Versalent

www.versalent.biz
Dec 2025

WLKBI2CS USB Wired/Wireless Keyboard RS232/I2C/SPI Converter Manual

Version 1.02
Revised Dec 22, 2025

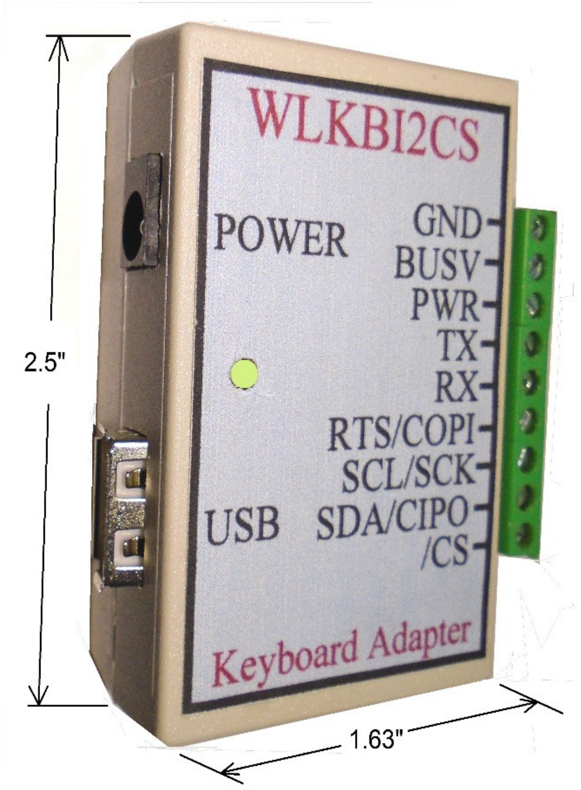


Table of Contents

General Description

WLKBI2CS is a small module (2.5" x 2.2" x 0.9") that allows a standard wired/wireless USB keyboard to be used with non-USB I2C or SPI or RS-232 serial systems. The characters printed on the keys are output from the RS232 port, and are available for I2C or SPI reads so that non-PC based hosts can easily use wired/wireless USB keyboards. USB is simple for users but contains complexities that require specialized software and hardware for even simple keyboard communications. WLKBI2CS simplifies all that providing standard interfaces which can connect to almost any computer system. Wired/wireless keyboards easily connect to the smallest of embedded systems through a UART, I2C or SPI port.

WLKBI2CS accepts the USB dongle of many keyboards/mice combos providing the USB interactions including power management, enumeration and report exchanges required to 'talk USB'. By default it also translates the USB keycodes to the ASCII character printed on the key so all you have to do is plug it in and characters arrive/are available. A companion WLKBI2CS Configuration Program is provided which allows any key output to be reprogrammed (via the RS232 port) to be a string of 0 to 10-characters. These custom strings are stored in the devices internal non-volatile memory so are retained indefinitely. CAPS Lock and NUM Lock states are managed transparently so the keyboard sends the modified key cases. If the keyboard has LEDs it manages those as well. It recognizes and acts on all keys including function keys, arrow keys, keypad keys with Numlock/Ctrl/Alt both on and off etc.

A wireless keyboard's power is provided by its own battery, but a dongle or wired keyboard must be 5-volt-powered from its USB host connector, and WLKBI2CS provides that. 4-30VDC (5VDC for the -5 model) power must be connected to screw-terminal PWR or the power jack that accepts a 'power-brick' wall supply.

LED Indicators

1) When the WLKBI2CS is powered and operating the green LED near the USB connector flashes very quickly before a USB keyboard or dongle is detected. Then after a keyboard is found and successfully initialized, the flash rate slows to every ½ second (noticeably slower). So the LED provides visual indicators of applied power as well as the state of the USB connection. Note that WLKBI2CS works with wired or wireless ***keyboards only*** – it is not a general purpose USB host like a desktop PC and will not operate with any other USB device or hub. WLKBI2CS has limited USB resources compared to a desktop computer and cannot provide access to different types/classes of devices.

2) Near the GND screw terminal is a yellow 'CMD' LED which flashes when a valid I2C, SPI, or RS232 command is received. Typically this will be on steadily as an I2C or SPI Master repeatedly interrogates the device requesting keypress data. Unlike the RS232

port which outputs keypress data unsolicited, the I2C and SPI ports only send data in response to a query command from a Master.

This LED also performs an 'interface-annunciator' function. At power up, it flashes a pattern of SHORT-PAUSE if the I2C interface is selected, or SHORT-SHORT-PAUSE if SPI is selected, or no flashes if no interface is selected. This annunciator ends when any command is received, or after the pattern repeats 10 times.

3) Near the /CS screw terminal is a red 'ERR' LED which flashes various patterns to indicate synchronous errors (I2C or SPI) such as 'invalid command received, 'port overrun' etc. The following table lists the possible errors and their associated flash patterns. Once an error occurs, its pattern will continue to flash until an error-reset (E) command is issued, or the unit is reset.

Error Code	**Error Description**	**Flash Pattern Repeating***
1	Command buffer overrun. New byte received before last one was read.	SHORT—SHORT--PAUSE
2	Unknown Command Received	SHORT--SHORT—SHORT --PAUSE
3	Received data addressed to someone else	SHORT—LONG—SHORT -PAUSE
4	Invalid Parameter Received	LONG--PAUSE

*SHORT = flash of 1/10th second, LONG = flash of ½ second, PAUSE = 2 seconds

Keypress Output

The WLKBI2CS is designed for simplicity – by default it outputs a single ASCII character (the one printed on the keyboard key) when the key is pressed. Or if reprogrammed, the key outputs the string of your choice. If the key is held then the WLKBI2CS auto-repeats the serial character .. strings do not auto-repeat. The keyboard/dongle generates USB keycodes and the WLKBI2CS translates those to serial characters/strings.

Since the standard 7-bit ASCII character set contains only 128 characters (0x00 thru 0x7f), in order to support the keyboard's function keys , keypad keys and CTRL keys, the WLKBI2CS sends 8-bit characters to identify these 'non-printable' keys which are beyond the 7-bit range. And to support the use of the ALT key as a character-modifier the WLKBI2CS sends two characters to identify ALT-x keys. Please refer to http://www.versalent.biz/manuals/wlkbmap.pdf which is a diagram of a 102/114-key PC keyboard showing all key positions, and the associated characters which the WLKBI2CS generates when the key is pressed. There are 5 characters (hex values) on each key to identify what 8-bit characters are output when pressed:

1) Normal keypress .. with no other keys pressed
2) Key pressed with SHIFT pressed (or CAPSLOCK active)
3) Key pressed with ALT pressed
4) Key pressed with CRTL pressed
5) Key pressed with NUMLOCK active

The NUMLOCK, SHIFT, ALT and CTRL keys do not cause any characters to be sent when pressed by themselves. These are 'modifier' keys only – which affect the values that other keys send. CAPSLOCK does send a code and then continues to act as a modifier when active.

