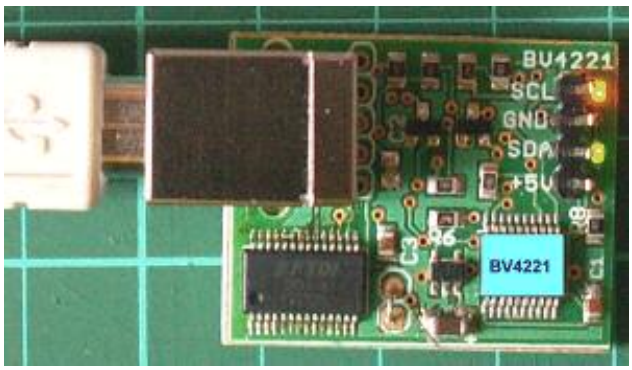


BV4221

USB to I2C Terminal

Product specification January 2008 V0.a

©ByVac

Contents

1. Introduction 3
2. Features 3
3. Physical Specification 3
4. Terminal Interface 3
5. Commands 4
6. The I2C Command Set 4
6.1. Command s 4
6.2. Command r 4
6.3. Command g 5
6.4. Command p 5
6.5. Command nn 5
7. System Commands 5
7.1. Command H 5
7.2. Command D 5
7.3. Command A 5
7.4. Command V 6
7.5. Command B 6
8. Examples 6
8.1. Using a 24LC256 EEPROM 6
8.2. Using BV4218 LCD & Keypad Board 6
9. Error Codes 6
10. Control by Software 6
10.1. End of Line 6
10.2. Waiting 6
11. Inspector Mode 7
11.1. Limitations 7

Rev	Change
Mar 2008	Preliminary
Dec 2008	Version 1.a, Now has clock stretch detect and higher speed. (free firmware upgrade available contact ah@byvac.com)
Jan 2009	Firmware works at approximately 100kHZ, previous datasheets incorrectly stated 400KHz.
Jun 2010	Added power & size spec.

1. Introduction

The BV4221 is a device for using I2C compatible equipment via USB. It provides a terminal like interface and is primarily intended for debugging and experimenting with I2C devices although could be used to drive I2C devices from a PC, see the section on programming.

In addition to the above there is an 'inspector' mode that allows 'viewing' of the I2C bus which is useful for debugging existing applications.

2. Features

- I2C up to 100kHz * v1.a
- I2C clock stretch detect * v1.a
- Command Driven
- Automatic Baud rate select
- USB driver easily obtainable
- 5V out to drive external I2C circuit – short circuit protected (100mA)
- I2C pull up resistors incorporated
- Inspect I2C up to 50kHz bus speed
- LED indication of SDA & SCL lines
- Size 25x38x15 high (mm)
- Weight 7g
- Power (idle) 15mA (powered from USB)

3. Physical Specification

The device consists of a USB connector and a 4 pin I2C connector. The USB supplies power to the I2C device via a 5V regulator. This gives short circuit protection for the USB bus.

The I2C output pins have the following designated pins.

Pin	Description
1	SCK – I2C clock
2	GND
3	SDA – I2C data line
4	+5V output to drive external equipment

Table 1 IC Pin Description

One or more external devices can be connected to this bus, the BV4221 acts as a master device that can write to and read from the external devices.

Two LED indicators are provided that give an indication of the SDA & SCL lines. When they are on the lines are high and when off the lines are low. This is useful as it indicates if the bus is free or has become locked up. There are two 5k6 resistors, one connected to the SCL line and the other connected to the SDA line. These are the required pull up resistors for the I2C bus so there is no need to provide them else ware.

4. Terminal Interface

The device should be plugged into a spare USB socket. It will then ask for a USB driver. The chip used for the interface is a FTDI chip (FT232R) and a driver for this can be found here:

http://www.ftdichip.com/

The driver is the **VCP** driver and should be downloaded for your operating system. The Unbutu Linux system already has the driver installed.

