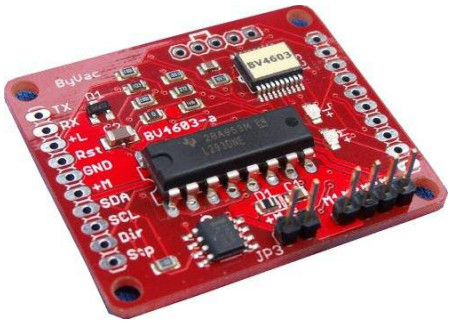
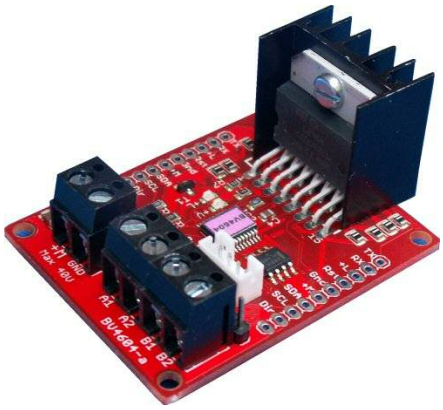
Serial Micro stepping Motor Driver BV4603/4



BV4603



BV4604

BV4603/4

Serial Micro stepping Motor Driver

Product specification August 2013

Contents

Serial Micro stepping Motor Driver BV4603/4

Rev	Change
August 2013	Preliminary
Oct 2013	Updates to command table
Dec 2013	Updated information about minimum number of steps
Apr 2014	Added command 'c'

1. Introduction

The BV4603 is a serial motor controller equipped with an L293 output driver.

The BV4604 is a serial motor conroller equipped with an L298 output driver.

It has a high resolution PWM output so that an exact amount of power can be delivered to the motor through the enable pins.

This also enables micro stepping of bipolar type stepper motors and full control over DC and unipolar stepper motors.

In addition to this it also has step and direction pins for direct control of stepper motors.

The main difference for the two versions if this device is the Motor driver IC and the connector options. Otherwise the BV4603 and BV4604 are identical in the way they work.

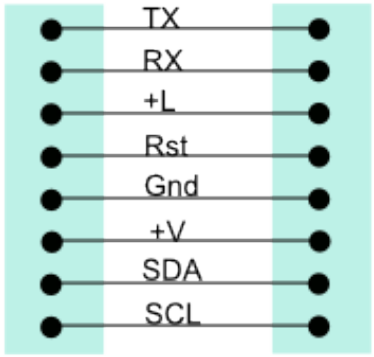
2. Features

- Can be daisy chained to control several motors.
- Wide Logic voltage range 2.5V to 5.5V
- 2 x PWM outputs
- 4 x Motor control outputs
- User step patterns
- DC motor control with soft start
- Stepper motor control
- Step rate up to 200kHz
- Step and direction mode
- L298 up to 4A (1.5A L293)
- Auto hold power
- On board 5V regulator up to 40V input
- Addressable many devices can share a single serial bus
- Baud rate selectable up to 115200
- User configurable

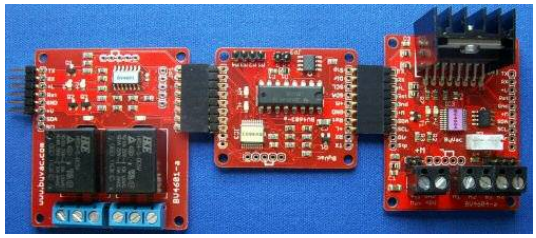
3. Sideways Stackable Connector

The connectors on either side of the board are through connectors and are designed to have a plug on the left hand side and a socket on the other. This enables devices to be stacked side by side without the need for any additional wiring.

This connector arrangement common to both devices and other devices in the SWS(Sideways Stackable) range.



If more than two devices are required then additional BV4603/4's can simply be stacked side by side. This is not limited to just this device.