Connecting WLKBI2CS RS232 Port

WLKBI2CS's RS232 serial port reports key-press characters, and is used with the WLKB232 Configuration app to reprogram key outputs. The WLKBI2CS port can be connected to a PC COM port using either:

1) Screw terminals wired to a DB9 female connector
   Terminal GND ➔ DB9 pin 5
   Terminal TX ➔ DB9 pin 2
   Terminal RX ➔ DB9 pin 3
   Appropriate power adapter (5VDC or 4-30VDC) plugged into the power jack)

2) ONLY the Versalent-provided DB9 flat cable (WLKBI2CS-PGM cable).

.

***** CRITICAL INFORMATION *****
A DB9 flat-cable is a convenient way to connect the WLKBI2CS serial port to a PC for programming, however ONLY the Versalent WLKBI2CS-PGM cable can be used since it passes only TX, RX and GND signals preventing other DB9/RS232 signals from damaging non-RS232 WLKBI2CS inputs. **DO NOT USE any other DB9 flat cable.**

(Warranty does not cover damage caused by negative-going RS232 signals to WLKBI2CS positive-only signal inputs).

Reliability Features

WLKBI2CS implements several reliability features:

1) **Watchdog timer** and brown-out detector -- If the internal microcontroller gets disrupted through static discharge or other temporary interference, the watchdog will automatically reset so that normal operation resumes with no user intervention. If the supplied power droops below an operational threshold the brown-out detector will suspend operation until normal power is restored.
2) **Baud Rate Validation** – If a power failure occurs during programming, an invalid RS232 baud rate could be set. At power-up the baud-rate setting is checked -- if not a valid rate, the unit is forced to 9600 baud no parity so the unit is always recoverable.
3) **Synchronous Command Error Led** – The I2C and SPI ports cannot automatically report communications errors. The red SERR (synchronous error) led flashes one of several patterns to indicate that an error has occurred. The pattern associated with the first error detected is flashed repeatedly until the error is retrieved/cleared with the 'E' command.

Connectors

Besides the 9 screw terminals provided for connection to the I2C and SPI signals, there is also a 10-conductor IDC (flat-cable) connector underneath the terminals. Using the IDC connector may provide a more convenient interconnect or a neater installation. When this manual refers to 'screw terminals' and its signals, it is also referring to the same signal-name (different pin numbers) on the IDC connector.
The standard IDC connector used is a non-latching type .. contact Versalent for replacement with horizontal or vertical-ear latching types.

SCREW TERMINAL PINS

Pin#	Pin Name	Pin Description
1	GND	the signal and power ground
2	BUSV	the host's I2C or SPI pull-up voltage (1.8V/3V/5V)
3	PWR	4-30V power input, or 5V power input for WLKBI2CS-5
4	TX	the RS232 TX output signal +/- 12V max
5	RX	the RS232 RX input signal +/- 20V max
6	RTS/COPI	RS232 RTS +/- 15V, SPI COPI (MOSI) input signal[1]
7	SCL/SCK	I2C SCL clock signal , SPI SCK clock signal[1]
8	SDA/CIPO	I2C bi-directional data , SPI CIPO (MISO) data output[1]
9	/CS	SPI /SELECT input signal[1] . Not used for I2C.

TABLE 1

10-PIN IDC* PINS

Pin#	Pin Name	Pin Description
1	SCL/SCK	I2C / SPI clock signal [1]
2	/CS	SPI /SELECT input signal[1] . Not used for I2C.
3	TX	the RS232 TX output signal +/- 12V max
4	GND	the signal and power ground
5	RX	the RS232 RX input signal +/- 20V max
6	SDA/CIPO	I2C bi-directional data, SPI CIPO (MISO) data[1] output
7	RTS/COPI	RS232 RTS +/- 15V, SPI COPI (MOSI) input signal [1]
8	BUSV	the host's I2C or SPI high-level voltage [1]
9	GND	2nd signal and power ground
10	PWR	4-30V power input, or 5V power input for WLKBI2CS-5

TABLE 2

*IDC = Insulation-Displacement Connector (flat-cable connector)

[1] Compatible with *SPI 2.7V to 5V logic*, and *I2C 1.8V to 5V logic*

E-Baud - Serial Port Baud Rate/Parity Control

This feature allows the baud rate and parity to be set electronically. Rates from 600 to 115.2k can be set by issuing the following RS232 command:

/BWLKBxy

- /B is the E-Baud command
- "WLKB" is a fixed 'key' which prevents inadvertent baud rate changes
- x is one ASCII character that defines the baud rate (see table below)
- y is one ASCII character that defines the parity (see table below)

This command can be issued by the WLKBI2CS Configuration application, or even manually with the use of a terminal emulation program.

	Baud Rate		**Parity**
x = ‘0’	600		
x = ‘1’	1200		
x = ‘2’	2400	y=’0’	NONE
x = ‘3’	4800	y=’1’	EVEN
x = ‘4’	9600	y=’2’	ODD
x = ‘5’	19.2k		
x = ‘6’	38.4k		
x = ‘7’	57.6k		
x =’8’	115.2k		

TABLE 4

The E-Baud command is issued at the device’s initial baud rate, and it responds with “SUCCESS” also at that initial baud rate. The baud rate/parity then changes and operation resumes with no reset or power cycling required. Because the new values are stored in non volatile memory, WLKBI2CS will then power up at this baud rate.

With E-Baud devices there is no way to visually confirm the unit’s current baud/parity settings. The WLKBI2CS Configuration app includes a ‘Find WLKBI2CS’ menu item that searches all serial ports, at all baud rates and all parity settings – sending a request for the firmware version (/V command) and awaiting a valid ‘WLKBI2CSvX.XX’ response.

Extended Function Keys (F13-F24)

WLKBI2CS devices support the standard 102-key keyboards (96 programmable keys), plus the F13-F24 available on some industrial keyboards for a total of 108 programmable keys. Firmware version numbers end with ‘F’ indicating that the extended Function keys are supported.

I2C/ SPI/RS232 Interfaces

Model WLKBI2CS offers an RS232 interface like its predecessor the WLKB232. Its main use is device configuration, but like the WLKB232, it outputs serial RS232 key-press characters -- the same characters available from the I2C and SPI ports. The RS232 port is always active along with neither or 1 of the I2C or SPI ports. See Connectors above.

Either the I2C interface, or a 3/4-wire SPI interface can be active – not both simultaneously. (Configuration application sets a value that determines which interface is active). The screw terminal and 10-pin IDC connections are shared for these two interfaces with both signal names shown on the case label.

