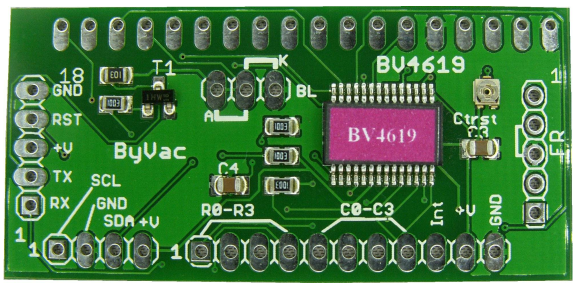
# Dual Interface LCD Display Controller BV4619



**BV4619**

**Dual Interface LCD Display & Keypad Controller**

Product specification Nov 2013 V0.a

## Contents

# Dual Interface LCD Display Controller BV4619

| Rev | Change |
|---|---|
| Dec 2013 | Preliminary |

## 1. Introduction

The BV4619 is a dual interface LCD display and keypad controller intended for the text display modules using the HD44780 or similar controller. These are most popular in 16x2, 20x4 and 20x2 formats. This controller will handle 1 to 4 lines and up to 40 characters or any combination of those.

NOTE: There are two similar controllers, BV4618 and BV4619. See below for the differences

## 2. Features

- Serial Input **and** I2C input
- Baud rate 9600 user configurable
- I2C up to 400KHz
- Addressable serial protocol
- Direct LCD control
- 16 byte keypad buffer
- keypad up to 16 switches (4x4)
- Supply voltage 5V DC
- Current: 6mA @ 4.7V
- Size: 55mm x 24mm

## 3. BV4618, BV4619 Comparison

The BV4619 is a later version of the BV4618 but does not replace it, the differences are as follows:

#### 3.1.1. BV4618

Automatic Baud rate detect, can be used with RS232 (12V) voltages. Only one BV4618 can be used on any one serial bus (not addressable). The LCD interface is fully controlled with scrolling at the end of a line and end of text. Only one back light output is provided either on or off.

The I2C interface is a mirror of the serial interface with no special commands.

#### 3.1.2. BV4619

The Baud rate is initially fixed at 9600 but can be changed by the user. Only TTL (3.3 or 5V) levels can be used on the serial interface. The device is addressable so up to 26 devices can be used on the same serial bus.

Simple direct commands to the LCD are provided, the host takes care of scrolling and selecting the LCD line.

Three back light outputs are provided all with PWM to give varying brightness (10 levels).

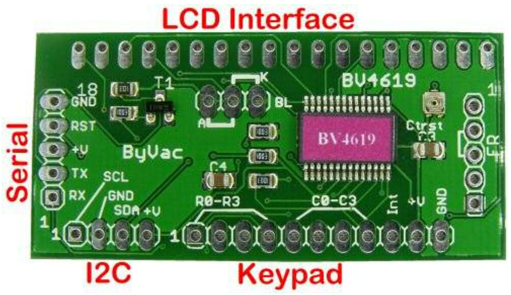
Much simpler I2C commands giving direct access to the LCD.

Only top connected LCD's can be used there is no side connector.

## 4. Physical Specification

The BV4619 is a small display and keypad controller designed to fit on the back of LCD display modules. There are four connection points:

1. LCD
2. Keypad
3. Serial
4. I2c



### 4.1. LCD

Most standard LCD display modules will fit onto the controller and there is a considerable amount of flexibility in the firmware for differing lines and character lengths. LCD's without backlights will usually only have 14 pads and so the last four can be left unconnected. Some LCD display have alternative arrangements for their back lights, see the text below (lcd Connector) for details.

