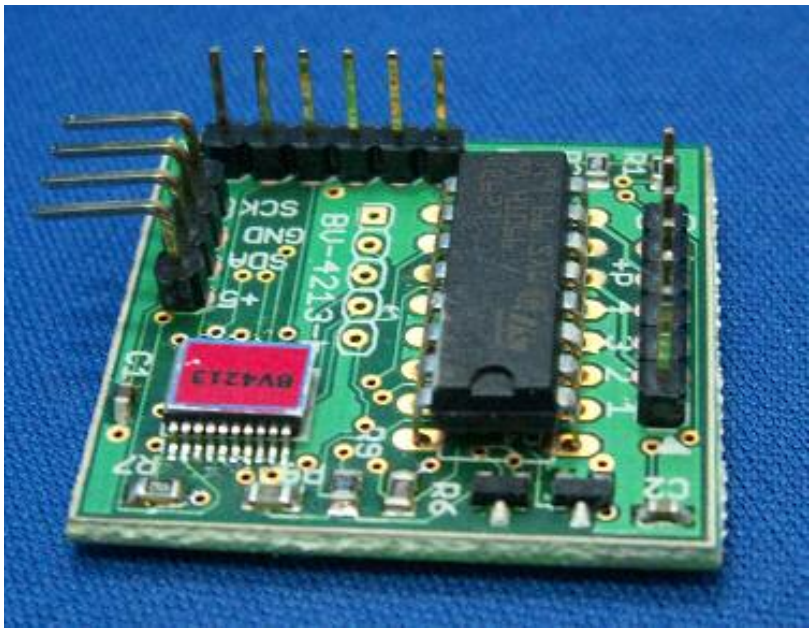
I2C-Motor Controller BV4213



BV4213

I2C-Motor Controller

Product specification October 2008 V1.a

©ByVac

Contents

Rev	Change
December 2007	Preliminary
Oct 2008	Updated ** Firmware version 2.a has some extra but is backward compatible apart form the factory reset command.
Jun 2009	Updated to make mode selection clearer – section 3.2

1. Introduction

The BV4213 is a multi purpose motor controller, capable of controlling DC motors with optional PWM (Pulse Width Modulation) and one stepper motor through an I2C interface or through a step interface with a choice of full or half step modes.

2. Features

- I2C up to 400kHz
- Simple command set
- Up to 4 DC motors single directions
- 2 DC motors with forward and reverse
- PWM for DC motors
- 1 stepper motor, full step half step
- Step complete output pin
- Step on change pin interface
- 1.2A peek output current
- Connects directly to motor, no external components required
- Motor supply up to 35V
- Operating voltage 4.5 to 5.5V
- Current <1mA @ 5V

3. Electrical Specification

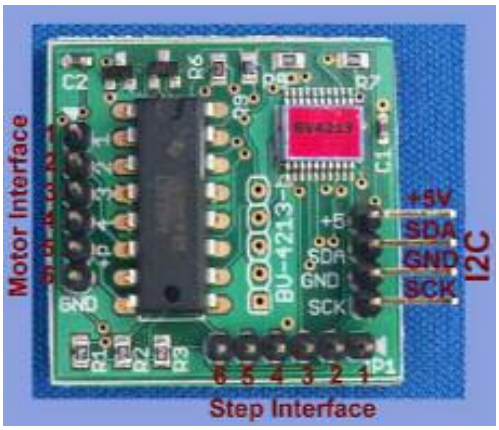


Figure 1 Annotation Diagram

There are three interfaces to the device, the I2C, the Step and the motor interface.

3.1.1.I2C

This interface provides the power to the device and the serial and clock lines.

Pin	Description I2C Pins
1	SCK
2	GND
3	SDA
4	+5V

Table 1 I2C Pin Description

The logic power is provided through this interface but the motor power is provided through the motor interface. The SDA and SCK lines require a pull up resistor to pin 4.

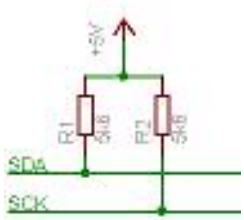


Figure 2 Pull up Resistors for I2C

3.2. Operation Mode

The device has two modes of operation, the I2C mode and the Step mode. This mode is determined at device reset or power up. The mode of operation is determined by pin 6 (/Step interface enable).

Pin 6	Mode
High when reset	I2C mode
Low when reset	Step mode

By default, no pins connected the device will start in I2C mode this is because there is a pull up resistor connected to pin 6.

