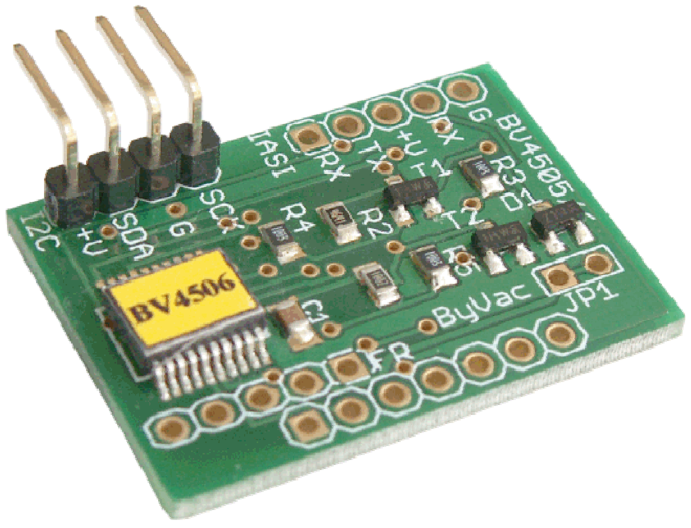
# I2C-Keypad Controller BV4506



**BV4506**

**I2C-Keypad Controller**

Product specification December 2008 V0.a

## Contents

# I2C-Keypad Controller BV4506

| Rev | Change |
|---|---|
| Dec. 2007 | Preliminary |
| October 2010 | The BV4506 is now sold separately as a controller. Description has been altered |

## 1. Introduction

The BV4506 is a 12 key keypad controller with an I2C interface. This makes it ideal for interconnecting to systems with the minimum of hardware.

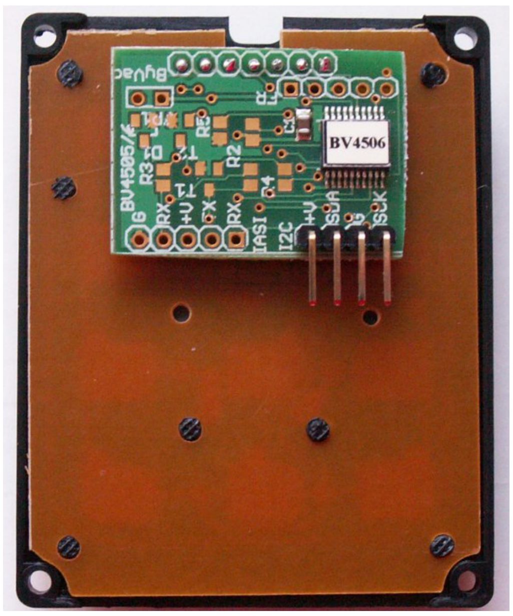
It has a 16 key first in first out buffer so that the host software does not need to continually scan the keypad. There is also a key down detect for key repeat applications or linear control.

**Default I2C address 0x62**

## 2. Features

- I2C up to 400kHz
- 16 key buffer
- 12 soft touch key switches
- Key down detect
- Programmable I2C address
- Size 65mm x 51mm x 22mm

## 3. Electrical Specification



The keypad has 4 pins for the I2C interface, Clock, Data, +V and Ground.

Pull up resistors are required for the I2C interface, these are not provided on the keypad board. The board is also used for the IASI2 interface and so any other holes should not be used.

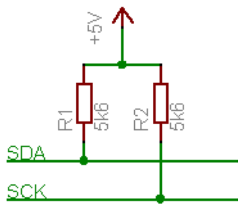
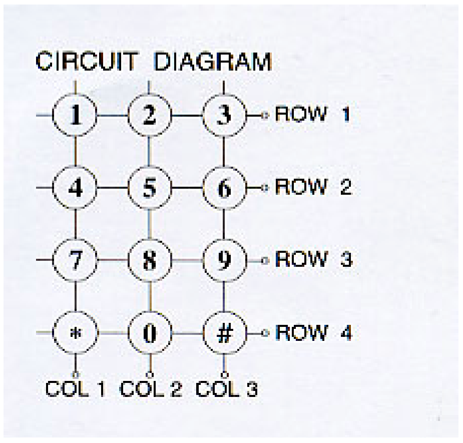


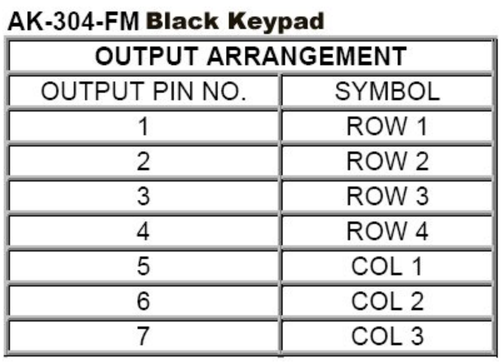
**Figure 1 I2C Bus pull up resistors**

### 3.1. Keypad Interface

There are two types of firmware available. One is intended for the white keypad and the other for the black keypad.