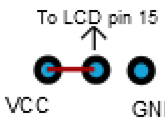
| Pin | Function |
|---|---|
| 1 | Gnd |
| 2 | V+ [1] |
| 3 | CL Mid wiper on the contrast trimmer |
| 4 | RS LCD control signal |
| 5 | RW control signal |
| 6 | RE control signal |
| 7 to 10 | not connected to anything |
| 11 | D4 data line on LCD |
| 12 | D5 data line on LCD |
| 13 | D6 data line on LCD |
| 14 | D7 data line on LCD |
| 15 | LCD Back Light Common [2] |
| 16 | LCD Back Light red |
| 17 | LCD Back Light green |

| | |
|---|---|
| 18 | LCD Back Light blue |

**LCD Connector**

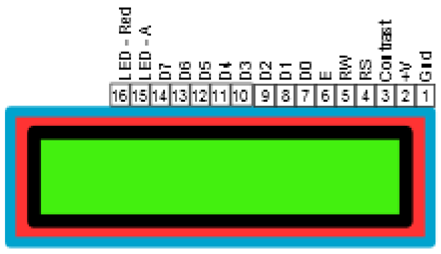
[1] Take care that the V+ and GND connections match the display you have, most do but there are a few that have them the other way round.

[2] By default the common connector is connected to VCC. This is the normal for most displays. The jumper pad marked BL has a shorting link on the left side thus:

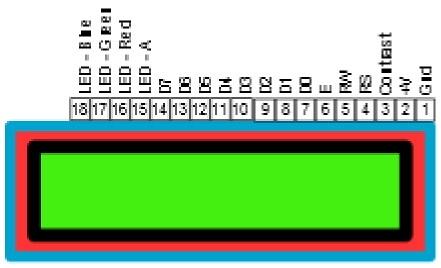


The arrangement can cater for both common Anode (usual) and common cathode types. Most displays only have one backlight and this is connected via a 16 way connector, the other two pins are ignored. Back light control is via the 'red' pin

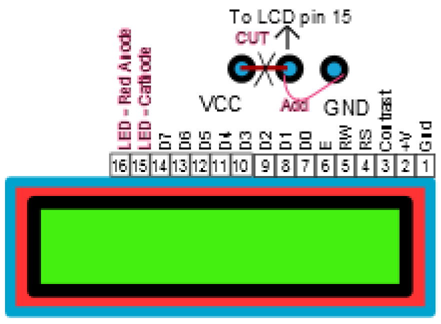
For the newer RGB back light displays all of the 18 pins are used.



Standard LCD display



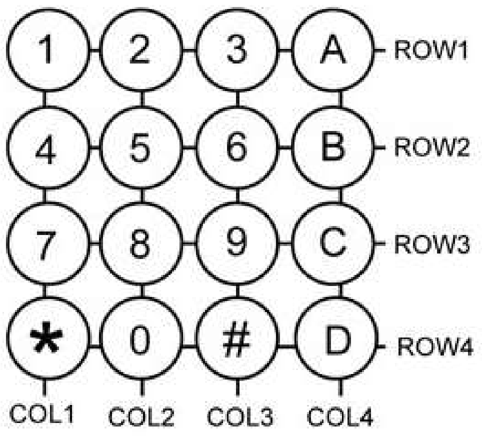
RGB Back light type display



Arrangements for when pin 15 is an anode rather than the more usual cathode. Cut the track to the jumper and add a link as shown.

## 4.2. Keypad

The keypad interface uses pins 1 to 8 and is designed for a 16 way cross point switch, i.e. a standard 16 key keypad with 8 connections. Four connections will be for the columns, and 4 for the rows. Smaller keypads can be used of course by leaving either the rows or columns disconnected.



A typical keypad would have the rows and columns connected as indicated on the PCB. The rows are in fact inputs that will be high if not connected to anything. The columns are outputs and are continually being scanned.

The keypad when wired correctly returns a scan code for that key. The scan codes are shown on the table below.

| | | | |
|---|---|---|---|
| EE [1] | DE [2] | BE [3] | 7E [A] |
| ED [4] | DD [5] | BD [6] | 7D [B] |
| EB [7] | DB [8] | BB [9] | 7B [C] |
| E7 [*] | D7 [0] | B7 [#] | 77 [D] |

**Keypad scan codes in hex**

Differing scan codes may be obtained by wiring the keypad differently.

The keypad is being scanned all of the time and there is a 16 byte buffer so poling is not constantly required. The host software can be 'relaxed' about collecting the input.

To assist this there is an interrupt pin which goes low if there are any keys in the buffer. This could be monitored by the host.

