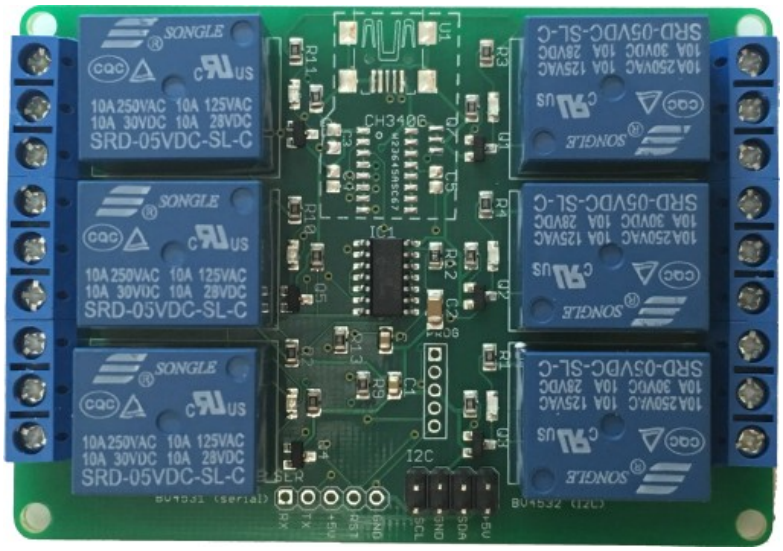
BV4533



I2C or Serial 6 Way Relay

Date	Firmware	Revision
February 2018	1.0.4	Preliminary
11 Feb. 2018	1.1.0	Serial Updated
3 Sep. 2018	1.1.0	I2C corrections, trigger is not used

Introduction

This is an I2C or Serial relay for use with microcontrollers, for example the Arduino, Raspberry Pi etc..

There is also a USB option.

Description

The device is access either by I2C OR Serial depending on the device supplied.

BV4531 Serial

BV4531U Serial with USB

BV4532 I2C

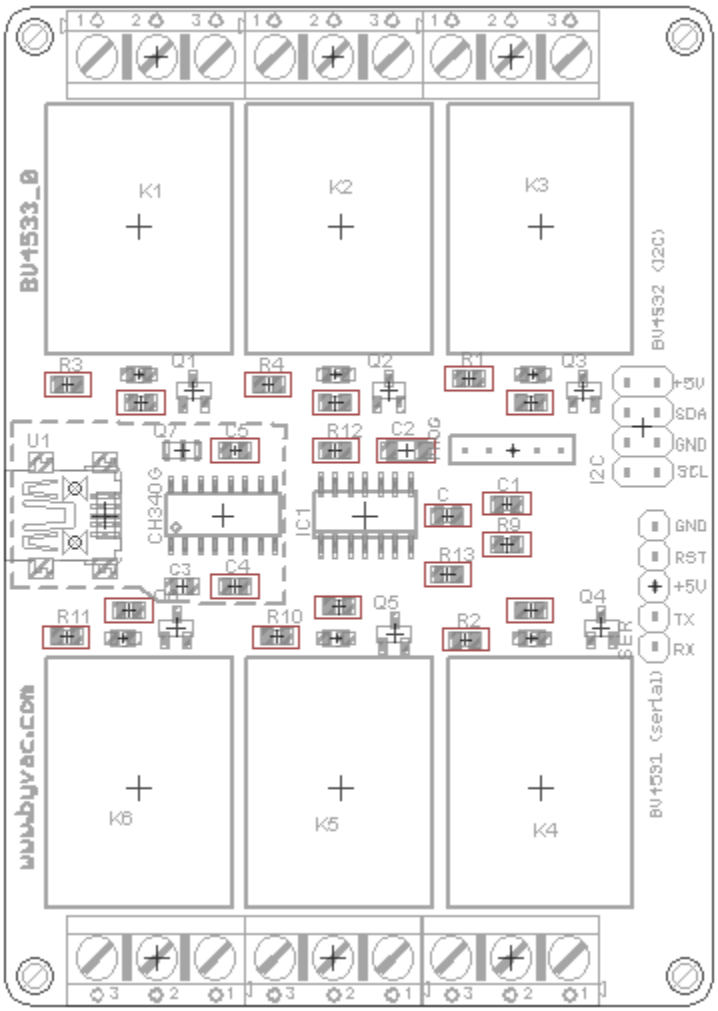
Relays can be switched on, off, timed with commands.

