

ITC232-A USER GUIDE

TABLE OF CONTENTS

A. INTRODUCTION	1
B. GENERAL CONVENTIONS	2
C. PIN ASSIGNMENTS	2
D. PIN OUT DESCRIPTION	2
E. MAIN FUNCTIONS OF THE ITC232-A	
1. DIGITAL INPUT/OUTPUT	4
2. SERIAL PERIPHERAL INTERFACE (SPI)	4
3. PULSE WIDTH MODULATION (PWM)	4
4. MEASURING A RESISTOR OR CAPACITOR	5
5. STEPPER MOTOR CONTROLLER	6
6. INTERRUPTS	7
7. REMOTE OPERATION VIA A PHONE LINK	7
F. COMMANDS	8
LIST OF COMMANDS (in alphabetical order)	9
G. ERROR LIST	15
H. ELECTRICAL SPECIFICATIONS	15
APPENDIX A: Stepper motor basics	
APPENDIX B: QBasic[®] set-up of serial port and the ITC232-A.	
APPENDIX C: Zero, Full Scale and Half Scale Tests in QBasic[®]	

RMV ELECTRONICS INC.
#230-2250 Boundary Road
Burnaby, B.C., V7M 3Z3 Canada
Tel: (604) 299-5173
Fax: (604) 299-5174
Web site: <http://rmv.com>
Email: customer@rmv.com

PLEASE READ THIS CAREFULLY:

RMV ELECTRONICS INC. does not assume any liability arising from the application and/or use of the product/s described herein, nor does it convey any license rights. RMV ELECTRONICS INC. products are not authorized for use as components in medical, life support or military devices without written permission from RMV ELECTRONICS INC.

The material enclosed in this package may not be copied, reproduced or imitated in any way, shape or form without the written consent of RMV ELECTRONICS INC. This limitation also applies to the firmware that the Integrated Circuit in this package might contain.

ITC232-A USER GUIDE

SECTION I: THE ITC232-A I.C.

A. INTRODUCTION

The ITC232-A is an integrated circuit containing a general purpose serial/parallel interface as well as a number of embedded functions which find application in data acquisition and control systems. It provides easy access, from a terminal or computer serial port (EIA RS-232C), to 32 input/output lines arranged in 5 ports which can be read or written with extremely simple ASCII commands. This allows control from within a custom written program as well as from any commercial communication software package (Terminal for Windows®, Procomm®, Telix®, MAC240®, etc). The possibility of operating the ITC232-A from both a dumb terminal program or a custom program is very convenient for system debugging.

The ITC232-A is hardware independent; it will work with any terminal or computer with an RS-232 serial port. The link requires 3 wires and operates at any standard speed between 300 and 115200 Bauds. The only external components required by the ITC232-A are a voltage driver to handle the RS-232 voltages and a 3.6864 MHz crystal.

In addition to serial/parallel/serial interfacing, the ITC232-A displays several other functions. These are:

- (1) To send or receive data in either Decimal, Hexadecimal or Binary format.
- (2) High and Low multiple interrupt recognition via 2 pins; IRQH and IRQL.
- (3) 10-10000 Hz, 0-100% duty cycle Pulse Width Modulation.
- (4) Hardware and software selectable Baud rate over the 300-115200 range.
- (5) Four channels for reading relative resistance or capacitance.
- (6) Three stepper motor ports with all the necessary logic to instrument monophasic, biphasic and half-step stepping modes. Emphasis has been placed in making the stepper interface easy to use.
- (7) A 1 keystroke "Again" command to repeat the previous command.
- (8) On-screen help (a summary of all commands is sent to the terminal).
- (9) A configuration report feature which sends the active configuration of any parallel port, the PWM or the stepper motor ports back to the terminal.

- (10) Remote operation via a phone link using a modem chip in place of the RS-232 voltage driver.

The flexibility, ease of use and multiple functions of the ITC232-A, make it ideal for any circuitry requiring computer control and/or data acquisition. Robotics, environmental control and instrumentation are just a few examples of fields where the ITC232-A finds application.

B. GENERAL CONVENTIONS

Throughout this manual High is used as synonymous to binary 1 and Low as synonymous to binary 0. The term "terminal" includes computers and the term "peripheral" is used as any circuit containing the ITC232-A.

C. PIN ASSIGNMENTS

ITC232-A Pin Assignments

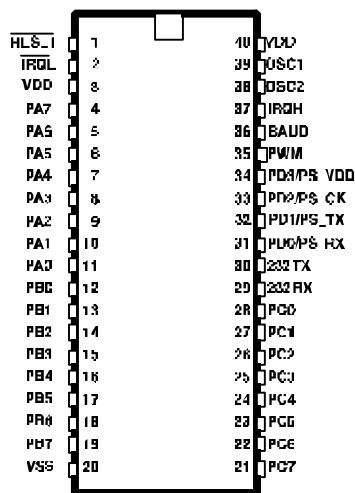


Figure 1

D. PIN OUT DESCRIPTION

- (1) **RESET:** Bringing this pin Low results in: (a) All previous configurations are lost. (b) PA, PB & PC are all configured as inputs. (c) The Baud rate is set according to BAUD pin. (d) The CRAD configuration is set as default. (e) Pin PWM is pulled Low. (f) The message:

Welcome to the ITC232-A
? or h for help

followed by ASCII(#7), CR, LF and the ">" prompt is sent out pin 232 TX.

- (2) **IRQL:** Edge sensitive only. Asserted on a High to Low transition. If BAUD pin = Low (300 Bauds) and IRQL is asserted before a command after power-up or RESET then IRQL forces the ITC232-A into Remote Mode. After the first command or if the ITC232-A is not in Remote Mode, IRQL sends an L to the terminal (see Interrupts).

- (3) **VDD:** +4.5 to +5.5 Volts referenced to VSS.