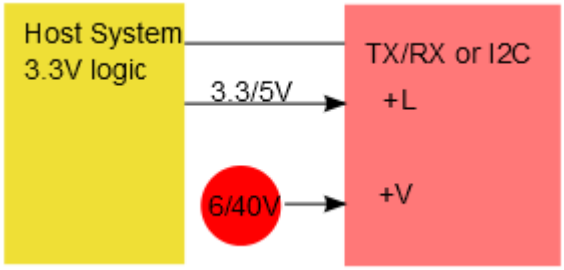


Several differing devices can be stacked in this way.

3.1. Power Supply

There are two power inputs to the connector, +L and +V. The +L will supply the logic and should be set to match the host system, i.e. 3.3V or 5V and +V should be set to match the Motor voltage that can be anywhere from 6V to 40V

The +V will also supply the motor driver chip with its logic voltage via a special 5V regulator that can accept a high voltageinput.



The logic supply can range from 2.5 to 5.5V

The motor supply voltage can range from 6V to 40V

4. Serial Interface

The serial interface is a standard 1 start bit 8 data bits and 1 stop bit and is initially set to 9600 Baud. This is user changeable from 2400 to 115200 in 8 steps.

The TX has an open collector output and so many devices can be connected to the same serial bus. Each device has its own address

ranging from 97 to 122 giving a maximum of 26 devices per serial bus.

TX: Serial output. Information from the motor controller to the host. This pin is capable of being connected to other TX pins in a multi controller set up.

RX: Serial input, this pin is used for either driving the motor with commands of for configuration.

+L: Logic voltage. This drives the logic circuits and should be set to the host voltage or supplied from the host. If the host is for example 5V then this should be 5V, if it is 3.3V then it should be 3.3V. This allows the board to be used in mixed voltage systems. The minimum is 2.5V and the maximum is 5.5V.

Reset: A momentary low on this pin will cause the controller to reset.

+V: Motor voltage. This is also connected to connectors (3) & (4) (BV4604). This supplies the power to the motor driver and can be up to 40V

5. I2C Interface

By default the serial interface is active. That is if no connections are made to the I2C pins. If an I2C host is connected to the I2C pins then this will pull up the SDA line as the I2C specification requires pull up resistors. If this is the case and power is applied to the device then the I2C interface will be selected.

6. Step and Direction

On this device there are two extra pins on the SWS connector for direct step control of the stepper motor. The step pin has TWO functions.

Mode 1: (default) Pin step mode it is an input and each time it is pulsed will step the motor.

Mode 0: the pin becomes an output and is normally high when the motor is stationary. When it is stepping the pin is held low. This is used as an indicator for the serial interface that the stepper motor has finished stepping.

7. Electrical interface BV4604

Some connectors are electrically duplicated but physically different to allow easier connectivity to various motor designs.

7.1.1.Connection Summary

1 Control plug (SWS) + step & direction

2 Control socket (SWS) + step & direction

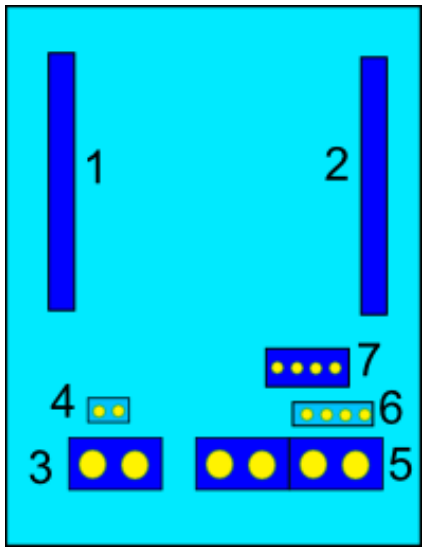
3 Motor power input +V (also on 1 and 2), screw terminal

4 Motor power input +V (also on 1 and 2), pin head plug.

5 Motor output, screw terminal

6 Motor output, pin head plug

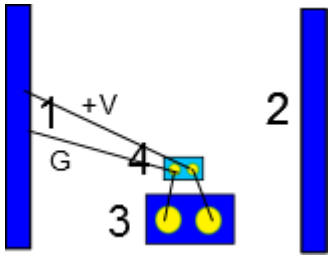
7 Motor output, 2mm JST type



Connector positions BV4604

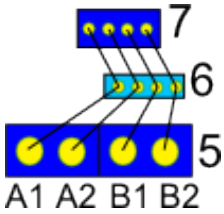
7.1.2.Motor Power (3) (4)

These supply power to the motor and three alternative connectors are provided, use one OR the other. See also the section on multiple controllers.