Features

- User Selectable I2C address (BV4532)

- Multiple serial devices
- 5V supply
- Relay action (on or off) can be timed 1 second to 18 Hours
- Requires only 2 control wires (TX/RX) ro (SDA/SCL)
- Fitted with 10A relays
- Size 60mm x 85mm (height approximately 20mm)

Physical Description



The PCB has 3 connection options, all the connector pads are in place but not all of them may be active for the particular device.

I2C	Applicable only to the BV4532 (see I2C section)
Serial	Applicable only to the BV4531 (see serial section)
USB	Applicable only to the BV4531U (see serial section)

EEPROM Locations

The values in the EERPOM effect the operation of the device and

are read on start up, some values are stored in RAM and so can be temporarily changed and some commands change the EEPROM values.

Adr	Default	Description
0	0x55	System Use
1	70	I2C address (only relevant for I2C)
2	17	Not used
3	4	Baud rate code (only relevant for serial)
4	1	Prompt (only relevant for serial)
5	0	Addressing (applies to serial only)
14	70	I2C address copy (only relevant for I2C)
240	70	I2C address copy (only relevant for I2C)

EEPROM Locations and default values

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.

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

I2C Address

This is only applicable to I2C devices, it is stored in 3 places for security, should one location become corrupt then it will be reset to the other two values. There is a command to change the I2C address but it can also be done by directly writing to the EEPROM, if so than at least two locations need changing.

Baud Rate

Applies only to serial devices. This is a code between 1 and 8 as follows:

1 1200 Baud
2 2400 Baud
3 4800 Baud
4 9600 Baud (Default)
5 19200 Baud
6 38400 Baud
7 57500 Baud
8 115200 Baud

Prompt

To help with serial communication the device when waiting for a command will output ‘>’ if this value is set to 1.

Addressing

It is possible to address more then one serial device using addressing, see serial section.

BV4532 (I2C)

This has the I2C interface.

I2C Interface

Pin	**Description**
SCL (*2)	Clock
GND (*2)	Ground
SDA (*2)	Data
V+ (*2)	Supply voltage for device *

I2C Electrical Connection

The I2C connector has duplicate pins, this is to allow for daisy chaining of I2C devices. Please remember that unless buffered the length of the cable should be limited.

*__Voltage:__ The device works with 5V mainly because of the relay specification, there are NO pull up resistors on the device as these are normally provided by the master device. The device can be supplied with 5V and still work with 3.3V without and detrimental effect as the pull up resistors (on the 3v3 host) take care of the 'high' value of the voltage.

The device has a standard I2C interface and will act as a slave.

0x46 (0x23 7 bit) Keypad & LCD address

Commands are realised by using the first byte written as the command byte.

There is one exception to this which is a general call (address 0) followed by 0x55. This will reset the EEPROM contents back to the factory defaults. The command is useful if the i2C address becomes corrupt due to inadvertently writing the wrong values to the location where the i2C address is held.

Address (0x46 or 0x23 7 bit)

Device I2C address is stored in EEPROM in three places (see eeprom locations). This address should be set to between 10 and 250 using EVEN numbers ONLY. It is the 8 bit address value so an address of 68 will be on some systems the read and write address 34.

There is a command however that will change all 3 addresses at once.

