Raw USB Interface

While we provide a Windows driver with a more convenient interface, users in other operating systems may need to directly send USB vendor requests to the board. This section describes that low-level interface.

Each USB control vendor request has two word-size parameters, Value and Index, plus a lengthed data buffer that can be written to the board or read from the board.

Digital I/O (DIO) is viewed as a series of bytes, from 0 on up. Counters are grouped into blocks based on 8254s, each block containing three counters(0, 1, and 2).

For the USB-DIO-32, port A is byte 0, port B is byte 1, port C is byte 2, and port D is byte 3. 8254 A is counter block 0, 8254 B is counter block 1, and 8254 C is counter block 2.

For the USB-IIRO-16 family, the relay outputs are bytes 0 and 1, and the isolated inputs are bytes 2 and 3.

For the USB-CTR-15, the counters are in counter blocks 0 through 4, unless otherwise noted.

VENDOR REQUEST 12H: DIO CONFIG

This request sets up the DIO for use; it sets the direction of each byte, provides initial data for each output byte, and turns tristate on or off. It's only used by the USB-DIO-32 family. At power-up, tristate is on, and this request must be issued before using the DIO.

Value: 0000h for tristate off, 0001h for tristate on. If you don't know to use tristate, you want tristate off.
Index: Reserved, use 0000h.
Data Written: 6 bytes. Bytes 0-3 are DO data, and will be written to the corresponding DIO bytes before tristate is set. (DO data for input bytes will be ignored.) Byte 4 controls the direction of the DIO bytes (see table below). Byte 5 is reserved for future COS functions (use 00h).

Table: DIO Config Direction Byte

Bit	7	6	5	4	3	2	1	0
Content	Reserved, leave 0				Byte 3 0 = in 1 = out	Byte 2 0 = in 1 = out	Byte 1 0 = in 1 = out	Byte 0 0 = in 1 = out

VENDOR REQUEST 10H: DIO WRITE

This request writes DO data to all bytes. (DO data for input bytes will be ignored.)

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Written: 4 bytes, written to the corresponding DO bytes.

VENDOR REQUEST 11H: DIO READ

This request reads all DIO bytes. For output bytes, this will read back the output values.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: 4 bytes, read from the corresponding DIO bytes.

VENDOR REQUEST 20H: COUNTER READ
This request reads the current value from the specified counter.

Value: The low byte is the counter block index. The high byte is the index of the counter to read, from 00h to 02h. (Note that the high byte is **not** a mode control byte, unlike other counter requests.)
Index: Reserved, use 0000h.
Data Read: 2 bytes, the current value in the counter.

VENDOR REQUEST 25H: READ ALL COUNTERS
This request reads the current value from all counters on the board.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: n×2 bytes, where n is the number of counters on your product. (USB-CTR-15: n=15) . Each pair of bytes is the LSB/MSB of the current value in the counter, with Counter 0 from Counter Block 0 first, followed by Counter 1 Block 0, Counter 2 Block 0, then Counter 0 Block 1, etc.

All counters are latched before this sequence of reads occurs, allowing as little time skew between sequential blocks as possible.

VENDOR REQUEST 23H: COUNTER READ/MODE/LOAD OR MODE/LOAD
This request read the current value from the specified counter, then puts it in the specified mode, then loads it with the specified load value. The read can be skipped.

Value: The low byte is the counter block index. The high byte is the mode control byte (see table below).
Index: The load value.
Data Read: Either none, or 2 bytes. If 2 bytes are read as part of the request, they will be the current value of the counter before the mode.

Table: Counter Mode Control Byte

Bit	7	6	5	4	3	2	1	0
Content	Counter Index, 0-2		1	1	Counter Mode, 0-5			0

VENDOR REQUEST 21H: COUNTER MODE
This request puts a counter in the specified mode, but does not load a load value. This can be useful for setting the output value of the counter, without actually making it count. (Mode 0 sets the output low, mode 1 sets it high.)

Value: The low byte is the counter block index. The high byte is the mode control byte (see table above).
Index: Reserved, use 0000h.
Data: None.

VENDOR REQUEST 22H: COUNTER LOAD
This request loads the specified counter with the specified load value.

Value: The low byte is the counter block index. The high byte is the mode control byte (see table above); however, only bits 6 and 7 are used.
Index: The load value.
Data: None.

VENDOR REQUEST 24H: SELECT GATE SOURCE (FOR FREQUENCY MEASUREMENT)

This request tells the board to start acquiring readings from the counters using the specified counter as the gate source. Please read and understand vendor request 26h before using this request. In order to measure frequency you need to connect the output of the counter you choose as a gate source to the gate of each of the counters on which you wish to measure frequency. Then connect each unknown frequency to be measured to the input of the counters you've configured with the gating signal. Each time the gate goes high the counters will count events, and each time the gate goes low the card will store the reading. The number of counts that occur during a known period is frequency.

Please note, for many applications a single counter may not be able to generate a long enough gating signal. Simply connect two counters in series. The output of the last counter in your daisy-chain is your gating signal, and simply report that single counter to this vendor request.

