USB IO Controller Command Reference

Table of Contents

[1. Notes for ALL commands: 3](#_Toc330106179)

[2. Command Classification 3](#_Toc330106180)

[3. System Commands 3](#_Toc330106181)

[4. Query Commands 3](#_Toc330106182)

[5. Configuration Commands 4](#_Toc330106183)

[6. Function Commands 4](#_Toc330106184)

[7. The "R" Command 4](#_Toc330106185)

[8. The "MR" Command 4](#_Toc330106186)

[9. The "MW" Command 5](#_Toc330106187)

[10. The "V" Command 5](#_Toc330106188)

[11. The "S" Command 5](#_Toc330106189)

[12. The "I" Command 6](#_Toc330106190)

[13. The "PI" Command 7](#_Toc330106191)

[14. The "T" Command 7](#_Toc330106192)

[15. The "A" Command 8](#_Toc330106193)

[16. The "CU" Command 9](#_Toc330106194)

[17. The "C" Command: 9](#_Toc330106195)

[18. The "PD" Command 11](#_Toc330106196)

[19. The "PP" Command 11](#_Toc330106197)

[20. The "BC" Command (warn) 11](#_Toc330106198)

[21. The "F" Command: 12](#_Toc330106199)

[22. The "O" Command: 12](#_Toc330106200)

[23. The "PO" Command 13](#_Toc330106201)

[24. The "RC" Command (warn) 13](#_Toc330106202)

[25. Binary Output Commands 13](#_Toc330106203)

[26. The "BO" Command 14](#_Toc330106204)

[27. The "BS" Command 15](#_Toc330106205)

[28. Errors Messages: 15](#_Toc330106206)

# Notes for ALL commands:

* You end a command by sending a <CR> or <LF> or some combination of the two. This is how all commands must be terminated to be considered valid.
* The total number of bytes of each command, counting from the very first byte of the command name up to and including the <CR> at the end of the command must be 64 bytes or less. If it is longer than 64 bytes, the command will be ignored, and other bad things may or may not happen. This limitation will hopefully be removed in future FW D versions.
* You can string together as many commands as you want into one string, and then send that string all at once to the USB IO CONTROLLER. As long as each individual command is not more than 64 bytes, this will work well. By putting many commands together (each with their own terminating <CR>) and sending it all to the USB IO CONTROLLER at once, you make the most efficient use of the USB bandwidth.
* After successful reception of a command, the USB IO CONTROLLER will always send back an OK packet, which will consist of "OK<CR><LF>". For just testing things out with a terminal emulator, this is very useful because it tells you that the USB IO CONTROLLER understood your command. However, it does add extra communications overhead that may not be appreciated in a higher speed application. You can use the CU command to turn off the sending of "OK" packets. Errors will still be sent, but not any "OK" packets.
* Currently, the backspace key does not work. For example, if you are typing a command to the USB IO CONTROLLER from a terminal emulator and you make a mistake and try to backspace to correct your mistake, the USB IO CONTROLLER will not recognize the backspace and will generate an error. This will hopefully be correct in a future FW D version.
* All command names ("C", "BC", etc.) are case insensitive.
* All port names ("A", "B", "C") are case insensitive

# Command Classification

* 1. System commands – Changes the operation of USB IO controller in a system level
  2. Query commands – Simply reports status or values & does not change anything
  3. Configuration commands – Changes status or operation modes. Used by subsequent commands
  4. Function commands – Directly acts on the ports and produces various outputs

# System Commands

1. R – Resets the system. All values are set to default as if system is powered ON
2. MR – Read memory location of USB IO controller internal RAM
3. MW – Write memory location of USB IO controller internal RAM

# Query Commands

1. V – Getting FW version
2. S – Getting status of USB IO controller, global variables, port usage status and functions running
3. I – Read & report all port pin value.
4. PI – Read & report a port pin value one at a time.
5. T – Read port values in digital or analog (if analog configured) form as configured on periodic basis automatically till stopped.
6. A – Sample analog inputs. Analog input should have been already configured.

