BV4242

I2C Front Panel User Interface

Product specification June 2014 V0.a

Contents

I2C User Interface BV4242

Rev	Change
June 2014	Preliminary
Aug 2016	Error KEY, INT meaning swapped in description

1. Introduction

This is an I2C user interface for use with microcontrollers, for example the Arduino or Raspberry Pi.

The output is in the form of an LCD display and the input is a user-configured touch keypad with 16 keys.

Full I/O control can be realised with only 2 wires. The keypad has a 79 byte buffer relieving the host microcontroller of a considerable burden.

2. Description

The device consists of two parts, an LCD display and a Touch panel. Both operate from the same I2C address (0x7a (0x3d 7 bit)).

Each touch pad consist of two capacitive touch channels and a key is determined as being pressed when both channels are activated. All of this is decoded internally so the host is presented with a simple key value.

For ease of use the keypad will buffer keys so they can be read at a later time by the host microcontroller.

The front of the PCB is designed so that a vinyl or similar overlay can be stuck to it thus the pads can be designed for the application in mind. Several pads can be grouped to make one larger pad if required.

3. Features

- I2C display 16x1 AND 16 x2
- User selectable I2C address
- Software adjustable contrast
- Software variable back light
- 16 Pad touch keypad
- 79 key buffer
- Interrupt pins
- Pads fully configurable
- Mounts is standard box
- User printable Front panel overlays
- Wide voltage 3.3V to 5V
- 7.44mA @ 3V3 BL full on, 4.71mA 1/2 on, 1.5mA BL off
- 270uA @ 3V3 sleep mode, BL off
- Only 2 wires for full I/O control

• Physical Description

The interface is very simple with a 7 pin connector designated as follows:

Pin	Description
SCL	I2C clock
GND	Ground
SDA	I2C data
V+	3.3V to 5V
RST	Reset; leave disconnected, pull low to reset the device.
INT	Normally high, will go low if there are any keys in the buffer.
KEY	Goes low only when a key is being touched otherwise it is high

4. I2C Interface

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

0x7a (0x3d) Keypad & LCD address

All commands go through the single I2C address that can be changed if required by the user.

NOTE: The address is stored in EEPROM in three places and a check is made at each reset to verify the value. At leas two address location values have to agree, if this is the case the third is set to that. If no addresses agree then the default address is used.

This is a robust method of storing addresses in a semi-volatile memory and in nearly all cases the address set by the user is maintained for ever. However if it is critical that the address cannot change under any circumstances then the part can be ordered with a fixed address.

5. Keypad

5.1. Tuning

The touch panel has been set with default values that are **suitable for most applications and should not really be altered**. Having said that the performance is greatly effected by the covering used over the PCB. Thin vinyl does not effect it much but thicker , glossy photo paper does.

This text is provided for changes in physical conditions. It will also inform on how the pad works.

With care it is possible to adjust the pads to make them more or less sensitive.

The adjustable values are all stored in EEPROM and so they can be changed. It is possible to stop the keypad working with unsuitable values, if this happens there is an i2c EEPROM reset command.

The following is a description of how the pads are read and how they work.

There are 8 channels that are constantly being scanned. Each pad is associated with 2 channels to give the 16 pads on the device.

5.1.1.Timebase

Under 'untouched' conditions the channel will reveal a value, the magnitude of the value is determined by the timebase.

A timebase of 8mS will give a value of about 3000 and a timebase of 16mS will give a value of about 5000. The higher the value the better, however as there are 8 channels a full scan takes 8 x timebase so increasing the timebase will lead to a slower response.

EPROM	mS	Full scan
16	8mS	64 mS
32	16mS	128mS

Timing Examples

The table gives an idea of the delay likely when setting a different timebase values.

The values are slowly averaged to form a stable 'untouched' condition.

5.1.2.Trigger

When a pad is touched the normal, average value drops. Depending on the conditions and the timebase this can vary between 100 and 1500.

An ideal trigger is set to half that amount, so if the drop was 1000 then the trigger would be set to 500, in practice probably just a bit less. The compromise of course is that if the trigger is too high the pads will be very unresponsive, if too low false triggering can occur.

