Serial (I2C)LCD Controller IC BV4638/9

BV4638/9

Serial (I2C) LCD Controller

Product specification February 2013

©ByVac

Contents

Rev	Change
Feb 2013	Preliminary
Mar 2013	V+ and GND corrected

1. Introduction

The BV4638/9 are two IC's. The BV4638 has an I2C interface whereas the BV4629 has a serial interface.

It is a programmed PIC microcontroller that makes interfacing to a typical character LCD display not only simple but it only uses a couple of processor lines instead of the normal 6 that would be required.

It is designed to interface with an LCD display that uses a HD44780 or similar controller. *This is just about every LCD character display available*.

The only difference between the two IC's is the programming.

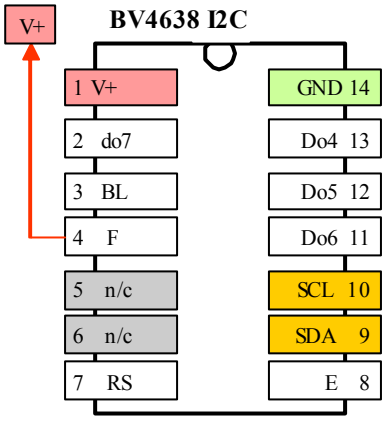


Figure 1 BV4618 I2C Layout

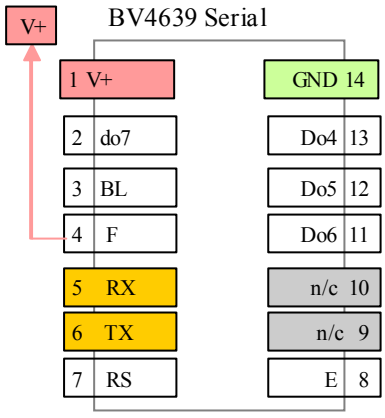


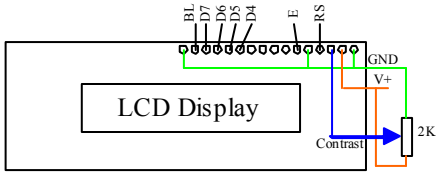
Figure 2 BV4619 Serial Layout

As can be seen from the figure the device is very small.

2. Features

- Back light fed by 20mA o/p port
- Serial interface and LCD features
- 1,2 and 4 line displays
- Supply current 3.5mA without display
- Easy to use asynchronous serial interface requiring only 4 connections or I2C interface
- Command set based on simple commands
- Only 2 data lines required, transmit and receive. The device will work with transmit only.
- Multiple devices can share the same data lines. This also applies to the serial device.
- Each device has it's own user configurable address, up to 26 devices, more on I2C.
- Automatic Baud rate detection (serial) up to 115200 from a select set of Baud rates.
- Common protocol used throughout range, devices can be mixed on same data bus
- Free Terminal software for Windows

3. Electrical interface



Most LCD displays need a contrast potentiometer connecting across the supply as shown, values anywhere between 1k and 10k will do. To try the display simply connect the contrast pin to ground it wont look very nice but it will give some results.

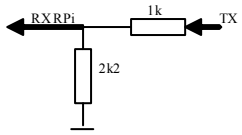
3.1. Voltage

The IC will work from 2.5 to 5.5 Volts and so will work with 3.3V displays. The logic will be at the supply voltage so if a 3.3V microcontroller is being used then the voltage should be 3.3V, see below under I2C for the exception.

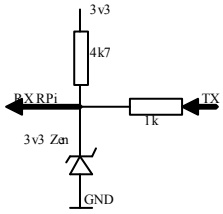
3.2. Serial

The serial device will automatically detect the Baud rate after sending 3 carriage returns. It will interface to microcontrollers or a USB to TTL serial controller. For interfacing to something like a Raspberry Pi it will be necessary to use 3.3V as this is the only voltage that the RPi will accept. This means that a 3.3V display is

required. As an alternative (see I2C) a 5V display could be used but some form of level translation on the o/p is required.



The above is a very simple circuit fro limiting the voltage from the TX line from 5V to approximately 3.3V.



This is probably a better circuit as it will not allow the input to go above 3v3 because of the Zenner diode.

3.3. I2C

The I2C default address for this device is **0x62, 8 bit and 0x31 7 bit**. When using the I2C device (BV4638) the problems associated with 5V and 3.3V are irrelevant as the pull up resistors on the master circuit will be connected to the 3.3V rail. As the I2C is open collector the voltage swing will be 3.3V.

In other words, it is possible to supply the IC with 5V and thus use a cheap LCD display, but connect the I2C SDA and SCL lines to a 3.3V device without doing any damage.

