	-
26	TXD	26	RXD
27	DSR	27	CTS
28	DTR	28	RTS
29	-	29	-
30	+12v	30	+12v
31	-12v	31	-12v
32	0v	32	0v

Motherboard connector identification.

PL1 14-way connector for the auxiliary inputs and outputs
PL2 25-way D-connector for the RS232C port connection
PL3 8-way connector for emergency stop, limit and external +24v supply connections
PL4 X drive connector
PL5 Y drive connector
PL6 Z drive connector
PL7 expander connection (for future use)

Connector PL1.

Terminal	Function	Circuit type (see page 37)
1	Output 5	D
2	0v for output 5	
3	Output 1	D
4	0v for output 1	
5	Output 2	D
6	0v for output 2	
7	0v for output 3	
8	Output 3	D
9	Output 4	D
10	0v for output 4	
11	External 0v connection	
12	Input 1	A
13	Input 3	A
14	Input 2	A

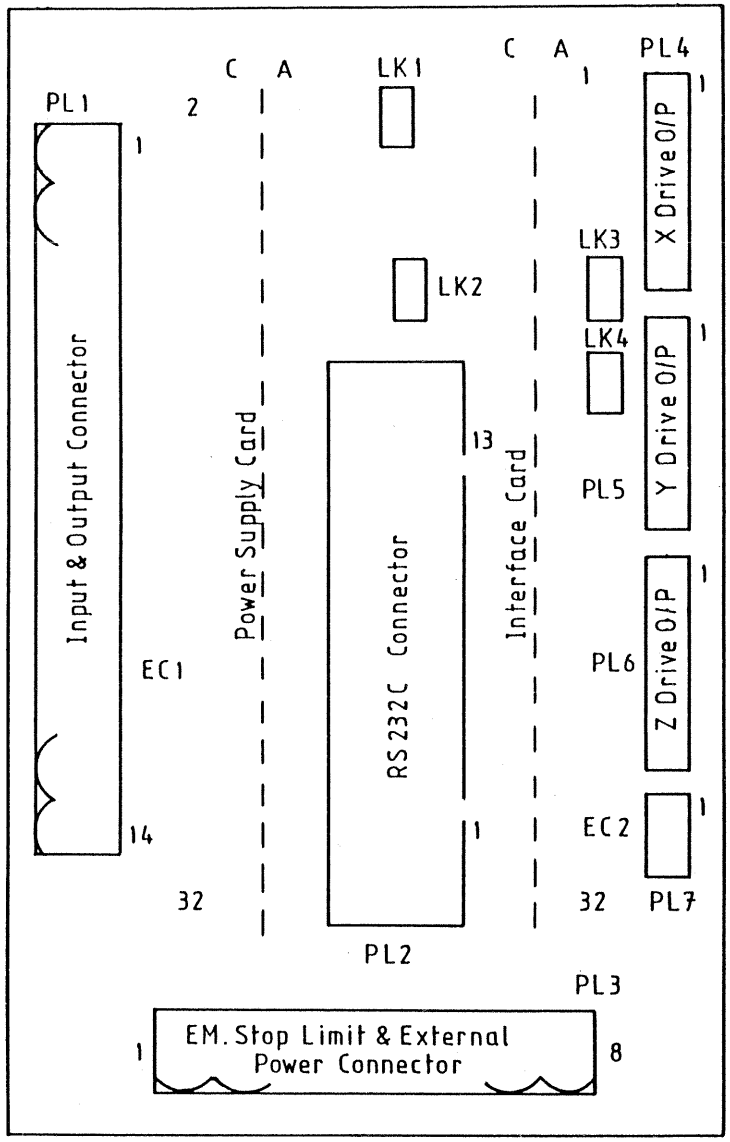
Connector PL2.

Pin no.	Function
1	Screen
2	TXD (transmitted data)
3	RXD (received data)
4	RTS (request to send)
5	CTS (clear to send)
6	DSR (data set ready)
7	Signal 0v
20	DTR (data terminal ready)

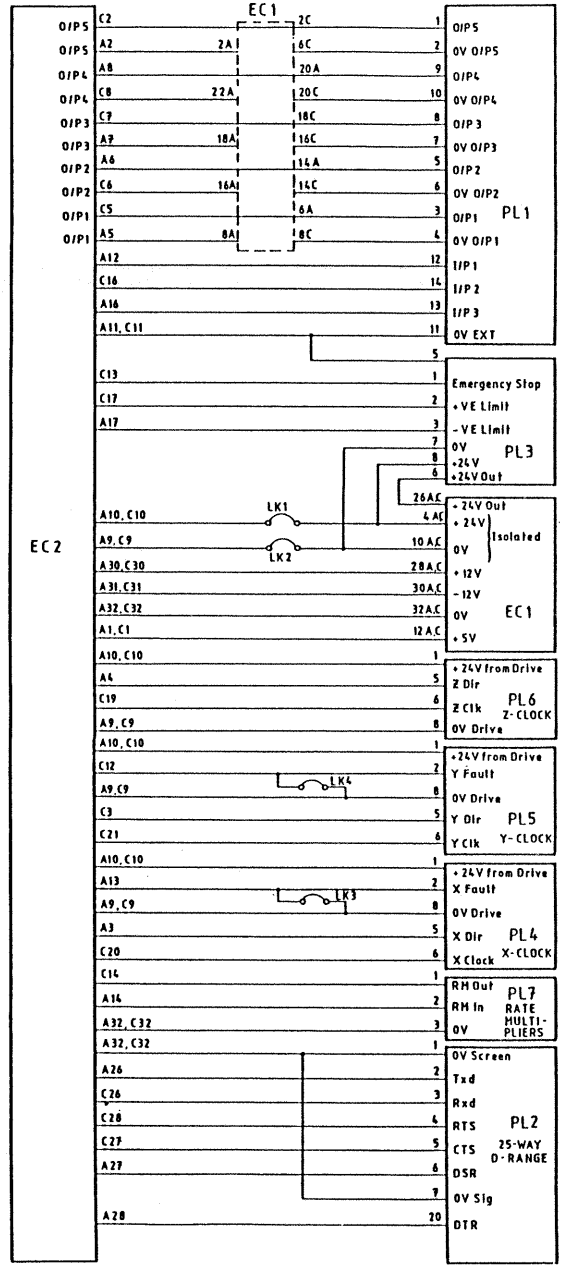
Connector PL3.