I2C and SPI both provide a 50-byte FIFO buffer which collects characters/strings from the keyboard. 50 bytes provides enough storage to hold 5-keys each programmed with 10 characters. An I2C/SPI Master host should read this buffer periodically (typically every 5-20ms) to retrieve characters generated. It is recommended that the host read a small number of bytes at a time (1-3) since keyboards do not generate data quickly. Non-zero bytes represent valid key data while 0x00 is the 'end-of-data' indicator. The host's low-millisecond read rate insures that keypress response is fast as it discards all 0x00 data which indicates that there are no keypresses to report. If the host does not read the device and its 50 character buffer begins overflowing, it starts overwriting the oldest keypress data. Reading is typically frequent enough that this never occurs, however I2C/SPI reading is not required. The device continues to operate and output RS232 characters while overwriting old data in its buffer.

The WLKBI2CS provides two pullup resistors for the I2C and SPI interfaces which can be enabled. They allow the level-shifters to operate and should be turned ON for both interfaces if there are no external resistors that pullup SCL/SCK and SDA/CIPO to the external BUSV voltage. (SPI output signals COPI and /CS are also level shifted but need no 'pullup control') . BUSV must always be connected to the external 'HIGH' voltage for both I2C and SPI providing the HIGH level for the shifters to deliver.

I2C Interface:

To use I2C connect 3 WLKBI2CS signals:

1) **SCL** **-** Connect to I2C Master SCL (clock signal)
2) **SDA** **-** Connect to I2C Master SDA (data signal)
3) **GND** **-** Connect to I2C Master signal and power ground

The I2C interface is 'level-shifted' and is compatible with external I2C levels of 1.8V to 5V. Internal 2.2k signal pullups are enabled by default and can be disabled if the external I2C bus provides resistors that pullup to BUSV (the high-level bus voltage). Any external I2C pullups should be >= 1k . The SCL clock signal can operate at up to 400kHz.

I2C Command Set:

Reading key-press data via I2C does not require sending a prior command. Simply reading the I2C address will return one byte of buffered data (there is no 'read' command to send .. this differs from SPI). Non-zero data represents key-press (or Version) data while 0x00 indicates 'no-data'. Typically a host will be in a periodic/continuous cycle of reading the I2C address awaiting key-press characters.
To send a command, the host halts its continuous read cycle, then writes a command byte then writes a parameter byte if needed. The command is then executed.

Ax Change the I2C Address. This is the 7-bit address that the device will respond to . **x** is the single binary byte of 0x08 to 0x77 which is the range of I2C addresses allowed for Slave devices. The I2C Master writes exactly two bytes and there is no command response (other than the standard I2C 'Ack' bit for each transferred byte that is part of the I2C protocol).
After a Master changes the address, it should send a 'V' command at the new address, or perform an I2C address-scan to verify that the command was successful. The new value is stored in non-volatile memory.

V Queue the device firmware version into the WLKBI2CS read buffer. Firmware version is a string of ~ 13 characters like "WLKBI2CSv1.61" which defines device features. This string is placed at the head of the buffer so it is returned before any pending key-press characters, and it ends with 0x00 so the host can always identify the end of the version string. The host should read and collect version-string characters until it encounters a 0x00 . Subsequent reads represent key-press data.

Px Set or return the state of the WLKBI2CS pullups. X is a binary parameter:

1) X = 0 to turn pullups OFF
2) X = 1 to turn pullups ON (factory default)
3) X = 2 to request the current state of the pullups. This queues up one byte to be read by a subsequent R1 command to retrieve that byte.

If internal pullups are turned OFF **.. external pullups MUST be connected** to BUSV or the Master will likely remain locked up waiting for valid high-level signals to arrive. Please refer to the free Arduino WLKBI2CS I2C Test Application on the Versalent downloads page. This app can serve as a programming reference, or as the starting base for your own Arduino app.

E Return and clear the last I2C error-code detected. The codes are returned as a single binary byte of 0-3, (0 indicates that no error has occurred). The following 3 errors can occur, and are annunciated by the red Error LED at one end of the screw terminals:

1) *Buffer Overrun* -- can occur if the I2C Master does not honor the clock-stretching. The red error LED will flash SHORT-SHORT-PAUSE until the error is read/ cleared.
2) *I2C Collision* – can occur if multiple I2C slaves are assigned the same address, or another slave responds to an address they do not own. The red error LED fill flash SHORT-SHORT-SHORT-PAUSE until the error is read/ cleared.
3) *Invalid Command or Parameter Received* -- will occur if an unknown command is sent to the WLKBI2CS I2C interface. The red error LED will flash SHORT-LONG-SHORT-PAUSE until the error is read/ cleared.

SPI Interface:

Limitations:

WLKBI2CS SPI clock speed can be up to 2.0 MHz, however because SPI provides no inherent ‘byte throttling’ like I2C does, each byte-access (both reads and writes) MUST be separated from others by at least 50usec. WLKBI2CS processes SPI bytes with software (interrupts) so it offers I2C commands which most SPI devices do not provide. The required ‘access-spacing’ prevents data overruns and gives the WLKBI2CS time to execute commands/retrieve data etc . Slowing the clock does not allow the access-spacing to be reduced.

Other devices sharing the SPI signals do not have to operate with this restriction – they can operate at their own speeds/response times. ‘Access-spacing’ only needs to be applied when accessing the WLKBI2CS (its /CS signal is active).

The WLKBI2CS SPI port offers 2.7V to 5V compatibility using the same level-shifters as the I2C interface described above. For SPI, pullups are always turned ON so the level-shifters can operate. The bus voltage of 2.7V to 5V must be connected to the BUSV terminal to set the HIGH logic level for the level-shifters.

To Use SPI Connect 3 or 4 WLKBI2CS Signals:

1) **SCK** - Connect to SPI Master SCK (Master-provided clock signal)
2) **COPI** - Connect to SPI Master COPI (previously MOSI .. data signal)
3) **CIPO** - Connect to SPI Master CIPO (previously MISO .. data signal)
4) **/CS** - For Multi-slave (4-wire) environment, connect to SPI Master device /select signal. For Single-slave (3-wire) environment connect to GND
5) **GND** - Connect to I2C Master signal and power ground

