WiZ232-A User's Guide
Table of Contents

#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174

WiZ232-A User's Guide

Chapter 1 INTRODUCTION

The WiZ232-A is an integrated circuit containing a general purpose serial to 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 24 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 WiZ232-A from both a dumb terminal program or a custom program is very convenient for system debugging.

The WiZ232-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 9600 and 230400 Bauds. The main external components required by the WiZ232-A are a voltage driver to handle the RS-232 voltages and a 7.3728 MHz crystal.

Features of the WiZ232-A include:

1 To send or receive data in either Decimal, Hexadecimal or Binary format.
2 Low interrupt recognition pin IRQL (serves multiple purposes).
3 15-15,000 Hz, 0-100% duty cycle Pulse Width Modulation with output pulse counting.
4 Software selectable Baud rates from 9600 to 230400.
5 Four channels for reading relative resistance or capacitance.
6 Advance stepper motor control port with all the necessary signals for using external drivers, with full and half-step stepping modes. Acceleration-deceleration feature for complex motion control. Stepping rates from 20 to 8500 steps/sec.
7 A one keystroke "Again" command to repeat the previous command.
8 A configuration report feature which sends the active configuration of any parallel port and the PWM.
9 Embedded command for reading serial ADC (8, 10 & 12 bit resolution).
10 Automatic Sampling feature allows data collection in real time.

The flexibility, ease of use and multiple functions of the WiZ232-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 WiZ232-A finds application.

General Conventions

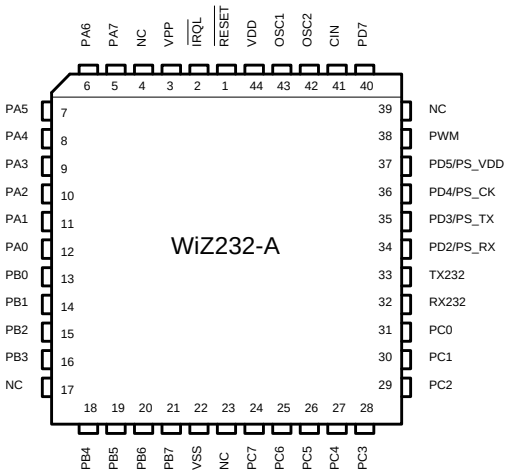
Throughout this manual 'High' is used as synonymous to binary 1 or 5V and 'Low' as synonymous to binary 0 or 0V. The term "terminal" includes computers and the term "peripheral" refers to any circuit containing the WiZ232-A.

#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174

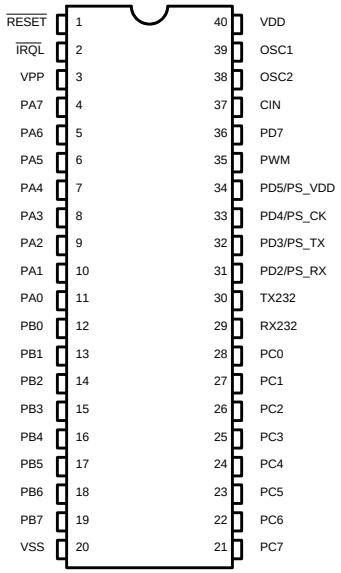
Chapter 2 WiZ232-A PIN ASSIGNMENTS

WiZ232-A PINOUT

Diagrams are for pin reference only. Package sizes are not to scale.



44 PIN - PLCC



40 PIN - DIP

Pin Out Description

Symbol	Pin Number		Type	Description
	Dip	PLCC		
RESET	1	1	I	Bringing this pin Low results in: • All previous configurations are lost. • PA, PB & PC are all configured as inputs. • The Baud rate is set to 9600. • The CRAD configuration is set as default. • Pin PWM is pulled Low. • The message: **WiZ232-A RMV Electronics Inc.** followed by ASCII(#7), CR, LF and the "**>**" (prompt) is sent out through pin 232 TX.