When the buffer is empty the pin will be high. There is no buffer full warning, when this happed previous keys will be overwritten.

| Pin | Function |
|---|---|
| 1 | Row 1 |
| 2 | Row 2 |
| 3 | Row 3 |
| 4 | Row 4 |

| | |
|---|---|
| 5 | Col 1 |
| 6 | Col 2 |
| 7 | Col 3 |
| 8 | Col 4 |
| 9 | nInt |
| 10 | +V |
| 12 | Gnd |

**Keypad Connections**

Pin 1 is to the left of the board and next to the I2C connector. It is important to get the rows and columns correct but it doesn't really matter if they are not in the correct order, they will simply give different scan codes.

The interrupt pin is high when no keys are in the buffer, it goes low as soon as there are keys in the buffer and remains low until it empties again. The key buffer is emptied either by reading all of the keys out or clearing the buffer.

If a smaller keypad is used then don't mix the columns and rows. If for example a 2x2 keypad is being used then just 2 of the rows are needed and 2 of the columns, don't put all four wires on say the columns, this will not work.

### 4.3. Serial Interface

(**default address 'h' 114**)

(**Baud rate 9600**)

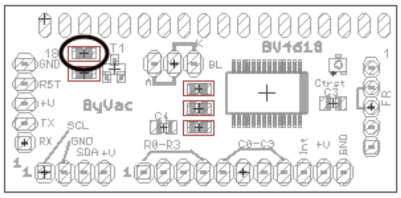
The serial interface will accept signals from both a microcontroller UART and a typical USB to serial converter. It will NOT directly accept signals from the standard 9 pin RS232 interface as this outputs + and - 12V

| Pin | Function |
|---|---|
| 5 | GND |
| 4 | Reset |
| 3 | +V |
| 2 | TX |
| 1 | RX |

The reset pin is normally high and can be left unconnected. When brought low it will reset the on board microcontroller.

The TX has a 'collector' output with a high value resistor and so several devices can be connected to the same bus.

If many devices are needed say more than 5 then the high value resistor can be removed from all of the boards except one. The resistor has a value of 10k and is marked below



To simplify the interface no hardware handshaking (CTS/RST) is used. All commands sent to the device are acknowledged which obviates the need for handshaking.

All commands are preceded with the device address which is what makes it possible to have more than one device on a single bus.

### 4.4. I2C

(**default address 0x72 8 bit and 0x39 7 bit**)

The I2C interface is a separate 4 pad connector next to the keypad connector and the pins are designated as follows:

| Pin | Function |
|---|---|
| 1 | SCL |
| 2 | GND |
| 3 | SDA |
| 4 | +V |

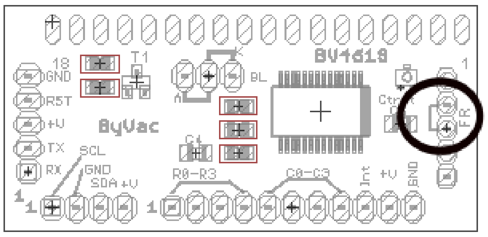
**Default I2C address is 0x72 (0x39)**

Pin 1 is to the left hand side furthest away from the keypad connector and is designated by a square pad.

There is a high value resistor connected to the SDA line and ground. When nothing is connected to the interface this condition is detected and the serial interface is selected.

If the device in on an I2C bus then there will be a pull up resistor on this pin which, at start up, will cause the device to select the I2C interface.

## 5. Hardware Reset



The purpose of the hardware reset is to reset the EEPROM back to a known position so that the display will work correctly. The procedure is:

1) Disconnect the display from power

2) Connect the reset pins together, these are shown on the picture above.
3) Apply power
4) Disconnect power
5) Remove the short

The display will now be reset back to the factory defaults.

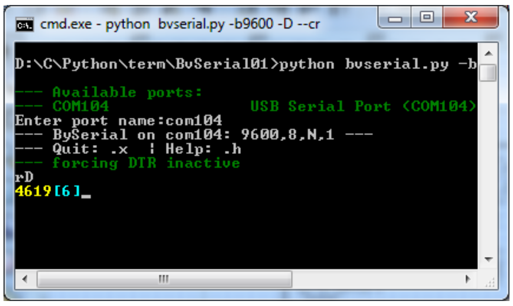
## 6. SV3

The serial interface uses 'Serial Version 3' which is a simple protocol for sending serial information to another device.