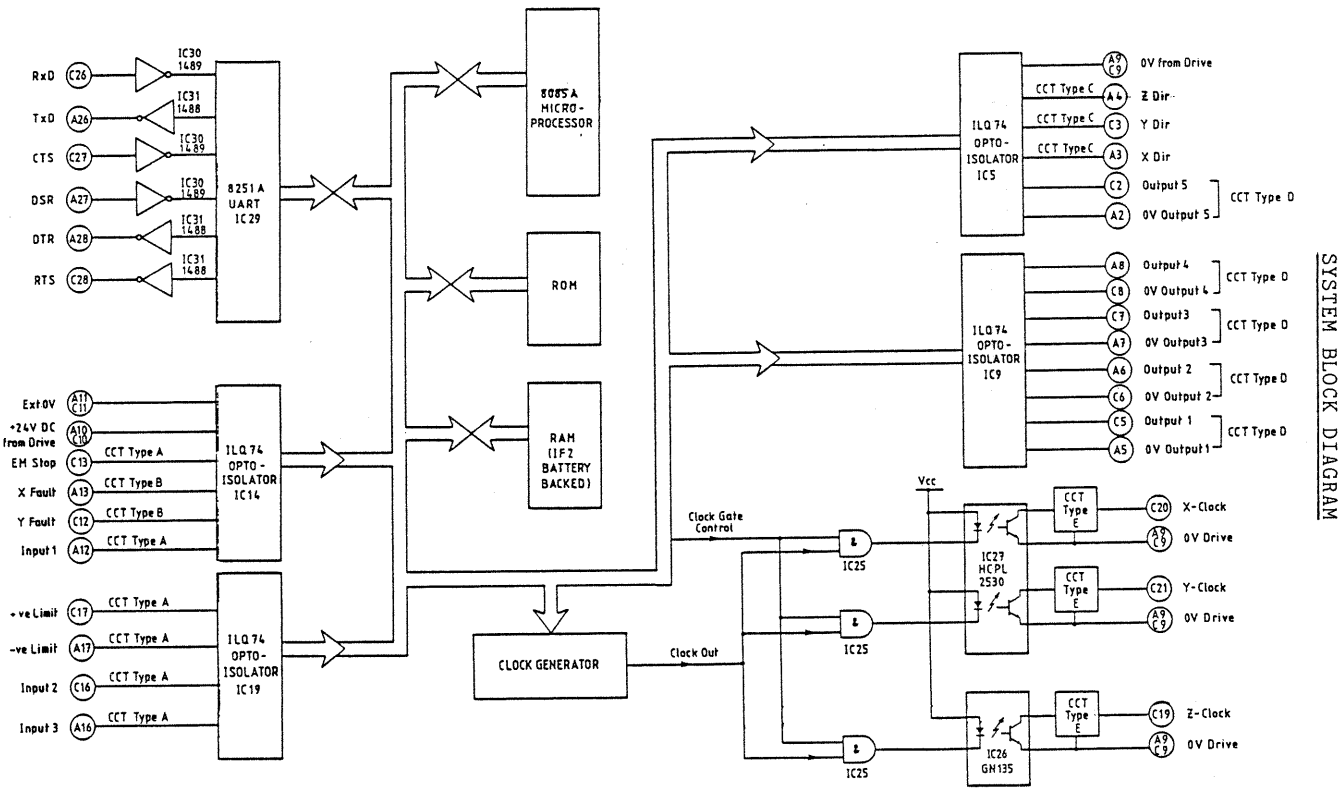
Terminal	Function	Circuit type (see page 34)
1	Emergency stop	A
2	Positive limit	A
3	Negative limit	A
4	No connection	
5	External 0v	
6	+24v out	
7	Auxiliary 0v in	
8	Auxiliary +24v in	

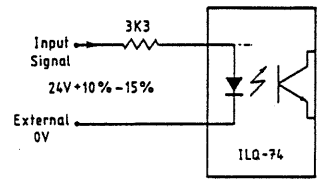
MOTHERBOARD LAYOUT



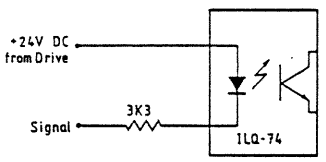
MOTHERBOARD LOGIC DIAGRAM



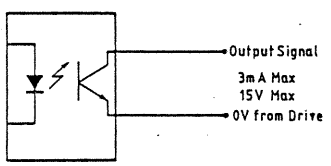

RxD
TxD
CTS
DSR
DTR
RTS
IC30 1489
IC31 1488
8251A UART IC29
8085A MICRO-PROCESSOR
ROM
RAM (IF2 BATTERY BACKED)
Ext 0V
+24V DC from Drive
EM Stop
X Fault
Y Fault
Input 1
ILQ74 OPTO-ISOLATOR IC14
CCT Type A
CCT Type B
+ve Limit
-ve Limit
Input 2
Input 3
ILQ74 OPTO-ISOLATOR IC19
CLOCK GENERATOR
Clock Gate Control
Clock Out
IC25
Vcc
IC27 HCPL 2530
IC26 GH135
CCT Type E
X-Clock
Y-Clock
Z-Clock
0V Drive
ILQ74 OPTO-ISOLATOR IC5
CCT Type C
0V from Drive
Z Dir
Y Dir
X Dir
Output 5
0V Output 5
CCT Type D
ILQ74 OPTO-ISOLATOR IC9
Output 4
0V Output 4
Output 3
0V Output 3
Output 2
0V Output 2
Output 1
0V Output 1