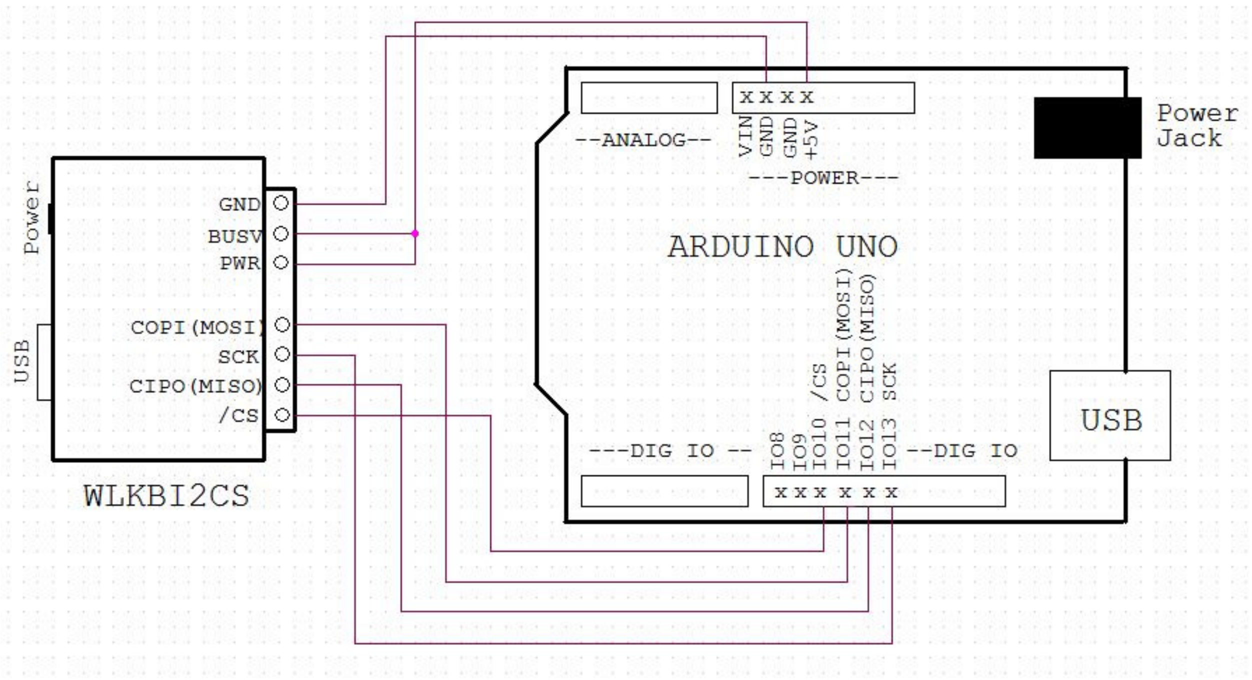
The 4-wire connection allows the WLKBI2CS to coexist on a multi-slave SPI bus. SCK clock-speed is limited to 2MHz maximum and a 50usec byte-spacing must be imposed by the SPI Master.   See above.

WLKBI2CS supports MSBFIRST (most significant bit first), and LSBFIRST. The WLKBI2CS Configuration app provides the choice, and this setting must match the Master for successful communications.

SPI operation:

SPI works by transferring a byte to the WLKBI2CS while simultaneously receiving a byte from it using complementary shift registers. When the Master sends a command, it sends a non-zero ASCII command-byte like 'R' for instance, then a binary parameter like 3 to retrieve 3 bytes. The data returned while sending these two command-bytes is ignored. With the command complete, the WLKBI2CS is configured to return 3 data bytes.
To then read the 3 bytes the SPI Master issues three successive SPI.transfer(0x00) * commands – sending anything, typically 0x00 which is ignored by the WLKBI2CS, with the data returned during each transfer representing the data requested. Please refer to the free Arduino WLKBI2CS <u>SPI</u> Test Application on the Versalent downloads page. This app can serve as a programming reference, or as the starting base for your own Arduino app.

*The Arduino SPI transfer command



Arduino-WLKBI2CS SPI Interconnect

Warning: WLKBI2CS TX/RX signals are RS232 levels (-10V to +10V) – to prevent damage, do not connect directly to Arduino 0-5V Digital IO signals

SPI Commands:

SPI data is transferred to/from the WLKBI2CS by first sending a command byte with its parameter byte(s) if required, then reading any resulting data. As described above, there must be a 50usec delay between sending the command character, and each subsequent parameter byte, and between each read of any expected data bytes.

Typically a host will periodically/continually read a small number (2-3) bytes awaiting key-press bytes. To send another command, this cycle is stopped, a different command is sent and any expected data is read.

Rn Read n bytes from the WLKBI2CS SPI buffer. n is a (binary) byte with a value of 0x01 – 0x14. Data may be key-press bytes or Version string characters or an Error code byte. This command must be followed by exactly n number of SPI reads (even if some or all of those reads = 0x00 indicating end-of-data).

Px Turn pullups ON/OFF or read back the current state of pullups. X is a binary-byte parameter :

1) X=0 turn pullups OFF
2) X=1 turn pullups ON
3) X=2 readback the current state of the pullups.

V Queue the device firmware version into the device read buffer. Firmware version is a string of ~ 13 characters like "WLKBI2CSv1.61" which defines device features. This string is placed at the head of the return buffer so it is returned before any key-press characters, and it ends with 0x00 so the host can identify the end of the version string even if there are also key-characters to be returned. The host should read and collect version-string characters until it encounters a 0x00 at the end of that string. Subsequent reads represent key-press data.

E Queue the last one-byte error code into the device read buffer. See SPI Errors below. The red LED flashes a pattern if an SPI error has occurred. This command reports and clears that error. After sending this command this host should read one byte which has a value of 0x0 – 0x3, with 0x0 indicating that no error has occurred. Possible errors:

1) *DATA OVERRUN* - occurs if a Master does not send exactly 8 bits, or an extra clock signal occurs due to noise. Red error LED flashes SHORT-SHORT-PAUSE until an 'E' command reads and clears it.
2) *UNKNOWN COMMAND* - WLKBI2CS received a command it does not recognize. Red error LED flashes SHORT-SHORT-SHORT-PAUSE until it is read/cleared with an 'E' command.
3) *NOT MY DATA* - electrical interference or noise has caused the WLKBI2CS to receive data that was not intended for it. The data is ignored, but the error is reported. The red error LED flashes SHORT-LONG-SHORT-PAUSE until an 'E' command reads/clears it.

4 *INVALID PARAMETER* - A command parameter supplied was out of range and invalid. The command is ignored, but the error is reported. The red error LED flashes LONG-PAUSE until an 'E' command reads/clears it.

RS-232 Signal Compatibility

The WLKBI2CS is compatible with standard RS-232 signals levels which go both positive and negative (above and below 0 volts) as well as non-standard 0-5 volt signal levels.

Power Inputs

The WLKBI2CS requires 4-30VDC (WLKBI2CS-5 operates on 5VDC) and to power the USB port. There are two ways to apply power to the WLKBI2CS :

1) Apply appropriate power to the PWR screw terminal
2) Plug a 'power brick' into the DC power jack.

USB Port Power

USB ports are sometimes used for battery chargers, or lighting power sources. The WLKBI2CS USB port must **not** be used for any of these functions. It should not be used with any USB device that requires more than 100mA . Typical keyboards or wireless dongles require a fraction of this available power.