It works by sending a command and then optionally receiving information. When sending data it is always terminated by CR (13). as an example to get the ID of this device the following bytes are sent and received:

114, 68, 13 // send command

52, 54, 49, 57, 6 // bytes returned



```
D:\C\Python\term\BvSerial01>python bvserial.py -b
--- Available ports:
--- COM104                  USB Serial Port (COM104)
Enter port name:com104
--- BySerial on com104: 9600,8,N,1 ---
--- Quit: .x  | Help: .h
--- forcing DTR inactive
rD
4619[6]_
```

The above is what it looks like on a terminal.

## 7. I2C

The default I2C address is:

| | |
|---|---|
| 8bit | write 0x72 read 0x73 |
| 7bit | 0x39 |

## 8. Custom Characters

To create a custom character you have to initially write to the CGRAM, this is done by using the direct LCD command code (c) or the I2C equivalent. Then send bytes as data, as an example this is the pseudo code for creating an up arrow for character 0

```
send_control(64) // set write to CGRAM pos 0
send(4)
send(10)
send(21)
send(4)
send(4)
send(4)
send(4)
send(4)
send_control(128)
```

There will now be an up arrow character in position 0. To display it send(0). The character positions are as follows:

| Character | Base Address |
|---|---|
| 0 | 64 |
| 1 | 72 |
| 2 | 80 |
| 3 | 88 |
| 4 | 96 |
| 5 | 104 |
| 6 | 112 |
| 7 | 120 |

## 9. EEPROM values

The EEPROM contains important values that control the way the device behaves. All of the values can be changed by the user using the 'W' command.

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.

| Adr | Default Value | Description |
|---|---|---|
| 0 | 0 | System Use |
| 1 | 114 | Device address |
| 2 | 6 | ACK value |
| 3 | 21 | NACK value |
| 4 | 3 | Baud rate (9600) |
| 5 | 1 | Error reporting 1 = on |
| 6 | 13 | End of line |
| 7 | 0 | Invert TX 0=invert |
| 14 | 114 | Device address copy |
| 16 | 10 | Back light red value |
| 17 | 10 | Back light green value |
| 18 | 10 | back light blue value |
| 19 | 50 | Keypad debounce value |
| 32 | | Sign on message start |
| 250 | 114 | Device address copy |

**Table 1 EEPROM use**

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.

### 9.1. 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.

### 9.2. 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.

### 9.3. 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.

### 9.4. 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

### 9.5. 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.

### 9.6. 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.

### 9.7. Back light values

The back light values can range from 0 (off) to 10 (full on) and the settings in the EERPOM hold the values that are set on reset.

### 9.8. Sign on Message

The sign on message can be changed from the default by writing to these EEPROM locations. To access an LCD command use a value of 1 first and always finish the message with 0.

For example to clear the display and then have "fred" written the following bytes would be in EEPROM

| EEPROM | Byte | Notes |
|---|---|---|
| 32 | 1 | Command follows |
| 33 | 1 | This is the command to clear the screen |
| 34 | 102 | 'f' |
| 35 | 114 | 'r' |
| 36 | 101 | 'e' |
| 37 | 100 | 'd' |
| 38 | 0 | Terminate message with 0 |

## 10. Commands

All serial commands are proceeded by an address and terminate with CR (0xd). In the examples given below the address is 'r' or 0x72