Pin Out Description (cont. I)

Symbol	Pin Number		Type	Description
	Dip	PLCC		
IRQL	2	2	I	Edge sensitive only. Asserted on a High to Low transition. Sends an L to the terminal (see Interrupts).
VPP	3	3	Power	+4.5 to +5.5 Volts referenced to VSS.
PA0-PA7	4-11	5-12	I/O	Parallel port A
PB0-PB7	12-19	13-16 18-21	I/O	Parallel port B.
VSS	20	22	Power	Lowest digital voltage connected to the WIZ232-A.
PC0-PC7	21-28	24-31	I/O	Parallel port C. PC0..PC3: Used to measure resistance or capacitance values. PC7… PC0 are used by Stepper port C.
RX232	29	32	I	Receives RS232-C serial data from terminal.
TX232	30	33	O	Transmits RS232-C serial data to terminal.
PD2/PS_RX	31	34	I/O	Pin common to PD 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).
PD3/PS_TX	32	35	I/O	Pin common to PD and PS. When SPI is active this pin sends data to a peripheral chip synchronized with PD2/PS_CK.
PD4/PS_CK	33	36	I/O	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 PD4/PS_CK is in a high impedance state until the SPI becomes active.
PD5/PS_VDD	34	37	I/O	Pin common to PD and PS. To use the SPI this pin must be tied to VDD using a appropriate resistor
PWM	35	38	O	Pulse Width Modulation output.

Pin Out Description (cont. II)

Symbol	Pin Number		Type	Description
	Dip	PLCC		
PD7	36	40	I/O	Port D bit 7
CIN	37	41	I	Period Measurement input.
OSC2	38	42	O	To crystal. If external clock applied to OSC1 then OSC2 should be left unconnected.
OSC1	39	43	I	Connect to external crystal or ceramic resonator. Also, if external clock is used, it must be applied here.
VDD	40	44	Power	+4.5 to +5.5 Volts referenced to VSS.

Chapter 3 MAIN FUNCTIONS OF THE WiZ232-A

Digital Input/Output

The WiZ232-A has 6 ports. One is the RS232 port which links it to the terminal via pins 232TX and 232RX. 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 can be configured as a 5 bit input/output 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 the WiZ232-A pin out diagram, PD and PS pins are labeled PDx/PSx.

PD is always available, while PS must be configured before being used. While all other ports can be read without being configured first, any 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 inputted 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 configured with PD3 and PD4 as outputs and PS is not configured at all. Caution must be exercised when changing the input/output configuration of Port D as the pins PD3 and PD4 must be configured as output in order to use the SPI. This is the default configuration after a reset.

Interrupts

Interrupts are characters sent from the WIZ232-A to the terminal to signal an event occurring in the peripheral even while the WiZ232-A is engaged in other tasks such as stepping a motor or generating a PWM signal. There is one interrupt pin: IRQL which detects a High to Low

transition. Note that the interrupt is only asserted when an EDGE is detected; the pin level is irrelevant. When it is in an idle state the IRQL must remain High. In order to achieve this, connect the IRQL pin to VCC (directly, if the IRQL is not used, or via a 10K resistor). The interrupt pin serves several functions including stepper motor control and analog sampling. Asserting the IRQL interrupt sends an 'L' to the terminal. No 'OK' is sent.

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 WiZ232-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 (115.2 KHz) circular shift register of which 8 bits are inside the WiZ232-A and the other 8 (or more) bits are in a peripheral chip. Thus, in order to clock data into the WiZ232-A, a value must be clocked out. When reading the SPI, the WiZ232-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 previous 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 resistor value (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 PD4/PS_CK. (2) To use PS, pin PD5/PS_VDD MUST be tied High. The best way to accomplish this is by using a 10k resistor connected to VCC.

Get Analog

The WiZ232-A provides an easy way to read a serial Analog to Digital Converter (ADC) using the SPI port. The command is very simple to use and the user just needs to provide the resolution and the channel to read. This command is based on industry standard ADCs as explained in the following table.