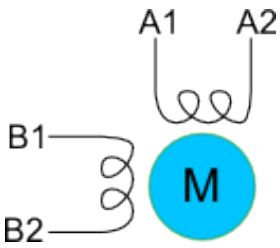


7.1.3.Motor (5) (6) (7)

Motor output provides three alternate ways to connect a motor or motors.



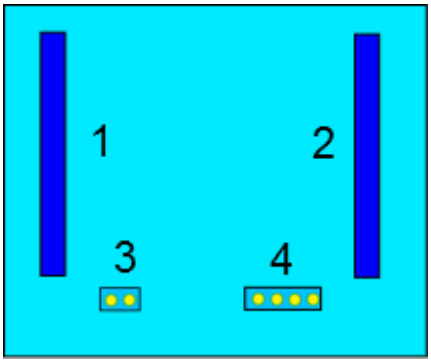
This shows how the connectors are connected.



A stepper motor is normally connected thus.

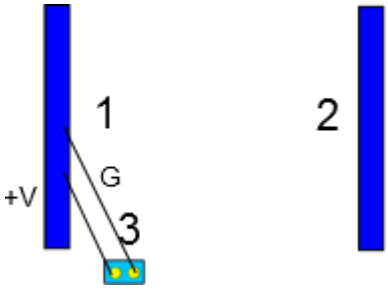
8. Electrical Interface BV4603

This is similar to the BV4604 but has a much simpler connection arrangement



Connector positions BV4603

(3) Power connector



(4) Motor connector

9. Operation

The device has two modes of operation. Mode 1 is designed for CNC systems where ultimate control over step and direction is required. In this mode it is possible to with the correct host software to operate the motors in unison so that correct lines and arcs can be created.

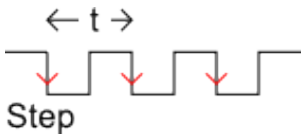
Mode 2 is for less stringent movements, a robot for example and the control of the device is done directly through the serial interface. Using this interface considerably reduces the overhead on the host as.

In both modes serial commands are available for interrogating the devices. If for example several are on the bus then the 'H' command will discover which devices are in use and working. Some serial commands are not available in mode 1.

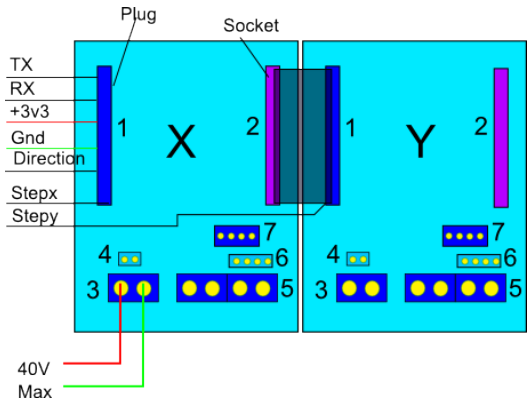
10. Mode 1 (step mode)

This is the default mode, out of the box so to speak. The serial interface is used for configuration and system checks and the step and direction pins control the stepper motor. It is possible to use the device without the serial interface at all in this mode, however the programmed defaults will prevail.

The step pin is active from high to low.



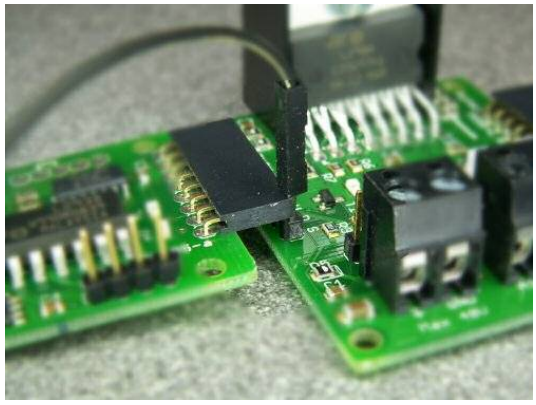
Any change in direction should be carried out before pulling the step line low. The host software will provide the step and direction information.