# Configuration Commands

1. CU – Configure USB IO command – Global parameter setting
2. C – Configure all Port & all Pin direction in one go
3. PD – Configure Port Pin direction one at a time
4. PP – Change Port protection status. All Function Commands run in a Port/Pin sets a protection bit so that the Port/Pin does not get changed by subsequent Function Command issued to the same Port/Pin till it is closed or status changed
5. BC – Bulk configure command. Requires one port fully and two more pins (strobe & busy) in another port for parallel bulk transfer

# Function Commands

1. O – Set Port output value for all Port & Pin at one go
2. PO – Set Port/Pin value one at a time
3. PT – Port B, Pins 0-2 as Channel 1-3 special timer operation
4. F – Give a frequency output in a specific one. At a time can run in only one pin
5. RC – Set RCServo output in Port/Pin
6. BO – Bulk output at Port B using strobe & busy at Port A configured using BC command
7. BS – Bulk stream using BC command

# The "R" Command

* The "R" Command stands for 'Reset to default state' and when you send the USB IO CONTROLLER an "R" command it will initialize all pins to digital inputs and stop any running timers.
* **Format:** "R"
* **Return Packet:** "OK"

# The "MR" Command

* The "MR" Command stands for 'Memory Read'.
* **Format:** "MR,<Address><CR>"
* **<Address>:** This is a number between and including 0 to 4095. It is the address in the USB IO CONTROLLER's RAM that you wish to read.
* **Example:** "MR,3968" (asks the USB IO CONTROLLER to read the value in the PORTA register)
* **Return Packet:** "MR,<Value>"
* **<Value>:** This is a number between and including 0 to 255. It is the result of reading <Address>.
* **Example Return Packet:** "MR,28"

# The "MW" Command

* The "MW" Command stands for 'Memory Write'.
* **Format:** "MR,<Address>,<Value><CR>"
* **<Address>:** This is a number between and including 0 to 4095. It is the address in the USB IO CONTROLLER's RAM that you wish to write into.
* **<Value>:** This is a number between and including 0 to 255. It is the value you wish to write into <Address>.
* **Example:** "MW,3968,56" (asks the USB IO CONTROLLER to write the value 56 into the PORTA register)
* **Return Packet:** "OK"
* **NOTE:** This command can be extremely dangerous unless you read the PIC datahseet and understand what you are doing. With this command you have the ability to write over all of the current RAM in the USB IO CONTROLLER - including all variables that the firmware is using, and all of the Special Function Registers in the PIC. It can be a very handy thing, but use it with caution.

# The "V" Command

* The "V" Command stands for 'Version' and when you send the USB IO CONTROLLER an "V" command, it will respond with a text string that looks something like this: "USB IO CONTROLLER FW Version - 1.4.0"
* **Format:** "V"
* **Return Packet:** "USB IO CONTROLLER FW Version - 1.4.0"

# The "S" Command

* The "S" Command stands for 'Status' and when you send the USB IO CONTROLLER an "S" command, it will respond with a series text string that looks something like this:   
  USB IO Controller FW Version - 1.2.7.9 -> Firmware Version

Up time - 00027:01 -> Uptime in minutes:seconds

Command Ack status – 1 -> To send OK packet or not

Port use protect status – 1 -> To protect used ports globally

Port A mode – 00000000 -> 0-Ouput, 1-Input

Port B mode – 00000000 -> 0-Ouput, 1-Input

Port C mode – 11000100 -> 0-Ouput, 1-Input

Port A use status – 00000000 -> 0- port not in use 1- in use

Port B use status – 00000000 -> 0- port not in use 1- in use

Port C use status – 00000000 -> 0- port not in use 1- in use

Detailed Port use details:

PORT A:

0 - 00

1 - 00

2 - 00

3 - 00

4 - 00

5 - 00

6 - 00

7 - 00

PORT B:

0 - 00

1 - 00

2 - 00

3 - 00

4 - 00

5 - 00

6 - 00

7 - 00

PORT C:

0 - 00

1 - 00

2 - 00

3 - 00

4 - 00

5 - 00

6 - 00

7 - 00

* **Format:** "S"
* **Return Packet:** As above

# The "I" Command

* The "I" Command stands for 'Input state' and when you send the USB IO CONTROLLER an "I" command, it will respond with an "I" packet back that will hold the value of each bit in each of the three ports A, B and C. It reads the state of the pin, no matter if the pin is an input or an output. If the pin is configured as an analog input, the bit will always read low (0) in the "I" packet.
* **Format:** "I<CR>"
* **Example:** "I"
* **Return Packet:** "I,<StatusA>,<StatusB>,<StatusC><CR>" where <StatusX> is a number from 0 to 255 that indicates the current value of the pins on that port. Note that <StatusX> will always be 3 characters long, which means that leading zeros will be added so that the return packet is always the same length regardless of the data values.
* **Example Return Packet:** "I,001,045,205"

# The "PI" Command

* The "PI" command stands for "Pin Input". It allows you to read the state of just one pin at a time. (High or Low)
* **Format:** "PI,<Port>,<Pin><CR>"
* **<Port>:** This is the character "A", "B", or "C" depending upon which port you want to change.
* **<Pin>:** This is a number between and including 0 to 7. It indicates which pin in the port you want to change the direction on.
* **Example:** "PI,C,6" - This would read the state of Port C pin 6.
* **Return Packet:** "PI,<Value>"
* **<Value>:** This is either a High (1) or a Low (0) depending upon the voltage on the pin at the time it was read.
* **Example Return Packet:** "PI,1" (Means that the pin was high.)

# The "T" Command

* The "T" Command stands for 'Timer read inputs' and when you send the USB IO CONTROLLER an "T" command, it will set the delay for one of two timers. When the timer times out, it will cause an "I" packet or "A" packet response to get sent to the PC.
* **Format:** "T,<TimeBetweenPacketsInMilliseconds>,<Mode><CR>"
* **<TimeBetweenPacketsInMilliseconds>:** The time between response packets is determined by the <TimeBetweenPacketsInMilliseconds> value, and is expressed as a number between (and including) 1 and 30000 . If you send a 10 for <TimeBetweenPacketsInMilliseonds> then a new packet response would be sent every 10ms. If you sent a value of 30000, then it would send a packet response every 30 seconds. If you want to turn off either timer so that no more packets are sent, send a <TimeBetweenPacketsInMilliseconds> of zero. You have to send a time of zero for both "I" and "A" timers if you want them both to turn off. **Note**: just because the UWB can kick out I and A packets every 1ms (at its fastest) doesn't mean that your PC app can read them in that fast. Some terminal emulators are not able to keep up with this data rate coming back from the USB IO CONTROLLER, and what happens is that the USB IO CONTROLLER's internal buffers overflow. This will generate error messages being sent back from the USB IO CONTROLLER. If you write your own custom application to receive data from the USB IO CONTROLLER, make sure to not read in one byte at a time from the serial port - always ask for large amounts (10K or more) and then internally parse the contents of the data coming in. (Realizing that the last packet may not be complete.) **Note 2**: It has been discovered that if an attempt is made to have all 13 channels of analog be reported any faster than every 4ms, then an internal USB IO CONTROLLER buffer overflow occurs. Be careful with the speed you choose for A packets. The maximum speed is based upon how many analog channels are being sent back.
* **<Mode>:** If <Mode> is "0" then the "I" packet timer (digital input packet) is set and will generate "I" packets back to the PC. If <Mode> is "1" then the "A" packet timer (analog input packet) is set and will generate "A" packets back to the PC. Both timers can be active and sending back their respective packets at different (or the same) rates.
* **Note**: The USB IO CONTROLLER is actually sampling the digital input pins at an extremely precise time interval of whatever you sent in the T command. The values of the pins are stored in a buffer, and then packet responses are generated whenever there is 'free time' on the USB back to the PC. So you can count the I packet responses between rising or falling edges of pin values and know the time between those events to the precision of the value of <TimeBetweenPacketsInMilliseconds>. This is true for <Mode>=0. For <Mode>=1, the analog inputs are sampled every 1ms. Each time the "A" timer times out, the latest set of analog values is used to create a new "A" packet and that is then sent out.
* **Example:** "T,100,0" - this would send back 10 "I" packets per second, sampled every 100ms.
* Example: "T,14,1" - this would sample all enabled analog inputs and send back an "A" packet every 14ms.
* **Return Packet:** "OK". Note however, the "I" packet or "A" packet responses will start flowing at regular intervals after the T command is received by the UWB.
* **Note:** If the "I" or "A" packet responses stop coming back after you've done a "T" command, and you didn't stop them yourself (with a "T,0,0" or "T,0,1") then what's happened is that the internal buffer in the USB IO CONTROLLER for I or A packet data has been filled up. (There is room for 3 I packets and 3 A packets.) This means that the USB system is too busy to get the packet responses back to the PC fast enough. You need to have less USB traffic (from other devices) or increase the time between packet responses.