Value: The low byte is the counter index, from 0 to 14 for the USB-CTR-15. Only implemented on USB-CTR-15.
Index: Reserved, use 0000h.
Data: None.

VENDOR REQUEST 26H: READ ALL LATCHED COUNTS (FOR FREQUENCY MEASUREMENT)

This request is designed to allow as many as 14 counters to be used to measure frequency simultaneously and efficiently. Several setup steps are required:

1) Issue request 23h to set all 14 frequency counters to mode 2; the load value should be "0" but that's technically up to your program.
2) Issue request 23h to set the gate source counter (which will be selected using request 24h) to mode 3, with a load value that will produce a slow enough output.
3) Issue request 24h to start measuring frequencies using the specified counter as the gate source.

The board is now storing counts onboard, measurement has started.
Poll all the counters using request 26h any time after this. The data returned is the count value most recently *latched by high-side gate periods*, for all counters, in the same order as for request 25h. The 31st byte of data indicates how many times the board read the frequency data since you last called request 26h. If this request is issued more often than new data is available, the 31st byte will be "0" until new data is ready. Ideally the 31st data byte would always read "1", indicating the board took data once for each read of the data by the customer's program.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: n×2+1 bytes, where n is the number of counters on your product. (USB-CTR-15: n=15) . Each pair of bytes is the LSB/MSB of the value latched by the counter, with Counter 0 from Counter Block 0 first, followed by Counter 1 Block 0, Counter 2 Block 0, then Counter 0 Block 1, etc. The last byte is "how many readings has the USB board taken since last asked this question". If zero, you're re-reading the same data as the last time you issued a vendor request 26h.

So, to calculate frequency from the data returned, divide the number of counts by the number of seconds your selected Gate was HIGH. For example, if you're using a 1Hz Mode 3 signal as the source of your gate signal, your gate will be high for 0.5 seconds. If you read 1100h counts from a counter using request 26h, the frequency would be "1100h / 0.5" or "4352 * 2" or 8704Hz.

VENDOR REQUEST B3H: DAC WRITE (SIMPLE FORM)

This request writes a value to a single DAC.

Value: Count value to write to the DAC.
Index: Which DAC channel to write to.
Data: None.

If any data is written, the request will instead be treated as multi-channel form (below).

VENDOR REQUEST B3H: DAC WRITE (MULTI-CHANNEL FORM)

This request writes values to any combination of DACs.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Written: 17 bytes. The first byte is a bit mask, the others are eight word-sized count values (in Intel byte order). For each DAC whose bit is set, the associated count value will be written to it. Count values associated with cleared mask bits will be ignored.

For future boards with more than 8 DAC channels, more than one 17-byte block can be written, each controlling 8 channels.

If no data is written, the request will instead be treated as simple form (above).

VENDOR REQUEST A2H: EEPROM READ OR EEPROM WRITE

This request writes to or reads from the onboard EEPROM. There's a section of the EEPROM set aside for customer use(most commonly, storing unique identifiers) which is accessed with this request. Note, however, that other parts of the EEPROM are used for important things, so make sure the start address and length are valid before issuing this request.

Value: The start address for the write. The custom EEPROM area is from 1E00h to 1FFFh.
Index: Reserved, use 0000h.
Data: Read or write some number of bytes. These will be read from/written to the EEPROM starting at the start address, so make sure the start address plus the data length does not exceed 2000h.

VENDOR REQUEST BFH: ACQUIRE "IMMEDIATE" A/D SAMPLE

This request simply takes one A/D sample, on a board of the USB-AI16-16 family. It doesn't use any of the intermediate features of the board, like calibration or even timing, much less any of the advanced features. It does use the channel's selected range and single-ended setting, though the defaults are often fine.

When using this request, the Scan bit in byte 11h of the configuration block should be cleared to 0. This request cannot return a scan of data, so if the Scan bit is set to 1, it's unspecified what data is returned.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: 2 bytes, being the sample acquired.

VENDOR REQUEST D2H: READ A/D CONFIGURATION BLOCK

This request reads the configuration block on boards of the USB-AI16-16 family, mainly for use in read-modify-write operations.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: For the USB-AI16-16 family, 20 bytes. See below for details.

VENDOR REQUEST BEH: WRITE A/D CONFIGURATION BLOCK

This request writes the configuration block on boards of the USB-AI16-16 family.

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Written: For the USB-AI16-16 family, 20 bytes. See below for details.

Configuration bytes for analog input boards(the USB-AI16-16 family) are as follows:

[00h]	…	[0Fh]	[10h]	[11h]	[12h]	[13h]
Channel 0 Gain Code	…	Channel 15 Gain Code	Calibration Code	Trigger & Counter Clock	Start & End Channel	Oversample

A configuration of all zeroes is close to an "ordinary" use; you'll likely want to set external or timer trigger, and start and end channels.

Gain codes (config bytes 00h-0Fh) for analog input boards(the USB-AI16-16 family) are as follows:

Gain Code	00h	01h	02h	03h	04h	05h	06h	07h
Range	0-10V	±10V	0-5V	±5V	0-2V	±2V	0-1V	±1V

Add 08h to the gain code for channel 0-7 to pair it with the respective channel 8-15 in differential mode.