Part Name	Resolution	Number of External Channels	Special channels	Manufacturer
MC145041	8	11	CH11: half scale (128)	Motorola
MC145051	10	11	CH11: half scale (511)	Motorola
TLC1543	10	11	CH11: half scale (511)	TI
TLC2543	12	11	CH11: half scale (2047) CH12: null (0) CH13: full scale (4095)	TI

Syntax:

GA<Resolution><;><Channel>

It returns a 3 digit number (0-255) for 8 bit ADC's and a 4 digit number (0-4095) for 10 & 12 bit ADC's.

Sampling AD Channels

The WiZ232 provides a simple way of getting samples from an analog signal. Speeds from 20 samples per second up to 2500 samples per second are attainable. This constitutes a Real Time data acquisition feature. Real time sampling is useful for many applications such as Spectrum Analysis using FFT, Digital Filtering of noise, etc. The basic idea is gathering data sampled by the WiZ232 and storing this data for later evaluation on the host computer. This supposes that a standard serial Analog to Digital Converter is connected to the SPI interface. Typically used ADC chips are:

8 bit MC145041
10bit MC145051
12bit TLC1543

Configuring the Sampling

The Syntax for this command is:

GS <ADCres><;><ChEnbMask><;><[SampSpeed><;><Samples><;>{Gate}

ADCres : this is the resolution of the ADC: 8, 10 or 12 bit.

ChEnbMask: represents which channels are enabled for sampling: from 1 to 2047.

In order to enable a channel for sampling, use the following table to calculate the Mask value:

channel	Add to mask
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Sampling Mask Channels

Example:
To enable channels #0 , #1 & #7 for sampling, the Mask value would be:

Mask Value = 1+2+128 = 131

#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174

SampSpeed: this is the sampling Speed in samples/second. '0' represents the highest While '7' the lowest.

Sampling Rates

Sampling Value	Sampling Period	Samples per second
0	400 µs	2500
1	500 µs	2000
2	1 ms	1000
3	2 ms	500
4	5 ms	200
5	10 ms	100
6	20 ms	50
7	50 ms	20

Samples: the number of samples to be taken: 1 to 65535.

Gate: add a G if software gating is desired. If a software gate is not desired, i.e. the 'G' is omitted in the GS command, a valid interrupt pulse must be sent to the interrupt input in order to start sampling.

When this command is accepted and no errors are detected, the WiZ232 will send the 'Acknowledge' string ('OK'), if not in CRAP mode. After this it will send all the samples and when done, the WiZ232 will send a prompt '>' (ASCII 62).

Gate Signal: In order to start the sampling, the user can send a gate pulse to the IRQL input. This must be a valid transition pulse (from high to low), after which the WiZ232-A controller will immediately start sampling.

Different formats are used for sending data to the host computer, depending on the ADC resolution:

for 8 bit ADC's : *123
for 10 & 12 bit ADC's : *1234

Samples are sent without any spacing characters, and it is up to the user to store them in a buffer properly arranged for this purpose.

High Speed Sampling: Sampling speeds with a sampling value less than '3' (1,000 to 2,500 samples per second) require the use of the WiZ232's internal RAM. Because of this, at those speeds, samples are first stored in this RAM and then dumped to the serial port. This means that the total number of samples that can be taken is limited to a maximum of 128 for 8 bit ADC, and 64 for 10 & 12 bit ADC's. These figures are also affected by the number of channels enabled. The stated sample maximums (128 & 64) are valid for only one enabled channel. For sampling speeds of 2,000 and 2,500 samples per second, only two channels can be sampled in the same sampling. A violation of this rule will result in an error message.

Several configurations can be tried in order to work out a solution for a given sampling application if a sampling error is encountered. The first solution may be to increase the serial port speed. The maximum number of samples available can be estimated by this equation:

Number of Samples = Serial Port Speed/50/Number of enabled channels.