# The "A" Command

* The "A" Command stands for 'Sample Analog Inputs'. When you send the "A" packet to the USB IO CONTROLLER, it will send back the last sampled set of analog inputs. All enabled analog inputs are sampled every 1ms, and stored. Whenever an "A" packet is received, the latest stored value for the analog inputs is sent back in a returning "A" packet.
* **Format:** "A<CR>"
* **Example:** "A"
* **Return Packet:** "A,0145,1004,0000,0045" (The return packet would look like this if there were 4 analog inputs enabled with the "C" command). There can be up to 12 analog input enabled, and thus there might 12 numbers between 0 and 1023 after the "A,". See the chart in the "C" command above for information on which analog inputs correspond to which pins. The numbers represent the analog voltage on each enabled analog input from 0V (0000) to 5V (1023). The first number after the "A," is for AN0, and the last number is for the highest analog input channel (ANx) that is currently enabled.

# The "CU" Command

* The "CU" command stands for "Configure USB IO CONTROLLER". It is designed to be a generic command for setting things that affect the general operation of the USB IO CONTROLLER.
* **Format:** "CU,<Parameter>,<Value><CR>"
* **<Parameter>:** This is an unsigned 8 bit value, representing the parameter number you wish to change. (See table below)
* **<Value>:** This is a value who's meaning depends upon the <Parameter> number chosen.
* **Example:** "CU,1,0" - This would turn off the sending of the "OK" packets after each command.
* **Return Packet:** "OK"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | <Parameter> | <Value> | <Value> meaning |
| Ack packet enable/disable | 1 | 0 or 1 | 0 = Turn OFF "OK" packets 1 = Turn ON "OK" packets (default) |
| Port already in use warning | 2 | 0 or 1 | 0 – Do not bother whether a Port/Pin is already used by another running command or not and execute a given command  1 – If a Port/Pin is already used, do not allow it to change and report “Port in use”. Use PP command to individually override |
| ON pulse duration in PT command Channel 2 | 3 | 0 or 1 to 990 | 0 – Pulse duration is set to default 250 milli seconds  1-990 – Pulse duration is set to the given number of milli seconds |
| ON pulse duration in PT command Channel 3 | 4 | 0 or 1 to 990 | 0 – Pulse duration is set to default 250 milli seconds  1-990 – Pulse duration is set to the given number of milli seconds |