WLKBI2CS Models Available

Model#	Features
WLKBI2CS	Screw Terminals and 10-pin IDC header 4-30VDC power to screw terminal or center-positive supply jack. RS232 port available for key outputs and programming/use with WLKBI2CS Configuration app.
WLKBI2CS-5	Screw Terminals and 10-pin IDC header 5 VDC +/- 5% power to screw terminal or center-positive supply jack. RS232 port available for key outputs and programming/use with WLKBI2CS Configuration app.

WLKBI2CS (& CS-5) Specifications

Physical

Size:	2.5" X 2.2" X 0.9"
Weight:	1.8 oz
USB Connector:	Standard Type-A
Host Connector:	9 Screw Terminals and 10-Pin IDC header
Case Color:	Bone/Beige
Power Connector:	5.5 mm X 2.1 mm (Ctr Positive)

Electrical Specifications

WLKBI2CS	
Absolute Max Input Voltage	+32VDC
Nominal Input Voltage	4-30VDC +/- 5%
Current Input (4-30VDC supply)	300mA surge, 150mA continuous
WLKBI2CS - 5	
Absolute Max Input Voltage	5.5VDC
Nominal Input Voltage	5VDC +/- 5%
Current Input (5VDC supply)	75mA + keyboard current
WLKBI2CS & WLKBI2CS - 5	
Max USB Keyboard Current	100mA
Baud Rate	600 to 115.2k selectable via E-Baud
Parity	Selectable NONE/EVEN/ODD
SPI Interface	Clock: 2MHz maximum Signals Levels: 0.4V maximum (low) 2.7V to 5V (high) Inter-byte delay: 50usec minimum
I2C	Clock: 400KHz maximum Signal Levels: 0.4V maximum (low) 1.8V to 5V (high) 7-bit addressing, 0x08 – 0x77

Environmental Specifications

Max Operating Temperature:	+60°C
Min Operating Temperature:	0°C
Max Storage Temperature:	+70°C
Min Storage Temperature:	-40°C
Humidity:	Non-condensing at all temperatures

Reading Key Data:

Read characters from I2C/SPI buffer:

I2C and SPI are synchronous interfaces, (unlike RS232 which automatically outputs key data as it is generated). For these two interfaces, the host must initiate each transfer. As keys are pressed, data is stored in a 50-byte buffer awaiting the host's data-reads.

For I2C, the host issues a read of the I2C address and a byte is returned. A non-zero value received represents data generated by the keyboard.

For SPI, the host sends the 'R' command specifying the number of bytes to transfer. It then executes that number of SPI transfers (sending 0x00 while simultaneously receiving a data byte). Non-zero values received represent data generated by the keyboard.

To provide fast key-press response, the host should read the interface every 3-20ms discarding all 0x00 bytes received. Keys may be pressed at any time, so the host should remain in this 'read mode' continually while awaiting key-press data. Read mode should be suspended while sending any RS232 configuration commands. See WLKBI2CS commands below.

RS232 Commands:

There are several commands used in erasing/programming/verifying key-string and other configuration data, however those are complex binary commands which are not user-accessible so are not published here. User-accessible RS232 commands are preceded with a '/' character:

Cmd Byte	Description
V	Get the WLKBI2CS firmware version. User sends /V and a 13 character string like "WLKBI2CSv1.60" is returned
B	Set the RS232 baud rate and parity (see E-Baud above for command details)

To program and verify key strings, setup I2C and SPI parameters, change baud rates, communicate with a device whose serial parameters are unknown etc .. the WLKBI2CS Configuration (Windows) application is provided. It can be downloaded from the Downloads page of the Versalent website: www.versalent.biz?P=downloads.htm . Refer to **Connecting WLKBI2CS RS232 Port** above.

.

Configuring Output Strings (using the WLKBI2CS Configuration app)

How to Configure WLKBI2CS for Custom Character/String Output :

Note: Any key reprogrammed to output a single ASCII character is subject to the CTRL, ALT, CAPS LOCK, SHIFT KEY modifications. The output of a key reprogrammed to send a string (2 or more characters) is not altered by any of the 'modifier' keys.

NOTE: During the following programming process, all keypresses described refer to your computer's keyboard. The WLKBI2CS 's keyboard/dongle should remain unplugged until the programming sequence is complete.

1) Download and install '*WLKBI2CS Config'* Windows program. It programs all versions of the WLKBI2CS using RS232.
2) Attach the WLKBI2CS to a PC COM port. See Connecting WLKBI2CS RS232 Port above.
3) Establish a software connection to the Configuration app by either:
   A) Set PC COM port to same serial communication settings as the WLKBI2CS and click the **Get WLKBI2CS Code Version** button -- or –
   B) Click **Find WLKBI2C** menu item
4) For any number of keys, click the mouse in the Keyboard Key box – a keyboard appears on screen. Select the key for which you want a custom output. Immediately to its right enter the character or string to be output.
5) Click 'the Add To List' button. Repeat steps 4 & 5 to alter any number of key outputs. You cannot add duplicates – only one entry per key.
6) After entering all the output strings desired, you can SAVE this list to a file for later recall, and/or just click SEND STRINGS TO WLKB.
7) The list will be programmed into non-volatile flash memory which is retained until changed. Note that even if you reprogram just one key, the configuration program updates the entire key-set. (Keys not in your list are programmed with their default characters.)

More Details For Each Step Above:

Step 1: The *WLKBI2CS Config* program is available from the DOWNLOADS section of the Versalent website: https://www.versalent.biz?P=dl.htm

Step 2: Connect the TX screw terminal to COM port DB9 pin 3, RX screw terminal to DB9 pin 2, screw terminal ground to DB9 pin 5 – OR – use ONLY a Versalent WLKBI2CS PGM CABLE … DB9 flat-cable.

Step 3: Serial settings for the *WLKBI2CS Config* app and the WLKBI2CS device must match. They can be set manually or by using the 'Find WLKBI2CS' menu item.

Step 4: The *WLKBI2CS Config* app is very easy to use. You can override the default output for any number of keyboard keys. The Output String accepts ASCII characters.. you can enter non-ASCII, 8-bit characters by enclosing their hexadecimal value in {} brackets. So a Line Feed character can be entered as {0A}. Entering a key but leaving the Output String box empty causes no output for that key. Examples: To make the Enter key output CRLF, you would enter {0D}{0A} as the Output String. Notice that each {xx} sequence represents just one character (of the maximum of 10) in the output string. Do not use {00} – this is an ASCII NULL character and is reserved.

Step 5: The Send Strings button updates the WLKBI2CS in about 15 seconds. When it completes, an auto-verify cycle runs to readback all the values to verify that the programming completed successfully. Below the Communications Details window a green checkbox will appear and a message indicating that all the strings were verified successfully, or a red question mark and text indicating an error.