Pull up resistors are required on the SDA and SCL lines. These may be provided already on the master device. In the case of the Raspberry Pi, they are.

3.4. Factory

Pin 4 is the factory reset pin and under normal use must be tied to V+. If this pin is connected to ground and then power applied for a short time (100mS). The EEPROM will revert to the factory values. This is useful if an unknown address has been saved to EEPROM. Without the address it will not be possible to access the device.

The factory reset will **not** reset the default sign on screen, that will remain what it was set to.

3.5. Back Light

The back light is supplied by pin 3. Pin 16 is connected to ground and pin 3 can source or sink up to 20mA.

4. Serial LCD Command Set

This is for the serial BV4619 IC, the I2C section is later in the text.

NOTE there are two distinct command sets. The system command set, commands normally using upper case and the device command set. The communication command set is described in the introduction to SV3 at the back of this text..

All commands MUST be followed by a single CR – This is a byte with a value of 13 . So for example the command "ab1" is actually the following bytes 97 98 49 13. The device will not respond until it receives the 13.

*** Default device address 98 ('b') ***

LCD Command Set	
'b' (98)	Back light
'c' (99	LCD controller command
'd' (100)	LCD controller data
'p' (112)	Print EEPROM to LCD
'h' (104)	Home cursor
'k' (107)	Home cursor and clear
's' (115)	Sets cursor position
't' (116)	Text mode

Table 1 Serial LCD Command Set

Table 1 is a command summary of all of the LCD commands.

All of the above commands require a device address to be specified and command examples are shown using the default address 'a'

NOTE: The buffer size is 32 bytes and so no single command should exceed this.

4.1. b

Name: **Back light**

Turns the back light on and off

Terminal Use **ab1**

Bytes sent in above example:

97 98 49 13

This command controls the digital output on pin 15, '1' (49) makes it high and '0' (48) makes it low. The command expects a text number.

4.2. c

Name: **LCD Controller Command**

Sends command directly to LCD controller

Terminal Use **ac1**

Bytes sent in above example:

97 99 49 13

This will send a command to the LCD controller. A command is sent when the RS line is low and so this will set the line low before sending the byte. The command is a number is decimal text.

Some examples:

ac1 clears the display

ac192 moves cursor to second line (on most displays)

Bytes sent in above example:

97 99 49 57 50 13

4.3. d

Sends data to the LCD display

Terminal Use **adJKL**

Bytes sent in above example:

97 100 74 75 76 13

Send data to the LCD controller (RS line high). The data in this case is not a text number but the actual byte and so in the above example JKL would be displayed on the LCD screen.

4.4. p

Name: **Print from EEPROM**

Sends data from the EEPROM to the LCD

Terminal Use **ap10**

Bytes sent in above example:

97 112 49 48 13

This will get the contents of the EEPROM starting at the address given and send it to the LCD screen until either the end of the EEPROM is reached or 0 is encountered.

If a 1 is encountered then the next byte will be sent as a command. In this way there is full control over what is displayed

This is very useful for storing fixed messages and then printing them out. The Sign on message is stored in EEPROM and so this can be customised to suit.

4.5. h

Name: **Home cursor**

Places cursor in the home position without clearing any text.

4.6. k

Name: **Home cursor and clear text**

This is the equivalent to clearing the screen, moves the cursor to the home position and clears the screen.

Terminal Use **ak**

Bytes sent in above example:

97 107 13

4.7. s

Name: **Positions cursor**

Positions cursor at a particular point on the screen. The command format is:

<address><s><row>,<column>

Row and column start at 0

Terminal Use **as0,8**

Bytes sent in above example:

97 115 48 44 56 13

4.8. t

Name: **text mode**

When this command is used the display will stay in this mode until esc (27) is received. All text entered is displayed on the screen, if it comes to the end of a line it will go to the start of the next line, if it gets to the end of the display the text will scroll.

This command does not support back space, carriage return or line feed.

5. I2C LCD Command Set

This is for the I2C BV4618 IC

NOTE there are two distinct command sets. The system command set, commands normally using upper case and the device command set. The communication command set is described in the introduction to SV3 at the back of this text..

LCD Command Set	
1	LCD controller command
2	LCD controller data
3	Back light
4	Print EEPROM to LCD
5	Home cursor
6	Home cursor and clear
7	Sets cursor position
8	Text mode

Table 2 I2C LCD Command Set

The I2C command examples will follow the BV4221.

The BV4221 is a USB to I2C convertor that works by translating text commands to I2C commands using the following simple single letter commands:

s – sends start along with a default write address

p – sends a stop

r – sends a restart using the write address +1

g – gets a byte from the i2c bus

g-2 gets two bytes form the i2c bus

5.1. 3

Name: **Back light**

Turns the back light on and off

s 2 <1 or 0> p

0 will turn off the back light

5.2. 1

Name: **LCD Controller Command**

Sends command directly to LCD controller

s 1 <byte> p

This will send a byet to the LCD controller. A command is sent when the RS line is low and so this will set the line low before sending the byte.