- (4-11) **PA0..PA7**: Parallel port A. PA0 is excluded from PA when the Remote Mode is asserted.
PA4..PA7 are used by Stepper port A.
- (12-19) **PB0..PB7**: Parallel port B. PB4..PB7: Stepper port B.
- (20) **VSS**: Lowest digital voltage connected to the ITC232-A.
- (21-28) **PC0..PC7**: Parallel port C.
PC0..PC3: Used to measure resistance or capacitance values.
PC4..PC7: Stepper port C.
- (29) **232 RX**: Receives RS232-C serial data from terminal.
- (30) **232 TX**: Transmits RS232-C serial data to terminal.
- (31) **PD0/PS_RX**: Pin common to PD (always 4 inputs) and PS (the Serial Peripheral Interface or SPI).
When SPI active this pin receives data from a peripheral chip synchronized with the clock on pin PD2/PS_CK (see Serial Peripheral Interface).
- (32) **PD1/PS_TX**: Pin common to PD and PS. When SPI is active this pin sends data to a peripheral chip synchronized with PD2/PS_CK.
- (33) **PD2/PS_CK**: Pin common to PD and PS. When SPI is active this pin clocks data in and out PS_TX and PS_RX. The clock can be in phase or out of phase with the data and idle Low or High according to the SPI configuration sent from the terminal. This pin must be tied via a suitable resistor to the clock idle voltage. Failure to do this might result in the peripheral missing the first clock pulse since PD2/PS_CK is in a high impedance state until the SPI becomes active.
- (34) **PD3/PS_VDD**: Pin common to PD and PS. To use the SPI this pin must be tied to VDD.
- (35) **PWM**: Pulse Width Modulation output.
- (36) **BAUD**: Selects the default Baud rate at reset or power-up. Low = 300 Bauds, High = 9600 Bauds. Changing the BAUD pin level after a reset is of no consequence.
- (37) **IRQH**: Edge sensitive only. Asserted on a Low to High transition. Sends an H to the terminal.
- (38) **OSC1**: To 3.6864 MHz crystal or external clock.
- (39) **OSC2**: To crystal. If external clock applied to OSC1 then OSC2 should be left unconnected.
- (40) **VDD**: +4.5 to +5.5 Volts referenced to VSS.

E. MAIN FUNCTIONS OF THE ITC232-A

1. DIGITAL INPUT/OUTPUT

The ITC232-A has 6 ports. One is the RS232 port which links it to the terminal via pins 232TX and 232RX (Figure 1). The other 5 ports can be used in your circuit and they are: PA,PB,PC,PD and PS. PA,PB and PC are general purpose I/O 8 bit ports. These 24 pins can be individually configured as inputs (high impedance) or outputs. In the latter case, pins can be individually addressed since even though the entire port value must be written to a port, the pins remaining in their former state will not change levels at any time. PD is always a 4 bit input port which shares pins with PS, a synchronous Serial Peripheral Interface (SPI) that can be used to communicate with other chips such as a parallel in/serial out shift register. In ITC232-A pin out diagram, PD and PS pins are labeled PDx/PSx.

PD is always available, PS must be configured before being used (while all other ports can be read without being configured first, an attempt to access PS before configuring it will result in error ?2 Port must be configured or enabled first). When a read or write command is issued to PS, PD yields its pins to the SPI, the transaction takes place using the previously input configuration for PS and then the pins are returned to PD.

While it is possible to use both PD and PS in an application, this is not recommended. Upon start-up or reset, all PA,PB and PC pins are configured as inputs (high impedance), PD is always inputs and PS is not configured at all.

For an example in Qbasic (easily carried to any other Basic) showing how to set-up the computer's serial port and the ITC232-A refer to Appendix B.

2. SERIAL PERIPHERAL INTERFACE (SPI)

In order to save pins, many IC manufacturers produce chips (such as A/D or D/A converters among others) that communicate through a synchronous serial interface. The ITC232-A SPI is designed to communicate with these chips and accommodate the various communication protocols they might require. The SPI operates as a fixed speed (57.6 KHz) circular shift register of which 8 bits are inside the ITC232-A and the other 8 (or more) bits are in a peripheral chip. Thus, in order to clock data into the ITC232-A, a value must be clocked out. When reading the SPI, the ITC232-A clocks out the last value written to the SPI or 0 if no value was previously written.

Port D, which shares pins with the SPI, is always active. When an SPI operation takes place, the corresponding configuration is loaded, the command is executed and the port returns to its normal high impedance state. Important: (1) ALWAYS pull the clock pin of the peripheral chip to ground or VDD (depending on whether the clock is required to idle Low or High) via a suitable value resistor (otherwise the first clock transition might not be detected at all). If you also need to use this pin for PD, place a 0.1-1 uF capacitor between your peripheral clock pin and PD2/PS_CK. (2) To use PS, pin PD3/PS_VDD MUST be tied High.

3. PULSE WIDTH MODULATION (PWM)

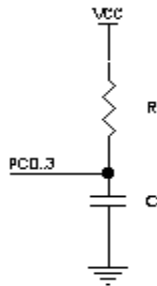
PWM can be used in a number of applications such as generating an analog voltage or regulating the speed of a D.C. motor (speeds down to a few turns per minute can thus be achieved with minimal torque reduction). The possibility to generate a given frequency signal is also advantageous. Besides the obvious creation of musical tones, this feature allows the production of accurate time intervals.

The ITC232-A features a PWM pin. The frequency and duty cycle of the pulses appearing on this pin can be specified from the terminal. The frequency range is 10-10,000 Hz and the duty

cycle can be set from 0 to 100% in 1 % intervals. However, as the frequency increases it becomes progressively more difficult to measure small time intervals accurately. Thus, the full duty cycle range is only available up to 220 Hz for 1% duty cycle and 230 Hz for 99% duty cycle. Increasing the frequency compresses the available duty cycle range around 50%. The PWM pin can be also set High or Low. At power-up or reset the PWM pin is set Low.

4. MEASURING A RESISTOR OR CAPACITOR

The input impedance of port C is extremely high. Thus, if an RC network is connected to an input pin as shown in the following figure, then the time constant of the network can be determined by measuring the time to reach a Low to High transition.



Since for a given pin the transition point is constant if VDD is kept constant, relative capacitances and resistances can be read. Values from one pin cannot be compared to those from another pin because the transition points are not identical. The sequence used by the ITC232-A to carry out a reading after receiving Rn is as follows:

- 1) The selected port is turned into a Low output for a short time in order to discharge capacitor C.
- 2) The pin is turned into an input. A loop linked to the internal clock measures the time required for the pin to become a logic High.
- 3) The result is sent to the terminal.

Note that the value returned is linearly proportional to the time to charge C to a certain voltage and hence there is a linear relationship with the values of C or R. Due to minor differences in the electrical characteristics of the input pins, readings are designed to be relative to a capacitance or resistance standard used to calibrate a given pin. Use high quality, low leakage capacitors. Polyethylene caps work best. Avoid electrolytic capacitors.