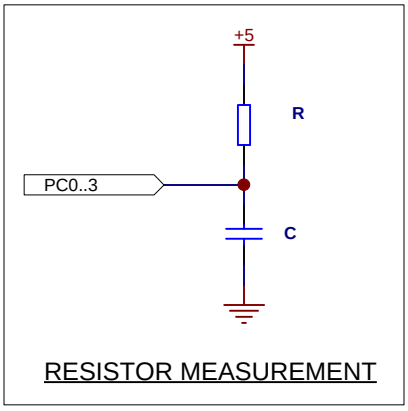
The actual value for the sampling speed should be less than this. If the error persists, then two alternative solutions are available. One is to enable fewer channels and perform the sampling in two batches. The other, is to decrease the sampling speed.

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 WiZ232-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 15-15,000 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.

Measuring a Resistor or Capacitor



RESISTOR MEASUREMENT

The input impedance of port C on the Wiz232 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 capacitance and resistance can be measured. Values from one pin cannot be compared to those from another pin because the transition points are not identical. The sequence used by the WiZ232-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 ranges from 200 Ohms to 10 MegaOhms using different capacitors for the desired range. We recommend against measuring resistors below 500 Ohms. For one, readings are less accurate than for those 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 need 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 resistors can replace in many instances 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 sulfide 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.

Period Measurement

Period measurement is done by using the GP (Get Period) function and uses the CIN pin of the WiZ232 controller in order to get a period reading from an external signal.

Syntax: <GP> (no parameters).

GP returns the period of the signal in microseconds. The maximum period frequency which can be measured is around 15 KHz (period = 66 µs) and 10 Hz (period = 100,000 µs) is the minimum. If the user tries to read the period of a signal whose frequency is lower than the minimum, or no signal is fed to the CIN pin, an error message will be sent (Error #7: Time out).

Stepper Motor Control

The WiZ232 features one stepper motor control port with all the necessary signals for controlling external drivers, with monophasic, biphasic and half-step stepping modes. For a diagram depicting the wave shapes for these three driving modes, refer to the Appendix. Acceleration and deceleration are available for controlling complex motion control. Stepping rates from 16 to 8,500 steps/sec are supported by the WiZ232.

In the application of the stepper motor function of the WiZ232, the following rules apply:

1) Steppers are controlled through the upper 4 bits of PC.

2) A configuration command must be sent in order to set the parameters for the motion of the stepper motor.

3) Enabling a stepper turns the upper 4 bits of PortC into outputs. Disabling a stepper leaves the corresponding pins as they were previously configured with the last value written to the stepper in the corresponding phantom port bits (see PORT Commands).

4) While stepping the PWM is inactive. If the PWM pin was set High (WH) or Low (WL) it will remain the same. If the PWM is in use while the stepper motor configuration command is sent, an error message (?C 'Timer not Available') is sent. The solution is to disable the PWM using either the WL or WH command.

When used for the control of stepper motors, the WZ232-A dedicates PORTC to handle the signals required for this function. Pins PC4 through PC7 are used for stepper motor phase outputs.

Four additional signals on PORTC: Pulse, Trigger, Direction and Pause, and a Stop signal on the IRQL (Interrupt pin) are also provided for greater functionality in controlling stepper motors with the WiZ232-A. These signals operate as follows:

1. Pulse (PC3): This is a hardware output signal which goes high for approximately 12uS every time a step is executed.
2. Direction (PC2): This hardware output signal is set to high (5V) for clockwise motion of the stepper motor or low (0V) for counter-clockwise movements.
3. Pause (PC1): This hardware input signal pauses the motion of the stepper motor when its input goes high. No further steps are taken until this signal is set to low (0V) once again. If this hardware pause signal is not to be used, then pin PC1 must be connected to ground using a 10 k Ohm resister.
4. Trigger (PC0): This hardware input signal is used to start the stepper motor. After sending a configuration command, the WZ232-A will read the Trigger input until a low pulse (0) is encountered. Upon reading '0', the WZ232-A triggers the first step of the stepper motor. For the trigger signal, the minimum pulse width is approximately 100uS. If this hardware only trigger is not used, then pin PC0 must be connected to ground using a 10 k Ohm resistor.
5. Stop (IRQL): This is also a hardware input signal which uses the Interrupt pin of the WZ232-A. During the movement of a stepper motor, if a negative edge signal (a transition from a high state (5V) to a low state (0V)) is sent to the IRQL input, the motion will be stopped and an ASCII 'S' will be sent to the host computer through the serial port.