Some examples:

s 1 1 p clears the display

s 1 0x94 p moves cursor to second line (on most displays)

If using the BV4221 the 0x prefix is left off.

5.3. 2

Sends data to the LCD display

s 2 <byte> p

Send a byte to the LCD controller (RS line high). Only one byte can be sent at any one time with this command.

5.4. 4

Name: **Print from EEPROM**

Sends data from the EEPROM to the LCD

s 4 <EEPROM address> p

This will get the contents of the EEPROM starting at the address given and send it to the LCD screen until either the end of the EEPROM is reached or 0 is encountered.

If a 1 is encountered then the next byte will be sent as a command. In this way there is full control over what is displayed

This is very useful for storing fixed messages and then printing them out. The Sign on message is stored in EEPROM and so this can be customised to suit.

Example, prints the default message:

s 4 0x20 p

If using the BV4221 the 0x prefix is left off.

5.5. 5

Name: **Home cursor**

s 5 p

Places cursor in the home position without clearing any text.

5.6. 6

Name: **Home cursor and clear text**

s 6 p

This is the equivalent to clearing the screen, moves the cursor to the home position and clears the screen.

5.7. 7

Name: **Positions cursor**

Positions cursor at a particular point on the screen. The command format is:

s 7 <row> <column> p

5.8. 8

Name: **text mode**

When this command is used the display will stay in this mode until esc (27) is received. All bytes entered is displayed on the screen, if it comes to the end of a line it will go to the start of the next line, if it gets to the end of the display the text will scroll.

This command does not support back space, carriage return or line feed.

6. EEPROM Values

The following locations are used to store information about the display. These of course can be changed using the standard SV3 write EEPROM command described at the end of this text.

See also that text for the system value locations.

EEPROM Location	Default	Description
16	0x80	Line 0 LCD command
17	0xc0	Line 1 LCD command
18	0x94	Line 2 LCD command
19	0xd4	Line 3 LCD command
20	2	Number of rows
21	16	Number of columns

32		"default message"

The above can be changed using the SV3 write to eeprom command.

7. User Notes

The display by default is set for 16 x 2 lines and so the use of commands that use this information (s and t) will rely on the values at EEPROM locations 16 to 21.

On most displays the values in locations 16 to 19 are correct, you can verify this using the 'c' command:

ac128<enter>

will put the cursor at the beginning of the first line.

ac192<enter>

will put the curser at the beginning of the second line

You may have a display that is not the same (unlikely) so these values can be adjusted. The default settings are for a 16x2 display so if you have a different configuration then change the values in the EEPROM locations 20 and 21

8. Revisions SV3

Rev	Change
Nov 2012	Update from IASI
Nov 2012	Version 1.0

9. Introduction to SV3

Serial Version 3 is an enhancement of the original IASI protocol that allows smaller more compact packets and thus increasing the efficiency. It is backward compatible but some of the superfluous commands have been removed. The default now is to connect to a microcontroller rather than RS232 and some devices do not support RS232 voltage levels any more.

The SV3 is a common standard that makes it much easier to control and use hardware from either a standard communication interface (terminal) or a microcontroller.

It is based on a very simple command set that does not require hardware handshaking and is therefore very easy to set up.

All of the transfers (unless otherwise stated) to and from the host are in text. This makes it easier to interface to common programming languages such as VB or Python.

10. SV3 Electrical Interface

The device has very simple requirements. A power supply, transmit and receive lines as shown in table E1.

The interface will always interface to 5V logic and may also have a provision for receiving 12V RS232 signals. A five pin connector is used with normally only 3 or four pins being connected at any one time.

There **may** be two receive lines, pin 1 receive line will accept normal 5V logic as presented by a microcontroller pin or UART and pin 4 will accept positive an negative voltages up to 15V that are normally present on a standard RS232 interface. Pin 4 will also invert the logic which is also normal for this interface.

The Baud rate is automatically detected at start up on the first or second receipt of Carriage Return (#13). The detection is from a fixed set of standard Baud rates normally: 9600, 14,400, 19,200 and 38,400 and sometimes 115200.

The transmit pin has an open collector output that has a pull-up resistor on board connected through a jumper. This allows more than one device to be used on the same serial line, only one jumper should be shorted. See the section on multiple devices for further information.