# The "C" Command:

* The "C" command stands for 'Configure' and allows you to set the state of the port direction registers for ports A, B and C, as well as enable analog inputs. This allows you to turn each pin into an input or an output on a pin by pin basis, or enable one or more of the pins to be analog inputs.
* **Format:** "C,<DirA>,<DirB>,<DirC>,<AnalogEnableCount><CR>" where <DirX> is a value between 0 and 255 that indicates the direction bits for that port. A 1 is an input, a 0 is an output.
  + <AnalogEnableCount> If this value is sent as a zero, then all analog inputs are turned off and all of the pins behave as just digital inputs or outputs.
  + If <AnalogEnableCount> is sent as a value from 1 to 13, then one or more of the analog inputs (see below) are enabled and will start sampling every millisecond. Any value over 13 is an error. Use the "A" command to retrieve the values of the enabled analog inputs.
* **Example:** "C,4,245,52,0"
* **Warning on Analog Inputs:** It is very important that if you enable an analog input on a pin, that you set that pin as an input (set the proper <DirX> bit). If you have a pin set as an output in the <DirX> bit but have it enabled as an analog input, your analog reading will simply convert the current digital output voltage on the pin (which may be what you want, but probably not). Also, if you have a pin set as a digital output with the <DirX> bit, but DON'T enable it as an analog input and then apply analog input levels to the pin, that pin may draw excessive power because it may float around between a High (5V) and Low (0V). PICs are very robust, but be careful.
* **Analog Input Matrix:** 
  + **If one or more of the analog inputs are enabled, then use this chart to see which number for <AnanlogEnableCount> enables which pins as analog inputs. The PIC core that all of the USB PICs are built on has 13 analog input channels. However, on the 28 pin parts (which are currently used in USB IO CONTROLLER designs) three of those analog inputs (AN5,AN6 and AN7) do not come out as pins. They still exist, and as far as Firmware D v1.4.4 is concerned, are treated exactly as all of the other analog input pins, so the show up in the "A" packet. What this means is that you can get up to 10 separate analog inputs on a USB IO CONTROLLER.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <AnalogEnableCount> value | AN12 RB0 | AN11 RB4 | AN10 RB1 | AN9 RB3 | AN8 RB2 | AN7 N.A. | AN6 N.A. | AN5 N.A. | AN4 RA5 | AN3 RA3 | AN2 RA2 | AN1 RA1 | AN0 RA0 |
| 0 | D | D | D | D | D | D | D | D | D | D | D | D | D |
| 1 | D | D | D | D | D | D | D | D | D | D | D | D | A |
| 2 | D | D | D | D | D | D | D | D | D | D | D | A | A |
| 3 | D | D | D | D | D | D | D | D | D | D | A | A | A |
| 4 | D | D | D | D | D | D | D | D | D | A | A | A | A |
| 5 | D | D | D | D | D | D | D | D | A | A | A | A | A |
| 6 | D | D | D | D | D | D | D | A | A | A | A | A | A |
| 7 | D | D | D | D | D | D | A | A | A | A | A | A | A |
| 8 | D | D | D | D | D | A | A | A | A | A | A | A | A |
| 9 | D | D | D | D | A | A | A | A | A | A | A | A | A |
| 10 | D | D | D | A | A | A | A | A | A | A | A | A | A |
| 11 | D | D | A | A | A | A | A | A | A | A | A | A | A |
| 12 | D | A | A | A | A | A | A | A | A | A | A | A | A |
| 13 | A | A | A | A | A | A | A | A | A | A | A | A | A |

* **Return Packet:** "OK"

# The "PD" Command

* The "PD" command stands for "Pin Direction". It allows you to set the direction on just one pin at a time. (Input or Output)
* **Format:** "PD,<Port>,<Pin>,<Direction><CR>"
* **<Port>:** This is the character "A", "B", or "C" depending upon which port you want to change.
* **<Pin>:** This is a number between and including 0 to 7. It indicates which pin in the port you want to change the direction on.
* **<Direction>:** This is either "0" or "1", for Output (0) or Input (1).
* **Example:** "PD,B,2,1" - This would change Port B, pin 2 to an input.
* **Return Packet:** "OK"