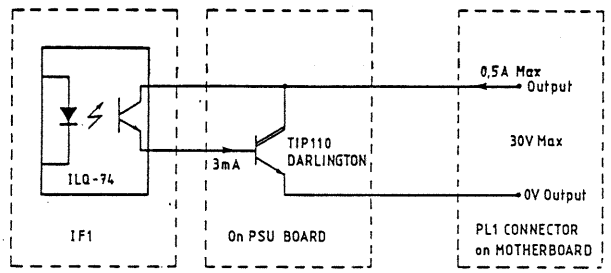
Circuit Type A



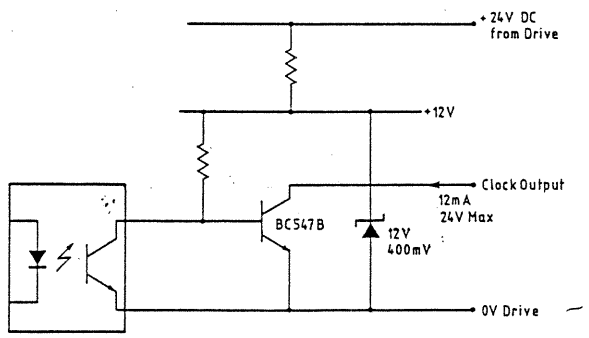
Circuit Type B



Circuit Type C

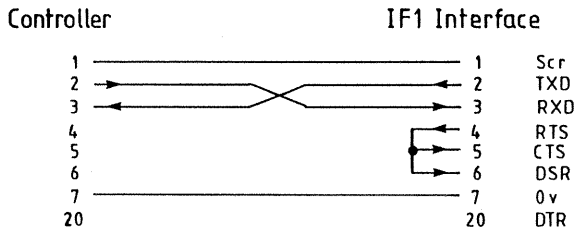


Circuit Type D



Circuit Type E

CONNECTION FOR THE NEC PC-8201A



SAMPLE PROGRAM 1 FOR THE NEC PC-8201A

This program is for 3-wire communication and uses echoback.

```
 60 '         OPEN NEC COMMUNICATIONS PORT
 70 ' 9600 baud,no parity,8 data bits,2 stop bits
 80 '
 90 OPEN"COM:8N82NN" FOR OUTPUT AS #1
100 OPEN"COM:8N82NN" FOR INPUT AS #2
120 '
130 '
140 '
150 '         INITIALISATION ROUTINE
160 '
170 PRINT#1,"(";
175 ON ERROR GOTO 5000
180 A$=INPUT$(2,#2)
190 B$=LEFT$(A$,1)
200 PRINTB$
210 IF B$="U" THEN 260 ELSE 220
220 PRINT "NO U" : GOTO 5000
230 END
240 '
250 '
259 '
260 '         SET DATA FORMAT
270 '
280 ' 8 data bits,2 stop bits,no parity,with echo,with LF's
290 '
295 ON ERROR GOTO 5020
300 PRINT#1,"U62"
320 GOTO 580
330 '
340 '
350 '
360 '         COMMANDS
370 '
380 '
390 A$="X1000@1000$"
400 GOSUB 1030
410 RETURN
420 '
430 A$="X-$"
440 GOSUB 1030
450 RETURN
460 '
470 A$="E"
480 GOSUB 1030
490 A$=INPUT$(2,#2)
495 S$=LEFT$(A$,1)
500 IF S$="C" THEN RETURN ELSE 470
510 '
520 A$="X$"
530 GOSUB 1030
540 RETURN
550 '
560 '
```

```
570 '
580 '       INDEX LOOP
590 '
600 PRINT"SPACE TO START TEST"
610 A$=INKEY$
620 IFA$=" " THEN 630 ELSE 610
630 GOSUB 390 : ' FIRST INDEX
640 FOR J=1 TO 10
650 GOSUB 470 : ' STATUS
660 GOSUB 430 : ' -INDEX
670 GOSUB 470 : ' STATUS
680 GOSUB 520 : ' +INDEX
690 NEXT J
700 END
710 '
720 '
730 '
800 '        TX CHARS AND CHECK ECHO
810 '
820 A=LEN(A$)
830 FOR I=1 TO A
840 B$=MID$(A$,I,1)
850 PRINT#1,B$;
860 E$=INPUT$(1,#2)
870 IF B$=E$ THEN 800 ELSE 930
880 NEXT I
890 RETURN
900 '
910 '
920 '
930 '        ECHO ERROR
940 '
950 PRINT "ECHO ERROR"
960 PRINT "CHAR No=";I
970 PRINT "TX-CHAR=";B$
980 PRINT "RX-CHAR=";E$
990 END
1000 '
1010 '
1020 '
1030 '        O/P STRING CHECKING ECHOS
1040 '
1050 GOSUB 800
1060 A$=CHR$(13)
1070 GOSUB 800
1080 RETURN
4995 '
4996 '
4997 '
4998 '  ERROR ROUTINES
4999 '
5000 PRINT "ERROR NUMBER" ; ERR
5010 CLOSE : GOTO 90
5020 PRINT "ERROR NUMBER" ; ERR
5030 END
```

SAMPLE PROGRAM 2 FOR THE NEC PC-8201A

This programm runs at 2400 baud without echoback.

```
60 ' OPEN NEC COMMUNICATIONS PORT
70 ' 2400 baud,no parity,8 data bits,2 stop bits
80 '
90  OPEN"COM:6N82NN" FOR OUTPUT AS #1
100 OPEN"COM:6N82NN" FOR INPUT  AS #2
110 '
120 '
130 '    INITIALISATION ROUTINE
140 '
150 PRINT#1,"(",
160 ON ERROR GOTO 5000
170 A$=INPUT$(2,#2)
180 B$=LEFT$(A$,1)
190 PRINT B$
200 IF B$="U" THEN 240 ELSE 210
210 PRINT "NO U" : GOTO 5000
220 '
230 '
240 ' Set data format
250 ' 8 data bits,2 stop bits,no parity,echo off,terminate with LF
260 '
270 ON ERROR GOTO 5020
280 PRINT#1,"V61"
290 '
300 '  Index Loop
310 '
320 PRINT "PRESS SPACE TO START TEST "
330 A$=INKEY$
340 IF A$=" " THEN 350 ELSE 330
350 GOSUB 470 : ' FIRST INDEX
360 FOR J=1 TO 10
370 GOSUB 550 : ' STATUS
380 GOSUB 510 : ' -INDEX
390 GOSUB 550 : ' STATUS
400 GOSUB 610 : ' +INDEX
410 NEXT J
420 END
430 '
440 '
450 '   COMMANDS
460 '
470 A$="X1000@1000$"
480 PRINT#1,A$
490 RETURN
500 '
510 A$="X-$"
520 PRINT#1,A$
530 RETURN
540 '
550 A$="E"
560 PRINT#1,A$
570 INPUT#2,S$
580 IF S$="E" THEN 550
590 IF S$="C" THEN RETURN ELSE PRINT "INTERFACE FAULT" : END
600 '
610 A$="X$"
620 PRINT#1,A$
630 RETURN
640 '
650 '
5000 PRINT "ERROR NUMBER" ; ERR
5010 CLOSE : GOTO 90
5020 PRINT "ERROR NUMBER" ; ERR
5030 CLOSE : PRINT "PROG ERROR" : END
```