The reading is linear (for both capacitance and resistance) between 10 and 32,767 relative units, with an error, as small as 0.5 %, depending on the capacitor used. Should the time to read the R network be excessive (a result > 32,767), Error ?7 (Time out error) will be sent to the terminal. The range of resistance that can be measured goes from 200 Ohms to 10 MegaOhms using different capacitors for the desired range. We recommend against measuring resistances below 500 Ohms. For one, readings are less accurate than above 500 Ohms. Secondly, the chip might

be permanently damaged due to excessive current flowing through the pin when it goes Low to discharge the capacitor.

Should PWM be enabled, expect the reading error to increase at frequencies over 5 KHz or when a combination of frequency and duty cycle is close to the admissible limits since PWM interrupts have priority over the timer used for the R command.

The port needs not be configured as an input in order to obtain a value (the <R>esistance command takes care of it). However, keep in mind that:

- (1) Changing the pin configuration and writing to it, prior to reading a resistance, might introduce differences as large as 10% in the readings (depending on the type of capacitor and its value (a Siemens® 0.47 uF Poly capacitor yields a difference below 1%)). Thus, we strongly recommend to keep the pin configuration and value unchanged between readings.
- (2) A circuit connected to the port might interfere with the reading obtained unless its impedance is very high compared to the resistance being measured.
- (3) Changing the PWM frequency over a wide range between resistance readings might lead to differences in readings which get larger as the returned value gets lower. Measuring resistances can in many instances replace an A/D converter and it finds application in many different situations. Some examples are robots, in which the absolute position of a mechanical element can be followed-up by measuring a variable resistor attached to it, in measuring light intensity (with a Cadmium sulphide cell) and even for measuring the conductance of a solution. For the latter, use the PWM pin at ~1000 Hz to control a 4066 analog switch which alternates the electrodes (representing here the resistance in the network) in order to avoid electrode polarization.

5. STEPPER MOTOR CONTROLLER

Note: For a detailed explanation of stepper motor basics, please refer to Appendix A.

The ITC232-A contains all the necessary logic to operate any stepper motor using very simple commands. The following rules apply:

- 1) Steppers are controlled through the upper 4 bits of PA,PB or PC.
- 2) The name of the motor equals the name of the port to which it is connected.
- 3) The steppers must be enabled (and configured) before use.
- 4) Enabling a stepper turns the upper 4 bits of the port into inputs (high impedance). Disabling a stepper leaves the corresponding pins as inputs with the last value written to the stepper in the corresponding phantom port bits (see under PORTS).
- 5) The configuration applies to all enabled steppers (you cannot have 2 or 3 motors configured differently).
- 6) If the configuration is changed, the last value written to the port is retained. Thus, the speed of a stepper may be changed without losing steps in the process.
- 7) While stepping the PWM is inactive. If the PWM pin was set High (WH) or Low (WL) it will remain the same. If pulsing, the PWM pin will be brought Low during stepping.

6. INTERRUPTS

Interrupts are not commands in the strict sense of the word. They are characters sent from the ITC232-A to the terminal to signal an event occurring in the peripheral even while the ITC232-A is engaged in other tasks such as stepping a motor or generating a PWM signal. There are 2 interrupt pins: IRQL which detects a High to Low transition and IRQH which detects a Low to High transition. Note that the interrupts are only asserted when an EDGE is detected; the pin level is irrelevant. Always tie both IRQ pins to the supply voltage opposite to the one they detect (directly if the IQ is not used or via a 10K resistor). Detecting edges rather than levels allows multiple interrupts from different devices. For multiple interrupt access, pull the IQ pin to the voltage opposite to what it detects via a 10K resistor and bring the interrupt edge to the IQ pin via a 0.1 uF capacitor in parallel with a 100K resistor. The interrupt pins are particularly useful in robotics and mechanical control where an end-of-excursion must be detected in order to stop the corresponding motor.

Asserting the IRQL interrupt sends an L to the terminal.

Asserting the IRQH sends an H.

No OK, CR, LF or the > prompt are sent.

Interrupt priority has been set as follows: (1)=IRQH (2)=IRQL, (3)= Stepper motor timing, (4)=PWM. Simultaneously asserting IRQL and IRQH results in only the latter being acknowledged. If the PWM pin is pulsing very close to the allowed frequency/percentage of duty cycle combination, an IRQL or IRQH might occasionally introduce a glitch in the PWM signal.

7. REMOTE OPERATION VIA A PHONE LINK (Phone mode)

The ITC232-A can be operated remotely using a low cost modem IC such as the AD7910 or AD7911 in-lieu of the RS- 232 driver. The following figure shows an example of the corresponding circuitry:

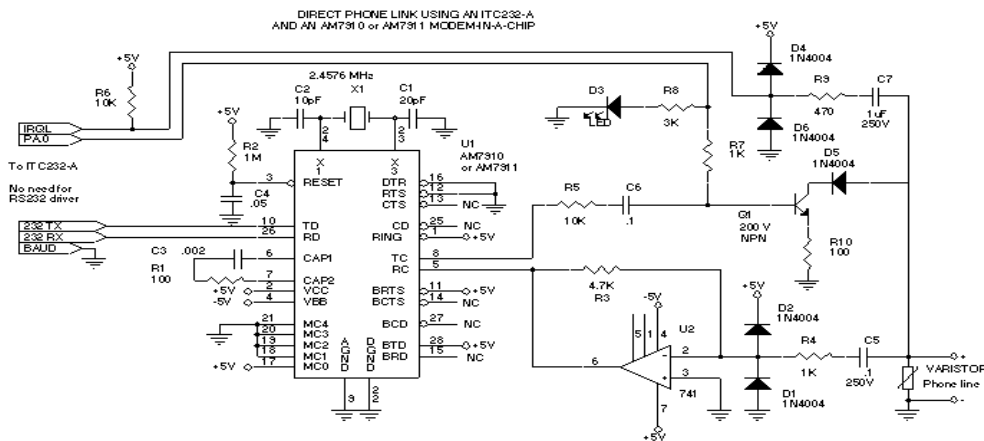


Figure 2

Remote operation requires that the ITC232-A operate at 300 Bauds (by pulling the BAUDS pin Low) and that an IRQL (generated from the ring signal of a phone line as shown in Figure 4) takes place prior to any command after power-up or reset (should a command be issued before the IRQL then the ITC232-A will operate normally (remote operation is disabled)). The first IRQL does not send an L to the terminal. Instead, it results in PA.0 = High. This is used to answer the phone. PA.0 is thereafter functionally excluded from PA; writing or configuring this

pin will have no effect. In order to limit the necessary lines to 1 (PA.0) as well as to allow for simple modem systems, there is no carrier detection. Instead, the message "Send a command within 30 sec or the ITC232 will hang up. This message will thereafter repeat itself if no commands within 5 minutes. Send OFF to hang up." followed by ASCII(#7) and the > prompt is sent out approximately 7 seconds after answering the phone. The ASCII(#7) not only provides an audible signal but is also useful to recognize this particular message from within a custom program. The 7 seconds delay allows commercial communications packages to detect the presence of the carrier. Should the ITC232-A not receive a command or Enter within 30 seconds, it will assume the phone call was unintentional and carry out the following sequence:

- (1) Send out the DISCONNECTING ASCII(#7) > message,
- (2) Hang-up by making PA.0 = 0,
- (3) Reconfigure PA.0 as an input,
- (4) Wait idle for another call.