Calibration codes (config byte 10h) for analog input boards(the USB-AI16-16 family) are as follows:

Cal. Code	00h	01h	03h
Effect	Acquire Normal Data	Acquire Cal. Ground (0.0114V @ 0-10V)	Acquire Cal. Reference (9.9V @ 0-10V)

Trigger & counter clock bits (config byte 11h) for analog input boards(the USB-AI16-16 family) are as follows:

Bit	7	6	5	4	3	2	1	0
Value	Reserved, use 0			CTR0 EXT	Falling Edge	Scan	External Trigger	Timer Trigger

• If CTR0 EXT is set, counter 0 is externally-triggered; otherwise, counter 0 is triggered by the onboard 10MHz clock.
• If Falling Edge is set, A/D is triggered by the falling edge of its trigger source; otherwise, A/D is triggered by the rising edge of its trigger source.
• If Scan is set, a single A/D trigger will acquire all channels from start to end, oversampling if so configured, at maximum speed. Otherwise, a single A/D trigger will cause a single acquisition, "walking" through oversamples and channels.
• If External Trigger is set, the external A/D trigger pin is an A/D trigger source. Otherwise, it's ignored. (Immediate (software) triggering is always an A/D trigger source, as well.)
• If Timer Trigger is set, counter 2 is an A/D trigger source. Otherwise, it's ignored.

Start & end channel (config byte 12h) for analog input boards(the USB-AI16-16 family) are the start channel (0h-Fh) in bits 0-3, and the end channel (0h-Fh) in bits 4-7. If the end channel is less than the start channel, then the board's behavior is unspecified.

Oversample (config byte 13h) for analog input boards(the USB-AI16-16 family) is a number indicating how many **extra** samples should be acquired from each channel before moving on to the next. In a noisy environment, the samples can be averaged together by software to effectively reduce noise.

VENDOR REQUEST BAH: QUERY A/D CALIBRATION FEATURE

This request determines whether A/D calibration is supported by the board(USB-AI16-16 family only).

Value: Reserved, use 0000h.
Index: Reserved, use 0000h.
Data Read: 1 byte. It will be equal to BBh if calibration is supported, or not equal if not supported.

VENDOR REQUEST BBH: LOAD BULK CALIBRATION BLOCK

This request completes the loading of a calibration block on a board of the USB-AI16-16 family.

When loading a calibration table, first send a block of up to 1024 words via bulk to endpoint 2, then use this vendor request to load it into the calibration SRAM.

Value: The start address for the block. The calibration SRAM is from 0000h to FFFFh.
Index: The length of the block, in words.
Data: None.

VENDOR REQUEST BCH: START ACQUIRING BLOCK

This request begins the acquisition of a block of A/D data on a board of the USB-AI16-16 family.

When actually acquiring A/D data, first use this vendor request to start acquisition of a block, then read the data via bulk from endpoint 6.

Value: The high word of the length of the block, in words.
Index: The low word of the length of the block, in words.
Data: None.
Note that if Value and Index are considered together as a double-word, the words are reversed.

The remaining vendor requests are not implemented, and are documented only for completeness. For the USB-CTR-15, the instructions for implementing these with multiple requests assume the “standard configuration” connector is attached.

VENDOR REQUEST 28H: COUNTER PURPOSE, OUTPUT FREQUENCY

This request would set up a counter block to output a square wave, at a frequency determined by dividing the oscillator frequency by the load values of counters 1 and 2. The result would be available on the 8254's OUT 2 pin. To achieve that without this request, use two counter mode/loads (vendor request 23h) instead. Counter 1 should be placed in mode 2, and counter 2 should be placed in mode 3.

VENDOR REQUEST 2CH: COUNTER PURPOSE, MEASURE FREQUENCY

This request would set up a counter block to measure an input frequency. Due to latencies in USB timing, this can't be achieved precisely without an atomic request. However, the USB-CTR-15 can measure frequency by a different method, using vendor requests 24h and 26h; see the discussion under vendor request 26h for details.

VENDOR REQUEST 2DH: COUNTER PURPOSE, EVENT COUNT

This request would set up a counter block to count falling edges on its COUNTER 0 IN pin. To achieve that without this request, use a counter mode (vendor request 21h) to put counter 1 in mode 1, a counter mode/load (vendor request 23h) to start counting on counter 0 (in mode 2), a counter read (vendor request 20h) to read counter 0 without affecting it, and a counter read/mode/load (vendor request 23h) to read counter 0 and reset it at the same time.

VENDOR REQUEST 2EH: COUNTER PURPOSE, MEASURE PULSE WIDTH

This request would set up a counter block to measure pulse width. Due to latencies in USB timing, this can't be achieved precisely without an atomic request.

VENDOR REQUEST 30H: READBACK GLOBAL STATE

This request would read back state saved by the board in non-volatile memory.

VENDOR REQUEST 31H: SAVE GLOBAL STATE

This request would cause the board to save its current state to non-volatile memory, for retrieval with the Readback Global State request(vendor request 30h).