SAMPLE PROGRAM 3 FOR THE NEC PC-8201A

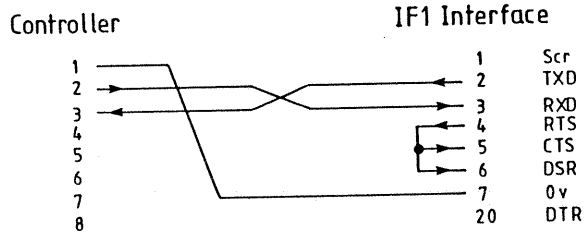
This program runs at 1200 baud without echoback and includes an example of a long sequence.

```
60 '    OPEN NEC COMMUNICATIONS PORT
70 ' 1200 baud,no parity,8 data bits,2 stop bits
80 '
90 OPEN"COM:5N82NN" FOR OUTPUT AS #1
100 OPEN"COM:5N82NN" FOR INPUT AS #2
110 '
120 '
130 '
150 '   INITIALISATION ROUTINE
160 '
170 PRINT#1,"(";
175 ON ERROR GOTO 5000
180 A$=INPUT$(2,#2)
190 B$=LEFT$(A$,1)
200 PRINTB$
210 IF B$="U" THEN 260 ELSE 220
220 PRINT"NO U" : GOTO 5000
240 '
250 '
259 '
260 '   SET DATA FORMAT
280 '   8 data bits,2 stop bits,no parity,echo off,with LF's
290 '
295 ON ERROR GOTO 5020
300 PRINT#1,"V61"
315 '
320 '       SEND COMPLETE SEQUENCE
325 '
330 PRINT  "PRESS SPACE TO START SEQUENCE"
335 A$=INKEY$
340 IF A$=" " THEN 345 ELSE 335
345 GOSUB 390 : ' SEND SEQUENCE
350 PRINT"SEQUENCE SENT" : END
360 '   COMMANDS
370 '
380 '
390 PRINT#1,":R12345:H1:X1000@10000S1:X-S2:YS3:Y-S4:ZS5:Z-R5:";
395 PRINT#1,"XYZR4:X-YZR3:X-Y-Z-R2:X-Y-Z-R1:=0,1:$"
400 RETURN
2000 '
5000 PRINT "ERROR NUMBER" ; ERR
5010 CLOSE : GOTO 90
5020 PRINT "ERROR NUMBER" ; ERR
5030 CLOSE : PRINT "PROG ERROR" : END
```

CONNECTION FOR THE EPSON HX20

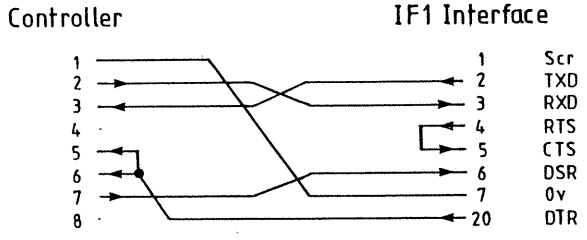
On the Epson machine the RS232 connection is via an 8-way DIN socket.

(a) 3-wire, with or without echo-back.



When using this connection scheme, note that the "control line active" mode should be set to "F" in the "OPEN" statement as shown in the listing. Since the Epson terminates on <CR> but sends <CR><LF>, either "U" or "V" can be specified in line 300 of the non-echoback program. Using "U" will speed up the communication.

(b) 5-wire with buffer control.



The connections shown above must be used with the "control line active" mode set to "3" in the "OPEN" statement as shown in the listing. This will ensure that the buffer control is detected on the CTS input of the Epson. If buffer control is not required, remove the DTR connection to pins 4 and 5 on the Epson together.

SAMPLE PROGRAM 4 FOR THE EPSON HX20

This program is for 3-wire communication without echoback.

```
10 '    USING THE EPSON WITH IF1 ( 3 WIRE )
20 '      ISS 4 - EPSON PROG WRITTEN 01/05/86
50 '
60  ' ******  OPEN EPSON COMMUNICATIONS PORT  ******
70  ' 2400 baud , no parity , 8 data bits,2 stop bits
80  OPEN"O",#1,"COM0:(58N2F)"
90  OPEN"I",#2,"COM0:(58N2F)"
130 ' ******  INITIALISATION ROUTINE  ******
150 PRINT#1,"(";
160 ON ERROR GOTO 5000
170 INPUT#2,A$
180 PRINTA$
190 IF A$="U" THEN 250 ELSE 200
200 PRINT "NO U" : GOTO 5000
210 STOP
250 ' ******  SET DATA FORMAT  ******
270 ON ERROR GOTO 5020
280 ' 8 data bits , 2 stop bits , no parity , no echo
290 '
300 PRINT #1,"U61";
305 PRINT#1,CHR$(13);
330 ' ******  INDEX LOOP  ******
350 PRINT "PRESS SPACE TO START TEST"
360 A$=INKEY$
370 IF A$=" " THEN 380 ELSE 360
380 GOSUB 510 : ' FIRST INDEX
390 FOR J=1 TO 10
400 GOSUB 590 : ' STATUS
410 GOSUB 550 : ' -INDEX
420 GOSUB 590 : ' STATUS
430 GOSUB 650 : ' +INDEX
440 NEXT J
450 GOSUB 590 : ' STATUS
460 STOP
470 '
480 ' ******  COMMANDS  ******
510 A$= "X1000@1000$"
520 PRINT#1,A$
530 RETURN
540 '
550 A$="X-$"
560 PRINT#1,A$
570 RETURN
580 '
590 A$="E"
600 PRINT#1,A$
610 INPUT#2,S$
620 IF S$="E" THEN 590
630 IF S$="C" THEN RETURN ELSE PRINT "INTERFACE FAULT" : STOP
640 '
650 A$="X$"
660 PRINT#1,A$
670 RETURN
700 '
720 ' ******  ERROR ROUTINES  ******
5000 PRINT"ERROR NUMBER" ; ERR
5010 CLOSE : GOTO 80
5020 PRINT"ERROR NUMBER" ; ERR
5030 CLOSE : PRINT "PROG ERROR" : STOP
```