Since the phone link could be unintentionally cut-off at any time, the ITC232-A will repeat the message above every ~5 minutes without a command (thus, make sure you send e.g. PRA, @ or Enter (ASCII(#13)) every 2-3 minutes to reset the timer). Should no command be received 30 seconds after the warning message, the ITC232-A will hang up and sit idle waiting for the next call. Operating a stepping motor will disable the 5 minute timer in order to prevent being disconnected during a stepping sequence. The timer is reset after the last step thus you have 5 minutes to issue a command before the warning message is issued again.

The <OFF> command allows the remote user to hang-up the phone connected to the ITC232-A (this command is not allowed unless the conditions for remote operation have been fulfilled). Whenever the ITC232-A receives <OFF>, the DISCONNECTING ASCII(#7) > message is sent to the terminal and the PA.0 bit is pulled Low. The various configurations and port values are retained until a reset or a power-down occurs but not the Baud rate which will be reset to 300 Bauds whenever the ITC232-A answers the phone again. Note that a <RESET> sent from the terminal will not only reset the chip but also hang up the phone. You can simulate remote operation while being connected to the ITC232-A directly from a terminal by using a LED to monitor PA.0 and generating an IRQL following a reset.

F. COMMANDS

Commands are always sent in ASCII format and they must be followed by a carriage return (CR or ASCII(#13)). The ITC232-A accepts upper or lower case and spaces or punctuation marks may be added for clarity (with the exception of the semicolon <;> which is reserved as a separator for command parameters). A successful command is always acknowledged with the OK string. The last character sent back to the terminal is always the > prompt (this is useful from within a custom program to determine when the ITC232-A is done with a command). Interrupts are an exception; they are signaled by only an L (IRQL) or an H (IRQH). An erroneous command returns ?n where ? indicates an error and n is the error code (from \$1 to \$B, see ERROR LIST). The error code number is followed by an error message unless the CRAP configuration is in use.

There are 2 types of commands: procedures and functions. A procedure command executes an action. A function command may or may not execute an action but it always returns a result. A successful procedure type returns OK (a CR and a LF is inserted before the OK and before the > unless the CRAP configuration is in use. A successful function command returns OK{\$}value where {} means optional and \$ applies only to results in Hexadecimal (this allows to input the hexadecimal result directly into a numeric variable in some programming languages). When Value is one byte long, results can be requested in either Decimal (3 digits), Hexadecimal (2 digits preceded by \$) or Binary (8 digits in two 4 bit groups (nibbles) separated by a space).

Functions returning numbers that might be higher than 255 are always returned in Decimal format

At power-up the default configuration for results is CRAD (Decimal). The default format can be changed at any time with the CONFIGURE RESULTS command. To override the default format for one reading, add B or % for Binary, D for Decimal or H or \$ for Hexadecimal AFTER the command (e.g. PRA will return the value in Port A in the default format, e.g. Decimal. PRAB will return it in ASCII binary, but only this once; next time PRA will return the result back in decimal).

When issuing a 1 byte number in a command, any of the three formats can be used. The default is Decimal (you can also place a D before the number). To send 1 byte Hexadecimal numbers to the ITC232-A, precede the number with an H or a \$ and for Binary numbers, precede the value with B or %. If you are issuing commands in Decimal you can use single, double or triple digit numbers (1 = 01 = 001). However, in Hexadecimal, two digits must be entered (\$F is incorrect, it must be \$0F). In binary, all 8 bits must be specified (B111 is incorrect, must be B00000111). Any number that might require more than 1 byte (e.g. the speed of a stepper motor) must be issued in Decimal.

Commands are arranged by item; commands starting with P are port commands, S are stepper motor commands and so on. Entering > or Esc (ASCII(#27) at any point of a command erases the command. A new ">" prompt is then issued.

LIST OF COMMANDS (in alphabetical order):

Characters within <> are mandatory, the rest of the command word is included for explanatory purposes. Items within {} are optional.

<@>gain.

Again command.

Procedure or Function type.

The @ character echoes the last command to the terminal preceded by @ (unless the CRAP configuration is active) and repeats its execution. If no previous command were issued @ has no effect.

aud rate <300>, <600>, <1200>, <2400>, <4800>, <9600>, <19200>, <38400>, <57600> or <115200>

Baud rate.

Procedure type.

Selects the Baud rate regardless of the Baud pin level. The command is acknowledged before the Baud rate changes. Make sure that your program adequately handles the few scrambled characters resulting from this action. Using the CRAP configuration at 115200 Bauds, ~1000 analog conversions per second from an MC145041 connected to the ITC232-A SPI or ~1250 parallel port readings per second can be achieved (using a compiled program and the @ command). Beware though that writing software to operate at high Baud rates is tricky and transmission errors are more frequent at higher Baud rates when long cables are used.

<C>onfigure <R>esults <A>SCII inary or <D>ecimal or <H>exadecimal or <P>rogram.

Configuring the default format for results

Procedure type.

The ITC232-A can return results in Decimal, Binary and Hexadecimal format. The default, upon reset or power-up, is Decimal. To change it (at any time) use this command. The CRAP configuration is optimized for operating the ITC232-A from a user written program and it differs from the other configurations in the following:

- * No CR or LF are inserted.
- * The format default for results is Decimal (it can be overridden adding B or %, D and H or \$ after a command).
- * The following are disabled:
- * Help.
- * Error messages (only ?n (n = error number) is sent to the terminal). ⌘ PCp? (where p = A, B, C or S).
- * S? (stepper motor configuration).
- * Actual frequency returned to the terminal when issuing a PWM command.
- * An attempt to obtain information not available with CRAP enabled will result in error ?3.

<H>elp or <?>.

Help (on screen)

Function type.