Commands

The first byte is the command byte that initiates all transactions, as an example to turn on relay 1 requires the command byte followed by the relay number followed by the action (on or off)

Example 8 bit

```
i2c_start()
i2c_write(0x46) // write address
i2c_putc(2) // command
i2c_putc(1) // relay
i2c_putc(1) // action 1=on
i2c_stop()
```

Where a read is needed say to get current timer values a restart can be used or start stop

Example 8 bit

```
i2c_start()
i2c_write(0x46) // write address
i2c_putc(3) // command to get timer value
i2c_putc(1) // relay to get value from
i2c_stop()
```

```
i2c_start()
i2c_putc(0x47) // address with read bit set
value = i2c_get(1) // **
i2c_stop()
```

** When reading I2C that last read should send NACK to indicate it is the last byte to receive.

Adruino and other controllers use 7 bit addressing where most of this is taken care of with the read and write commands.

Command	Range	Notes
1 c,r,a,th,tl	th+tl: 1 to 65535 a: 0 or 1	**Set Relay 0 to 5** c: is the command 1 r: is the relay number from 0 to 5 a: is the action for the relay to perform, 0 is turn off an 1 is turn on. th and tl: from a 16 bit number from 1 to 65535 which is a counter to set the action. This time is approximately seconds. Example Turn on relay 2 in 15 seconds i2c.write(2,1,0,15) Turn off relay 3 in 5 minutes (300 seconds) i2c.write(3,0,1,44) (300 = 0x12c)
2 c,r,a		**Turn on or off relay now 0 to 5** c: is the command 2 r: is the relay number from 0 to 5 a: is the action 0=off, 1=on

		Example Turn on relay 4 i2c.write(2,4,1)

3
c,r
	0 to 65535	**Gets Relay current timer value for relay 0 to 5**
		c: is the command 3
		r: is the relay number from 0 to 5
		Reads the relay timer value and returns TWO bytes high and low.
		Example
		Read timer value of relay 5
		i2c.write(3,5)
		i2c.read() // high byte
		i2c.read() // low byte

4	0-0x3f	**Get state of all relays as a byte value** Each relay is represented by 1 bit either 0 or 1 Relay Bit A 0000 000x B 0000 00x0 C 0000 0x00 etc. 00AB CDEF Example i2c.write(4) i2c.read()
5		**All relays off**
6		**All relays on**
System		
		EEPROM reset to defaults This will restore the default EEPROM settings which has the I2C address. No effect will take place until the device is reset. This can take a few ms and so some master I2C devices will need to take this into account if a timeout is to be avoided. This is a special command in that it can be called regardless of the state of the EEPROM, in other words it does not rely on the I2C address value, so if the EEPROM values are random then this will restore as factory defaults. Example i2c.write(0,0x55)
60	Adr 6-255	**Change I2C address** The address must be an even value, this is the 8 bit or full address where an even address is write and an odd address is read. It will not take effect until reset. The same effect can be achieved by writing to the EEPROM Example i2c.write(60,adr)
61	Adr 0-255	**Read EEPROM** This will read a single value from an address in

		EEPROM Example i2c.write(61,adr) adr is the new I2C address.
62	Adr 0-255 data 0-255	**Write EEPROM** Writes a single byte to the given address Example i2c.write(62,adr,data) adr is the address of the EEPROM 0 to 255, data is what to store there.
63		**Gets device ID as 2 bytes** The first byte is the high byte of a 16 bit number and the second byte is the low byte Example i2c.write(63) i2c.read(2) Returns 2 bytes, to resolve the is (first byte * 256 + second byte)
64		**Gets firmware version as 3 bytes** x.x.x Example i2c.write(64) i2c.read(3) Returns 3 individual bytes e.g. 1,1,0 for v1.1.0
66	N/a	**Reset** Resets the device as at first switch on. Depending on the I2C master this will likely cause a time out as there will be no reply from the device and so this may cause an I2C error from the master Example i2c.write(66)

BV4531 (Serial)

This section refers to the serial and USB interfaces

Pin	Description
TX	Transmit, output
RX	Receive, input
V+	5V supply for controller and relays
Rst	Reset, leave disconnected or high, pulling low will reset the controller
Gn	Ground

Serial connector, single 5 way

USB

This is an option of the serial interface and by default the USB will supply power to the relays. However the USB cannot supply sufficient guaranteed power to all of the relays if they are all on at the same time.