# The "PP" Command

* The "PP" command stands for "Pin Protect". It allows you to protect the status of Port/Pin from Function Command to safe guard the already running Function commands at the given Port/Pin. By default all Function commands protects the specific Port/Pin. So the use of this command is normally to unprotect a Port/Pin. Another way looking at this command is like Busy(1)/Free(0). This is what reported in “S” command as Port busy status.
* **Format:** "PP,<Port>,<Pin>,<Value><CR>"
* **<Port>:** This is the character "A", "B", or "C" depending upon which port you want to protect/un protect.
* **<Pin>:** This is a number between and including 0 to 7. It indicates which pin in the port for which you want to set the state.
* **<Value>:** This is either "0" or "1", for do not protect (0) or protect (1).
* **Example:** "PP,A,3,0" - This would make Port A pin 3 un protected.
* **Return Packet:** "OK"

# The "BC" Command (warn)

* The "BC" command stands for "Bulk Configure". It allows you to configure the options for the BO and BS commands.
* **Format:** "BC,<Init>,<WaitMask>,<WaitDelay>,<StrobeMask>,<StrobeDelay>,<CR>"
* **<Init>:** This is the inital value written to PortA.
* **<WaitMask>:** Each bit that is set in this mask indicates a 'busy' bit coming back from the LCD (or other hardware). This value is only used if <WaitDelay> is not zero.
* **<WaitDelay>:** The <WaitDelay> is the maximum amount of time to wait for the busy bit to become asserted, and then to become de-asserted. If <WaitDelay> expires, then the next byte is just sent out. <WaitDelay> is in units of about 400ns.
* **<StrobeMask>:** Each bit that is set in this mask indicates the strobe bits that are to be inverted after the byte is written to PortB. When <StrobeDelay> is over, the initial value (<Init>) is written back to PortA.
* **<StrobeDelay>:** The length of time that the strobe bits (from <StrobeMask>) are inverted from their intial values. <StrobeDelay> is in units of about 830ns.
* **Example:** "BC,1,1,1,1,1"
* **Return Packet:** "OK"

# The "F" Command:

* The "F" command stands for 'Frequency output' and allows you to output a square wave at a particular frequency on any pin. Only one pin may have this frequency output at a time.
* **Format:** "F,<Frequency>,<Port>,<Pin>,<Percentage><CR>"
  + <Frequency> The frequency in Hz.
  + <Port> A port name A, B, or C.
  + <Pin> A pin number, from 0 to 7
  + <Percentage> (optional) A number, from 1 to 99, representing the percentage of each cycle that the pin should stay high for (basically turning this into a PWM command)
* To turn off the frequency output, use a frequency value of zero. The <Port> and <Pin> paramters are optional if <Frequency> is zero.
* This command will automatically set the specified pin to be an output. If the pin was previously an analog input, you must set it to digital mode (using the "C" command below) in order for the F command to work properly.
* The usable frequency range is from 1HZ to 30KHz. The output is very accurate (better than 0.5%) from 1Hz to about 15Khz, then it starts to get a little bit worse as you go higher in frequency.
* **Example:** "F,7621,B,2" - this will output a square wave of frequency 7621Hz on PortB pin 2.
* **Return Packet:** "OK"

# The "O" Command:

* The "O" command stands for 'Output state' and will take the values you give it and write them to the port A, B and C data registers. This allows you to set the state of all pins that are outputs.
* **Format:** "O,<PortA>,<PortB>,<PortC><CR>" where <PortX> is a value between 0 and 255 that indicates the value of the port pins for that register.
* **Example:** "O,0,255,22"
* **Return Packet:** "OK"

# The "PO" Command

* The "PO" command stands for "Pin Output". It allows you to set the output value (if it is currently set to be an output) on just one pin at a time. (High or Low)
* **Format:** "PO,<Port>,<Pin>,<Value><CR>"
* **<Port>:** This is the character "A", "B", or "C" depending upon which port you want to set.
* **<Pin>:** This is a number between and including 0 to 7. It indicates which pin in the port for which you want to set the state.
* **<Value>:** This is either "0" or "1", for Low (0) or High (1).
* **Example:** "PO,A,3,0" - This would make Port A pin 3 low.
* **Return Packet:** "OK"