This is a typical setup for driving an x-y axis. The serial interface is useful but not required. The direction pin is common to all devices and the motor power is carried from one device to the other via the main 10 way connector.

The step pin is individual to the device and for this arrangement the host will need to provide one direction output and two step outputs.

When using the 10 way socket the step pin can be bent to make a suitable connection.



10.1. Mode 0 (Serial Mode)

NOTE: The serial mode also applies to the I2C mode as I2C is also serial just a different protocol.

Default Addresses:

Device	Serial	I2C(8)	I2C(7)
BV4603	'b'	0x62	0x31
BV4604	'd'	0x64	0x32

In this mode the motor is controlled by serial commands. The greatly simplifies the host

software requirement as the stepping and direction are controlled by the device. This also includes acceleration and deceleration.

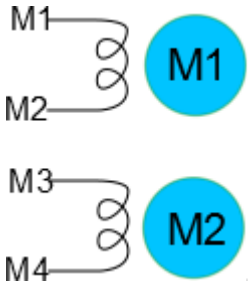
The disadvantage of this mode is that in a multiple device set up the motors cannot be accurately co-ordinated (driven in unison) and so if used as an X-Y axis it will probably not be possible to produce accurate arcs and lines.

The motors can though be operated simultaneously as once a command is issued for one device another command can be issued for the next. This will probably be suitable for a robot arm arrangement where it is important that the arm moves to the correct place but it is not critical how it gets there.

The full command set is at the back of this text.

11. DC Motors

The device is not just for stepper motors but can also drive up to 2 DC motors in both forward and reverse.



There is also a soft start arrangement that will gradually apply power to the motor until up to full speed.

12. Device Parameters

The EEPROM contains important values that control the way the device behaves. All of the values can be changed by the user using the serial or i2c interface.

The EEPROM consists of 255 bytes and in general the first 16 bytes are used by the system, the second 16 byte are used by the device and the rest of the bytes can normally be used by the user.

For this device there are step tables stored at 30 onwasrds.

Adr	Default Value	Description
0	0	System Use
1	device	Device address
2	6	ACK value
3	21	NACK value
4	3	Baud rate
5	1	Error reporting 1 = on
6	13	End of line
7	1	Invert TX 1=invert
14	device	Device address copy
16	20	Hold power
17	10	Stop Power
18	1	Microstep 1=yes
19	30	Step Pattern pointer
20	1	Mode 0=serial
21	150	Ramp
22	255	Global power
23	20	DC soft start
24	50	Hold delay
250	device	Device address copy

Table 1 System EEPROM use

Bipolar		
Location	**Name**	**Content**
30	# Steps	4
31	Step 1	5
32	Step 2	6
33	Step 3	10
34	Step 4	9
Unipolar full step		
Location	**Name**	**Content**
40	# Steps	4
41	Step 1	1
42	Step 2	2
43	Step 3	4
44	Step 4	8
Unipolar half step		
Location	**Name**	**Content**
50	# Steps	8
51	Step 1	1
52	Step 2	3
53	Step 3	2
54	Step 4	6
55	Step 5	4
56	Step 6	12
57	Step 7	8
58	Step 8	9

Table 2 Step tables

The user is free to use any locations that are not occupied by the system but for future use it is best to avoid locations below 32.

EEPROM values are only read on start up so when changing values they will not normally take effect until the device is reset.

12.1. Address

These EEPROM locations contains the device address. By convention the address is set to values between the values 97 to 122, no checking is made by the device so setting values outside this range may or may not work.

For security the address is stored in three places and to change the address of the device at least two of the locations need to be set otherwise the device will detect the anomaly at start up and revert to the majority value.