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 |
|---|---|---|---|---|---|
| **b,r,g,b** | 1,r,g,b | 0-10 | | | **Back light Control**<br>The back light brightness is specified as 3 values to cater for displays that have colour back lights. If there is just one backlight then normally the red values is used but all three still need specifying.<br>**Examples**:<br>For a single colour backlight LCD set the value to 1/2 brightness<br>**r5,0,0**<br>For a multi colour display set it as yellow, full brightness.<br>**r10,10,0 (red+gree=yellow)**<br>**I2C**<br>a single byte is used, for the previous example:<br>**s 1 10 10 0 p** |
| **c** | 2 | 0-255 | | | **LCD Command**<br>This will send a command (as opposed to data) to the LCD. Commands can control the cursor or clear the screen etc. Consult the data sheet for the LCD display concerned.<br>**Examples**:<br>Clear the screen<br>**rc1**<br>Turn off the cursor<br>**rc12**<br>Cursor block flashing<br>**rc15**<br>Normal cursor<br>**rc14**<br>**I2C**<br>Example of the above<br>**s 2 14 p** |
| **d** | 3 | Send byte's | | | **Data**<br>This sends data to the display, it will display what is sent. Multiple bytes can be sent up to the internal buffer size which is 80 bytes. The data will be sent on receiving CR<br>Example: |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | **rdfred** (prints 'fred' to the screen)<br>**I2C**<br>This will also accept multiple bytes but **MUST** be terminated by 13<br>Example:<br>Send 'A'<br>**s 3 65 13 p**<br>Send 'fred'<br>**s 3 102 114 101 100 13 p** |
| **g** | 4 | 0-255 | | | **Gets key**<br>Returns the scan code for the key that was placed in the buffer on a first in first out basis so if keys 1,2 and 3 were pressed. Using this command 3 times would return the scan codes for 1,2 and 3 in order.<br>When a key is read with this command it is removed from the buffer. If the command is used when there are no keys in the buffer it returns 0.<br>Example:<br>**rg**<br>**I2C**<br>s 4 r g-1 p |
| **n** | 5 | 0-32 | | | **Number of bytes in buffer**<br>Each time a key is pressed it is entered into the buffer. This command returns the number of keys in the buffer |
| **dn** | 6 | 0-255 | 50 | 19 | **Sets debounce value**<br>This is set to 50 by default and should not really need changing. It is also stored in EEPROM so that the value can be picked up on reset.<br>The larger the value the greater the debounce but the slower the response time. The maximum value is 255. |
| **r** | 7 | | | | **Clears the Keypad buffer**<br>The keypad has a 32 byte buffer, this command clears it. |
| | | | | | |
| **Wn,m** | 0x91 | n=0-255<br>m=0-255 | | | **Write to EEPROM**<br>This will write a single byte to an EEPROM location, the command format is:<br>&lt;adr&gt;W&lt;EEPROM address&gt;,&lt;value&gt;<br>Care should be taken when using this command for two reasons:<br>1) The EEPROM can only be written to a certain number of times all be it a large number.<br>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.<br><br>**I2C Example write 23 to location 7**<br><br>s 0x91 7 23 p |
| **Rn,m** | 0x90 | n=0-255<br><br>m=0-255 | | | **Read from EEPROM**<br><br>The EEPROM values can be read with this command given a starting address and the number of bytes to read.<br><br><adr>R<"EEPROM adr><#bytes><br><br>This example will output the first 16 bytes of EEPROM<br><br>**rR0,16**<br><br>The output from the device will commence after receiving CR and will consist of a string of data terminated with ACK.<br><br>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:<br><br>"0,97,6,21,0"<ACK><br><br>**I2C**<br><br>I2C will read only single values at a time, to read from location 3:<br><br>s 0x90 3 r g-1 p |
| **D** | 0xa1 | | | | **Device ID**<br><br>Returns a number representing the device product number as a string<br><br>**rD**<br><br>Output from the above would be:<br><br>"4601"<ACK><br><br>**I2C** returns two bytes representing a 16 bit number, high byte first<br><br>s 0xa1 r g-2 p |
| **I** | n/a | | 1 | 7 | **Toggle Inverted**<br><br>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 from one to the other.<br><br>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.<br><br>Example<br><br>**rI**<br><br>This is only needed as a **one time** operation as the change is automatically written to EEPROM |
| **C** | 0x95 | | | | **Reset**<br><br>Resets an individual device. This is a soft |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | reset.<br><br>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.<br><br>Example<br><br>**rC**<br><br>**I2C**<br><br>**s 0x95 p** |
| **V** | 0xa0 | | | | **Version**<br><br>Returns the firmware version as a string in the format "H.L"<br><br>Example<br><br>**rV**<br><br>**I2C** Sends two bytes, major revision first so 2.7 world be 2 and 7 |
| **H** | n/a | | | | **Hello**<br><br>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:<br><br>for j = 97 to 122<br>  Send(chr$(j)+"H\r")<br>  if ack received then<br>     print device j found<br><br>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. |