SAMPLE PROGRAM 5 FOR THE EPSON HX20

This program is for 3-wire communication using echoback.

```
 70 '      OPEN EPSON COMMUNICATIONS
 80 '  4800 baud,no parity,8 data bits,2 stop bits
 90 '
100 OPEN"O",#1,"COM0:(68N2F)"
110 OPEN"I",#2,"COM0:(68N2F)"
120 '
130 '
140 '     INITIALISATION  ROUTINE
150 '
160 PRINT#1,"(";
165 ON ERROR GOTO 5000
170 INPUT#2,A$
180 B$=LEFT$(A$,1)
190 PRINT B$
200 IF B$="U" THEN 260 ELSE 210
210 PRINT "NO U" : GOTO 5000
230 '
240 '
250 '
260 '      SET DATA FORMAT
270 '
280 '  8 data bits,2 stop bits,no parity,echo on
290 '
295 ON ERROR GOTO 5020
300 PRINT#1,"U62";
310 PRINT#1,CHR$(13);
320 GOTO 580
330 '
340 '
350 '
360 '       COMMANDS
370 '
380 '
390 A$="X1000@1000$"
400 GOSUB 1030
410 RETURN
420 '
430 A$="X-$"
440 GOSUB 1030
450 RETURN
460 '
470 A$="E"
480 GOSUB 1030
485 A$=INPUT$(2,#2)
486 S$=LEFT$(A$,1)
500 IF S$="C" THEN RETURN ELSE 470
510 '
520 A$="X$"
530 GOSUB 1030
540 RETURN
550 '
560 '
```

```
570 '
580 '   INDEX LOOP
590 '
600 PRINT "PRESS SPACE TO START TEST"
610 A$=INKEY$
620 IF A$=" " THEN 630 ELSE 610
630 GOSUB 390 : ' FIRST INDEX
640 FOR J=1 TO 10
650 GOSUB 470 : ' STATUS
660 GOSUB 430 : ' -INDEX
670 GOSUB 470 : ' STATUS
680 GOSUB 520 : ' +INDEX
690 NEXT J
695 '
700 STOP
710 '
720 '
730 '
800 '  TX CHARS AND CHECK ECHO
810 '
820 A=LEN(A$)
830 FOR I=1 TO A
840 B$=MID$(A$,I,1)
850 PRINT#1,B$;
860 E$=INPUT$(1,#2)
870 IF B$=E$ THEN 880 ELSE 930
880 NEXT I
890 RETURN
900 '
910 '
920 '
930 '    ECHO  ERROR
940 '
950 PRINT "ECHO ERROR"
960 PRINT "CHAR No=";I
970 PRINT "TX-CHAR=";B$
980 PRINT "RX-CHAR=";E$
990 STOP
1000 '
1010 '
1020 '
1030 '     O/P STRING CHECKING ECHOS
1040 '
1050 GOSUB 800
1060 A$=CHR$(13)
1070 GOSUB 800
1080 RETURN
4995 '
4996 '
4997 '
4998 '  ERROR ROUTINES
4999 '
5000 PRINT "ERROR NUMBER";ERR
5010 CLOSE : GOTO 90
5020 PRINT"ERROR NUMBER";ERR
5030 STOP
```

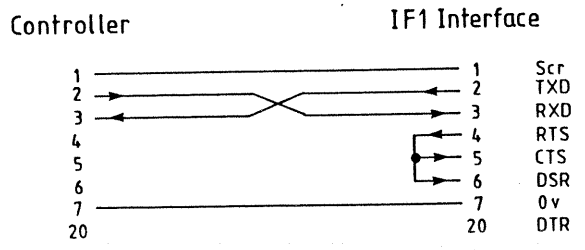
SAMPLE PROGRAM 6 FOR THE EPSON HX20

This program is for 5-wire communication with buffer control.

```
10 'Program to install a battery backed up sequence on the IF2
20 '
30 '    OPEN EPSON COMMUNICATIONS PORT
40 '    2400 baud,no parity,8 data bits,2 stop bits
60 '
70 OPEN"O",#1,"COM0:(58N23)"
80 OPEN"I",#2,"COM0:(58N23)"
90 '
100 PRINT "SWITCH ON IF2"
110 INPUT " PRESS CR TO CONTINUE",A$
120 '
130 '   INITIALISATION ROUTINE
150 PRINT#1,"(";
160 INPUT#2,A$ : REM INPUT AND DISCARD
170 PRINT#1,"*": REM CLEAR ALL CONTENTS OF BATTERY RAM
180 ON ERROR GOTO 550
190 INPUT#2,A$
200 PRINTA$
210 IFA$="U" THEN 270 ELSE 220
220 PRINT "NO U" :GOTO 550
230 STOP
240 '
270 '   SET DATA FORMAT
290 ON ERROR GOTO 570
300 ' 8 data bits,2 stop bits,no parity,echo off.
310 '
320 PRINT#1,"V61"
330 '
340 '   SEND COMPLETE SEQUENCE
360 PRINT#1,"E"
370 INPUT#2,S$
380 IF S$ = "E" THEN 360
390 GOSUB 420
400 PRINT"SEQUENCE SENT":STOP
410 '
420 '   COMMANDS
440 PRINT#1,":R12345:H1:X1000@1000S1:X-S2:YS3:Y-S4:ZS5:Z-R5:";
450 PRINT#1,"XYZR4:X-YR3:X-Y-ZR2:X-Y-ZR2:X-Y-Z-R1:+0,1:$"
460 PRINT#1,"[" : REM LOCK THE SYSTEM
470 RETURN
480 '
530 '   ERROR ROUTINES
550 PRINT "ERROR NUMBER" ; ERR
560 CLOSE : GOTO 80
570 PRINT "ERROR NUMBER" ; ERR
580 CLOSE :PRINT"PROGRAM ERROR" :STOP
```

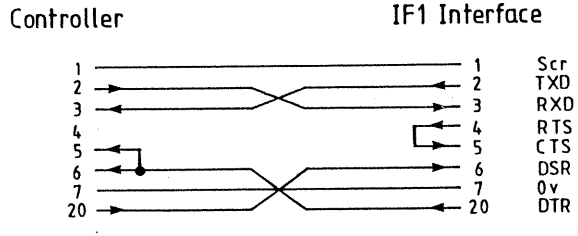
CONNECTIONS FOR THE IBM PC

(a) 3-wire with or without echo-back.



This connection mode assumes that all "control line active" checking is suppressed as specified in the "OPEN" statement (see sample listing line 90).

(b) 5-wire with buffer control.



This connection mode assumes that "CTS line active" is specified in the "OPEN" statement (see sample listing line 90). The timeout on CTS can be varied, thew minimum value can be found experimentally by sending a long string of characters to the IF1. The sample listing gives a suggested timeout period.