# The "RC" Command (warn)

* The "RC" command stands for "RC Servo Output". It will turn any pin into an RC servo output, if that pin is already configured as a digital output.
* **Format:** "RC,<Port>,<Pin>,<Value><CR>"
* **<Port>:** This is the character "A", "B", or "C" depending upon which port you want to set.
* **<Pin>:** This is a number between and including 0 to 7. It indicates which pin in the port for which you want to set the state. Note that some pins do not come out of the chip (RA6, RA7, RC3, RC4 and RC5), and some pins are not accessible via headers. You can still set RC outputs on those pins, but the non-existent ones will just be skipped by the RC code, and if you set RC outputs on RC0, RC1 or RC2, you may see interesting results (since RC0 and RC1 have LEDs on them).
* **<Value>:** This is a value between 0 and 11890.
  + A <Value> of 0 (zero) will turn the RC output (for that pin) completely off. A <Value> of 1 will cause a 1ms high pulse on the pin. A <Value> of 11890 will cause a 2ms high pulse on the pin. Any <Value> inbetween 1 and 11890 will cause a high pulse whose duration is proportionally between 1ms and 2ms. These pulses repeat every 19ms.
* **Example:** "RC,B,3,5945" - If PortB pin 3 was already an output, then there would be a 1.5ms (which is 'center' to an RC servo) high pulse coming out of PortB pin 3 every 19ms.
* **Return Packet:** "OK"

1. Binary Output Commands  
   The "BC", "BO" and "BS" commands all work together to allow for high speed parallel output to a hardware device like an LCD panel or other latched 8-bit parallel interface. The basic idea is to take a byte, write it out to PortB, set a strobe bit on PortA, wait a bit, then clear the strobe bit on PortA, then wait for a busy bit to go high (or low) on PortA, then wait for the busy bit to go low (or high) on PortA and then repeat for as many bytes as there are to send out PortB. So PortB is used as the output to the parallel bus and two bits on PortA are used as a strobe bit (output) and a busy bit (input).  
     
   The BC command sets up all of the parameters, and then the BO or BS commands stream the data out PortB. Before this scheme will be very successful, make sure to set the direction bits on PortB and PortA properly.

# The "BO" Command

* The "BO" command stands for "Bulk Output". It uses the settings from the BC command and outputs bytes to PortB. PortA bits are used as control bits, with at least one used as a strobe output (indicating that a new byte is present on PortB) and an optional second PortA bit used as a 'busy' input to prevent the next byte from being sent out before the receiver is able to process it.
* **Format:** "BO,<ASCII\_HEX\_Bytes><CR>"
* **<ASCII\_HEX\_Bytes>:** This group of characters are the hexadecimal representation of the bytes that you whish to send out PortB. For example, if you wanted to send 3 bytes, of values 0x55, 0xA7 and 0x21, you would use "BO,55A721" as your command. The total length of this command must not be more than 63 bytes, so at most you can pack 30 bytes worth of ASCII hex characters into this command.
* The way that the bytes get sent out PortB is by using the WaitMask, WaitDelay, StrobeMask, StrobeDelay and Init values from the BC command above. When the BC command is accepted, PortA will be initalized to the value in the <Init> parameter. Note that you must have already set up PortB as outputs and have the direction of each bit in PortA set up according to what you need. When it is time to output a byte (with the BO or BS commands), the chain of events is as follows:
  + The byte is output on PortB.
  + The bits that are high in the <StrobeMask> are inverted on PortA.
  + A delay is executed for <StrobeDelay> units (each unit is about 830ns)
  + The PortA is then returned to the <Init> value.
  + If <WaitDelay> is greater than zero,
    - Wait (up to <WaitDelay> units) for the busy bit to become the state it is in <Init>
    - Wait (up to <WaitDelay> units) for the busy bit to become the inverse of what it is in <Init>
  + Repeat
* **Example:** "BO,55A721"
* **Return Packet:** "OK"

# The "BS" Command

The "BS" command stands for "Bulk Stream". It uses the settings from the BC command and streams raw binary bytes to PortB, just like the "BO" command does.