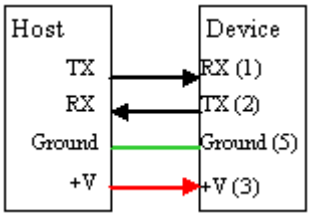
The data byte is represented by 1 start bit, 8 data bits and 1 stop bit. The idle state is normally high so a stop bit begins by pulling the serial line low.

Pin	Description
1	RX receives data from the host
2	TX sends data to the host
3	+V 2.5 to 5.5V (but see data sheet for that device)
4	RX input, can accept _15V and -15V will also invert the data– **not implemented on all devices**
5	Ground

5 Pin connector

The +V pin is in the centre and so if the plug is inadvertently connected backwards there is no consequence.

11. Serial Connections



The device is designed to work with a microcontroller or similar device that outputs logic level signals, in turn the device will also output a logic level signal.

The power supply to the device depends on the device, the electronics will work from 2.5 to 5.5V, however it may be connected to a peripheral that requires more voltage, relays and some LCD displays require 5V for example. The logic level of the output pin will be the same as the +V pin 5.

There is an alternative input provided on some devices for the RS232 standard on pin 4. This is just the input, the output on pin 2 will remain at logic levels. The input on pin 4 can accept up to + and – 15V. It will also invert the input as this is what is needed for RS232.

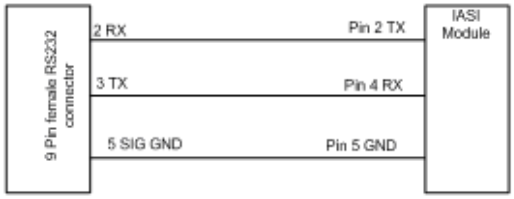


Figure 3 Connection to a PC

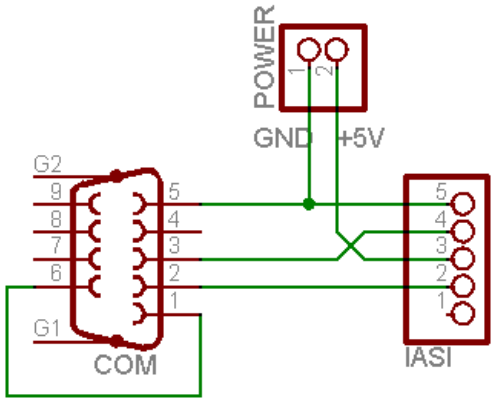


Figure shows the connections to a 9 pin D type connector found on some PC's. This is the RS23 connection. The RX line cannot be used for multiple devices in this configuration.

12. Start Up

The Start up procedure will establish the Baud rate of the host. The Baud rate will be selected from a table of standard Baud rates as follows:

1. 2400
2. 4800
3. 9600
4. 14400
5. 19200
6. 38400
7. 57600 [1]
8. 115200 [1]

[1] These two rates are not available on all devices.

The device will match the host baud rate by using the following procedure:

Send CR with a delay or 50ms between until the device sends back '*'. This will normally take 3 CR's

Pseudo code:

for j = 1 to 5

putc("\r")

delay_ms(50)

if getc() = '*' break

next

CR is 13 or 0x0d or "\r"

'*' is 42 or 0x2a

The auto detect can be overridden with a fixed Baud rate by setting the appropriate value on the devices EEPROM. If multi (see eeprom) is set to 0 then any time CR is sent on its own, the device will respond with '*'.

13. Command Format

All devices have an address which is one byte and can be any value within the range 97 to 122. The user specifies the address and so can be set up for multiple devices.

The **default address is given in the data sheet for the individual device** and all devices must be addressed although there are some global commands that address all of the devices at once.

There are basically two sets of commands, those which are common to all devices, these are described here and normally be in the range 65 to 90 and those that are device specific which are described in the devices datasheet.

This section deals with the system commands.

The command format in general is therefore:

<address>{commands}<EOL>

The device will always (unless reset) respond with ACK which by default is 6. EOL is the end of line character which by default is 13

14. Numbers

Some commands are followed by data. Where this is the case and a number is required it is sent as text. Numbers are also received as text.

As an example if a command required two numbers say 12 and 120, the command format may look something like below.

<address><command><"12"><delimiter><"120"><EOL>

The address would be a single byte corresponding to the devices address. The command would also normally be a single byte. The "12" would be 2 bytes, and "120", 3 bytes.

The delimiter between the numbers is normally a ',' but can be a space or any non digit, just as ling as it separates the two numbers.

If the address was 99 and the command say 50 then the actual bytes sent to the device would be:

99 50 49 50 44 49 50 48 13

Just to re-iterate, the "12" is sent as text so this is 2 bytes with values of 49 and 50 that correspond to the ASCII values '1' and '2'.