SAMPLE PROGRAM 7 FOR THE IBM PC

This program is for 3-wire communication without echoback.

```
20 ' P.K.S.- DIGIPLAN LTD. 08 Jan 1987
30 ' Power up the IF1 before running program
40 '
50 '
60 '       OPEN COMMUNICATIONS PORT
70 ' 2400 Baud,No parity,8 Data bits,2 Stop bits
80  COM(1) ON
90  OPEN"COM1:2400,N,8,2,RS,CS,DS,CD" AS #1
100 ON ERROR GOTO 5000 : ' This is to empty the IBM input buffer
110 '      INITIALISATION ROUTINE
120 CLS:PRINT "PLEASE WAIT"
130 ON ERROR GOTO 4000
140 PRINT#1,"(";
150 ON ERROR GOTO 5000
160 INPUT #1,B$
170 PRINT B$
180 IF B$="U" THEN 220 ELSE 190
190 PRINT" NO U":GOTO 5000
220 '      SET DATA FORMAT
240 ' 8 Data Bits,2 Stop Bits,No Parity,No Echoback.
260 PRINT #1,"U61"
270 FOR W = 1 TO 10000:NEXT:' Wait for initial ramp calculation.
280 '
290 '      SOME EXAMPLE MOVES
300 '
310 PRINT #1,"X1000@1000^10$"
320 GOSUB 1000: ' Check Motion Status.
330 PRINT #1,"Y$"
340 GOSUB 1000
350 PRINT #1,"X-$"
360 GOSUB 1000
370 PRINT #1,"Y-$"
380 GOSUB 1000
390 '
400 '
410 CLOSE: ' CLOSE COMMS PORT BEFORE ENDING PROGRAM
420 '
430 END :  END OF PROGRAM
440 ' Note. Before running program again reset IF1 by powering down!!!!
450  '
1000 '     MOTION AND FAULT CHECK ROUTINE
1010 PRINT #1,"E" : ' Check Busy Status
1020 INPUT #1,S$              : ' Input Status Byte
1030 IF S$ = "E" THEN 1010    : ' Loop until not Busy
1040 IF S$ = "C" THEN RETURN : ' Motion Stopped return for next move
1050 IF S$ = "F" THEN PRINT "INTERFACE FAULT": GOTO 5000
1060 '      END OF MOTION AND FAULT CHECK ROUTINE
4000 '      IGNORE INPUT BUFFER NULL CHARACTER
4010 IF ERR = 57 THEN 140 ELSE GOTO 5000
5000 '      ERROR CHECKING ROUTINE
5010 PRINT "ERROR NUMBER";ERR
5020 CLOSE :END
```

SAMPLE PROGRAM 8 FOR THE IBM PC

This program is for 3-wire communication using echoback.

```
10 ' COMM IBM-PC TO IF1 ISS 4 - WITH ECHO-BACK (3 WIRE).
20 ' IBM PROG ISS 1 15/05/86
30 ' THIS ASSUMES IF1 IS ALREADY POWERED UP
40 ' ************  OPEN IBM COMMUNICATIONS PORT  ******************
50 ' 9600 baud,no parity,8 data bits,2 stop bits
60 COM(1) ON
70 OPEN"COM1:9600,N,8,2,RS,CS,DS,CD" AS #1
80 ' *************** INITIALISATION ROUTINE  ************************
90 PRINT#1,"(";  :'          SEND BAUD DETECT COMMAND
100 ON ERROR GOTO 580
110 INPUT#1,B$:PRINT B$:'  READ AND PRINT BAUD SET CHARACTER
120 IF B$="U" THEN 140 ELSE 130
130 PRINT" NO U" : GOTO 580 : '  IF "U" IS NOT RECEIVED RETRY.
140 ' ************  SET DATA FORMAT  *****************************
150 ' 8 data bits,2 stop bits,no parity,echo on
160 PRINT#1,"U62": '         SEND DATA FORMAT COMMAND
170 GOTO 300 :'              SEND COMMANDS TO INTERFACE.
180 ' ************  COMMANDS  ************************************
190 A$="X1000@1000$" :'      SEND POSITIVE INDEX
200 GOSUB 530:RETURN:'       CHECK ECHOBACK
210 A$="X-$"    :'           SEND NEGATIVE INDEX
220 GOSUB 530:RETURN:'       CHECK ECHOBACK
230 A$="E"      :'           SEND STATUS COMMAND
240 GOSUB 530 :'             CHECK ECHOBACK
250 INPUT#1,S$ :PRINT S$
260 IF S$="E" THEN 230
270 IF S$="C" THEN RETURN ELSE PRINT "INTERFACE FAULT" : END
280 A$="X$"
290 GOSUB 530:RETURN
300 ' *************  INDEX LOOP  *********************************
310 PRINT "PRESS SPACE TO START TEST"
320 A$=INKEY$: IF A$<> " " THEN 320
330 GOSUB 190 : '        FIRST INDEX
340 FOR J = 1 TO 10
350 GOSUB 230 : '        STATUS
360 GOSUB 210 : '        -INDEX
370 GOSUB 230 : '        STATUS
380 GOSUB 280 : '        +INDEX
390 NEXT J
400 END
410 ' *************  TX CHARS AND CHECK ECHO  ***********************
420 A=LEN(A$)
430 FOR I=1 TO A
440 B$=MID$(A$,I,1)
450 PRINT#1,B$;
460 E$=INPUT$(1,#1)
470 IF B$=E$ THEN 480 ELSE 500
480 NEXT I
490 RETURN
500 ' ***************  ECHO ERROR  *********************************
510 PRINT "ECHO ERROR","CHAR NO.";I,"TX-CHAR=";B$,"RX-CHAR=";E$
520 END
530 ' ************  O/P STRING CHECKING ECHOS  *********************
540 GOSUB 410
550 A$=CHR$(13)
560 GOSUB 410
570 RETURN
580 PRINT "ERROR NUMBER" ; ERR
590 CLOSE : GOTO 60
```

SAMPLE PROGRAM 9 FOR THE IBM PC

This program is for 5-wire communication with buffer control.

```
10 ' COMM IBM-PC TO IF1 ISS 4 (5-WIRE WITH BUFFER CONTROL)
20 ' IBM PROG ISS 1 15/05/86
30 ' THIS ASSUMES IF1 IS ALREADY POWERED UP
40 '
50 '
60 '        OPEN IBM COMMUNICATIONS PORT
70 ' 2400 baud,no parity,8 data bits,2 stop bits
80   COM(1) ON
90   OPEN"COM1:2400,N,8,2,CS350,DS" AS #1
100 '
110 '
120 '
130 '
140 '       INITIALISATION ROUTINE
150 '
160 PRINT#1,"(";
165 ON ERROR GOTO 5000
170 INPUT#1,B$
180 '
190 PRINT B$
200 IF B$="U" THEN 260 ELSE 210
210 PRINT" NO U" : GOTO 5000
220 '
227 '
228 '
230 '
240 '
250 '
260 '            SET DATA FORMAT
270 '
280 ' 8 data bits,2 stop bits,no parity,echo off
290 '
300 PRINT#1,"U61"
310 '
320 '            SEND COMPLETE SEQUENCE
325 '
330 PRINT "PRESS SPACE TO START SEQUENCE"
335 A$=INKEY$
340 IF A$= " " THEN 345 ELSE 335
345 GOSUB 390 : ' SEND SEQUENCE
350 END
360 '            COMMANDS
370 '
380 '
390 PRINT#1,":R12345:H1:X1000@10000S1:X-S2:YS3:Y-S4:ZS5:Z-R5:";
395 PRINT#1,"XYZR4:X-YZR3:X-Y-ZR2:X-Y-Z-R1:=0,1:$"
400 RETURN
570 '
5000 PRINT "ERROR NUMBER" ; ERR
5010 CLOSE : GOTO 80
```