Returns a summary of all the commands. To scroll the screen press any key. To leave Help press > or Esc and you will be returned to the > prompt for a new command. Mind that some communication packages (particularly older ones) have a small serial port buffer and send out XOFF to tell the peripheral to stop sending data until the buffer is cleared and then XON to continue. The ITC232-A does not recognize the XOFF-XON protocol and therefore you might lose some characters. Should this be the case, use a slower Baud rate or another communications package. On screen help is not available when the CRAP configuration is active.

<P>ort <C>onfigure <A> or or <C> <Value>.

Configure Parallel ports PA, PB and PC

Procedure type.

Each pin can be individually configured as an input or an output according to the corresponding bit in <Value> (0 for input, 1 for output). Preceding <Value> with an H or \$ for Hexadecimal, B or % for Binary and D for Decimal overrides the default. For example, to configure the lower 4 bits of PB as inputs and the higher 4 bits as outputs the following commands can be used: PCB \$F0, PCB B1111 0000, PCB 240 or PCB D240 (spaces are allowed but not mandatory). Trying to configure PD will result in error ?A Port D is always a 4 bit input port.

<P>ort <C>onfiguration <A> or or <C> or <S> <?> {B,%,D,H or \$}.

Return port configuration

Function type.

Returns the port configuration (not available when CRAP is active). For example, after configuring PB as above and having CRAB as default, PCB? will yield OK 1111 0000 and PCB?D will yield OK 240. <S> applies to the SPI.

<P>ort <C>onfigure <S>erial <R>ead or <W>rite or <A>ll <V>alue.

To configure the SPI

Procedure type.

Some serial chips need only be read. Others might only be written to and finally, some devices require to be both read and written to. Both read only and write only peripheral chips might be simultaneously connected to the SPI and they might require different communication configuration. Thus the <R>, <W> and <A> in the command (<A>ll means both read and write).

When writing out a value, the SPI is first configured as per the last PCSW command (if no previous configuration entered, error ?2 Port must be configured or enabled first will occur). When reading data in, the SPI is configured as per the last PCSR command.

When a read operation takes place, the <R>ead configuration is enabled, the last value written to the SPI (or 0 if none yet) is clocked out and the value from the peripheral is clocked in and sent to the terminal. Beware that the value is written out in the <R>ead configuration and therefore should another peripheral to which you normally write (using a different <W>rite configuration) be connected to the SPI, it might interpret the data wrongly. It is recommended that you use a parallel port pin to select the device when more than one chip is connected to the SPI.

<V>alue specifies whether the SPI is enabled or not, the clock polarity, whether the clock and the data are in phase or not and the sense of the byte (some chips send data MSB first while others send the LSB first). The Table below shows the function of the different bits in <V>alue:

Bit 7 Must be 1 in order to enable the SPI.

Bits 6,5,4,3 Irrelevant.

Bit.2 Determines the polarity of the clock (PD2/PS_CK pin).
POL 0 = the clock idles Low.
POL 1 = the clock idles High.

Bit.1 Controls the clock phase (PD2/PS_CK pin).
PHASE 0 = In phase with data.
PHASE 1 = Out of phase with data.

Bit.0 Controls the order of the bits received by the SPI. This is useful in some cases when a peripheral sends the byte "backwards".
ORD 0 = The order is preserved.
ORD 1 = The order of the bits is reversed (MSB <--> LSB)

The following Table shows all possibilities:

BIT	7	6	5	4	3	2	1	0	HEX	DEC
	1	0	0	0	0	0	0	0	\$80	128
	1	0	0	0	0	0	0	1	\$81	129
	1	0	0	0	0	0	1	0	\$82	130
	1	0	0	0	0	0	1	1	\$83	131
	1	0	0	0	0	1	0	0	\$84	132
	1	0	0	0	0	1	0	1	\$85	133
	1	0	0	0	0	1	1	0	\$86	134
	1	0	0	0	0	1	1	1	\$87	135

To disable the SPI, any configuration with Bit.7 = 0 (any decimal < 128 or hex < \$80) will do. Important: Pin PD3/PS_VDD need not be pulled High for a configuration command to be successful but it is mandatory to read or write to the SPI (otherwise error ?B will be issued). As with the other ports, PCS?{number base} returns <Value> in the specified {number base} or in the default format.

<P>ort <R>ead <A> or or <C> or <D> {B,%,D,H,\$}.

Reading a parallel port

Function type.

When reading a parallel port which has pins configured as outputs, the value returned in the corresponding bits is the present state of those pins (0 if nothing written to the port since power-up or reset).

<P>ort <W>rite <A> or or <C> {B,%,D,H,\$} <Value>.

Writing to a parallel port:

Procedure type.

When writing to a pin configured as an input, the pin remains in a high impedance state. However, the value is stored in a phantom bit which might be thought of being present behind the actual port bit. Thus, if you write to an input pin and then configure it as an output pin, the value in the phantom bit will be immediately transferred to the corresponding pin (this can lead to unwanted values in a port pin if this is not kept in mind). After a reset or upon start-up the value in all phantom bits is 0.

<P>ort <R>ead <S>PI {B,%,D,H,\$}

Reading from the SPI

Function type.

<P>ort <W>rite <S>PI {B,%,D,H,\$} <Value>

Writing to the SPI

Procedure type.

The <P>ort <R>ead and <P>ort <W>rite commands work in a fashion similar to that of the parallel ports with the following exceptions:

- 1) Pin PD3/PS_VDD must be tied High. Otherwise, the error message: “?B SPI requires pin PD3/PS_VDD always high, change and try again.” is sent to the terminal.
- 2) The relationship between configuration and pin value is different.
- 3) There is no phantom port storing the last written value as happens with PA, PB and PC.
- 4) IMPORTANT: The SPI can be considered as a circular serial shift register in which the 8 bits in the port are circulated through a peripheral chip. In order to read a value into the SPI, a value must be written out. This value is the last written value to the SPI or 0 if no writing took place between the configuration and a reading. There are some particular situations that should be avoided or handled with care:
 - 4.a. Connecting 2 peripheral chips to the SPI, one which is read from and the other which is written to while having them both enabled simultaneously. This will result in the re-writing of the latter when reading the former. The value written will be the same as last time, but the data on the peripheral chip will jump up and down as the byte is shifted in at a speed of 57.6 KHz. In addition to this, if the write and read configurations are different it is almost certain that the chip to be written will misinterpret the data. To prevent this, use pins from PA, PB or PC to enable or disable the chips connected to the SPI as required.
 - 4.b. Connecting a peripheral chip that requires both writing and reading to operate, e.g. the Motorola® MC145041 Analog/Digital converter.