Normally to change the address of a device locations 1 and 14 are both changed. The device will detect this at start up and change the address in location 250 to match.

12.2. 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.

12.3. 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.

12.4. Baud Rate

The Baud rate has the following values:

0. no valid
1. Baud rate is fixed at 2400
2. Baud rate is fixed at 4800
3. Baud rate is fixed at 9600 (default*)
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 57600
8. Baud rate is fixed at 115200

12.5. 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. This does not apply to I2C if available.

12.6. 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.

12.7. Invert

This should not be changed for this device. It will invert the TX output and is used for devices that do not have a PCB.

12.8. Hold Power, Stop Power & Delay

A unique feature of this device is that after stepping the motor power can be reduced to a holding level. The level of the power and the time it takes to apply this power can be controlled with this command.

There are three values that work together and come into play after stepping has finished as follows.

Immediately stepping has finished the stop power is applied. This should be less than full power as when the motor is not turning it will consume much more current. It will be held at this power for a delay set by the hold delay and then drop down to the holding power.

The power (holding and stop) is in the range 0-255 and it is applied to ENA and ENB. The delay is also in the range 0-255. 0 will give no delay and 255 will give approximately 2 seconds.

12.9. Microstepping

Microstepping controls the power to the motor in discrete steps. Each normal step is divided into 8 steps and then an internal control applies the correct amount of power to each winding.

By default microstepping is switched on '1'

12.10. Step Pattern Pointer

This points to an area of EEPROM that contains the step pattern. By default this is a standard full step pattern and works well with microstepping.

The pattern has in its first location the number of steps and then the actual pattern bits fed to M1 to M4 for each step. Only the 4 lower bits of the byte are used.

The step tables can be changed by the user if required. The first byte must be the number of steps. The byte value is sent directly to the driver output with the LSB as follows:

M4	M3	M2	M1
1	1	0	0

In the above a value of 12 (0xc) has been output as a step.

New tables can be created as the EEPROM is free from location 59 to 240. It is best not to use locations 240 onwards as this may be used by the system to duplicate important EEPROM values.

12.11. Mode

The device will operate in two modes, mode 1 is the default and this expects a step pulse on the step pin (described earlier in the text). Mode 0 is the serial mode. To use the serial mode in full this needs setting to 0 and the device requires a reset.

12.12. Ramp

This is sometimes called acceleration and deceleration and is usually necessary to overcome mechanical inertia. The scheme used is simple in that the value set in ramp is the number of steps away from the start or stop position.

As an example if the ramp is 10 and the number of steps requested is 100 then steps 0 to 9 will be accelerating and steps 90 to 100 will be decelerating.

The start and stop steps per second is half the number of the running value, ramping up smoothly to the running value or ramping down to half again.

12.13. Global Power

This value is applied to all motor outputs where 255 is full power and 0 is no power. The value stored in the EEPROM is the **initial** global power that will be applied on reset. The power can be changed dynamically via a serial command. The default is full power because the mode default is 1. This however would be better in a practical situation to set this to 0 and then apply the power by software control.

12.14. DC Soft Start

This only applies to the DC commands ‘a’ and ‘b’. The value is in steps of 0.1mS. This also varies with the actual amount of power that is applied.

It works by delaying each step in the requested power by the amount given. For example if a power of 500 were applied and the soft start value was 20 (2mS) then it would take 500 x 2mS to reach full power.

13. Commands

All serial commands are proceeded by an address and terminate with CR (0xd). In the examples given below the address is 'b' (4603) 'd' (4604) or 0x62 (0x31) 0x64 (0x32)

When a command is accepted by the device it always returns ACK which by default is the value 6. If the device rejects the command then it will return NACK, value 21