   Note 1: A 'Stop' command can also be executed by software by sending the 'SS' command. If a software 'Stop' is executed before a valid interrupt signal is sent to the Interrupt pin, the hardware stop will not function.

   Note 2: Attempting to execute a PWCxx command while a motor is enabled will result in error ? C (Stepper enabled, do disable first).

The Direction and Pulse signals can be used for external driver output, and the Pause and Trigger signals for external control inputs.

Stepper Motor Configuration Command

In order to control a stepper motor, a configuration command with all the control parameters must be sent to the WZ232-A. The command syntax for configuring the stepper controller is:

<S> <Mode> <Direction> <;> < Steps to Go> <;> <Slew Rate> <;> <Acceleration/Deceleration>

<;> <Initial Rate>

Mode:	H [half step] B [full step - biphasic] M [full step – monophasic]
Direction:	+ [clockwise] - [counter-clockwise]
Steps to Go:	1 to 8 million steps
Slew Rate:	16 to 8500 steps/second
Accel/Decel:	0 to 255 steps/second^2
Initial Rate:	16 to 8500 steps/second

Note 1: There is no ';' inserted between the [Mode] and [Direction] parameters in the Configuration Command.

Note 2: The motion profile for acceleration and deceleration describes a trapezoid with the slew rate the plateau of the trapezoid. If the number of steps to go is insufficient to reach the desired speed, the slew rate is reduced until either the desired speed is achieved, or acceleration and deceleration are equally split within the number of steps to go available. In this second case, the desired speed may not be attained unless the acceleration/deceleration rate is increased.

For the initial configuration of the stepper motor, all the parameters must be given.

Other Stepper Control Related Commands

For increased functionality, other stepper motor control commands are available. These commands can be sent at any time after the configuration command has been sent.

SN Gets stepper position. By executing the SN command, the current position of the stepper motor is sent to the WZ232-A. This command can be used to provide an 'on-the-fly' position reading. The position register is a 24 bit register which thought of as a single axis bisected by a '0' point. A single clockwise step of the stepper motor from an initial position of '0' results in a position register value of '1'. A single counter-clockwise step from '0' will result in a position register value of 8,388,609. As a practical rule, any clockwise or positive movement increases the position register value closer to the value 8,388,607 and any counter-clockwise movement of the stepper motor increases the position register value toward the value 16,777,215. If the position value goes higher than 16,777,215, the position register will overflow and reset to '0'. If the position value goes higher than 8,388,607 the register will also reset to '0'.

In order to calculate the sign of the position value apply the following calculation:

For position values greater than 8,388,607 (which is the largest positive position):

Position = 8,388,608 – Current position value

For position values less than 8,388,607 no calculation is necessary. The value is positive and read as it is.

SR Reset stepper position. This command resets the position value to '0'. This command can only be executed while the motor is not moving.

SD Disable stepper motor. This shuts off all outputs to the external driver. Once this command is executed, the stepper configuration command and all the parameters must be sent again to operate the motor.

SS Stop stepping. This command is the software equivalent of a hardware stop.

S? Returns the current status of the motor. A returned value of '1' means that the motor is not running and can accept configuration commands, while a returned value of '0' indicates that the motor is running and cannot accept any configuration commands.

Using the WiZ232 Timer

Several functions make use of the 16 bit Timer Counter on the WiZ232. This resource can be only dedicated to perform a single task at a time. Because of this limitation, some functions can not be executed at the same time. This means that these tasks are exclusively executed. The functions that can not be executed in a 'shared mode' are:

- Stepper Motor Control
- PWM
- Analog Sampling

If any of these three is enabled, none of the others can be executed until the one which is being performed is finished or is disabled by the user.
For example, if an application requires that a stepper motor and PWM work at the same time, this represents a violation of the above mentioned. In this case, trying to enable the PWM while the stepper motor is running will cause an error message to be generated. (#C Timer not available).