Once the trigger value has been exceeded averaging stops and the pad is deemed to be touched.

5.1.3.Channels

There are 8 channels but physically the touch pads have two channels per pad, this enables a 4x4 matrix of 16 pads to be used.

A further advantage is that 2 readings must be obtained before a pad is active making the system more reliable.

The actual physical arrangement is:

4,7	3,7	2,7	5,6	4,6	3,6	2,6	5,8
4,9	3,9	2,9	4,8	3,8	2,8	5,7	5,9

Table mapping channel numbers to pad layout

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16

Key pad numbers

The table shows the actual channels used which rang from 2-9. So for example channel 5 and 6 must be active for key 4 to be active.

5.1.4.Scan Code

At the lowest level a scan code is derived from the channel numbers.

2	3	4	5	6	7	8	9
MSB							LSB

Scan code derived from channel numbers

The scan code is a byte value on the second row of the table. An active channel will represent a bit 1 and an inactive will be bit 0. For example, refer to the channel numbers and key position tables above.

2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0

Scan code for key pad 3

In the above example channels 2 and 7 are active, this produces a scan code of 0b10000100 or 0x84.

The instantaneous scan code values can be read with the appropriate I2C command. These are not stored but the decoded value from the key table is.

5.2. Tuning Commands

In order to assist with tuning some commands have been provided. Command 10 will return 8 x 16 bit channel average values.

Note: The I2C command will need to fetch 16 values made up of the high and low bytes for each channel.

The average values will indicate what trigger level to have, this is an example:

3081 2932 3175 3020 3272 3285 2687 3135

Channel 2 is the first number and channel 9 the last. This can be combined with command 11 that will return the delta value, thus:

```
Avg:  3058 2901 3153 2990 3246 3269 2658 3093
Dlt:     0    0 1222    0    0 1147    0    0
```

In this example a finger has been placed on the first pad, command 11 is the second line and shows the difference between the average value and the touched value. A trigger of greater than

1300 would not register, the actual trigger value for this is about 500.

5.3. Tuning Summary

Step 1. The average values should be set so that they read around 2000 the greater the better. The values are adjusted with the timebase setting, the larger the value of the timebase the higher the average value. However this will effect the period between scans and so the response time of the keys.

The formula is timebase * 4mS, this gives the scan time. For example setting the timebase to 25 will give a scan time of (25*4) 100mS. This means that it will take 100mS to see any change in condition. In practice it may be possible to go to 200 or 300mS or even more depending on the application.

Step 2. Set the trigger to a low value say 100 and observe the delta output. This output is the difference between the average value and the pressed value. The delta output will only be observed when pressing a key. The trigger should be set to half the delta value.

5.4. Key Buffer

There is a 79 key, key buffer to store pressed keys. It is a circular buffer for maximum flexibility. It is up to the user to ensure that the buffer does not become full as this will overwrite previous keys.

There is an indicator bit (see Device Parameters section) that will send a message to the display if the buffer becomes full.

5.5. Sleep Mode

The device can be set to sleep mode via an I2C command. In this mode the keypad is inactive however the device can be awakened by an I2C read or write.

6. LCD

The LCD is a chip on glass (COG) type.

The LCD uses the ST7032i controller. This is almost identical to the very familiar HD44780 controller in that you can write commands and data to it.

It also has more advanced features such as double height characters and software controlled contrast. The character set itself is more comprehensive.

Access to the LCD is via just a few simple I2C commands. The commands will enable any manipulation of the LCD to be carried out. Here are a few examples.

Turn on the cursor:

bus.i2c(0x3d,[31,0xe],0)

Set cursor to beginning of line 1:

bus.i2c(0x3d,[31,0x80],0)

Set cursor to beginning of line 2:

bus.i2c(0x3d,[31,0xc0],0)

Clear display:

bus.i2c(0x3d,[31,1],0)

7. 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 i2c interface.

This mostly applies to the keypad as the LCD is a separate entity and is not controlled by the on board microcontroller but by the user.

