Documentation UBW32 Firmware version 1.6.3 for UBW32 Boards

Back to main UBW32 page

Description:
The firmware (application) that comes with the UBW32 gives the user a basic set of input/output commands. For example, the user can set any I/O pin to be an input or output, read the inputs, write to the outputs, etc. Future versions of the firmware will allow more complicated commands (like analog input, timing related commands, PWM, SPI, USART, etc.)

Version Notes:

- Version 1.0
  - First publicly released version, and the version that SparkFun is shipping on their first run (v2.4) of boards.
- Version 1.1 (bug fixes, no new features, released 2/16/09)
  - Fixed pin number limitation on PD, PI and PO command - allow pin values up to 31 now
  - Fixed port offset bug in C command - now able to configure port A properly (as well as the others)
- Version 1.2 (bug fix, no new features, released 9/07/09)
  - Fixed problem with location of SoftwareKey variable. Needed a "." in section name to correctly allocate at 0xA0000000. BL command now works better.
- Version 1.3 released 8/26/09
  - Edited project files to use new version of Microchip USB stack, version 2.5a. No new functionality, other than the bug fixes present in the new USB stack.
- Version 1.4 released 11/01/10
  - Added T2 testing command for enhanced testing at SparkFun
- Version 1.5 internal release only 11/01/10
  - Added PM command (PWM output)
- Version 1.6 internal release only 01/08/11
  - Added SP command to allow for software PWM from 100Khz ISR
- Version 1.6.2 released 09/02/11
  - Updated Software_Key variable so it works with the new C32 v2.00 compiler. From now on, we will always need to use this compiler (or above) with this firmware.
  - Updated extract_number() to use 32-bit signed and unsigned ints (called longs) aswell as floating point values (using sscanf()).
  - Tested with USB stack 2.9a, MPLAB 8.76, C32 v2.01
- Version 1.6.3 released 09/05/11
  - Andrew Prudil added CA and IA commands, for configuring and reading analog inputs. See below.
  - Updated on 11/30/11 - added projects for both MPLAB X and MPLAB 8 in the same project, compiled with C32 v2.01, MAL USB stack 2.9b

Commands:

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.
- You can string together as many commands as you want into one string, and then send that string all at once to the UBW32. 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 UBW32 at once, you make the most efficient use of the USB bandwidth.
- After successful reception of a command, the UBW32 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 UBW32 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.
- All command names ("C", "BC", etc.) are case sensitive. In other words, "BC" is a different command from "bc".
- All port names ("A", "B", "C") are case insensitive. You can use "B" or "b" for port names.

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>,<DirD>,<DirE>,<DirF>,<DirG><CR>" where <DirX> is a value between 0 and 65,535 that indicates the direction bits for that port. A 1 is an input, a 0 is an output.
- **Example:** "C,0,0,0,0,65535,0,0" - This would set all ports as outputs except port E which would be all input
- **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>,<PortD>,<PortE>,<PortF>,<PortG><CR>" where <PortX> is a value between 0 and 65,535 that indicates the value of the port pins for that register.
- **Example:** "O,0,0,131,0,0,0,0"
- **Return Packet:** "OK"

The "I" Command

- The "I" Command stands for 'Input state' and when you send the UBW32 an "I" command, it will respond with an "I" packet back that will hold the value of each bit in each of the seven ports A, B, C, D, E, F and G. It reads the state of the pin, no matter if the pin is an input or an output.
- **Format:** "I<CR>"
- **Example:** "I"
- **Return Packet:** "I,<StatusA>,<StatusB>,<StatusC>,<StatusD>,<StatusE>,<StatusF>,<StatusG><CR>" where <StatusX> is a number from 0 to 65,535 that indicates the current value of the pins on that port. Note that <StatusX> will always be 5 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,50943,64479,24606,65535,01015,12607,62403"

The "V" Command

- The "V" Command stands for 'Version' and when you send the UBW an "V" command, it will respond with a text string that looks something like this: "UBW32 Version 1.0"
- **Format:** "V<CR>"
- **Return Packet:** (example) "UBW32 Version 1.1"

The "R" Command

- The "R" Command stands for 'Reset to default state' and when you send the UBW32 an "R" command it will initialize all pins to digital inputs.
- **Format:** "R<CR>"
- **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, C, D, E, F or G depending upon which port you want to change.
- **<Pin>:** This is a number between and including 0 to 31. 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 "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, C, D, E, F or G depending upon which port you want to change.
- **<Pin>:** This is a number between and including 0 to 31. 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 "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, C, D, E, F or G depending upon which port you want to set.
- **<Pin>:** This is a number between and including 0 to 31. 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:** "PD,A,3,0" - This would make Port A pin 3 low.
- **Return Packet:** "OK"