Step 6: At any time you can verify that the key outputs stored in the WLKBI2CS match your on-screen list (perhaps retrieved from a file or entered manually). For example if you are unsure if you already programmed a WLKBI2CS , you can either

a) Click the "Read Existing Key Strings' button which will load the screen with the key current WLKBI2CS key outputs for visual verification – or—

b) Open a file previously SAVED using the FILE menu (loads those values to the screen), then click the 'Verify Strings' button which will compare those stored in the WLKBI2CS with those onscreen. A Green checkmark appears if all compare successfully, and Red question mark appears if any fail. And as mentioned previously, any keys not specified in your list assume their default character values. All characters/strings, including defaults are verified.

Additional Configuration Notes:

Programmed keys can be entered in any order. Since all keys are programmed as a single block, no incremental updates are allowed -- all your key entries must be in a single list. So it is a good idea to save any key string sets to a file for easy recall/edit.

Verifying WLKBI2CS Character/String Outputs

After configuring the WLKBI2CS , you can confirm that during actual operation, the correct output characters/strings are being generated for the specified keys. This can be done using the:

1) **RS232 Port** .. Connect the RS232 serial port to a COM port, and using any terminal emulator (the Versalent Simple Term is available for download from https://www.versalent.biz?P=dl.htm). Plug in a keyboard and press keys. The programmed values will appear in the terminal window.

2) **I2C Port** .. Connect to an Arduino Uno I2C port and run the Versalent WLKBI2CS I2C Tester app -- free to download from the above webpate. The 'C' command reads/displays keypresses continually in the Serial Monitor window.

3) **SPI Port** .. Connect to an Arduino Uno SPI port and run the Versalent WLKBI2CS SPI Tester app – free to download from the above webpage. The 'C' command reads/displays keypresses continually in the Serial Monitor window.

Document Revision Record

Revision #	Revision Date	Description
V1.00	11/3/25	Prelim
V1.01	12/5/25	Minor corrections and additions
V1.02	12/22/25	Add 'interface-annunciator' LED flash feature

VERSALENT INFO ONLY

All subsequent information is not for publication – it is for Versalent use only. When printing this manual for the website, stop at page 18. And make sure that none of the following info has headings that will appear in the table of contents.

WLKB232 Config Process Description

J. Ursoleo 1-22-11
Updated 2-13-11
Updated 8-21-25 To add Function keys F13-F24

Any number of USB keyboard keys (up to 96 of them) can be re-programmed to output character strings of 0-10 characters instead of their default key-legend characters. Unlike the PS2PRO these keys can reside anywhere on the keyboard.

An output string is not altered by the application of the SHIFT, CTRL or ALT keys as are the single-character outputs. So if the operator programs the key to output a single character, then the modifier keys are applied to the key. The WLKBConfig program runs on a PC and allows for generating the correct strings and downloading them to the WLKB232. These strings are stored in the Flash Program Memory section of the PIC24FJ64GB002 (512 3-byte words total) . The string length of 10 was chosen because although the PIC24 erases program memory in 512-word sections = 1536 bytes, it actually stores keychar[] bytes in two of them (the lower word only). With each of 96 key strings up to 10 characters, including one optional NULL character marking the end of a shortened string, the total amount of memory required is 96 * 10 = 960 characters(bytes). There are 512 X 2 = 1024 bytes available in a single code page (with the upper byte of each 3-byte word unused). In this scheme, strings are either 10-charcters long .. or if shorter they are terminated with a NULL which marks their end.

The full 1024 bytes is allocated so that the linker will not position any other PVS variables in this section, so nothing will have to be saved when a key-string erase/update occurs. The entire 1024 bytes (plus the unused 512 upper bytes in each word for a total of 1536) is erased each time, so the PC program has to send the entire key string array each time even one key is updated/altered. Therefore updates are not incremental since the entire array must be cleared. However the operator has only to specify which keys they want overridden .. and the PC program will send all the other key strings as the key-default, single character so that when the entire array gets erased, then re-written, unspecified keys are returned to their default values.

The rest of this discussion ignores the upper 3rd byte in each program code word since it is ignored by the ‘C’ compiler and is not stuffed as part of the keychar array.

Although a whole code page of 512 words is erased as a block .. that page is re-written in blocks of 64 words (128 bytes) So following a single page-erase, there must be 8 64-word transfers into the PIC24 write-buffers each followed by a write of that 64-word block to fill the previously erased 512 word page. This is somewhat awkward but that is the requirement of the PIC24F family. It is the responsibility of the PC program to organize all this. The PIC24 firmware simply accepts /W commands to write into the 64-word block buffer, then a /F command must flash this block to the correct ‘offset’ within the 512 word flash page. This requires some interaction since there must be delays during the flash-write, and the PIC must respond within a reasonable time. The block can be verified after it is written, or the entire key-string array can be verified after all 8 block-writes.

Note that since each string occupies 10 bytes , an even number of these strings do not fit in the 64 word = 128 byte write-block. What does fit is 12 full strings (120 bytes), then 8 characters from the 13th string. So the PC program sends the 12 full strings, then a shortened (8 byte) 13th string so that exactly 128 bytes are loaded into the PIC24F write-register block for writing. After this block is Flashed, the PC program resumes by sending a 2- byte strings from the point it left off (part-way thru the 13th string). This completes the 13th key-string so that the next /W command is string-aligned with the 14th string, and the string can be seen on-screen at the beginning of a /W command. This is not necessary, but helps when viewing on-screen messages, otherwise the assigned strings would 'roll-thru' the W commands at odd positions. To handle this, the PC program sends the 4th byte and 5th bytes as the character offset (integer) into the 96-key (960 byte) data array. That with the number of bytes in the command allows the PIC24F to place each byte/word at the correct place. So

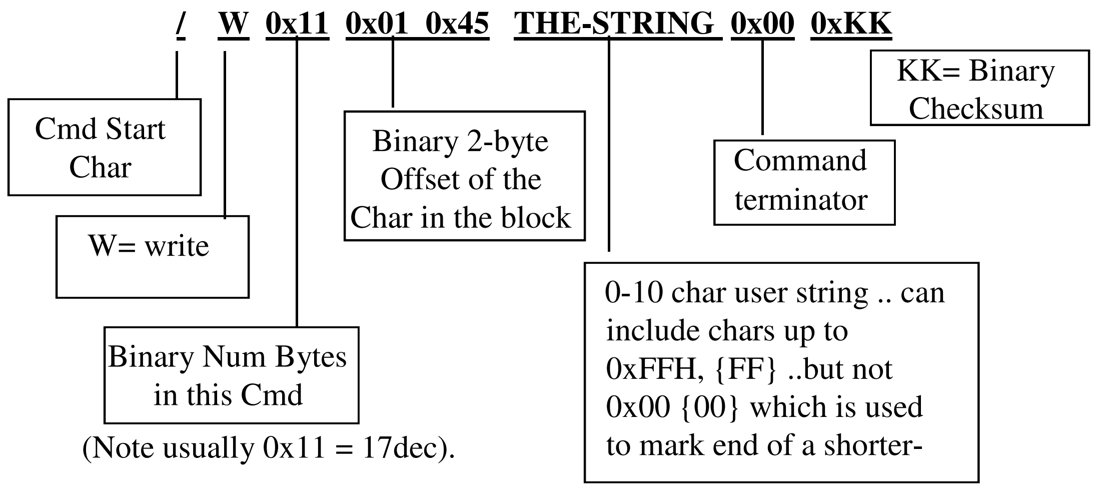
1st /W cmd offset = 0, string length =10
2nd /W cmd offset = 10, string len = 10
.
.
12th /W cmd offset = 110, string length = 10
13th /W cmd offset = 120, string length = 8
//this ends the first 128-byte block and is followed by a /Flash command