The EEPROM consists of 255 bytes and in general the first 16 bytes are used by the system

Adr	Default Value	Description
0	0	System Use
1	122	Device address
2	45	Default contrast
3	4	Indicator flag
14	122	Device address copy
16	0	Reserved
17	0	Mode (keep to 0)
18	0	Trigger H
19	0xc8	Trigger L
20	5	Hysteresis
21	30	Key table pointer (KP)
22	16	Key table size
23	1	Debounce
24	9	Repeat H
25	0xc4	Repeat L
26	25	Timebase in 0.5mS
27	130	Back light
28	60	Sign on message location
250	122	Device address copy

Table 1 System EEPROM use

Key Code Table		
Location	**Name**	**Content**
KP+0	K1	0x24
KP+1	K2	0x44
KP+2	K3	0x84

KP+3	K4	0x18
KP+4	K5	0x28
KP+5	K6	0x48
KP+6	K7	0x88
KP+7	K8	0x12
KP+8	K9	0x21
KP+9	K10	0x41
KP+0	K11	0x81
KP+1	K12	0x22
KP+2	K13	0x42
KP+3	K14	0x82
KP+4	K15	0x14
KP+5	K16	0x11

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.

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

7.1. 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.

7.2. Contrast

This is the default contrast setting for the LCD display and the default value will give good results for a supply of 3.3V. If a 5V supply is used then it will be too dark a value of about 25 is required.

The contrast can be set at any time so this value does not need changing it is simply the value that is used for initialisation.

7.3. Indicator Flag

NOTE: This flag is intended for debugging mainly. The 'features' will more than likely get in the way of a user program and so should probably be switched off. There is one exception and that is the buffer full flag. As the buffer should never get full it will indicate programming errors.

This is a byte that has three bit value, when set to 1 the indicator is on, when set to 0 it is off:

0b00000ABC

If bit A is set (**key buffer full**)

When this flag is set and the key buffer becomes full, a message is printed on the bottom line of the LCD display.

If bit B is set (**keys in buffer**)

If this bit is set the cursor will blink when there are keys in the buffer.

If bit C is set (**BL key flash**)

If set then when a valid key is detected the back light will flash off and then on.

7.4. Mode

This is used for testing purposes and should always be 0

7.5. Trigger

This is a 16 bit value. The high and low values are stored separately. If the trigger value is for example 420 then this should be converted to hex (0x1a4). The least significant digits 'a4' are the low value and the most significant '1' is the high value.

In this example 1 would be stored in location 18 and 0xa4 would be stored in location 19.

7.6. Hysteresis

This value should be set low, somewhere between 3 and 20. It is difficult to determine the exact effect but will go some way towards preventing jitter (on/off/on) when a pad is touched.

7.7. Key Table Pointer

This holds the address of where the key table is. It would of course be possible to have other key tables stored by adjusting this pointer

7.8. Key table size

As it says

7.9. Debounce

This is the number of full scans before a key pad touch is accepted. See the text and timebase for how long a full scan takes. Increasing this value will delay the response time.

7.10. Repeat

This is a 16 bit number stored high and low (see trigger). The actual value is found by trial and error. When a pad is touched the value is immediately recorded, if the finger is held there

another, same value is recorded until the pad is untouched.

The time delay between each key record is determined by this value. The default value of 1256 (0x4e8) gives about 1/2 second.

7.11. Timebase

This value is multiplied by 0.512mS, so the default value of 16 gives 8.2mS. Further details about what this does and how to adjust it is given in the 'Tuning' section of the text.

7.12. Back Light

This is the back light brightness at start up and can be set between 0 and 130 with 130 being maximum brightness.

Values greater then 130 will make no difference.

7.13. Key Table

When a key (made up of 2 or more channels) is touched it produces a unique scan code depending on which channels have been touched.

A further explanation of this is given in the 'tuning' section of the text.

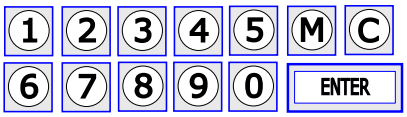
The key table is searched for the scan code and if it is found then the POSITION of the code is stored in the key buffer.