In I2C mode pin 4 is an output the goes low at the end of step count, no other pins, except for ground are active.

With pin 6 enabled (low), a low to high or high to low transition of pin 5 will cause the motor to step once, direction being determined by pin 3 and half or full step, stepping being determined by pin 2. When pin 2 is low, full stepping is selected.

In this mode pin 6 is used to cut power to the motor when high (disable). This can save energy as the enable lines are activated while ever this line is low.

The I2C interface can still be used in this mode but should not be used to drive motors as unpredictable results may occur.

By default pins 2 to 4 and 6 are high and maintained so by a pull up resistor and so can

be left disconnected if this interface is not required.

Factory Reset

Pin 4 of the Step interface is the factory reset pin and this has a resistor to hold it high at switch on. By holding this pin low at power on a factory reset will take place, see section 8.

3.2.1.Step Interface

The step interface provides the end of step interrupt and an alternative to driving the motor with an I2C interface.

Pin	Description Step Interface
1	GND
2	Half - /Full Step
3	Direction
4	/End of Step (factory)
5	Step
6	/Step interface enable

Table 2 Step Pin Description

With pin 6 not connected to anything, it has an internal pull up resistor, this interface will be disabled. The only relevant pin that is always enabled is the end of step pin 4. When an I2C command for stepping has been given, this pin goes low when the end of step count has been reached. This pin can be monitored by the host processor.

With pin 6 enabled (low), a low to high or high to low transition of pin 5 will cause the motor to step once, direction being determined by pin 3 and half or full step, stepping being determined by pin 2. When pin 2 is low, full stepping is selected.

By default pins 2 to 4 and 6 are high and maintained so by a pull up resistor and so can be left disconnected if this interface is not required.

Factory Reset

Pin 4 of the Step interface is the factory reset pin and this has a resistor to hold it high at switch on. By holding this pin low at power on a factory reset will take place, see section **Error! Reference source not found.**.

3.2.2.Motor Interface

This is the output from the H-Bridge chip (L293D). The input is fed from a microcontroller that is controlled either by the I2C interface r the Step interface.

The supply for the motor is also provided by pin 5 which can be up to 35V. This is independent of the logic supply but must be present for a motor output to occur.

Pin	Description Motor Interface
1	Y1
2	Y2
3	Y3
4	Y4
5	Motor Power
6	GND

Table 3 Motor Pin Description

Y1-Y4 should be either connected to either 4 DC motors, 2 DC motors or 1 stepper motor

3.3. Typical Motor Connections

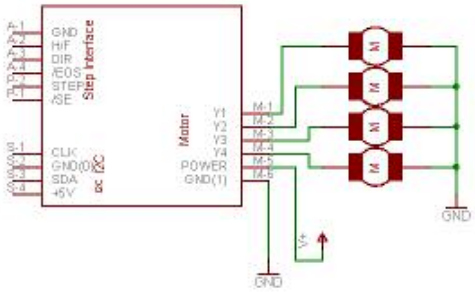


Figure 3 4 DC Motors

This is probably the least used but can provide an output for up to four motors. The motors will be capable of only going in one direction. I2C commands are provided that can control the output of each individual Y output.

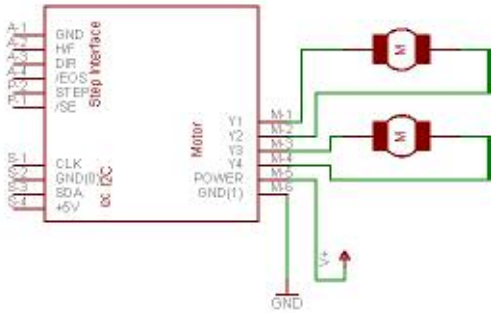


Figure 4 Differential Control

This shows the connection for driving two motors bi-directionally. In this mode Y1-Y2 is considered one channel and Y3-Y4 is considered the other. I2C commands are provided that can control these as a channel.

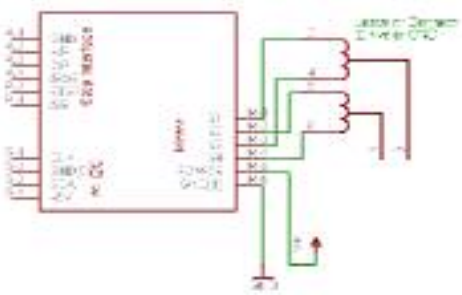


Figure 5 Stepper Motor

Shown here is a typical stepper set up. Stepper motors vary from 4 to 6 wires and shown is a 6 wire motor. Five wire motors have the centre tap wires connected together and 4 wire motors do not have a centre tap.