<RESET>

Resets the ITC232-A

Procedure type.

This command is self explanatory. It has the same effect as a hardware reset.

<R>esistance <0> or <1> or <2> or <3>.

Measure resistor or capacitor

Function type.

0-3 represent the bits or pins on PC.

**<S>tepper <E>nable <A> or or <C> <M>onophasic
or iphasic or <H>alf step <Speed> <;> <Stop delay>**

Enable and set-up stepper
motor

Procedure type.

<A>, and <C> refer to the 4 upper bits of the corresponding ports. <M>onophasic, iphasic and <H>alf step refer to the modes described above. <Speed> is in steps/second (10 to 4,000). <Stop delay> is a value within 0-255 decimal representing the number of steps during which the last value on the stepper port remains fixed before the port returns to a high impedance state. (E.g. SEAB500;10 will make stepper A turn at 500 steps/s (2 ms/step) and hold the last step for $2 \times 10 = 20$ ms.) This serves the purpose of braking the motor thus preventing it from continuing turning due to inertia (The larger the motor and the higher the speed the larger this value should be. Start trying with 10% the speed value).

Once the configuration for a stepper has been entered, enabling a second or third stepper only requires <S>tepper <E>nable <A> or or <C>. The remaining parameters are the same for all motors. The configuration entered last becomes current for all motors. The maximal speed error for a stepper is < 2.2 %.

<S>tepper <D>isable <A> or or <C>.

Disable stepper motor

Procedure type.

Note that you can disable all the steppers while the configuration is retained in memory. Thus, a subsequent <S>tepper <E>nable <A> or or <C> will work using the same parameters as previously entered.

<S>tepper <?> {B or % or D or H or \$}

or

<S>tepper <E>nable <?> {B or % or D or H or \$}

Request stepper motor configuration

Function type.

These commands return the stepping motor configuration, the enabled ports (A,B & C) and the last output for each motor. These commands are disabled when the CRAP configuration is active. {B, %, D, \$ or H} specify the format for the last value on the corresponding port (otherwise this value is returned in the default configuration).

You can reconfigure a port with PC p nr while the corresponding stepper is enabled and you can even write to the pins by issuing a PW p nr (WARNING: the last value written to the stepper port will immediately appear on a pin should it be configured as an output). The stepper port will be reconfigured when stepping again and it will be left as all inputs once the stepping stops).

Re-configuring a parallel port connected to a stepper motor driver is strongly discouraged since it might lead to catastrophic output combinations that can destroy the stepper driver transistors.

<S>tep <A> or or <C> <L>eft or <R>ight <Number of steps>.

To step a motor

Procedure type.

<L>eft or <R>ight are arbitrary mnemonics to specify the direction of spinning. <Number of steps> can be any number up to 65535 (a value of 0 is valid and results in no action).

To stop a motor while stepping:

Sometimes you might want to force a stepper to stop before all the steps are completed. This can be done by sending: a space (ASCII(#32)), "S", "s", ">", Esc (ASCII(#27)), or Enter (ASCII(#13)). This action results in the ITC232-A sending the remaining number of steps (5 digits, ALWAYS in decimal, with leading 0's if required) followed by the message steps to go (the latter omitted if the CRAP configuration is active).

<W>idth <frequency>.

Sets PWM frequency

Function type.

<frequency> can be any value between 10 and 10,000 Hz and it must be specified in decimal format. This command assumes a 50% duty cycle is required. The actual frequency is sent to the terminal (see note #1 below).

<W>idth <frequency> <;> <Duty cycle>.

Sets PWM duty cycle

Function type.

<Frequency> is the same as above, the <;> is mandatory and <Duty cycle> is an integer from 0 to 100 indicating the percentage of the cycle during which the PWM pin is High.

Notes:

1) Upon entering these commands the ITC232-A returns f=XXXXXX (not available in the CRAP configuration). XXXXXX is always a 5 digit long decimal number and is the integer part of the actual frequency the ITC232-A is putting out (due to calculation rounding and timing restrictions). The actual frequency is given by the expression: $Af = 460800 \div \text{Round}(460800 \div Rf)$ where Af is the actual frequency and Rf is the requested frequency. Af should be within the precision of the crystal.

2) During the stepping of a stepper motor:

- (a) If WL or WH are active then the pin stays as it is.
- (b) If PWM is pulsing, the PWM pin is brought Low during stepping.

3) If <Duty cycle> = 0 or 100, then <frequency> is irrelevant, the PWM pin will stay Low or High respectively. Another way to achieve this is by using the <WL> and <WH> commands.

4) <Duty cycle> can be varied in 1 % intervals, however, as the frequency increases it becomes progressively more difficult to generate very small or very large duty cycles. If a particular duty cycle cannot be achieved for a given frequency, the ?8 Frequency too high for requested duty cycle error will occur. The highest frequencies require duty cycles around 50% whereas below 220 Hz, 1% or 99% duty cycles are possible.

<W>idth <H>.

Forces the PWM pin High.

Procedure type.

<W>idth <L>.

Forces the PWM pin Low.

Procedure type.

This command is internally issued at power-up or reset.

<W>idth <?>.

Return last <W> command issued

Function type.

Returns the last <W> command issued (the requested frequency, not the actual frequency). Not available under the CRAP configuration. Issued immediately after a reset or start-up, the W? command returns WL.

G. ERROR LIST

- ?1 Syntax error.
- ?2 Port must be configured or enabled first.
- ?3 Command not allowed in current configuration.
- ?4 No such port.
- ?5 Value out of range or syntax error.
- ?6 Pin configured as an output.
- ?7 Time out error.
- ?8 Frequency too high for required duty cycle.
- ?9 Baud rate not supported.
- ?A Port D is always a 4 bit input port.
- ?B SPI requires pin PD3/PS_VDD always high, change and try again.