SAMPLE PROGRAM 10 FOR USE WITH THE TANDY 100/102 PORTABLES

This program 3-wire communication at 9600 baud using echoback.

```
10 ' SAMPLE PROGRAM FOR THE TANDY 100/102 PORTABLES
20 ' FOR USE WITH THE P.K.S.DIGIPLAN IF1 ISSUE 5
30 ' JAN 12TH 1987
60 ' OPEN TANDY COMMS PORT
70 ' 9600 baud,no parity,8 data bits,2 stop bits
80 '
90 OPEN"COM:88N2D" FOR OUTPUTAS #1
100 OPEN"COM:88N2D" FOR INPUT AS #2
120 '
130 '
140 '
150 ' INITIALISATION ROUTINE
160 '
170 PRINT#1,"(";
180 ON ERROR GOTO 5000
190 A$ = INPUT$(2,#2)
200 B$ = LEFT$(A$,1)
210 IF B$ = "U" THEN 260 ELSE 230
220 IF B$ = "U" THEN 260 ELSE 230
230 PRINT "NO U" :GOTO 5000
240 END
250 '
260 ' SET DATA FORMAT
270 ' 8 data bits,2 stop bits, no parity,with echo,with LF termination
280 '
290 ON ERROR GOTO 5020
300 PRINT #1,"U62"
310 GOTO 580
320 '
330 '
340 '
350 '
360 ' COMMANDS
370 '
380 '
390 A$ = "X1000@1000^40$"
400 GOSUB 1030
410 RETURN
420 '
430 A$="X-$"
440 GOSUB 1030
450 RETURN
460 '
470 A$="E"
480 GOSUB 1030
490 A$=INPUT$(3,#2)
500 S$ = LEFT$(A$,1)
510 IF S$ ="C" THEN RETURN ELSE 470
520 A$ = "X$"
530 GOSUB 1030
540 RETURN
550 '
560 '
570 '
580 ' INDEX LOOP
590 '
```

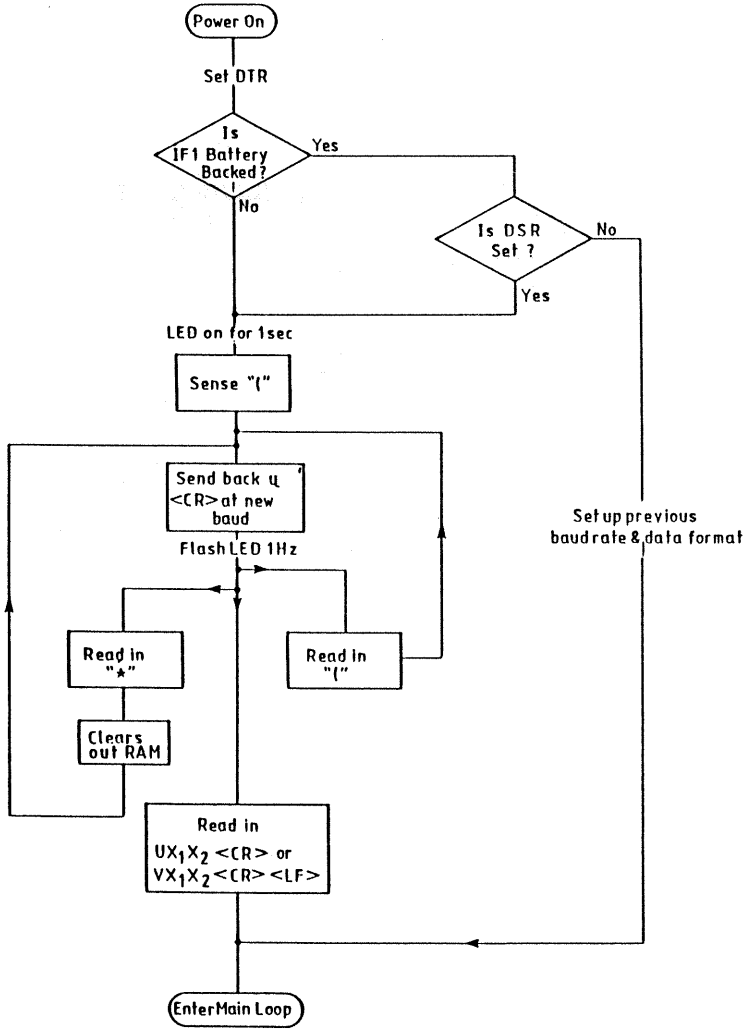
```
600 PRINT "PRESS SPACE TO START TEST
610 A$=INKEY$
620 IF A$= " " THEN 630 ELSE 610
630 GOSUB 390:' FIRST INDEX
640 FOR J=1 TO 10
650 GOSUB 470:' STATUS
660 GOSUB 430:' -INDEX
670 GOSUB 470:' STATUS
680 GOSUB 520:' +INDEX
690 NEXT J
700 A$ = "X5000/Y500@6500$"
710 GOSUB 1030
720 GOSUB 470
730 A$ = "X-/Y-$"
740 GOSUB 1030
750 GOSUB 470
760 END
800 ' TRANSMIT CHARS. AND CHECK ECHO
810 '
820 A=LEN(A$)
830 FOR I = 1 TO A
840 B$=MID$(A$,I,1)
850 PRINT #1,B$;
860 E$=INPUT$(1,#2)
870 IF B$=E$ THEN 880 ELSE 930
880 NEXT I
890 RETURN
900 '
910 '
920 '
930 ' ECHO ERROR
940 '
950 PRINT "ECHO ERROR"
960 PRINT "CHAR. NO.";I
970 PRINT "TX. CHAR.";B$
980 PRINT "RX. CHAR.";E$
990 END
1000 '
1010 '
1020 '
1030 ' O/P STRING CHECKING ECHO'S
1040 '
1050 GOSUB 800
1060 A$=CHR$(10)
1070 GOSUB 800
1080 RETURN
5000 ' ERROR ROUTINES
5010 PRINT "ERROR NO.";ERR
5015 CLOSE:GOTO90
5020 PRINT " ERROR NO.";ERR
5030 END
```