The "CU" Command

- The "CU" command stands for "Configure UBW32". It is designed to be a generic command for setting things that affect the general operation of the UBW32.
- **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"

<Parameter>	<Value>	<Value> meaning
1	0 or 1	0 = Turn off "OK" packets 1 = Turn on "OK" packets (default)

The "BL" Command : Enter Boot Loader mode

- The "BL" command stands for "Boot Loader". When you issue a BL command, the UBW32 will reset itself into bootloader mode and wait for a download from the HID Bootloader application.
- **Format:** "BL<CR>"
- **Example:** "BL"
- **Return Packet:** none

The "T1" Command : Execute Test 1

- The "T1" command stands for "Run Test 1". It set all pins to be digital outputs, and set them all high. It will then pull each one down, in turn, for the length of time specified. It will start with A15, and progress clockwise around the UBW to D8, then cycle back around to A15. It will repeat this the number of times specified. The purpose of this command is to test all 78 I/Os, and show that they are all connected properly to the CPU, and that there are no opens or shorts. If you put an LED through a resistor to +3.3V on each I/O pin, you'll end up with a nice Knight Rider effect.
- **Format:** "T1,<duration>,<iterations><CR>"
- **<Duration>:** This is an unsigned 16 bit integer, and represents the time, in milliseconds, that each I/O will be pulled low
- **<Iterations>:** This is an unsigned 16 bit integer, and represents the number of times to repeat the pattern
- **Example:** "T1,200,4" - This would run the pattern four times, with each pin going low for 200ms.
- **Return Packet:** "OK"

The "T2" Command : Execute Test 12

- This command assumes a proper test jig, where certain pins are shorted together (Contact Schmalz Haus for details if you need to use this command). It checks for opens and shorts on as many pins as can be tested. For example, RD8 and RD10, RD9 and RD11, and so on around the UBW32's pins

  Procedure:

1) Set every other pair of pins to be inputs, every other pair to be outputs.
Example: RD8 = Out, RD9 = Out, RD10 = In, RD11 = In, etc.
2) Set all outputs low
3) Set RD8 high, and check that RD10 is high.
4) Set RD8 low, and check that RD10 is low.
5) Repeat steps 3 and 4 for each pair of connected pins
6) Wait for up to 30 seconds for the user to push both PRG and USER buttons
7) Output a response - either "PASS", or "FAIL - "<some error>

- **Format:** "T2<CR>"
- **Example:** "T2"
- **Return Packet:** "OK"

The "PM" Command : Set hardware PWM output values command

- Sets a PWM value for any of the 5 PWM hardware channels
- **Format:** "PM,<Channel>,<DutyCycle><CR>"
- **<Channel>** is required and is 1 through 5, which correspond to the output compare (OC0 through OC4 = RD0 through RD4) pins.
- **<DutyCycle>** is required and is 0 through 65535. If <DutyCycle> is 0, then the hardware PWM is shut off.
- The PWM frequency is 1220Hz (80MHz/0x10000)
- **Example:** "PM,3,512"
- **Return Packet:** "OK"

The "SP" Command : Set software PWM output values command

- SP command (Software PWM output) Sets a PWM value for any of 18 PWM channels
- **Format:** "SP,<Channel1>,<DutyCycle1>,<Channel2>,<DutyCycle2>,...,<Channel18>,<DutyCycle18><CR>"
- **<Channel1>** is required
- **<DutyCycle1>** is required and is 0 through 4096
- But **<Channel2>** through **<Channel18>** are optional. Note that you must have pairs of values - you can't skip a channel number and have more PWM values than channels, for example.
- Setting a PWM value of a channel to 0 turns the PWM of that channel off, and allows normal digitial I/O on that pin.
- PWM value of 4095 will be %100 on time.

- **Example:** "SP,4,2899,7,1955,2,0<CR>" - Would set channel 4 to 2899, channel 7 to 1955 and channel 5 to 0.
- **Return Packet:** "OK"

The "PC" Command : Configure software PWM paramters

- PC command (configure software PWM parameters) is used in conjunction with the SP command (see above). The SC command should be used to set up each of the various paramters for the SP command, before starting up any of the software PMW channels with the SP command.
- **Format:** "PC,<SubCommand>,<Value1>,<Value2>,<Value3><CR>"
- **<SubCommand>** is required, and what <Value> means (and how many of the 3 are actually needed) depends upon the <Command> value.