H. ELECTRICAL SPECIFICATIONS

Maximum Ratings (Voltages referenced to VSS)

Rating	Value	Units
Supply Voltage	-0.3 to + 7.0	V
Input Voltage	VSS-0.3 to VDD+0.3	V
Current drain per pin	25	mA
Storage temperature range	-65 to +150	Deg. Celsius
Operating temperature range	0 to +70	Deg Celsius

DC Electrical Characteristics (VDD-VSS = 5.0 VDC)

Characteristic	Min	Typ	Max	Units
Output Voltage (I<10uA)	VDD-0.1	-	0.1	V
Output Voltage (i=0.8mA)	VDD-0.8	-	0.4	V
Input High (PA,PB,PC,PD, IRQ's,BAUD,232 RX,RESET)	0.7 x VDD	-	VDD	V
Input Low (PA,PB,PC,PD, IRQ's,BAUD,232 RX,RESET)	VSS	-	0.2 x VDD	V
Supply Current	4.7	7.0	-	mA
Hi-Z input leakage current (PA,PB,PC,PD)	-	-	10	uA
Capacitance PA,PB,PC,PD	-	-	12	pF
Capacitance RESET,IRQ's, 232 TX,232 RX, BAUD	-	-	8	pF

Notes:

1. All values show average measurements.
2. Measurements were made at 25°C.

APPENDIX A:

Stepper motor basics:

A stepper motor is designed in such a way that it turns one step at a time. This allows precise motion and absolute positioning of the rotor. Figure 6A shows a simplified version of a stepper motor.

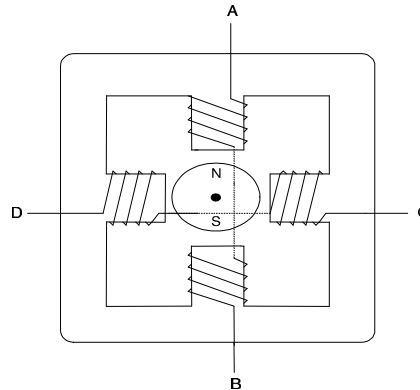


Figure 6A

The rotor consists of a permanent magnet surrounded by the stator poles (electromagnets, also referred to as phases). The magnetic polarity of the stator can be changed according to the current flowing through its windings. The motor in the example can be made to turn with three possible drive sequences: The first is to energize the windings in the sequence AB/CD/BA/DC (BA is the same winding as AB but the current flows in the opposite direction) as shown in Figure 6B.

This sequence is known as "one-phase-on" or "monophasic" because at any one time only 1 winding (phase) is energized. The second possibility is to always energize both windings. This is called "two-phase-on" or "biphasic" and it is represented in Figure 6C. Note that now the rotor aligns itself between two pole positions rather than with the poles themselves. This configuration, which is the one most commonly used, yields the highest torque. Finally, the third option is to energize one phase, then two, then one, etc. as shown in Figure 6D. This sequence halves the angle per step and therefore is called "half step mode". All three modes are implemented in the ITC232-A as we will see later.

Please note that the very first step can result in (a) no motion if the rotor happens to be aligned with the stator, (b) a clockwise motion (as wanted) if the rotor is behind the stator or (c) a counter clockwise motion if the rotor is ahead of the stator. This is only valid for the first step of a sequence but it should be kept in mind when designing a system. The angle per step is 90° (Figures 6A,B & C), or 45° (Figure 6D). Real motors have many more poles on the stator and rotor thus allowing for much smaller angles per step. Typical values are 15° (24 steps/turn), 7.5° (48 steps/turn), 3.75° (96 steps/turn), 3.6° (100 steps/turn) and 1.8° (200 steps/turn). Four bits of a port can be used to drive a stepper motor. For example, if we connected the A,B,C and D winding ends to a suitable current driver which buffers the output from a 4 bit binary word (nibble) created by the ITC232-A, the connection would be as follows:

Winding	C	D	B	A
Bit	3	2	1	0

In monophasic mode (Figure 6B), the driving sequence would then be 0001, 1000, 0010, 0100 (to spin in one direction and backwards to spin in the opposite direction). In biphasic mode

(Figure 6C), the sequence is 1001, 1010, 0110, 0101. Interleaving the former we have the half step mode (Figure 6D) with the driving sequence 0001, 1001, 1000, 1010, 0010, 0110, 0100, 0101.

Once the last value is output, the sequence is repeated. Stepping can start on any value provided that the sequence order is maintained.

Inverting the voltage on a winding requires 4 transistor H bridges such as the ones shown in Figure 7.

Even though the drivers can be made using 8 discrete transistors per motor, a much easier, cheaper and efficient way, is to use a driver IC such as the L298 from SGS®. Another way to approach the driver problem is to use a so called unipolar or bifilar winding motor. This kind of stepper normally has 5 or 6 cables coming out instead of 4. The coils in bifilar motors are double; one is wound clockwise and the other counter-clockwise. To invert the magnetic field, power is sent to one or the other coil rather than changing the current flow through the same coil. The circuit to drive this kind of stepper only requires 4 transistors (Figure 8). This figure also illustrates why only 5 or 6 cables come out the motor instead of 8. Two disadvantages of bifilar steppers compared to monofilar ones are higher cost and less power (both due to an extra set of windings which are only used half of the time).

The driving sequence for both kinds of steppers is identical.

Working with steppers can be frustrating particularly if one is using a stepper without having the specifications on-hand. However, using the ITC232-A in conjunction with an L298 from SGS® or equivalent, or a bifilar stepper with 4 Darlington NPN transistors, makes the task trivial: To use a 4 wire (or monofilar or biphasic) stepper, first, determine with the aid of an Ohm meter which cables correspond to each phase. Then connect them according to Figure 9 and issue the commands described below. Inverting one phase results in the motor turning in the opposite direction. If you are working with a 5 or 6 wire stepper first determine the phases and the common wire to each phase (the one showing half the resistance of the other two). Then, configure the circuit as shown in Figure 8. Again, reversing one phase will invert the sense of spinning.

Using more than 3 steppers:

This can be easily done by multiplexing the stepper motor ports using a parallel port pin to select the active motor driver (if a L298 driver is used (Figure 9), connect ENA and ENB to the controlling pin). Keep in mind that the values left on the port from the last stepper will likely not be the required ones to start the next stepper. This will lead to the equivalent of losing steps. To prevent this, disengage all motors and make the required port to "step" as many steps as necessary to bring the next stepper to a complete turn.

Example: Stepper 1 and 2 are connected to port C. Both have a resolution of 3.6 degrees/step thus requiring 100 steps for a complete turn.

- 1) Activate stepper 1 via a port A pin and make it step 1230 steps.
- 2) Disengage both steppers and make the port step $100-30=70$ steps in the same direction.
- 3) Activate stepper 2 via another port A pin and make it move 13 steps.
- 4) Disengage both steppers and make the port step $100-13=87$ steps in the same direction.
- 5) Activate stepper #1 and repeat the process.

Note: Avoid changing modes (<M>, and <H>) from one stepper to another.

APPENDIX B

QBasic[®] set-up of serial port and the ITC232-A.