When running the USB presents itself as a Com port. HyperTerminal or BV-Terminal (better) can be used to communicate. BV-Terminal is free and can be downloaded from:

www.pin1.org

Alternatively a programming language such as VB or Just Basic can be used for automated applications. The requirement is that it can talk to the COM device.

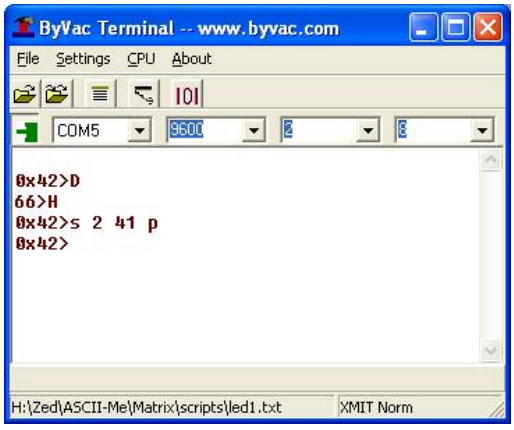


Figure 1 Using BV Terminal

The Baud rate is selected by interrogating the first CR (Carriage Return) character from the following five Baud rates:

- 9,600
- 14,400
- 19,200
- 38,400

- 11,5200

It will select the closest to the above and so the terminal should be set to one of those. There is no handshake or parity and the number of stop bits can be one or two.

When the device is first switched on a CR character should be sent first to establish the Baud rate, sending any other character will give unpredictable results.

nn>

The nn is the current I2C address. This can be changed with the A or s commands. There are two modes of operation, Decimal and Hex. In Hex mode the 'nn' is shown with a leading 0x. In hex mode all values are assumed to be hex so entering 37 for example would give a decimal value of 55.

At the prompt the BV4221 is ready to send commands to the connected I2C device/s.

5. Commands

There is a simple command set that controls, reads and writes to the device and the external I2C devices.

A command is a single letter and is **case sensitive**.

Command	I2C Command set
s[-nn]	Send start condition and address
r	Send restart command with address+1
g[-nn]	Get data bytes from slave
p	Send stop condition
nn	Send byte to slave
Command	**System Command Set**
?	List commands available
H	Set Hex mode
D	Set Decimal mode
A	Change I2C address
B	Re-invoke AutoBaud
V	Show firmware version
Command	Bus Debug
I	Inspect I2C data interactive
i	Inspect I2C data buffered

Table 2 I2C & System Command Set

Table 2 is a command summary of all of the I2C and system commands. Note that the system commands are upper case and the I2C commands a lower case.

6. The I2C Command Set

Sending I2C information to an I2C device usually consists of a start condition, the start address followed by the information and then a stop condition. Each device is different and so the commands that are sent will depend on the device.

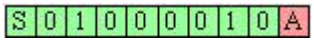
The terminal will send whatever is requested using the command format, this can be done individually (line by line) or it can be done as a long line up to the capacity of the input buffer which is 63 characters.

The diagrams use the following key colours:

6.1. Command s

Name: **Send start and address**

Format: **s[-nn]**



(The A (ack) will come from the slave)

This command is usually the first to be sent to an I2C device. This command does two things:

1. Sends the start condition
2. Send the current address or the address specified

The current address is the number before the prompt, this is the address that will be used if just a single 's' is used. To send a different address, specify this after the s with a leading '-' thus s-44

This will not effect the current address (see command A). It is used to temporarily switch addresses. Using this method enables several different devices to be addressed.

The temporary address set by s-nn will remain until the next stop 'p' command.

Examples

s 2 34 p

s-44 90 0 r g p

6.2. Command r

Name: **Restart command**

Format: **r**

Some devices, serial EEPROMS for example, can be written to and then read from using the restart command. This command does two things:

1. Sends a restart condition
2. Sends the current I2C address + 1

The address is always one more than the current address, the current address should be

an even number so that the restart will send a read request to the slave.