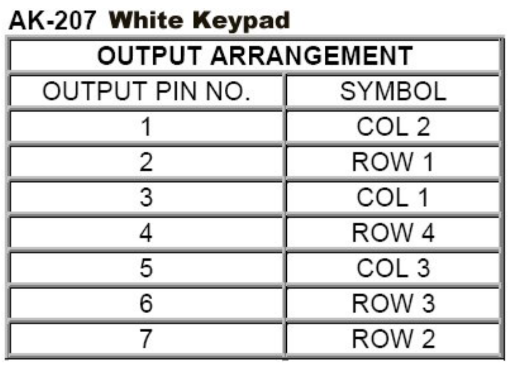
Both keypads are laid out as shown above but the roes and columns come to differing pins. The black keypad firmware expects the pins to be as follows:



**AK-304-FM Black Keypad**

| OUTPUT ARRANGEMENT | |
|---|---|
| OUTPUT PIN NO. | SYMBOL |
| 1 | ROW 1 |
| 2 | ROW 2 |
| 3 | ROW 3 |
| 4 | ROW 4 |
| 5 | COL 1 |
| 6 | COL 2 |
| 7 | COL 3 |

Pin number one is the square pad on the controller. The white keypad is laid as:



AK-207 **White Keypad**

| OUTPUT ARRANGEMENT | |
|---|---|
| OUTPUT PIN NO. | SYMBOL |
| 1 | COL 2 |
| 2 | ROW 1 |
| 3 | COL 1 |
| 4 | ROW 4 |
| 5 | COL 3 |
| 6 | ROW 3 |
| 7 | ROW 2 |

It is important to specify which firmware is required to match which keypad. For keypads with a completely differing layout then wires could always be used instead of soldering the board directly to the keypad in which case it would be probably better to go for the black keypad firmware.

## 4. Key Values Returned

| Key | Value |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 0 | 0 |
| # | 10 |
| * | 11 |

** The values returned are specifically designed for the matching keypad. Other keypads may give different values that must be interpreted by software.

In general command 1 is used to discover the number of keys that have been pressed and command 4 is used to read out those keys.

Command 1 can be used as the status since when there are no keys in the buffer 0 will be returned. Command 4 could also be used as a status because an attempt to read an empty buffer returns 255 (0xff).

## 5. Factory Reset

As the I2C address is stored in EEPROM and is fully configurable, it is possible that the user may forget what the address is. To restore factory defaults do the following:

1) remove power
2) Connect the square hole marked FR to the hole next to it, this will in fact connect the square hole to +V
3) Apply power
4) Remove power

The address and any other parameters that may be stored in the first half of the EEPROM will now be reset.

In general the first 16 bytes of EEPROM are normally reserved for system use but these areas can still be overwritten by the user.

## 6. 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 two type of command, those referring to the device and those referring to the system. The system commands enable changing of the device address etc.

**Device default address is 0x62**

| Command | AtoD Command Set |
|---|---|
| 1 | # of keys in the buffer |
| 2 | Clear key buffer |
| 3 | Key down |
| 4 | Get key/s in buffer |
| 0x55 | Test |
| **Command** | **System Command Set** |
| 0x90 | Read EEPROM |
| 0x91 | Write EEPROM |
| 0x92 | Confirm Command Complete |
| 0x93 | End of EEPROM |
| 0x94 | Sleep mode |
| 0x95 | Reset |
| 0x98 | Temporarily change address |
| 0x99 | Change Device Address Permanent |
| 0xA0 | Firmware Version |

**Table 1 Device & System Command Set**

Table 1 is a command summary of all of the device and system commands.

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 (0x62). 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..><Stop>

The restart address will be one greater then the start address, thus if the start address is 0x62, the restart address will be 0x63. Again the data can be one or more bytes read from the device.

***For more information about the command formats see section 9.***

Each command will have it's own format and is described in the following text. A start condition and address is always followed by a command.

### 6.1. Command 1

Name: **Number of keys in buffer**

Format:<**S-addr**><**1**><**R-Addr**><**Keys**>**<Stop>**

The keypad has a 16 byte (key) buffer and this command will return the number of keys in the buffer so that command 4 can read them out.

This can be used as a status command to determine if any keys need reading.

Note that if the buffer becomes full it will be cleared and all of the key data lost.

### 6.2. Command 2

Name: **Clear key buffer**

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

Clears the 16 key buffer and resets the pointer back to 0. It is only necessary to do this after initialisation.

### 6.3. Command 3

Name: **Key down**

Format:<**S-addr**><**3**><**R-Addr**><**1or0**>**<Stop>**

This will return 1 if a key is actually being pressed at the time of the command, returns 0 otherwise.

This command can be used for auto-repeat or linear control. For example keeping key 1 pressed increases the volume.

### 6.4. Command 4

Name: **Get key/s in buffer**

Format:<**S-addr**><**4**><**R-Addr**><**Value...**>**<Stop>**

If the key buffer has keys in it, this command will return the value of the keys. A single key value can be read or several values can be read at the same time.

In the following example keys 1,5 and 9 have been pressed.

**BV4221 Example**

```
0x62>s 4 r g-1 p
```

The result returned is 1, representing the first key pressed

```
0x62>s 4 r g-3 p
```

The result returned is : 5 9 255

Because, in total 4 keys have been requested and there were only 3 keys in the buffer, the last request returns 0xff (255), this is the default 'no-key' value.

| 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

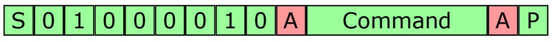
N Not acknowledge = 0

### 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:

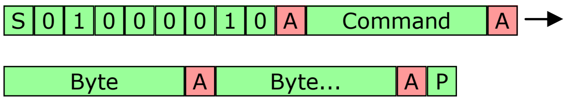


### 9.2. Sending a command with a parameter byte/s

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

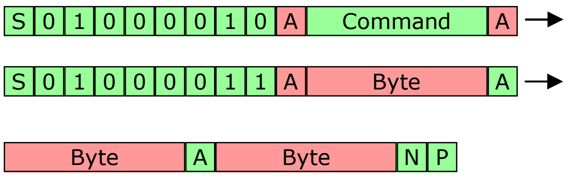
When optionally multiple reads of a slave is required (the 0x55 command is a good example) the last read should send a NACK rather then a ACK. This informs the slave that no more reads from that command are required.

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.


S 0 1 0 0 0 0 1 0 A Command A
S 0 1 0 0 0 0 1 1 A Byte A
Byte A Byte N P