What follows is an example in QBasic[®] (easily carried to any other Basic) showing how to set-up the computer's serial port and the ITC232-A. The comments in italics should not be part of the program.

```
CLS
TRUE = 1
FALSE = 0

REM Open COM port
OPEN "Com1: 9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB2048,RB2048" FOR RANDOM AS #1
'This opens a 2K buffer for receiving and transmitting data at 9600 Bauds. You could use Com2 and any other speed up to 19200 (the highest speed attainable in Basic. If your computer is fast enough you can go beyond this point by directly poking values into the 8250 IC registers in the serial port).

PRINT "PLEASE RESET THE ITC232-A"
PRINT
GOSUB READSERIAL
'Wait until the "Welcome..." message is sent and

PRINT $$
'print it on the screen ($$ contains the string received).

W$ = "crap": GOSUB WRITESERIAL
'A command always goes to the writing subroutine in W$. This particular command gets the ITC232-A into program mode to optimize the speed and handling of results.

W$ = "prF": GOSUB WRITESERIAL
'This is an error example: "prF" will generate an error since there is no F port.
END
```

SUBROUTINES

```
REM Writing to the serial port
WRITESERIAL:
PRINT #1, W$
'This sends the command previously stored in W$ out the serial port. Since the ITC232-A always acknowledges a command with OK or an error message, we now read what the ITC232-A has to say by polling the serial port

GOSUB READSERIAL
RETURN
'and we return to the main program. The results from the reading of the serial port are returned in $$, V$ and V as described below.

REM Reading the serial port
READSERIAL:
$$ = ""
```

'Empty the string before reading

IF LOC(1) = 0 THEN GOTO READSERIAL

'In case a polling is done before sending any command, as in this program in which at the very beginning the computer waits for the "Welcome..." message from the ITC232-A.

REM Get the received string into S\$

Lp1:

C\$ = INPUT\$(1, #1)

S\$ = S\$ + C\$

IF C\$ <> ">" THEN GOTO Lp1

'The program loops until it finds an ">". This is because the ITC232-A ALWAYS finishes sending data with an ">".

REM decode string (V\$) and value (V)

'What follows is not really necessary but handy for getting the results of a function type command in a string variable (V\$) and in a numeric variable (V). It also handles errors. If these features are unnecessary, place a RETURN here and ignore the program below.

VALIDERROR = TRUE

ERRORCODE\$ = ""

V\$ = ""

'Reset variables.

FOR H = 1 TO LEN(S\$)

IF MID\$(S\$, H, 1) = CHR\$(7) THEN VALIDERROR = FALSE

IF MID\$(S\$, H, 1) = "?" THEN ERRORCODE\$ = MID\$(S\$, H + 1, 1)

NEXT H

'Detect if there is a "?" in S\$ that would indicate an error. Eliminate the false signaling of an error in the "Welcome..." message which contains a "?" but also a CHR\$(7) (see next line).

IF (VALIDERROR = TRUE AND ERRORCODE\$ <> "") THEN GOSUB

ERRORSUB: RETURN

IF LEN(S\$) > 3 THEN V\$ = RIGHT\$(S\$, LEN(S\$) - 2): V\$ = LEFT\$(V\$, LEN(V\$) - 1)

'If the command was of the function type, then the ITC232-A will return a string longer than 3 characters. In such case, extract the result in V\$ and

V = VAL(V\$)

'get the value into a numeric variable.

RETURN

ERRORSUB:

PRINT

PRINT "Error #"; ERRORCODE\$

RETURN

'This subroutine is self explanatory.

APPENDIX C

Zero, Full Scale and Half Scale Tests in QBasic®

The example below in QBasic® shows how to perform the 3 tests (Zero, Full scale and half scale) the MC145041 provides. Other channels are read identically:

```
CLS
START:
PRINT "CONNECT BOARD AND PRESS A KEY WHEN READY"
START1: DO UNTIL INKEY$ <> "": LOOP
CLEAR
TRUE = 1: FALSE = 0
REM Open COM port
OPEN "Com1: 9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB2048,RB2048" FOR RANDOM AS #1
PRINT "PLEASE RESET THE ITC232-A or ESC TO ABORT"
PRINT
RES:
IF INKEY$ = CHR$(27) THEN END

IF LOC(1) = 0 THEN GOTO RES
GOSUB READSERIAL

W$ = "CRAP": GOSUB WRITESERIAL
W$ = "PCSA128": GOSUB WRITESERIAL

FOR number = 1 TO 8
PRINT number,
W$ = "$B0": PRINT W$, " "; : GOSUB READ10BITS: ' Half scale test
W$ = "$D0": PRINT W$, " "; : GOSUB READ10BITS: ' Full scale test
W$ = "$C0": PRINT W$, " "; : GOSUB READ10BITS: ' Zero test
PRINT
NEXT number
END

REM Subroutines

REM Writing to serial port
WRITESERIAL:
PRINT #1, W$
GOSUB READSERIAL
RETURN

REM Reading serial port
READSERIAL:
S$ = ""
IF LOC(1) = 0 THEN GOTO READSERIAL

REM Get received string into S$
Lp1:
C$ = INPUT$(1, #1)
S$ = S$ + C$
IF C$ <> ">" THEN GOTO Lp1
```

```

REM decode string (V$) and value (V)
VALIDERROR = TRUE
ERRORCODE$ = ""
V$ = ""
FOR H = 1 TO LEN(S$)
IF MID$(S$, H, 1) = CHR$(7) THEN VALIDERROR = FALSE
IF MID$(S$, H, 1) = "?" THEN ERRORCODE$ = MID$(S$, H + 1, 1)
NEXT H
IF (VALIDERROR = TRUE AND ERRORCODE$ <> "") THEN GOSUB ERRORSUB:
RETURN
V$ = ""
FOR n = 1 TO LEN(S$)
x$ = MID$(S$, n, 1)
IF x$ <> CHR$(13) THEN IF x$ >= "0" AND x$ <= "9" THEN V$ = V$ + x$
V = VAL(V$)
NEXT n
RETURN

ERRORSUB:
PRINT
PRINT "Error #"; ERRORCODE$
RETURN

READ10BITS:
W$ = "PWS" + W$: GOSUB WRITESERIAL
W$ = "PRS": GOSUB WRITESERIAL: ' dummy read
W$ = "PRS": GOSUB WRITESERIAL: ' Get MSByte
MSB = V
W$ = "PRS": GOSUB WRITESERIAL: ' Get LSByte
LSB = V
RESULT = LSB + 256 * MSB
RESULT = RESULT / 64
PRINT RESULT,
RETURN

```