The centre taps can be either left disconnected, connected to +ve or connected to ground for whichever gives the best results.

4. I2C Command set

The format used by this device consists of a command, this is a number, followed by other bytes depending on that command.

There are three type of command, those referring to the DC-Motor, those referring to the Stepper-Motor and those referring to the system. The system commands enable changing of the device address etc.

Device default I2C address is 0x42

All I2C commands have one of two formats. Fro writing to the device:

<S-Addr><Command><data..><Stop>

and for reading from the device:

<S-Addr><Command><RS-add><data..[NACK]><Stop>

S-Addr is a start condition followed by the device address. The address here will always be an even number, i.e. with bit 0 clear (0).

All command begin by writing to the device and the first thing that is written is the command number. What follows after this is dependant on the command.

Where the command requires information from the device (read) it is necessary to send a start condition again followed by the address + 1, this is shown by RS-Addr above.

Some read commands can be terminated by sending not-acknowledge (NACK) before the supply of data is exhausted.

All command strings whether read or write terminate with a stop condition.

Command	DC-Motor Command Set
1	Enable output Y1,Y2
2	Enable output Y3,Y4
3	power saver 0 power off 1 power on
4	Differential control for channel A - Y1 and Y2
5	Differential control for channel B - Y3 and Y4
0x11	Output for Y1
0x12	Output for Y2
0x13	Output for Y3
0x14	Output for Y4
0x15	PWM for Channel A, 0 is off
0x16	PWM for Channel B 0 is off
Command	**Stepper**
0x20	Step Continuous
0x21	Step Stop
0x22	Step number of steps
0x23	Set step parameters (4 bytes)
0x24	Read step parameters (up to 6 bytes)
0x30	Set direction
0x31	Set Speed
0x32	Set mode half/full
0x33	Set Ramp

Table 4 LCD & System Command Set

0x30	**Set direction**
0x31	Set Speed
0x32	Set mode half/full
0x33	Set Ramp

Table 4 is a command summary.

5. The DC Motor Command Set

There are two sets of I2C commands, those dealing with the operation of one or more DC motors and those dealing with the stepper motor. This section deals with the former.

The method of writing to the device using the I2C protocol follows a consistent format, typically:

<S-Addr><Command><data..><Stop>

Where S-Addr is the start condition followed by the device address (0x42). Command is one of the commands given in the table. Data is one or more bytes and Stop is the stop condition.

Reading data requires a restart and this will be in the format:

<S-Addr><Command><R-Addr><data..[NACK]><Stop>

The restart address will be one greater then the start address, thus if the start address is 0x42, the restart address will be 0x43. Again the data can be one or more bytes read from the device.

Each command will have it's own format and is described in the text that follows. A start condition and address is always followed by a command. The device has an internal 32 byte I2C buffer, the effect of this is two fold:

1. Only 32 bytes can be sent to the I2C bus at any one time, this includes the

command itself and any restart address that may be required.

2. The device will only respond after the stop command is received.

Using a buffer enables the I2C bus to work at full speed even though the device it is connected to may be slower.

5.1. Command 1

Name: **Enable output Y1 & Y2**

Format: <**S-addr**><**1**><**byte**>**<Stop>**

Byte is either 1 for enable and 0 for disable. This directly controls the enable lines of the L293 H-Bridge driver chip.

5.2. Command 2

Name: **Enable output Y3 & Y4**

Format: <**S-addr**><**2**><**byte**>**<Stop>**

Byte is either 1 for enable and 0 for disable. This directly controls the enable lines of the L293 H-Bridge driver chip.

5.3. Command 3

Name: **Power Saver**

Format: <**S-addr**><**3**><**byte><Stop>**

Byte is either 0 to 1.

The circuit board is capable of switching off the power to the L293 chip. This saves approximately 16mA if the motor is powered down. The default is powered up (1), this command does not need activating at reset.

5.4. Command 4

Name: **Differential control for channel A**

Format: <**S-addr**><**4**><**byte><Stop>**

Byte is 0 to 4.