There is a jumper pad marked USB that is closed which supplies power to the board. If power is supplied externally then this jumper should be open.

Serial Signals

The serial expects (TTL logic either 3.3V or 5V) NOT + and -12V RS232. If you have the old 9 pin connector then a conversion device will be required to transform the 12V levels to 5V or 3.3V levels.

The serial interface uses 1 start bit, 8 data bits,1 or 2 stop bits, no parity and no handshake, the default Baud rate is 9600 but this can be changed via an EEPROM setting. Or temporarily by the B command.

Handshake

There is no hardware handshaking. This is provided in the way the software works.

When the device is free to accept a command it will output a prompt character (default '>'). The host will enter a command sequence followed by CR (13, \r). The device will not interpret the command until this is received. When received the command will be carried out and when finished send a prompt.

The host must wait until the prompt is received as the device will ignore any incoming characters whilst it is doing the command.

Addressing (TX only)

This is switched off by default and so a command will have immediate effect, for example to set relay 3 to on the command is r3,1.

If more than one device is connected to the serial bus then all devices will respond, to enable more than one device to take action it is possible to address the device individually and so it is possible to turn relay 3 on on one board and without turning it on another board.

The address will be the same as that used for I2C, in this case 0x46 which is the ASCII code for 'F'. When using addressing therefore the command would now be:

Fr3,1

Turn on addressing: W5,1
Turn off addressing FW5,0

Reception. *Serial devices are not bus devices and strictly*

speaking only one device is allowed per bus. The TX line is normally held low thus preventing any other device from responding. This is not the case with receiving though and so many devices can is suitably addressed receive instructions.

Commands

General Rules

- CaSe is important.
- Values returned are in decimal
- Command act as soon as '\r' is received.
- Numbers are separated by a ',' unless it is a number after the command in which case no comma is needed

Command	Range	Notes
t<r>,<a>,<y>	t: 1 to 65535 r: 0 to 5 a: 0 or 1	**Timed action** a: is the action for the relay to perform, 0 is turn off an 1 is turn on. t: is the time the action should take in seconds, this is approximate. **Example**: turn relay 3 on in8 seconds t3,1,8\r
r<r>,<a>		**Turn on or off – not timed** a: is 0 for off 1 for on **Example**: turn off relay 4 r4,0\r
g<r>	R: 0 to 5	**Gets Relay current timer value A to F** Reads the relay timer value. When using timed values this will return the remaining time left to the action.
s	0-0x3f	**Get state of all relays as a byte value** Each relay is represented by 1 bit either 0 or 1 Relay Bit A 0000 000x B 0000 00x0 C 0000 0x00 etc. 00AB CDEF
o		**All relays off**
f		**All relays on**
System		
E	N/a	**EEPROM reset** This will restore the default EEPROM settings No effect will take place until the device is reset.
B	1 to 8	**Set Baud Rate** This is a temporary setting, for a permanent change the EEPROM values must be set. The value is a choice of Baud rates from 1 to 8, see the table in the EEPROM settings section.
R<a>,<n>	a: 0 to	**Read EEPROM**

	255 n: 0 to 255	Read the eeprom starting at address <a> for the number of bytes <n> **Example:** Read 5 bytes from eeprom starting at 8 R8,5\r
W<a>,<b>	a: 0 to 255 b: 0 to 255	**Write EEPROM** Writes byte <b> to address <a> Only one byte can be written at once Example: Write 27 to location 75 W75,27\r
I		**Gets device ID** As a decimal number
F		**Gets firmware version** As a string
X		**Reset** Resets the device as at first switch on