In I2C speak an odd address is an address with the w/r bit set to 1, i.e. read.

6.3. Command g

Name: **Get Bytes**

Format: **g[-nn]**

The g command is used for receiving bytes from the slave, issuing a **g** on its own will fetch one byte, when a number is used with a '-' prefix, that number of bytes will be fetched.

A slave device expects either a ACK or NACK depending on whether another byte is going to be fetched or this is the last byte. An ACK is sent if another byte is going to be fetched whereas a NACK is sent if this is the last byte. The g command takes care of this.

The programmer should make use of this as follows:

If a random number of bytes needs to be fetched from the slave device then this can be achieved by repeatedly sending the g command without any following parameter. This will fetch a byte from the slave with ACK indicating that another byte will be required. The LAST fetch should be 'g-1'. This will fetch a byte with NACK, telling the slave that this was the last byte.

Example:

The above represents a g on its own, the slave will send a byte and the BV4221 will respond with an ACK.

Here **g-2** has been sent, the first byte is received with ACK and the second byte (being the final byte required form the slave) is followed by a NACK

In summary:

g on its own will fetch a byte and tell the slave that another g will be coming along.

g-nn with a number following will get precisely that number of bytes and tell the slave no more fetches will be done in this session (i.e not without another start).

6.4. Command p

Name: **Stop**

Format: **p**

The stop command must be issued when a complete command sequence has finished. The master will release the bus so that another device can be addressed.

6.5. Command nn

Name: **Send byte**

Format: **nn**

A number on its own will be sent to the I2C device, see the examples section.

7. System Commands

Command	System Command Set
?	List commands available
H	Set Hex mode
D	Set Decimal mode
A	Change I2C address
V	Show firmware version
B	Re-invoke AutoBaud

The system commands enable the user to change the way the device works.

7.1. Command H

Name: **Hex mode**

Format: **H**

It is sometimes more convenient to work in Hex. After this command is issued all numbers, input and output will be treated as hex numbers. The prompt indicates which mode is in use. In hex mode the current I2C address is preceded by 0x, in decimal mode it isn't.

The mode is stored in EEPROM so that it will be remembered after power down.

7.2. Command D

Name: **Decimal mode**

Format: **D**

This turns off Hex mode and all numbers will be treated as decimal.

The mode is stored in EEPROM so that it will be remembered after power down.

7.3. Command A

Name: **Set default I2C address**

Format: **Ann**

For convenience the current I2C address is stored in EEPROM and is used with both the s and r commands. This address can be changed on a temporary basis using the s command but after power down it will revert to the address that is stored on the BV4221's EEPROM. The A command is used to change this address in EEPROM so that it will be remembered after power down.

7.4. Command V

Name: **Show firmware version**

Format: **V**

Simply displays the firmware version.

7.5. Command B

Name: **Re-Invoke AutoBaud**

Format: **B**

The automatic Baud rate select feature is invoked at the device switch on. The BV4221 waits for a CR and then sets the baud rate to what it finds in this first character.

The Baud rate can be changed by using B<CR>

changing the Baud rate on the terminals and then

<CR>

again to set the new Baud rate.

8. Examples

8.1. Using a 24LC256 EEPROM

Write 1 to 5 to EEPROM address 0. The EEPROM has a device address of 0xA0

H (set hex mode)

Aa0 (set device address)

s 0 0 1 2 3 4 5 p (write 1-5)

Read from those locations

s 0 0 r g-2 p

In the above read operation the EEPROM address counter has been reset back to 0000 and a restart command issued to read the slave. g-2 reads the first 2 bytes which will return 1 and 2.

s-a1 g-2 p

This next operation sets the address to read (odd number) and reads the next two bytes that return 3 & 4.