Channel A is Y1 and Y2 working together to supply a forward and reverse control to a DC motor. A value is sent from between 0 to 4 that will have the following effect:

BYTE	ENA	Y1	Y2
0	0	0	0
1	1	1	0
2	1	0	1
3	1	1	1
4	1	0	0

Byte is the value of the byte in the command and ENA is the enable for Y1 and Y2. The circuit for use with this command is shown in Figure 4.

Sending 0 will disable the channel by setting ENA low and thus the motor will be disabled. Setting 4 and 3 are similar but with the chip enabled, depending on the motor this may have a breaking effect.

1 & 2 send the motor in a forward and reverse direction.

5.5. Command 5

Name: **Differential control for channel B**

Format: <**S-addr**><**5**><**byte><Stop>**

Byte is 0 to 4.

This is the same as command 4 except this acts on Y3 and Y4.

5.6. Command 0x11

Name: **Y1 output**

Format: <**S-addr**><**0x11**><**byte**>**<Stop>**

Byte is either 0 or 1.

This directly controls the Y1 output and sets it or either 1 or 0. The channel must be enabled using commend 1 for anything to happen.

5.7. Command 0x12

Name: **Y1 output**

Format: <**S-addr**><**0x12**><**byte**>**<Stop>**

Byte is either 0 or 1.

This directly controls the Y2 output and sets it or either 1 or 0. The channel must be enabled using commend 1 for anything to happen.

5.8. Command 0x13

Name: **Y1 output**

Format: <**S-addr**><**0x13**><**byte**>**<Stop>**

Byte is either 0 or 1.

This directly controls the Y3 output and sets it or either 1 or 0. The channel must be enabled using commend 2 for anything to happen.

5.9. Command 0x14

Name: **Y1 output**

Format: <**S-addr**><**0x14**><**byte**>**<Stop>**

Byte is either 0 or 1.

This directly controls the Y4 output and sets it or either 1 or 0. The channel must be enabled using commend 2 for anything to happen.

5.10. Command 0x15

Name: **PWM for channel A**

Format: <**S-addr**><**0x15**><**byte**>**<Stop>**

Byte is a value from 0 to 255

This command modulates the output to ENA and therefore controls the power applied to Y1

and Y2. This can be used with any DC motor command.

5.11. Command 0x16

Name: **PWM for channel B**

Format: <**S-addr**><**0x16**><**byte**><**Stop**>

Byte is a value from 0 to 255

This command modulates the output to ENB and therefore controls the power applied to Y3 and Y4. This can be used with any DC motor command.

6. I2C Stepper Commands

Command	Stepper
0x20	Step Continuous
0x21	Step Stop
0x22	Step number of steps
0x23	Set step parameters (4 bytes)
0x24	Read step parameters (up to 6 bytes)
0x30	Set direction
0x31	Set Speed
0x32	Set mode half/full
0x33	Set Ramp

These commands relate directly to driving a stepper motor connected in a similar way to Figure 5. The default set up should work for most motors sufficiently to get it working. A command can be issued to step the motor a number of steps. There is an internal counter (16 bit) that counts down and this can be checked with command 0x26.

Any of the commands can be issued at any time even if the motor is still stepping.

6.1. Command 0x20

Name: **Step Continuous**

Format: <**S-addr**><**0x20**><**Stop**>

Causes the motor to continue receiving step pulses until the Step Stop command is received.

6.2. Command 0x21

Name: **Step Stop**

Format: <**S-addr**><**0x21**><**Stop**>

or

Format: <**S-addr**><**0x21**><**1**><**Stop**>

This will stop the motor. There are two acceptable versions. The first version will stop the motor and disable the output (ENA, ENB = 0). The second will stop the motor and leave the enable lines activated. This has the effect of breaking the motor but uses a considerable amount of power.

6.3. Command 0x22

Name: **Step number of steps**

Format: <**S-addr**><**0x22**><**2-bytes**> **<Stop>**

The number of steps is stored internally in a 16 byte register. This means that the maximum number of steps is 65,535. The host must split this number into two bytes and sent the highest byte first.

As an example suppose 20,452 steps are required. This is 0x4FE4, so the command would be:

<S-addr><0x22><0x4f><0xe4><Stop>

The high byte is sent before the low byte.

6.4. Command 0x23

Name: **Set step parameters**

Format: <**S-addr**><**0x23**><**x**> **<Stop>**

Where x is 4 bytes as follows:

First	Direction	1 or 0
Second	Speed	0 to 255
Third	Half/Full	0=full 1=half (stepping mode)
Forth	Ramp	0 to 7, 7= slow ramp up to speed.