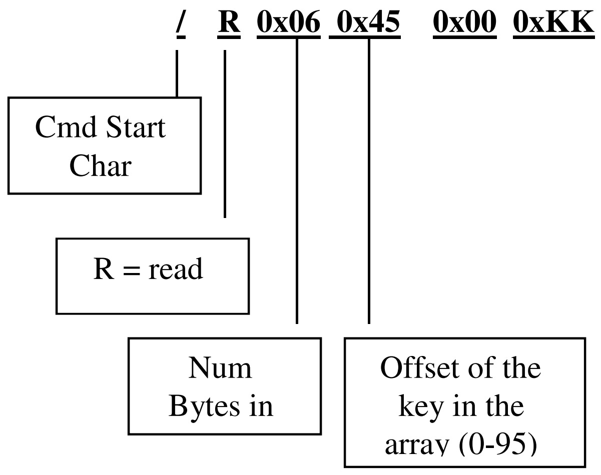
//start new block
14th /W cmd offset = 128, string length=10 (contains 2byte remainder of 13th str)
15th /W cmd offset = 130, string length = 10

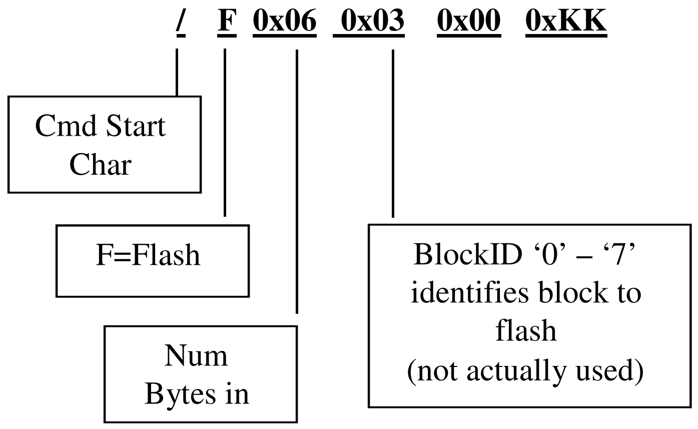
Also, as the program is building /W commands from the key-strings ..if a string occupies less than 10 characters, it is stuffed with 0x00's to a length of 10. So initially, /W commands start at the beginning of key-strings .. but at the 13th, only a partial 8-byte /W is sent because it bumps into the 128-character/byte limit. Then a Flash command is sent and a new 128-byte block is started, with the 14th /W which contains the last two characters of key-string[13] (which may be 0x00's).

And note that a 960-byte array is not evenly divisible by 128 which is the block-size that the WLKB232 must flash. The total number of bytes sent is increased to the next higher multiple of 128 (=1024) so that the WLKB232 always receives 128 bytes to flash a block of 64 words.

A command set is built into the WLKB232 to allow uploading these strings and programming them into individual blocks. And they can be read-back as well. To program a particular 64-word block issue the command (to the WLKB232 serial port):


/ W 0x11 0x01 0x45 THE-STRING 0x00 0xKK
Cmd Start Char
W= write
Binary Num Bytes in this Cmd
(Note usually 0x11 = 17dec).
Binary 2-byte Offset of the Char in the block
0-10 char user string .. can include chars up to 0xFFH, {FF} ..but not 0x00 {00} which is used to mark end of a shorter-
Command terminator
KK= Binary Checksum



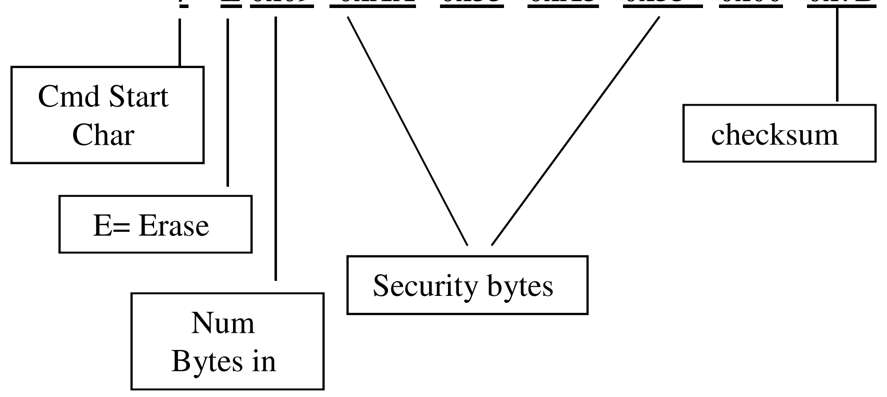


The KK checksum is a simple 8-bit checksum which includes all prior command bytes starting with the '/' command indicator .. and the 0x00 command terminator. It is a simple addition of all previous bytes ignoring any rollovers.

To erase the entire page use

(0xKK)

/ E 0x09 0xAA 0x55 0xA5 0x55 0x00 0x7B



This is the ERASE command which includes some security values which have to be correct, otherwise the command will not be executed. Once executed, the entire array of 512 words (1024 bytes) is emptied (Set=0xFF). The WLKBconfig program then must either restore default values (characters) by entering the default key outputs for keys not specified with overrides, .. or by entering some override strings as specified by the user.

UPRADE TO ALLOW FOR F13-F24

To add the function keys F13-F24 would be an easy matter if there were room in the existing 1024 byte flash memory block, however there is not. The additional 12 keys requires an additional 12 X 10 =120 memory locations, and we were already up to 960 bytes for keys, then more bytes for baud/parity, and the keypad fixed-numeric values. So another block of 1024 bytes is now allocated making the full array 2048 in size instead of the previous 1024. Again reserving the entire block of 2048 since I don't want to share any of it with code space .. and take the chance that a code section gets erased, then fails to restore – unit would be dead.

To implement F13-F24 . the Config Application needs to detect if the connected unit is an F version. If so, it needs to write one additional block of 64 words, most of which are unused, but the full 64 word writes must occur. The remainder of the erased

1024 block can remain un-programmed. Note however that the 112 keys X 10 bytes/key = 1120 bytes which is not divisible by 128. And we must send complete 128-byte blocks to the device, so the Total_Bytes sent is increased to the next higher increment of 128 = 1152. The last block is filled with repeated/dummy data from the highest character in the keydef[index] array.