8.2. Using BV4218 LCD & Keypad Board

Assuming the default address of 0x42 is used:

s-42 1 1 p

This will clear the display

s 2 41 42 43 p (no need to set address again)

This writes ABC to the display

s 90 28 r g-5 p

This gets the first 5 bytes of the sign on message

9. Error Codes

The following error codes may be seen:

1. End of input buffer reached
2. Unknown command (sometimes displays as 3)
3. Number expected
4. I2C bus not free on master start condition, caused by slave not releasing bus.

There is also the LED indicators for the I2C SDA and SCL lines. When the bus is free these should both be high, illuminated. The master will release the bus on issuing a stop command 'p'. If both lights are not on after this command then one or more slave devices are holding the bus. Sometimes issuing a start 's' and then a stop 'p' can clear a slave device.

10. Control by Software

The BV4221 can be controlled by software hosted on a PC, this will enable simple hardware to perform complex applications.

The website www.pin1.org contains some example code written in Just Basic (JB). This language was chosen because it is extremely simple, has a good serial interface and is FREE.

Other languages of course can be used provided they can talk to the COMM port. Here are some hint's and tips that will be needed if a programming interface is used.

10.1. End of Line

The serial communication works by receiving any and all characters up to and including the CR (Carriage Return 13) OR LF (Line Feed 10). When a CR or LF is received the whole line is processed and the instructions within that line are passed to the I2C bus.

It is important that only CR is sent and not a combination of CRLF or LFCR as this will slow down the interface and delays may need to be added.

This is often carried out in languages using an escape character such as:

Print("some commands \r");

The \r or whatever it may be tells the output only to append ASCII 13, i.e. CR

In JB this is handled explicitly by a print routine that suppressed the normal CRLF behaviour and replaces it with a CR.

10.2. Waiting

The prompt **0x42>** will only appear once all of the commands have been processed so it is important to wait for this before sending the next command. The examples given in JB wait the '**>**' character.

11. Inspector Mode

Command	Bus Debug
I	Inspect I2C data interactive
i	Inspect I2C data buffered

The BV4221 is also capable of 'spying' on the I2C bus and displaying what it sees. Because of the limitation of the display, the maximum speed is 50k but this is useful enough for debugging purposes.

There are two options; the first option will display anything on the I2c bus immediately and the second option will buffer any received data until a stop condition is received whereby it will display what has been captured.

This second option allows faster capturing as the device does not have to display each event.

Both modes, monitor the I2C bus and report on the results. Pressing any key on the console will break out of the inspector mode.

When displaying the results the following key is used:

S Start condition

P Stop condition

A ACK

N NACK

To give an example, writing values 1,2 and 3 to the beginning of a serial EEPROM would be this command:

s 0 0 1 2 3 p

(The address is set to 0xA0)

The inspector would display:

SA0A00A00A01A02AP

Note, this command would need to be sent by another I2C master device for the inspector to see it. From the output it can be seen that the sequence begins with the Start condition (S) followed by the address (A0) followed by and ACK (A), etc. up until the stop command (P)

Reading back from the EEPROM:

s 0 0 r g-2 p

Produces this from the inspector:

SA0A00A00A**SA1A**01A02**N**P

There are some things to note about this. The 'r' command produces this SA1A, highlighted in bold above. The restart is S followed by the address plus 1 (A1) and the last A is the acknowledge from the slave. Note also that the last byte received is followed by a NACK prior to the stop command.

11.1. Limitations

I2C Analyser

The inspector mode will not work for a busy I2C, it is not intended as an I2C analyser and so don't expect to be able to connect it to a fully operational system and inspect packet after packet at high speed.

The inspector mode is intended for simple debugging and even learning about how I2C works. It is extremely useful for getting a system to work particularly with unfamiliar devices.

Buffer

The buffer when used in the 'I' mode is limited to about 90 bytes. There is no warning or checking for overflow. If more than 90 bytes are received before a valid stop condition than this is likely to crash the BV4221 so that it will need restarting.