The command expects all four bytes even if only one is changed. NOTE, the above parameters can also be changed individually, see commands 0x30 to 0x33

6.5. Command 0x24

Name: **Read step parameters**

Format: <**S-addr**><**0x24**><**RS-Add**><**x**> **<Stop>**

Where x is UP TO 6 bytes as follows:

First	High	Number of steps to go
Second	Low	Number of steps to go
Third	Direction	1 or 0
Forth	Speed	0 to 255
Fifth	Half/Full	0=full 1=half

		(stepping mode)
Sixth	Ramp	0 to 7, 7= slow ramp up to speed.

There is no need to read all of the 6 bytes, a NACK can be sent for the lat byte that is read. For example if only the number of steps to go are required then the command would be:

<S-addr><0x24><RS-Add><hb><lb><NACK><Stop>

The two bytes read hb and lb would indicate the number of steps to go and the NACK is issued to the I2C bus to indicate that there are no more bytes required from this command.

6.6. Command 0x30

Name: **Direction**

Format: <**S-addr**><**0x30**><**1 or 0**>
<Stop>

Changes the direction of stepping

Default at reset = 0/

6.7. Command 0x31

Name: **Speed**

Format: <**S-addr**><**0x31**><**0 to 255**>
<Stop>

Sets the stepping rate where 0 is the slowest and 255 is the fastest speed.

Default at reset = 100 (0x64).

6.8. Command 0x32

Name: **Mode**

Format: <**S-addr**><**0x32**><**0 or 1**>
<Stop>

Sets the step mode to either half step or full step. 0 is full step and 1 is half step

Default at reset = 0.

6.9. Command 0x33

Name: **Ramp**

Format: <**S-addr**><**0x33**><**0 to 7**>
<Stop>

This will set the number of steps that the stepping rate will take to obtain full speed, most motors, depending on what they are connected to will not start at full speed. Setting this to any value other than 0 will slow don the first few (as set) steps.

Default at reset = 3.

Rev	System Section Changes
Oct 2008	Preliminary
Dec 2008	Removed command 96
Aug 2008	Added command 0xa1 (not applicable to all devices)

7. I2C System Commands

The following section deals with system commands that are common to all I2C devices. Note that not all of theses commands are available for all devices.

Command	System Command Set
0x55	Test
0x90	Read EEPROM
0x91	Write EEPROM
0x93	End of EEPROM
0x94	Sleep
0x95	Reset
0x98	Change Device Address Temporary
0x99	Change Device Address Permanent
0xA0	Firmware Version
0xA1	Returns device ID

Most but not all devices contain an EEPROM that can store data when the power is off. The first 16 bytes of the memory is reserved for system use and should not be changed by using these commands.

If the contents of the first 16 bytes are changed then, depending on the device unpredictable results may occur. A factory reset will put the contents back to normal. In some devices not all 16 bytes are used.

The rest of the EEPROM can be used by the user for any purpose.

7.1. 0x55

Name: **Test**

Format: < **S-addr**><**0x55**><**start**><**R-Addr**><**Value..**><**NACK**><**Stop>**

BV4221 Example

0x42>s 55 r g-3 p

The above command will return 1,2,3 if the device is connected and working correctly.

This command simply returns an incrementing value until NACK is sent by the master prior to stop. This can be useful for testing the interface.

It can be used for testing the presence or other wise of a device at a particular address. It is also useful during the development stage to ensure that the I2C is working for that device.

7.2. Command 0x90

Name: **Read EEPROM**

Format: <**S-addr**><**0x90**><**EE-Address**><**R-Addr**><**data…**><**Stop>**

BV4221 Example

0x42>s 90 0 r g-3 p

The above will fetch 3 bytes from the EERPOM addresses 0, 1 and 2

This command will allow a single or several bytes to be read from a specified EEPROM address.

7.3. Command 0x91

Name: **Write EEPROM**

Format: <**S-addr**><**0x91**><**EE-Address**><**data…**><**Stop>**

BV4221 Example

0x42>s 91 10 1 2 3 p

The above write 1,2 and 3 to EEPROM addresses 0x10, 0x11 & 0x12

This command will write one or more, up to a maximum of 30 bytes at any one time, to be written to the EEPROM. Address 0 of the EEPROM is the device address and this cannot be written to by this command. A special command 0x99 is used for this purpose.

The first 16 bytes 0 to 15 are reserved for system use.