* **Format:** "BC,<ByteCount>,<BinaryStreamOfBytes><CR>"
* **<ByteCount>:** This is the number of bytes in <BinaryStreamOfBytes>. It must be an exact number, because this his how the USB IO CONTROLLER knows when the end of <BinaryStreamOfBytes> is and when to begin looking for the <CR>. The range of acceptable values for <ByteCount> are from 1 to 56 inclusive.
* **<BinaryStreamOfBytes>:** Must be exactly <BinaryStreamOfBytes> bytes long. This stream is the \_binary\_ (NOT ASCII) bytes that you want to send out PortB. In other words, if you wanted to send three bytes who's values are 0x23, 0x49 and 0x6A then the three bytes in <BinaryStreamOfBytes> would be "#Ij" and <ByteCount> would be 3. Note that since this is pure binary, it does not suffer from the lower bandwidth utilization that the "BO" command has (where each byte to be output is represented by two ASCII HEX bytes in the command string). But it is also more difficult to use because many of the characters inside the binary stream are not printable, and/or are difficult to generate with a terminal emulator.
* Note that this comand now (as of 1.4.8) has special processing rules applied to it when it is being parsed. This means that if you are typing the BS command, as soon as you start typing the binary data section, you can no longer use things like the backspace key to edit what you type (duh) or the up arrow key to bring back the previous command, etc. The USB IO CONTROLLER will continue to accept bytes of data and put them in the buffer until the number of bytes in the <ByteCount> paramter have been seen, then normal command parsing rules return.
* **Example:** "BS,3,#Ij"
* **Return Packet:** "OK"

# Errors Messages:

There are two (or more) scenarios that one might use a USB IO CONTROLLER:

1. By typing commands into a terminal emulator on a computer, to 'test out' commands and how the system is working.
2. By writing a computer program that will automatically generate commands to send to a USB IO CONTROLLER.

The long error messages are very useful for debugging the system, and especially when using the USB IO CONTROLLER by hand from a terminal emulator. The long messages are not as useful when running under scenario 2) above, as the PC application has a much harder time parsing the long error messages.  
  
To help make the error messages useful in both scenarios, each error message starts out with an exclamation mark "!" and then is immediately followed by an integer error number, then a space, and then the long text of the error message with a <CR><LF> at the end. This means that if your PC application wants to parse the error message, it can look in the data coming back from the USB IO CONTROLLER for the exclamation mark "!" and then read in the error number and ignore everything else until the next <CR><LF>.  
  
Error Message List:

* "!0" (unused)
* "!1 Err: Port Pin already under use"
* "!2 Err: TX Buffer overrun"
  + This error is generated if, for some reason, the internal code of the USB IO CONTROLLER tries to send too much back to the PC at once, and the internal transmit buffer back to the PC overflows.
* "!3 Err: RX Buffer overrun"
  + This error is generated if, while the USB IO CONTROLLER is receiving data from the PC, the internal receive buffer is overfilled.
* "!4 Err: Missing parameter(s)"
  + The USB IO CONTROLLER will send back this error if it expected to find another parameter in the command, but instead found a <CR> or <LF>.
* "!5 Err: Need comma next, found: '<some\_char>'"
  + The USB IO CONTROLLER will send back this error if it expected to find a comma, but found something else instead. The <some\_char> will be the character it found instead of the comma.
* "!6 Err: Invalid parameter value"
  + This error means that the USB IO CONTROLLER found a parameter, but its value was outside of the acceptable range for that particular parameter.
* "!7 Err: Extra parameter"
  + This error indicates that the USB IO CONTROLLER expected to see a <CR> or <LF> command terminator, but instead found an extra comma or extra parameter.
* "!8 Err: Unknown command '<command\_chars>'"
  + This error indicates that the single or double byte command name was not understood or doesn't exist. <command\_chars> will be the one or two bytes that the USB IO CONTROLLER received that did not match any know commands.