Chapter 4 COMMANDS

Commands are always sent in ASCII format and they must be followed by a carriage return (CR or ASCII(#13)). Upper or lower case and spaces may be added for clarity (with the exception of the semicolon <;> which is reserved as a separator for command parameters). The last character sent back to the terminal is always the > prompt (this is useful from within a custom program to determine when the WiZ232-A is done with a command). An erroneous command returns ?n where ? indicates an error and n is the error code (from $1 to $F, 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).

As previously noted, when issuing a 1 byte number in a command, any of the three formats can be used. The default is Decimal. To send 1 byte Hexadecimal numbers to the WiZ232-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 format. In addition, some commands require that all arguments or parameters are passed in Decimal format.

Commands are arranged alphabetically; commands starting with P are port commands, S are stepper motor commands and so on. Entering 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 (unless the CRAP configuration is active in which case the command will be executed but no echo will be sent) and repeats its execution. If no previous command were issued @ has no effect.

<B>aud rate <9600>, <57600>,<115200> or <230400> Baud rate.

Procedure type.

Selects the Baud rate regardless of the Baud pin level. The command is acknowledged before the Baud rate changes and a prompt is also sent. This means that the next command must be sent to the WiZ232 using the new baud rate.

<C>onfigure <R>esults <A>SCII <B>inary or <D>ecimal or <H>exadecimal or <P>rogram. Configuring the default format for results

Procedure type.

The WiZ232-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 WiZ232-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 %, H or $ after a command).
* The following are disabled:
  Error messages (only ?n (n = error number) is sent to the terminal). ¤ PCp? (where p = A, B, C or S).
  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.
.

GA<Resolution><;><Channel> Get Analog Signal

Function type.

It returns a 3 digit number (0-255) for 8 bit ADC's and a 4 digit number (0-4095) for 10 & 12 bit ADC's.

<GP> (no parameters). Get Period of Signal

Function type.

GP returns the period of the signal in microseconds. The maximum period frequency which can be measured is around 15 KHz (period = 66 µs) and 10 Hz (period = 100,000 µs) is the minimum. If the user tries to read the period of a signal whose frequency is lower than the minimum, or no signal is fed to the CIN pin, an error message will be sent (Error #7: time out).

GS <ADCres><;><ChEnbMask><;><SampSpeed><;><Samples><;>{Gate} Get Sample

Function type.

<ADCres>: this is the resolution of the ADC: 8, 10 or 12 bit.
<ChEnbMask>: represents which channels are enabled for sampling: from 1 to 2047.
<SampSpeed>: this is the sampling Speed in samples/second. '0' represents the highest While '7' the lowest.
<Samples>: the number of samples to be taken: 1 to 65535.
{Gate}: add a G if software gating is desired

For the parameter ChEnbMask, in order to enable a channel for sampling, the user should use the following table:

Channel	Add to mask
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Sampling Mask Channels

Example:
For sampling channels #0 , #1 & #7 the Mask value would be:
Mask Value = 1+2+128 = 131

For the parameter SampSpeed use the following table to determine the correct sampling speed value:

SAMPLING RATES

Sampling Value	Sampling Period	Samples per second
0	400 µs	2500
1	500 µs	2000
2	1 ms	1000
3	2 ms	500
4	5 ms	200
5	10 ms	100
6	20 ms	50
7	50 ms	20

<P>ort <C>onfigure <A> or <B> or <C> or <D> <Value>. Configure Parallel ports PA, PB, PC and PD

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).

<P>ort <C>onfiguration <A> or <B> or <C> or <D> 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 to 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 (Most Significant Bit) first while others send the LSB (Least Significant Bit) 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 (PD4/PS_CK pin).
POL 0 = the clock idles Low.
POL 1 = the clock idles High.

Bit.1 Controls the clock phase (PD4/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 PD5/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 <B> 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 <B> or <C> or <D> {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 the 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.

Note: Attempting to execute a PWAxx command while a motor is enabled will result in error ?A (Stepper enabled, do disable first).