7.4. Command 0x93

Name: **End of EEPROM**

Format: <**S-addr**><**0x93**><**R-Addr**><**data**><**Stop>**

BV4221 Example

0x42>s 93 r g-1 p

Returns the address of the end of the EEPROM, normally 0xff

The system only uses a small portion of the first part of the EEPROM, the rest of the EEPROM can be used for user data or other purposes depending on the device. This command returns a single byte that will determine the last writeable address of EEPROM, normally 0xFF.

7.5. Command 0x94

Name: **Sleep**

Format: <**S-addr**><**0x94**><**Stop**>

BV4221 Example

0x42>s 94 p

This will put the IC into sleep mode. Any other command will wake the IC. Depending on the device this can be a considerable power saving.

7.6. Command 0x95

Name: **Reset**

Format: <**S-addr**><**0x95**><**Stop**>

BV4221 Example

0x42>s 95 p

Resets the device, this is equivalent to disconnecting and then connecting the power again.

7.7. Command 0x98

Name: **Change Device Address Temporary**

Format: <**S-addr**><**0x98**><**New-Addr**><**Stop**>

BV4221 Example

0x42>s 98 62 p

Changes the device address to 0x62, the device will revert back to 0x42 at reset.

This will change the device address with immediate effect and so the next command must use the new address. The address must be a write address (even number) Odd numbers will simply be ignored. The effect will last as long as the device is switched on. Resetting the device will restore the address to its original value. The address is stored in EEPROM location 0.

7.8. Command 0x99

Name: **Change Device Address Permanent**

Format: <**S-addr**><**0x99**><**New-Addr**><**0x55**><**0xaa**><**Current-Addr**><**Stop**>

BV4221 Example

0x42>s 99 62 55 aa 42 p

Permanently changes the device address to 0x62.

This command changes the address immediately (the next command will need to use the new address) and permanently (see hardware reset). The address must be a write address (even number) and follow the sequence exactly.

Permanent in this case means that the device will retain this address after power down, i.e. it is stored in EEPROM. Should anything go wrong the default address can be restored by using a hardware reset.

7.9. Command 0xA0

Name: **Firmware version**

Format: <**S-addr**><**a0**><**R-Addr**><**byte**><**byte**><**Stop**>

BV4221 Example

0x42>s a0 r g-2 p

This will return the two firmware bytes.

This simply returns two bytes that represents the firmware version.

7.10. Command 0xA1

Name: **Device ID**

Format: <**S-addr**><**a0**><**R-Addr**><**byte**><**byte**><**Stop**>

BV4221 Example

0x42>s a1 r g-2 p

This returns two bytes that represent the device ID. This is a later addition to the command sent and so may not be available on all devices.

8. Hardware Reset

A hardware reset has been provided should the device address be changed to some unknown value.

The method of restoring the factory defaults and thus the default device address is as follows:

1) Remove power
2) Hold the designated pin low or high or connect two pins together. The actual pins are device dependant and will be referenced in the sections above this text.
3) Apply power
4) Remove power
5) Remove shorting link

When power is now restored the device will have the default I2C address, normally 0x42.

9. Command Diagrams

To further explain the format of the commands this section has been provided. 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

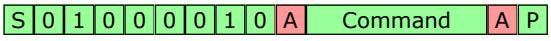
N Not acknowledge = 0

9.1. Sending a single command

This is designated in the list as:

<**S-addr**><**cmd**><**Stop**>

This sequence is used for simple functions where no data is involved. The I2C sequence, using the default address is:

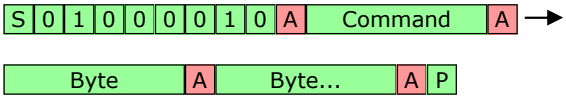


9.2. Sending a command with a parameter byte/s

This is designated in the list as:

<**S-addr**><**cmd**><**data…**><**Stop**>

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.



9.3. Receiving bytes from the salve

<**S-addr**><**cmd-Addr**><**byte**><**Stop**>

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). When the slave acknowledges the command another start condition is sent with the R/W bit set to 1. This is called a restart. After this restart the slave will continue to send bytes until the master sends a not acknowledge (N or NACK).

If the master does not send a NACK at the last byte the slave will be expecting another byte to be requested and this may cause either unpredictable results or the I2C bus to lock.

10. 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.

10.1. 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.

10.2. 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.

The typical symptoms are that when the 0x55 command is implemented no other commands will work after that.

10.3. 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.