SAMPLE PROGRAM 11 FOR THE TANDY 100/102 PORTABLES

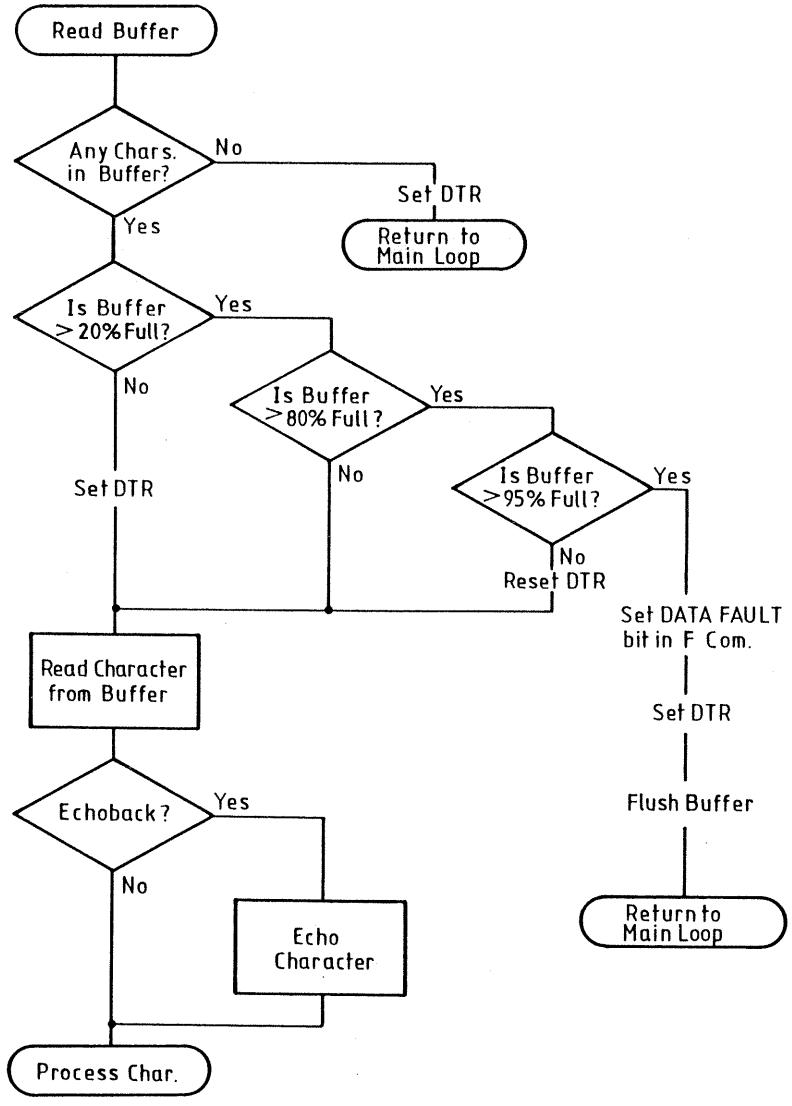
This program uses 3-wire communication at 2400 baud without echoback.

```
20 ' P.K.S.- DIGIPLAN LTD. 08 Jan 1987
30 ' Power up the IF1 before running program
40 '
50 '
60 '       OPEN COMMUNICATIONS PORT
70 ' 2400 Baud,No parity,8 Data bits,2 Stop bits
80  COM ON
90  OPEN"COM:68N2D" FOR OUTPUT AS #1
100 OPEN"COM:68N2D" FOR INPUT AS #2
110 '      INITIALISATION ROUTINE
120 CLS:PRINT "PLEASE WAIT"
130 ON ERROR GOTO 4000
140 PRINT#1,"(";
150 ON ERROR GOTO 5000
160 INPUT #1,B$
170 PRINT B$
180 IF B$="U" THEN 220 ELSE 190
190 PRINT" NO U":GOTO 5000
220 '      SET DATA FORMAT
240 ' 8 Data Bits,2 Stop Bits,No Parity,No Echoback.
260 PRINT #1,"V61"
270 FOR W = 1 TO 10000:NEXT:' Wait for data configuration
280 '
290 '      SOME EXAMPLE MOVES
300 '
310 PRINT #1,"X1000@1000^10$"
320 GOSUB 1000: ' Check Motion Status.
330 PRINT #1,"Y$"
340 GOSUB 1000
350 PRINT #1,"X-$"
360 GOSUB 1000
370 PRINT #1,"Y-$"
380 GOSUB 1000
385 PRINT #1,"X5000/Y500@6500$"
386 GOSUB1000
387 PRINT #1,"X-/Y-$"
388 GOSUB1000
390 '
400 '
410 CLOSE: ' CLOSE COMMS PORT BEFORE ENDING PROGRAM
420 '
430 END :  END OF PROGRAM
440 ' Note. Before running program again reset IF1 by powering
down!!!!
450  '
1000 '     MOTION AND FAULT CHECK ROUTINE
1010 PRINT #1,"E" : ' Check Busy Status
1020 INPUT #1,S$              : ' Input Status Byte
1030 IF S$ = "E" THEN 1010    : ' Loop until not Busy
1040 IF S$ = "C" THEN RETURN : ' Motion Stopped return for next move
1050 IF S$ = "F" THEN PRINT "INTERFACE FAULT": GOTO 5000
1060 '      END OF MOTION AND FAULT CHECK ROUTINE
5000 '      ERROR CHECKING ROUTINE
5010 PRINT "ERROR NUMBER";ERR
5020 CLOSE :END
```

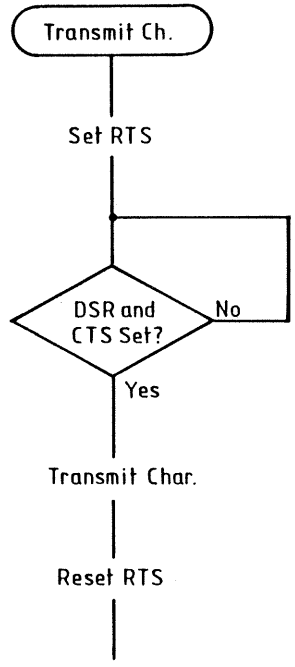
INITIALISATION FLOWCHART



INPUT BUFFER STROBE FLOWCHART



FLOWCHART FOR TRANSMITTING DATA FROM IF1



INDEX