    - SubCommand = 0, Value1 = Roll Over Value, 32 bit unsigned int, units = ISR ticks
        - Use this commnd to set the number of ISR ticks between rising edges of all PWM channels. This effectively sets the PWM frequency, and the effective resolution (i.e. maximum value for DutyCycle on each PWM channel)
    - SubCommand = 1, Value1 = ISR Rate, 32 bit unsigned int (Currently unimplemented in 1.6.3)
        - This sub commands sets the time between ISR ticks. The units are 80MHz clocks. Don't make this too fast, or your processor will lock up.
    - SubCommand = 2, Value1 = PWM Channel, Value2 = Port (use 'A' through 'G'), Value3 = pin (0 through 15)
        - This sub command uses all 3 values. It assigns a physical pin to a software PWM channel number. Send one of these for each pin you want to set up.
    - SubCommand = 3, Value1 = anything, print out current SP command paramters
        - Ths sub command just prints out the current paramters for the SP command
    - SubCommand = 4, Value1 = MaxUsedChannel, 8-bit unsigned value from 0 to 64
        - Use this sub command first (before SubCommand 2) to set the maximum number of channels that the software PWM system can handle. The smaller it is, the faster you can go.
    - SubCommand = 5, Value1 = SoftwarePWMState (0 = Off/1 = On) - use to turn off software PWM
        - Use this sub command after you're done with a channel to turn it off.
- **Example:**
    - "PC,4,3<CR>"
    - "PC,2,1,D,1<CR>"
    - "PC,1,1000<CR>" - Sets ISR tick rate at 80,000,000/1,000 or 80KHz
    - "PC,0,4096<CR>" - Sets PWM resolution to 12 bits, and PWM frequency to 80Khz/4096 = 19.5Hz
- **Return Packet:** "OK"

The "CA" Command : Configure Analog inputs

- CA command configures and turns on the analog inputs - use before the IA command (see below).
- **Format:** "CA,<PinBitmask><CR>"
- **<PinBitmask>** is required, represents the bits of the analog inputs that you want to 'turn on' to be analog inputs. Valid values are 0 to 65,535. Each bit corresponds to an ANx (analog input pin). See the UBW32 scheamatic to see what pins each analog input is on.
- **Example:** "CA,11<CR>" would set AN0, AN1 and AN3 to be analog inputs.
- **Return Packet:** "OK"

The "IA" Command : Read Analog inputs

- IA command reads analog inputs and prints out their values. You must use the "CA" command first to turn on the analog inputs you want to use. The command will loop through all bits that are high in <PinBitmask> and print out those analog input's values, then wait for <Delay> microseconds, then do the whole thing <Count> times. Note that all values are simply printed out in a long stream (with commas between each value), after all values are read. For this command to work properly, the <PinBitmask> used in the IA command must not have any bits set that were not set in the last CA command's <PinBitmask>. In other words, if you tell the IA command to read analog inputs which

you have not yet configured as analog inputs using the CA command, you will not get the results you expect.
- The analog to digital converter on the PIC32 is 10 bits, so you will get values from around 0 (for 0.0V input) to 1023 (for 3.3V input).
- **Format:** "IA,<PinBitmask>,<DelayUS>,<Count><CR>"
- **<PinBitmask>** is required, and represents the a bitmask of the bits you want to read. For example, if you wanted to read AN1 and AN3, you would use 9 for this paramter.
- **<DelayUS>** is required, and is in microseconds. This is the delay between <Count> reads of <PinBitmask> analog inputs.
- **<Count>** is required, and is the number of times you want to read the inputs, with <Delay> microseconds between each set of readings.
- **Example:** "IA,11,1000,5<CR>" would read AN0, AN1 and AN3 and print their values, then delay for 1000us, then read and print their values again, etc. five times in a row.
- **Return Packet:** "IA,16,873,651,17,32,19,14,15,14,17,15,14,15,15,13<CR>OK<CR>", which represents the values of AN0,AN1,AN3,AN0,AN1,AN3,AN0,AN1,AN3,AN0,AN1,AN3,AN0,AN1,AN3.

Errors Messages:

There are two ways that one might communicate with a UBW32:

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 UBW32.

The long error messages are very useful for debugging the system, and especially when using the UBW32 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 UBW32 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" (unused)
- "!2 Err: TX Buffer overrun"
    - This error is generated if, for some reason, the internal code of the UBW32 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 UBW32 is receiving data from the PC, the internal receive buffer is overfilled.
- "!4 Err: Missing parameter(s)"
    - The UBW32 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 UBW32 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 UBW32 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 UBW32 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 UBW32 received that did not match any know commands.

Version 1.6.3 Files:

ZIP file of all project and source files necessary to build 1.6.3 here.
v1.6.3 HEX here (for programming a UBW32)

Questions? E-mail me at brian_schmalz@yahoo.com