So for example if the scan code was 0x28 then 5 would be stored in the key buffer. The codes here give an extra level of stability as it is necessary for two and only two channels to be activated for the code to be accepted.

If just one channel is pressed by a finger not quite on the pad then this will not be accepted and also if the finger is across 2 pads this will also not be accepted.

There may be occasions when this is infract desirable to create a larger keypad made up of several keypads for example. The key table can be used for that.

Here is a practical example where two keys make a larger key that forms an enter key.



Avg: 4401 4051 4441 4086 4488 4562 3692 4414

Dlt: 634 0 0 0 0 0 577 0

Scan Code: 0x82

This code is for the left hand side of the enter key.

Avg: 4389 4047 4442 4088 4482 4559 3683 4409

Dlt: 322 0 0 330 0 482 296 0

Scan Code: 0x86

This code is when both the left and right keys are touched - in the middle of the enter key.

Avg: 4364 4038 4432 4090 4469 4561 3667 4388

Dlt: 0 0 0 528 0 730 0 0

Scan Code: 0x14

This code is for the right hand side of the enter key.

We could therefore create an extra key code for the scan code of 0x86. The host could then look for 3 codes; any of the three will be the enter key.

7.14. Sign On

The start up message is stored in EEPROM and so can be changed using the write to EEPROM command. The start of the message location is given in the table above.

The EEPROM is read from that location and will send any byte as data to the display. If is command is required then this is preceded by a 0 and the byte command is sent.

The sequence should be terminated with 0xff.

Example: "Hello" on first line "World" on second line:

"Hello",0,0xc0,"World",0xff

0xc0 is the command to send the cursor to the start of line 2. In bytes this would look like:

72,101,108,105,111,0,192,87,111,114,108,100, 255

0,192 is the command.

255 is the terminator.

7.15. Tips

Don't put in codes that only have one channel (0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01) as each channel is connected to 4 pads then it is likely that a false reading will occur.

When using multiple keys it may be possible to use the code or channel from between the two keys. Thought must be put into this though as this may be the code for another key.

The design of the keypad overlay will need careful consideration of this fact if 'in between' values are to be used..

8. Keypad Commands

Key pad commands **I2C address 0x7a (0x3d 7 bit address)**

All I2C transactions start with a command for example:

8 bit pseudo code get value from buffer	**7 bit pseudo code get value from buffer**

i2c_start(0x7a) // write
i2c_putc(3)
i2c_stop()
i2c_start(0x7b) // read
value = i2c_getc()
i2c_stop()
	i2c_start()
	i2c_write(0x3d,3)
	i2c_stop()
	i2c_start()
	value = i2c_read(0x3d)
	i2c_stop()

The examples given in this table user notSMB (http://www.pichips.co.uk/index.php/RPi_Not_smBUS) that has three parameters:

optional value = bus.i2c(<i2c 7 bit address>[write to i2c],read from i2c), example :

value = bus.i2c(0x3d,[3,7],2)

This will address a device 0x3d, send bytes 3 and 7 and then read two bytes. In Python 'value' will be a list that can handle multiple bytes.

I2C Command	range	Default Value	EEPROM Location	Description
1	n/a			**Clears keypad buffer** **bus.i2c(0x3d[1],0)**
2	0-79			**Gets number of keys in buffer** Returns 0 if no keys are in the buffer **value = bus.i2c(0x3d[2],1)**
3	0-16			**Get key value from buffer** Key values are from 1 to 16 but this can be reduced or increased by configuring the key pad table in the EEPROM 0 is returned if no keys are in the buffer **value = bus.i2c(0x3d[3],3)**
4	0-16			**Key in Buffer** Checks to see if a particular key is in the buffer. It returns 0 if the key has not been found or a number representing the position of the key in the buffer. **value = bus.i2c(0x3d[4,keyToFind],1)**
5	0-255			**Get scan code** See 'tuning' section in the text for an explanation of what a scan code is. This will return a scan code if a pad is being touched and 0 if not. The command will produce unreliable results for a particular key however it may useful for something like a volume control. Any valid key in the key table will still be stored in the buffer so this should be cleared from time t time. **value = bus.i2c(0x3d[5],1)**
10	0-65535			**Returns 8 average values representing channels 2 through 9**