<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 PD5/PS_VDD must be tied High. Otherwise, the error message: "?B SPI requires pin PD5/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:

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 115.2 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.

b. Connecting a peripheral chip that requires both writing and reading to operate, e.g. the Motorola® MC145041 Analog/Digital converter.

<RESET> Resets the WiZ232-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 <Mode> <Direction> <;> < Steps to Go> <;> <SlewRate> <;> <Acceleration/Deceleration> <;> <Initial Rate> Stepper Configuration & Control

Process type

Mode:	H [half step] B [full step - biphasic] M [full step – monophasic]
Direction:	+ [clockwise] - [counter-clockwise]
Steps to Go:	1 to 8 million steps
Slew Rate:	16 to 8500 steps/second
Accel/Decel:	0 to 255 steps/second^2
Initial Rate:	16 to 8500 steps/second

Note 1: There is no ';' inserted between the [Mode] and [Direction] parameters in the Configuration Command.

<SN> Stepper Position

Function type

Gets stepper position. By executing the SN command, the current position of the stepper motor is sent to the WZ232-A. This command can be used to provide an 'on-the-fly' position reading. The position register is a 24 bit register which can be thought of as a single axis bisected by a '0' point. A single clockwise step of the stepper motor from an initial position of '0' results in a position register value of '1'. A single counter-clockwise step from '0' will result in a position register value of 8,388,609. As a practical rule, any clockwise or positive movement increases the position register value closer to the value 8,388,607 and any counter-clockwise movement of the stepper motor increases the position register value toward the value 16,777,215. If the position value goes higher than 16,777,215, the position register will overflow and reset to '0'. If the position value goes higher than 8,388,607 the register will also reset to '0'.

In order to calculate the sign of the position value apply the following calculation:

For position values greater than 8,388,607 (which is the largest positive position):

Position = 8,388,608 – Current position value

For position values less than 8,388,607 no calculation is necessary. The value is positive and read as it is.

If the stepper has not been previously configured, the use of this command will return an error #D (Stepper Motor not Enabled)

<SD> Disable stepper motor

Process type

This turns off all outputs to the external driver. Once this command is executed, the stepper configuration command and all the parameters must be sent again in order to operate the motor.

<SR> Reset stepper position.

Process type

This command resets the position register value to '0'.

<SS> Stepper stop

Process type

Stops the stepper motor and holds the state of the phase outputs. (i.e. it keeps the windings on) This command is the software equivalent of a hardware stop.

<S?> Get stepper motor status

Function type

Returns the current status of the motor. A returned value of '1' means that the motor is not running and can accept configuration commands, while a returned value of '0' indicates that the motor is running and cannot accept any configuration commands.

<W>idth <frequency>. Sets PWM frequency

Function type.

<frequency> can be any value between 15 and 15,000 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 WiZ232-A returns f=XXXXX (not available in the CRAP configuration). XXXXX is always a 5 digit long decimal number and is the integer part of the actual frequency the WiZ232-A is putting out (due to calculation rounding and timing restrictions). The actual frequency is given by the expression: Af = 921600 ÷ Round (921600 ÷ 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 for frequencies below 220 Hz, 1% or 99% duty cycles are possible.

<W>idth <frequency> <;> <Duty cycle> <;> <no. of pulses> Sets PWM duty cycle & output pulse counter

Procedure type

The same as above, but with this command the number of pulses to be generated by the PWM pin can be controlled. This number can be between 1 and 16 million. If the duty cycle parameter is not needed, then the syntax can be <W>idth <frequency> <;> <;> <no. of pulses>. When all the pulses have been generated, an ASCII 'P' is sent to the host computer signaling the end of the task.

<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.

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 Stepper enabled, do disable first
?B SPI requires pin PD3/SS always high, change and try again.

?C Timer not available.
?D Stepper motor not enabled.
?E Stepper running, do stop first.
?F Too many channels or speed too high.