Serial	I2C	range	Default Value	EEPROM Location	Description
ap,d **(*)**	1p<2>,d	p 0-1023 d 0,1,2			**DC Motor power and direction for M1-M2** See notes at foot of table
bp,d **(*)**	2p<2>,d	p 0-1023 d 0,1,2			**DC Motor power and direction for M2-M3** See notes at foot of table
cp1,d1,p2,d2	7,p1H,p1L,d1, p2H,p2L,d2	Power 0-1023 Direction 0,1,2			**DC Motor power and direction for both M1-M2 and M2-M3** See notes at foot of table
f (*)	3(4bytes)	0-2147483647			**Steps to Go** When the motor is in the process of stepping, this will return the number of steps to go before it is finished. A more useful indication when using mode 0 (serial mode) is to use the step pin which in serial mode functions as an output indication when stepping has finished. **I2C** Because it is a large number the I2C returns 4 bytes, highest first. **bus.i2c(0x31,[3],4)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature **NOTE: When the motor is stepping there is not much time for communications, the step pin should be used in preference, See the section on mode.
gn	4n	0-255	255	16	**Global power** The initial value is read from EEPROM but this command can change it dynamically. This command does not store the value to EEPROM. **I2C** Set global power to 55 **bus.i2c(0x31,[4,55],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature

					See notes at foot of table
k	5				**Stop** Sets M1-M4 to 0 and switches off PWM to both enable channels. Issuing this command will stop all motor operations. Example: **bk** **I2C** Example **bus.i2c(0x31,[5],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature
sn,p,d **(*)**	9<4>, <2>,<1>	s: 0-2147483647 p: 70-65000 d: 0,1			**Step** Steps a stepper motor given the number of steps, the speed in steps per second and the direction 0 or 1 Minimum step rate 70 See notes at bottom of table. This is the main serial control for use with a stepper motor. The specification is described by: <**#of steps**><**step speed**><**direction**> The number of steps is a 32bit positive value so the range can be 1 to 2147483647 The step speed is a 16 bit value and specifies the number of steps per second. In practice most stepper motors will struggle to achieve more than 12,000 steps per second in microstepping mode, much less sun 1000 in non-microstepping mode. The direction is a value of either 0 or 1 Example: **bs10000,3200,1** The above will step 10,000 at a speed of 3200 steps per second. **I2C** A total of 8 bytes (including the command) is required for I2C, using the values in the above example: 10000 = 0,0,0x27,0x10 3200 = 0xc, 0x80 **bus.i2c(0x31,[9,0,0,0x27,0x10, 0xc,0x80,1],0)**

					See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature
t	17				**Test** This simply outputs the step pattern with a delay of 1mS. This will cause any connected motor to revolve slowly. It is used for determining the correct polarity of the wires by trial and error. ** **IMPORTANT** the only way out of this command is to reset. **I2C** test **bus.i2c(0x31,[17],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature
Wn,m	0x91	n=0-255 m=0-255			**Write to EEPROM** This will write a single byte to an EEPROM location, the command format is: <adr>W<EEPROM address>,<value> Care should be taken when using this command for two reasons: 1) The EEPROM can only be written to a certain number of times all be it a large number. 2) The EEPROM contains system information that is used at start up a wrong value could mean loss of communication with the device that would require a factory reset. **I2C Example write 23 to location 7** **s 0x91 7 23 p** or **bus.i2c(0x34,[0x91,7,23],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature.
Rn,m	0x90	n=0-255 m=0-255			**Read from EEPROM** The EEPROM values can be read with this command given a starting address and the number of bytes to read. <adr>R<"EEPROM adr><#bytes> This example will output the first 16 bytes of EEPROM

					fR0,16 The output from the device will commence after receiving CR 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> **I2C** I2C will read only single values at a time, to read from location 3: **s 0x90 3 r g-1 p** or **bus.i2c(0x34,[0x90,3]1)** See www.pichips.co.uk and ‘notsmb’ for an explanation of the above nomenclature.
D	0xa1				**Device ID** Returns a number representing the device product number as a string **fD** Output from the above would be: “4601”<ACK> **I2C** returns two bytes representing a 16 bit number, high byte first **s 0xa1 r g-2 p** or **bus.i2c(0x34,[0xa1],2)** See www.pichips.co.uk and ‘notsmb’ for an explanation of the above nomenclature.
I	n/a		1	7	**Toggle Inverted** This command will invert the output of the TX pin and store the value to EEPROM. A value of 1 is inverted and 0 is not inverted. The command will toggle form one to the other. If the TX pin requires changing, instead of ACK being returned by the device a value of 0x3e (‘>’) is returned. This is easily detected and this command can be issued to correct it. Example **fI** This is only needed as a **one time** operation as the change is automatically written to EEPROM