				This will in fact return **sixteen** values as I2C can only return 8 bits at a time. The value is sent as high low. To get the actual value requires something like: value = i2c_get() << 8 value = value + i2c_get() The value will now contain a 16 bit number. **value = bus.i2c(0x3d[10],16)**
11	0-65535			**Delta values for all channels** This returns 16 values (8 16 bit channels see command 10) The value returned represents channels 2 through 9, 2 is the first 9 is the last. The actual value returned is the difference between the average value and the touched value. If no pads are touched then the return value will be 0. The trigger value is important in that the touched value has to be lower then the average minus the trigger for this value to change from 0. If ny zeros are being returned when the pad is touched then the trigger is set too high. See also the 'tuning' section of this text. **value = bus.i2c(0x3d[11],16)**
20				**EEPROM reset** Will reset the EEPROM back to the default values. **bus.i2c(0x3d,[20],0)**
21				**Sleep** This will put the device into sleep mode. Once in this mode the only way to wake is either by reset or an I2C read/write. The keypad will not work in sleep mode. **bus.i2c(0x3d,[21],0)**
LCD	**Range**	**Time**		
30	0-255	500mS		**Reset LCD** Resets LCD. This is just the LCD and will not print the sign on message NOTE: This command requires 500mS to complete before sending the next I2C command. **bus.i2c(0x3d,[30],0)**
31	0-255			**LCD Command** Sends a command to the LCD controller; a command usually effects the way the display behaves. Example: to clear the screen: **bus.i2c(0x3d,[31,1],0)** Example to put cursor on the second line:

				bus.i2c(0x3d,[31,0xc0],0)
32	0-255			**LCD Data** Writes characters to the display at the current cursor position. Example, writes 'f' **bus.i2c(0x3d,[32,66],0)** **NOTE:** This will only write single bytes, each byte to be written must be preceded by the 32 command.
33	string of characters followed by 0			**LCD Data String** Writes a string of characters to the display at the current cursor position. Example, writes 'abcd' **bus.i2c(0x3d,[32,61,62,62,64,0],0)** **WARNING:** Leaving the terminating 0 off may cause indeterminate results for the next command and even cause the I2C bus to lock up.
35		10mS		**LCD Sign on** Displays the current sign on string stored in EEPROM, this is useful for testing **bus.i2c(0x3d,[35],0)**
36	0-130			**Sets Back light brightness level** The level can be set from 0 to 130 where 0 is off and 130 is full on, values higher then 130 are accepted but it makes no difference Example to dim the back light, set it to say 50. **bus.i2c(0x3d[50],0)**
37	0-63			**Sets Contrast level** The level required depends on the supply voltage and there is a considerable difference between 3.3V and 5V settings. Normally 25 is okay for 5V and 45 is okay for 3.3V. Example, set contrast to 25 **bus.i2c(0x3d,[37,25],0)** This is a hybrid command and can also be derived from using the lcd command (31) above.
38	0 or 1			**Sets Mode (1 line, 2 line)** The display can operate as a 1 line x 16 with double height letters or the more normal 16x2 with single height letters. Using this command 0 is 2 line and 1 is single line. Example, set to singe line mode **bus.i2c(0x3d,[38,1],0)** This is a hybrid command and can also be derived from using the lcd command (31) above.
System				

0x91	n=0-255 m=0-255			**Write to EEPROM** This will write a single byte to an EEPROM location **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.
0x90	n=0-255 m=0-255			**Read from EEPROM** Reads a singe EEPROM values from a given address. **Example** 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.
0xa1				**Device ID** 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.
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. Example **s 0x95 p** or **bus.i2c(0x34,[0x95],0)** See www.pichips.co.uk and 'notsmb' for an explanation of the above nomenclature.
0xa0				**Version** Returns the firmware version as two bytes **value = bus.i2c(0x3d[0xa0],2)**