This is the general 'rule' however for efficiency this may be overridden by the individual device, if that is the case it will be clearly explained in the data sheet for that device.

15. Non/Inverted Mode

As previously mentioned the device is capable of operating with a standard RS232 communication port (inverted) and a microcontroller (non-inverted - default). The device will accept either signal as it has two inputs but it only has one output.

The output can be set as inverted to comply with RS232 but doing so will prevent any other device from working on the same bus. The inverted mode is not the normal way of working as this protocol allows many devices to share a serial bus.

The recommended method if an RS232 9 pin connection has to be used is to interface through a conversion device to make the RS232 behave as a microcontroller word. A MX232 for example.

16. Data Packet

A packet consists of a series of bytes followed by byte 13 (Carriage Return CR). It follows then that all commands must end with CR for the device to accept the command. The host should then wait for the device to return ACK (6) or NACK (21) before proceeding.

Using this protocol means that no hardware handshaking is required, for a microcontroller this means two less lines are required. There are some implications however and that is that the device must wait until the full command is received before acting on it. The buffer size default for a device is 64 bytes but this may vary. The buffer cannot be exceeded so will restrict some forms of communication.

17. Receiving Data

Some commands will return information. Where this happens the ACK or NACK will be received at the end of the received data. The ACK will instruct the host that it is okay to collect the data. See the Version command for a simple example of this behaviour.

18. Non-Addressable Commands

The interface is completely software driven, all commands and configuration are done through a serial interface. The only exception to this is the hardware factory default restore.

The following commands do not require a device address. These are 'global' in that all devices on the bus will respond or take action.

18.1. Command 1

This is the discovery command it purpose is to find out what devices are on the bus in an automated way. It can also be used to check that the devices exist as expected. The command requires two bytes to be sent:

1 13

Following this each device will place its address on the bus in turn using the address value in a delay calculation.

Each device has 2mS to place its address on the bus, the lowest address 97 will send its address immediately where as a device with address 122 will take (122 - 97)*2mS = 50mS.

Example, two devices are on the bus:

1 13

99 102 <output from devices>

No ACK is given for this command so the host must wait at least 50ms for all the devices to respond.

18.2. Command 2

Toggle inverted. The output signal will be inverted, this is mainly to cater for devices that are connected to RS232 that requires an inverted signal.

18.3. Command 3

This will reset all devices as if they had just been powered up. Following this command one or two CR is required to establish the Baud rate.

19. Addressable Commands

The following commands require the device address to be sent first, only the device with that address will respond.

The command byte values have been chosen so that they are in the printable range. This makes it easier to debug using a simple terminal emulator. To read the first 6 bytes of the EERPOM for example would simply be:

aR0,6<CR>

The device will then output readable text.

19.1. Summary

Command	Description
'W'(87)	Write to EEPROM
'R'(82)	Read EEPROM
	Reserved
'I'(73)	Toggle inverted
'C'(67)	Reset device
'V'(86)	Version
'D'(68)	Device ID

Note that examples will use the default address of 97.

The first few bytes of the EEPROM contain system information and can be changed with the above commands. The system details along with the EEPROM address is as follows:

EEPROM Address	Default Value	Description
0	0	0xff causes factory reset
1	97	Device address
2	6	ACK character
3	21	NACK character

4	0	Baud rate [1]
5	1	Error reporting 0 is off
6	13 or 0xfc	Default CR or end of line character
7	0	Multi
8		
9		
10		

Bytes up 10 reserved for device or future use.

[1] The Baud rate has the following values:

0. This is the automatic Baud rate. The actual Baud rate will be determined by the host when 13 is first sent.
1. Baud rate is fixed at 2400
2. Baud rate is fixed at 4800
3. Baud rate is fixed at 9600
4. Baud rate is fixed at 14400
5. Baud rate is fixed at 19200
6. Baud rate is fixed at 38400
7. Baud rate is fixed at 115200

The latter Baud rate may not be available on all devices.

19.2. Write to EEPROM

This device has an internal EEPROM with an address range 0 to 255. The user can use this as general non-volatile storage but should refrain from using addresses below 10 as they may be used for the system.

The command has the following format:

<address><”eeprom address”><delim><”value”><EOL>

If things do go wrong with any of the system values then a hardware factory reset can be performed to restore the EEPROM back to its default settings.

As an example to change the address from 97 to 102 the following bytes will be sent by the host:

97 87 49 44 49 48 50 13

A comma (44) is used as a delimiter to separate the EEPROM address from the value. EEPROM address 1 contains the device address.

NOTE: When altering a system EEPROM setting a reset is required for it to take effect.

19.2.1. Address

This EEPROM location contains the device address. It is important to set the address between the values 97 to 122, no checking is made by the device.