Issues That Occurred:

I discovered that the last block of 128 bytes sent to the WLKB232 were not always filled with 128. Early version of firmware did not enforce the 128-bytes, and the PIC24F datasheet says that the entire 64 words should be written. Now that I force the full 128 bytes in the last block .. I discovered that the V2 devices were not retaining the 'fixed' value array for the keypad keys (that get used when the user presses NumLock). And now that I am filling that last block, it must be filled with 0xC6,0,0xC7,0,0xC8 etc. starting at offset=960.

Interface Command:

This command reads or sets the interface parameters including interface selection, i2c address, spi mode, spi bit order, and pullup state. The read command is short and simple as below.

<u>/</u> <u>I</u> <u>0x5</u> <u>0x00</u> <u>0xKK</u>

It returns 6 ASCII characters (no other formatting):

1) '0', '1', or '2' for NONE, I2C or SPI interface
2) Next 2 characters are the i2c address as hex characters .. '21' or '3F' etc
3) 4th character is '0'-'3' representing SPI_MODE0 - SPI_MODE3
4) 5th character is '0' = MSBFIRST or '1' = LSBFIRST
5) 6th character is '0' for Pullups OFF or '1' for Pullups ON

To set a different interface or parameters the command is:

<u>/</u> <u>I</u> <u>0xA</u> cmdchars <u>0x00</u> <u>0xKK</u>

When sending this command, all cmdchars are ASCII .. EXCEPT the i2c address which is a single binary character. Not sure why I did this but there you go. So:

1) First cmdchar = '0' '1' or '2' to specify the Interface NONE, I2C, SPI
2) 2nd cmdchar is <u>binary</u> I2C address, like 0x21, 0x5D
3) 3rd cmdchar is '0', '1', '2', '3', spi mode
4) 4th cmdchar is '0' = MSBFIRST or '1' = LSBFIRST
5) 5th cmdchar is '0' = pullups OFF, '1' = pullups ON

The response to valid parameters is an ASCII string "ACK". Anything else will return a string "NAK".

I2C Command:

Needed a command to set the I2C address, and read it back as well so the configuration app can display it for the user. Here is the command to read back the value from the RS232 port:

<u>/</u> **<u>A</u>** **<u>0x5</u>** **<u>0x00</u>** **<u>0xKK</u>**

It returns two hex characters representing the address .. 08 – 77.

And here is the command to write a new I2C address:

<u>/</u> **<u>A</u>** **<u>0x6</u>** **<u>0x71</u>** **<u>0x00</u>** **<u>0xKK</u>**

This command sends one binary byte to set the address to 0x71

I2C Info

An I2C interface was added in Oct, 2025. This is a level-shifted interface that allows for 1.8V to 5V I2C busses to be connected with the (new model) WLKBI2CS in the role of slave. Its I2C address defaults to 0x21, but can be set to 0x08 to 0x77 through the serial port, or the I2C port. The RS232 port is retained and used for configuration of all other device properties. The DB9 is replaced with 8-screw terminals plus an 8-pin flat-cable connector – either can be used.
The command set is very limited ..

1) Ax .. set the WLKBI2CS i2c address to (7-bit) x
2) Rx .. read 1-20 bytes of data from WLKBI2CS. When data ends it returns 0x00 to fill the data request. I2C master has to detect 0x00 and toss. If master requests more than 20, WLKBI2CS returns only 20.
3) V .. return WLKBI2CS firmware version. This command causes WLKBI2CS to put the firmware version into the output buffer. Rx command must then be issued to read it (typically R6 but can be R1 – R10

Something I did not understand about the way i2c works: In the PIC24F .. slave interrupts are generated for (optional)START, ADDRESS, DATA, STOP. So when reading, after address is detected with the D_NOT_A status bit, the first byte of data has to be 'queued up' in the transfer register so it is ready for the next DATA interrupt. Then each subsequent DATA byte read must queue up the next. But the last one CANNOT queue up another byte .. because it would be overwritten by the one that gets queued when the next ADDRESS arrives. So how does the slave handle this when it does not know how many bytes the master will retrieve? It cannot .. so the master must tell it with a WRITE that precedes each read. The first transfer of any read is a write of the number of bytes that will be requested, followed by that number of reads. The slave then knows to skip the 'data queueing' on the last one so that the next read ADDRESS received can 'pre-queue' the first byte.
Update 11/8/25. The above is too complicated for users .. first write a command to tell it how many bytes will be transferred … then detect when the last one has been read and DO NOT QUEUE up the next one for return. If the user gets out of sequence, they can loose a byte which will confuse them – and I cannot debug their logic. The simplest way is to return one byte at a time. User sends a 'R'ead command, it gets queued up, and user reads one byte. No need to tell it how many bytes, OR TO READ THAT EXACT NUMBER or get out of sync and loose a byte.

I2C Buffer on PIC24F .. Min HIGH level = .7VDD = 2.31V
Max LOW level = .3VDD = 0.99V

My 2.4V pullup voltage on the level shifters meets this, and allows this pullup voltage to work with 1.8V external. When ext voltage is 1.8V, the .6V drop of the Mosfet body diode lets the internal voltage pullup to 2.4V -- and does not pull higher so that the external voltage is not pulled up over its rail. Tis is why had to drop the pullup to 2.4V – if it was higher, it would pullup higher and would also pullup the external voltage higher possibly damaging the external 1.8V circuits.

SPI Info

If user wants to tie /CS low (in single-slave SPI system), that works. I tested it – was afraid that the SPI engine might expect the /CS to toggle before it will accept data and cause interrupts. But it works fine with /CS tied permanently low, and can be Written to, and read just as if the /CS line was toggled at the appropriate times. So this allows users to use a 3-wire SPI.

SPI has a different lower limit of I/O voltage. This is because the PIC24F outputs are not open drain. So the MISO signal which is the only SPI output, goes up to 3.3V, the forward diode drop of the FET pulls the external input up to 3.3 - .6 = 2.7V. So 1.8V logic would be damaged.

Additional Diagnostic Commands for I2C and SPI

For testing during development, as well as testing each production unit, these additional unpublished I2C and SPI commands will be added:

1) 'T..' Stuff the included 40 bytes of data into the i2c_buf -- to be read-back and compared with the pattern sent. When WLKBI2CS receives this command it flushes the existing buffer so that the provided data is the only thing in the buffer. Test program then uses normal R40 command to return the 40 bytes and compare.
2) 'Z' repeatedly write 40 random bytes, and readback and compare and continue is a loop that can be run 50k times or more.
3) 'E' .. return the last error code (0x1 – 0xff). It is put into the isr buffer just like the version string .. This command also clears the error so the LED stops flashing