C	0x95				**Reset** Resets an individual device. This is a soft reset. 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. Example **fC** **I2C** **s 0x95 p** or **bus.i2c(0x34,[0x95],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature.
V	0xa0				**Version** Returns the firmware version as a string in the format "H.L" Example **fV** **I2C** Sends two bytes, major revision first so 2.7 world be 2 and 7
H	n/a				**Hello** This command is used to check what devices are on the bus. It simply returns ACK but where there is more then one device on the bus the following sudo code will list them: for j = 97 to 122 Send(chr$(j)+"H\r") if ack received then print device j found If a device is found then the other attributes such as device ID can be obtained. Also user information could be stored in the devices EEPROM and retrieved.

(*) These commands only work in mode 0, I2C will ignore input, serial will give an error output if used in mode 1.

13.1. DC Motor Power and direction

This controls the output of either M1-M2 (command 'a') or M3-M3 (command 'b'), M1-M2 + M2-M3 (commend c). The power and direction are set with this command. Power is the power applied to the PWM 0 is zero power and 1023 is full power. If more then 1023 is specified than this will default to 1023.

The power will also be effected by the global power as a multiplier from 0 to 1. As an example if a power of 500 is set and the global power is set to 128 (1/2 of 255) then the power will be 500 * 0.5 which will be 250.

The direction applies voltage to the M pins as follows:

Direction	M1(3)	M2(4)
0	0	0
1	1	0
2	0	1

If follows then that a direction of 0 is off or no power to the motor, 1 and 2 will control forward or reverse.

The power is applied via the **soft start delay**, this can beset to 0 if not required.

Example:

To set motor 'b' in a 'forward' direction using 700 power

bb700,1

I2C

The I2C command is 2 and 2 bytes (high) (low) make up the 16 bit number; 700 is 0x2bc

s 2 2 0xbc 1 p

or

bus.i2c(0x31,[2,2,0xbc,1],0)

See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature.

DC Motor command 'c' (i2c command 7) will effect both motor outputs simultaneously but does not have the soft start feature. The command requires both power and direction of each motor to be specified.

Example:

set both motors to be power 700 and direction 1:

bc700,1,700,1

I2C

The i2c command requires 6 bytes as the power can be greater than 256 and so the power is specified as 2 bytes high and low.

bus.i2c(0x31,[7,2,0xbc,1, 2,0xbc,1],0)

13.2. Global Power

This set an overall power that will effect all motor operations and modes. The power is a value form 0 to 255, where 0 is no power and 255 is full power. It works by multiplying the requested power by the global power value as a percentage for 0 to 100.

To give an example the output power will be the following:

0	25%	50%	75%	100%
0	63	128	191	255

The formula for obtaining the values is 255 * percentage / 100

13.3. Minimum Step Rate

The minimum accurate step rate is 70 steps per second, setting the rate lower than this will not be accurate. If slower stepper speeds are required then here are some options:

- Use microsteppeing as this will step 8 logical steps for one physical step. This effectively gives about 9 steps per second when the step rate is set to 70.
- Use a different step table, the Unipolar half step will effectively give 35 steps when the step rate is set to 70
- Create a table in EEPROM that repeats steps for example in the table below there are 4 logical steps to 1 physical step. There is plenty of room to increase the size of this table.

Location	Name	Content
	# Steps	

61	Step 1	5
62	Step 1	5
63	Step 1	5
64	Step 1	5
65	Step 2	6
66	Step 2	6
67	Step 2	6
68	Step 2	6
69	Step 3	10
70	Step 3	10
71	Step 3	10
72	Step 3	10
73	Step 4	9
74	Step 4	9
75	Step 4	9
76	Step 4	9