19.2.2. ACK character

By default this is 6 but can be changed using the EERPOM Write command. The effect will not be implemented until the device is reset.

19.2.3. NACK character

By default this is 21 but can be changed using the EERPOM Write command. The effect will not be implemented until the device is reset.

19.2.4. Turn off Error reporting

By default error reporting is enabled and this will be reported and an output prefixed by Error, for example ‘**Error 2**’. This may get in the way of the program trying to control the device and so it can be disabled with this command. The effect will not be implemented until the device is reset.

19.2.5. CR Character

By default this is 13 which is the standard ASCII CR and the whole protocol relies on this being at the end of every command. It may be that this is unsuitable in some systems and so this can be changed.

Important: This does not apply to the automatic Baud rate detection that must be 13

19.2.6. Multi

Where there is only one device, sending EOL (CR) on its own will get a ‘*’ response back. This is very reassuring when testing and using the device. However in a multi-device environment. This behaviour can have a detrimental effect if all of the devices try to do it at the same time. Setting this value to 1 will turn off this behaviour.

19.3. Read EEPROM

The EEPROM values can be read with this command given a starting address and the number of bytes to read.

<address><read eeprom><”start”><”#bytes”>

97 82 48 44 49 54 13<host>

The output from the device will commence after receiving 13 and will consist of a string of data terminated with ACK.

The sting will be in the form of text delimited by ‘,’ and all of the values will be decimal. An example of output for the first 5 bytes of EERPOM would be:

“0,97,6,21,0”<ACK>

19.4. Device Number

Returns a number representing the device product number as a string

97 68 13<host>

"4111"<ACK> Returned by device

19.5. Toggle Inverted

Pin 2 on the electrical interface that supplies the output information (Tx line), can be supplied inverted or non-inverted (at reset, start up). Inverted is used if the device is connected directly to an RS232 PC Com (where the interface is available) port and non-inverted is used when the device goes through a converter (BV201, BV101) or is connected to a microcontroller.

At reset the device is always in the non-inverted mode. Some device do not support this command in which case an error will be generated.

97 73 13 – output is now inverted

97 73 13 – output is now non-inverted

Just ACK will be returned by the device, but this will be an inverted ACK.

19.6. Reset

Resets an individual device. The baud rate will need establishing again after this command is used.

This is similar to command 3 but works on a single device. A soft reset will normally be the same as a reset at start-up but this may not always be the case. Obviously no ACK will be returned by this command.

19.7. Version

Returns the firmware version as a string in the format "H.L"

An example of the transaction would be:

97 86 13 <host>

49 16 19 <ACK>

20. Error Codes

Error codes will be displayed if they have not been switched off.

An error code is output as **text** followed by NACK, as an example if error 2 occurred then the output would be:

0x45 0x72 0x72 0x6f 0x72 0x32 0x15

Code	Description
2	Unknown command, the command issued is not in the command table for this device.
3	Bad device address, the address specified is outside the address range.
4	Bad number, out of range for the command specified
5	Incomplete command, usually because not enough bytes have been sent for the specified command.
6	reserved

21. Connection and Configuration

21.1. Multiple Devices

The output of the device is an open collector, this means that many devices can be connected to the same serial bus. The only proviso is that there needs to be a pull up resistor somewhere on the bus.

On all SV3 devices there is a built in pull up resistor connected to a jumper that has is shorted out by a PCB track. In practice around three devices can be connected to the same bus without regard to this jumper or pull up.

If more devices need to be connected then the cumulative impedance of all of the individual pull up resistors will be too great for the serial bus to overcome. In this instance the track between the jumper pins needs cutting except on one device so as to retain at least one pull up.

The above of course applies to the output from the device (input to the host). If this is not required then it doesn't matter about the pull ups.

As the normal idle state is high (+V) the pull up system works well as only one device at a time will pull the bud sown to create a signal. This however does not apply to the inverted mode where the idle state is low. In this mode only one device can be connected to the host input. This is why this mode is not recommended.

22. Restoring Factory Defaults

The configuration of an SV3 device is contained in the EEPROM. As the user has full access to this it is possible that it may render the device unreachable. If the end of line character has been accidently changed to some unknown value it would take up to 256 attempts to find out what it was.

The default values for the eeprom can be reset by the following procedure.

1. Power down the device.
2. Use a shorting link (bit of wire – paper clip etc.) on the appropriate pins.
3. Power up the device, this will restore the factory settings.
4. Power down the device.
5. Remove the shorting link.

The shorting link position varies from device to device. See the data sheet for where it should go.