Chapter 5 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, 232 TX,232 RX, BAUD	-	-	8	pF

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

PLEASE READ THIS CAREFULLY:

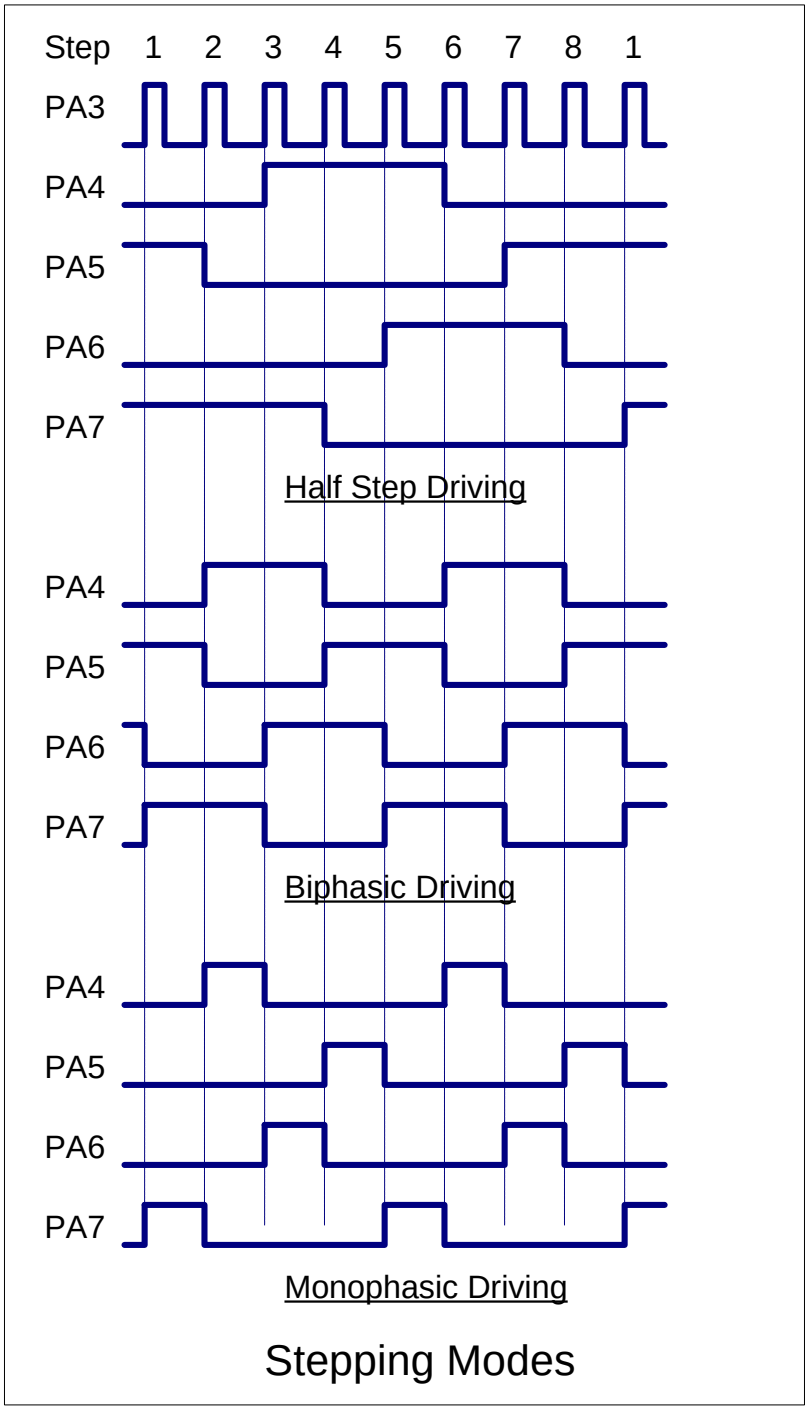
RMV ELECTRONICS INC. does not assume any liability arising from the application and/or use of the product/s described herein, nor does it convey any license rights. RMV ELECTRONICS INC. products are not authorized for use as components in medical, life support or military devices without written permission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

© 1999 RMV Electronics Inc.

#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174

APPENDIX A



#300 - 3665 Kingsway, Vancouver, BC, V5M 5W2, Canada
Phone: 604-299-5173 Fax: 604-299-5174