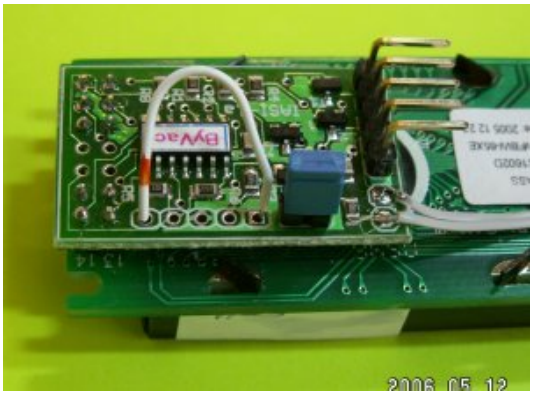


Figure 4 Example Shorting link

23. Revisions SV3

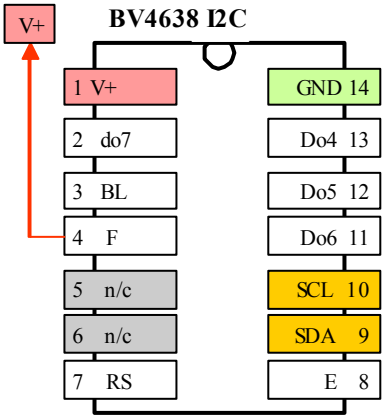
Rev	Change
Feb 2013	Preliminary

24. Introduction to I2C_SV3

This is the I2C equivalent of the serial SV3. This document contains the common instruction set (commands) for the I2C devices that use this protocol.

25. I2C_SV3 Electrical Interface

The interface follows the standard I2C.



Example of an I2C_SV3 Device.

Consult with the datasheet for the actual device as the pin locations may vary, the above is just an example of one particular device.

All I2C devices require a pull-up resistor to be attached to the SCL and SDA lines. The value of the resistor can vary widely but about 5k is the average used.

Most master devices already have the pull-up resistor incorporated into their circuitry and so the slave will not need these.

If the pull-up resistors are left out of the slave and master the device may still work but very unreliably.

26. Addressing

The most common problem with an I2C device is getting the addressing right, this is doubly confusing as there are specified, 8 bit and 7 bit addresses.

To make it clear I2C only has 7 bit or 10 bit addressing according to the standard specification. Where the confusion arises is that the read/write bit is the least significant bit of the address. When writing the bit is set to 0 and when reading the bit is set to 1.

When looking at this will a digital analyser you effectively get 8 bits of information, the first 7 bits are the address and the last bit is read or write.

As most microcontrollers and other digital equipment consider 8 bits as the norm it is easy to equate the 7 bits + 1 as an 8 bit value where an even address is for writing and an odd address is for reading. As an example if the 8 bit write address was 0x42 then the read address would be 0x43.

Where this becomes important is how the master software treats the I2C interface, if for example there is a function something like:

i2cWrite(char adr, char value);

Then it is likely that the address will require 7 bits as the function is a **write** function then it would be expected that it will add the last bit itself. There will also quite likely be a corresponding read function.

On better software the functions are more granular allowing the programmer more control over the bus. In this situation there will probably be separate start and stop functions and so the equivalent of the above would be something like:

i2cStart();
i2cData(char byte);

In this case the i2cData function will simply send that byte to the I2C bus and so an 8 bit address should be specified.

To convert an 8 bit address to a 7 bit address simply device by 2, so if an 8 bit address is specified as say 0x62 then the 7 bit address will be 0x31

27. Command Diagrams

The design of the interface has been purposely kept simple and so there are only a few standard sequences required.

Key

Master

Slave

S Start condition

P Stop Condition

A Acknowledge = 1

N Not acknowledge = 0

The BV4221 is a USB to I2C convertor that works by translating text commands to I2C commands using the following simple single letter commands:

s – sends start along with a default write address

p – sends a stop

r – sends a restart using the write address +1

g – gets a byte from the i2c bus

g-2 gets two bytes form the i2c bus

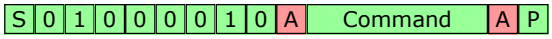
This format can be used to describe the commands that follow

27.1. Sending a command

This is designated as:

s <cmd> p

This sequence is used for simple functions where no data is involved. The I2C sequence, using an 8 bit address of 0x42 is:

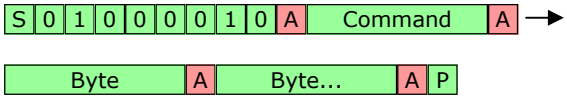


27.2. Sending a command with a parameter byte/s

This is designated as:

s <cmd> <param> p

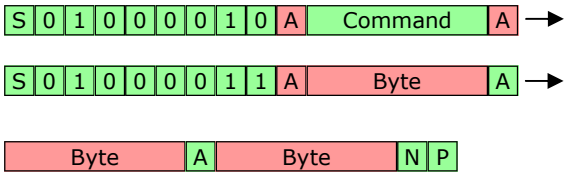
Some commands expect a parameter after the command. In this case the bytes are sent one after the other up to the maximum of 31 bytes. The stop command tells the slave that there is a command ready to be executed.



27.3. Receiving bytes from the salve

s <cmd> r g p

When receiving one or a number of bytes from the device a restart is required.



The command is sent with an even address (the R/W bit 0). The salve acknowledges this and then the command is sent followed by another acknowledge from the slave. At this point a restart 'r' is sent, this is the same as 's' except that the address is plus 1. The salve will acknowledge this and wait for the master to clock out the bytes.

It is the master that will now send the ACK or NACK (as shown in green). The correct protocol for the master is to send a NACK (not acknowledge) on the last byte, this will inform the slave that no more bytes are required.

This is only necessary where an undetermined amount of data is fetched form the slave, for a fixed number of bytes this protocol will not be necessary.

28. Commands

Check with the device data sheet for the I2C address.

28.1. Summary

Command	Description
0x90	Read EEPROM
0x91	Write to EEPROM
0x95	Reset device
0xa0	Version
0xa1	Device ID

28.2. EEPROM

The first few bytes of the EEPROM contain system information and can be changed with the above commands. The system details along with the EEPROM address is as follows:

EEPROM Address	**Default Value**	**Description**
0	0	0xff causes factory reset
1	<varies>	Device address

Bytes up to 10 reserved for device or future use.

28.2.1. Address

This EEPROM location contains the device I2C address. It is important to set the address to and **EVEN** value, no checks are made by the firmware.

28.3. Write to EEPROM

This device has an internal EEPROM with an address range 0 to 255. The user can use this as general non-volatile storage but should refrain from using addresses below 10 as they may be used for the system.

The command has the following format:

s 0x91 <eeprom address> <value> p

If things do go wrong with any of the system values then a hardware factory reset can be performed to restore the EEPROM back to its default settings.

As an example to change the I2C address from 0x62 to 0x82 the following is required:

s 0x91 1 0x82 p

If using the BV4221 the 0x prefix is left off.

NOTE: When altering a system EEPROM setting a reset is required for it to take effect.

28.4. Read EEPROM

The EEPROM values can be read with this command only one value at a time can be read.

s 0x90 <EEPROM address> r g p

If using the BV4221 the 0x prefix is left off.

28.5. Device Number

Returns a number representing the device product this is a 16 bit value returned as two bytes, the high byte being the first byte

s 0xa0 r g-2 p

If using the BV4221 the 0x prefix is left off.

This will return tow bytes representing the device number.

28.6. Reset

Resets an individual device.

s 0x95 p

If using the BV4221 the 0x prefix is left off.

28.7. Version

Returns the firmware version as a two bytes in the format "H.L"

s 0xa1 r g-2 p

If using the BV4221 the 0x prefix is left off.

29. Error Codes

There are no error codes for the I2C deviice

30. Restoring Factory Defaults

This will only re-instate the EEPROM values in the first 10 locations. For this device the factory reset pin is the one indicated by 'F' on the diagram for that particular device in question.

The sequence is:

6. Power down the device.
7. Connect the pin to ground
8. Power up the device, this will restore the factory settings.
9. Power down the device.
10. Remove the shorting link.

31. Trouble Shooting

This section has been added to answer frequently asked questions. The problems are usually caused because the master device has not had the I2C specification fully implemented.

31.1. Pulse Stretching

This is a method of I2C handshaking which is used in BV slave devices but it is not always supported by the master system. The symptoms are erratic behaviour, some commands will be accepted an others will not.

To explain: when the slave device is busy it holds the clock line low (normally only the master controls the clock line), the master should check that the clock line is high before sending the start condition. If it is low the master should wait until it is free.

Quite a few slave devices do not use pulse stretching and so this not being implemented in the master does not show up. However when dealing with relatively slow hardware, an LCD display for example (i.e. BV4219), this will become a problem. The work round is to make sure that the master recognises pulse stretching properly or introduce delays after each command.

31.2. Last Read NACK

When optionally multiple reads of a slave is required (the 0x55 command is a good example) the last read should send a NACK rather then a ACK. This informs the slave that no more reads from that command are required.

It has been found that some master implementations do not send a NACK on the last read. This causes the BV slave to remain in the (multi read) command effectively blocking any other commands.

31.3. Pull Up's

The most common problem when trying to get a new device going is to forget to put the pull up resistors somewhere on the bus. BV Slave devices do not have pull up resistor on board so they must be provided by the master (the BV4221 has pull up's) or provided externally. A value of around 5k is okay but this is not usually critical.