

Basic I/O Shield™ Reference Manual

Revised June 27, 2017

This manual applies to the Basic I/O Shield rev. C

Overview

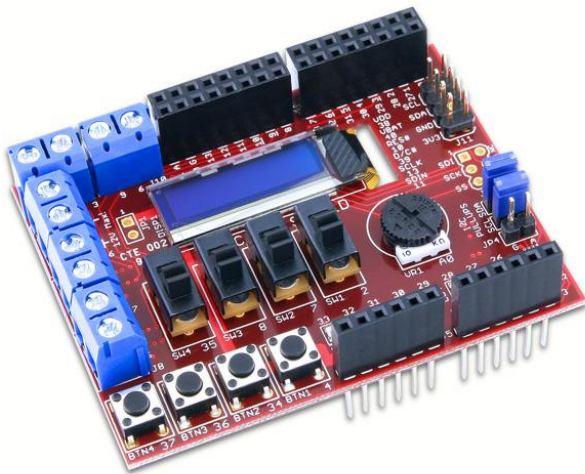
The Basic I/O Shield is an input/output expansion shield designed for use with microcontroller boards such as the Uno32™, uC32™, and the Max32™. These boards are based on the popular Arduino® IDE open-source hardware prototyping platform.

The Basic I/O Shield provides a range of input/output devices for both beginners and advanced users. It provides simple digital input devices such as switches and buttons, and digital output devices such as discrete LEDs and high-current open FET drivers. It also provides more advanced devices such as an I²C EEPROM, an I²C temperature sensor, an organic LED graphic display, and a potentiometer for use as an analog input device.

The Basic I/O Shield has the same form factor as the Uno32 board, but is also useable with the Max32 board.

Features include:

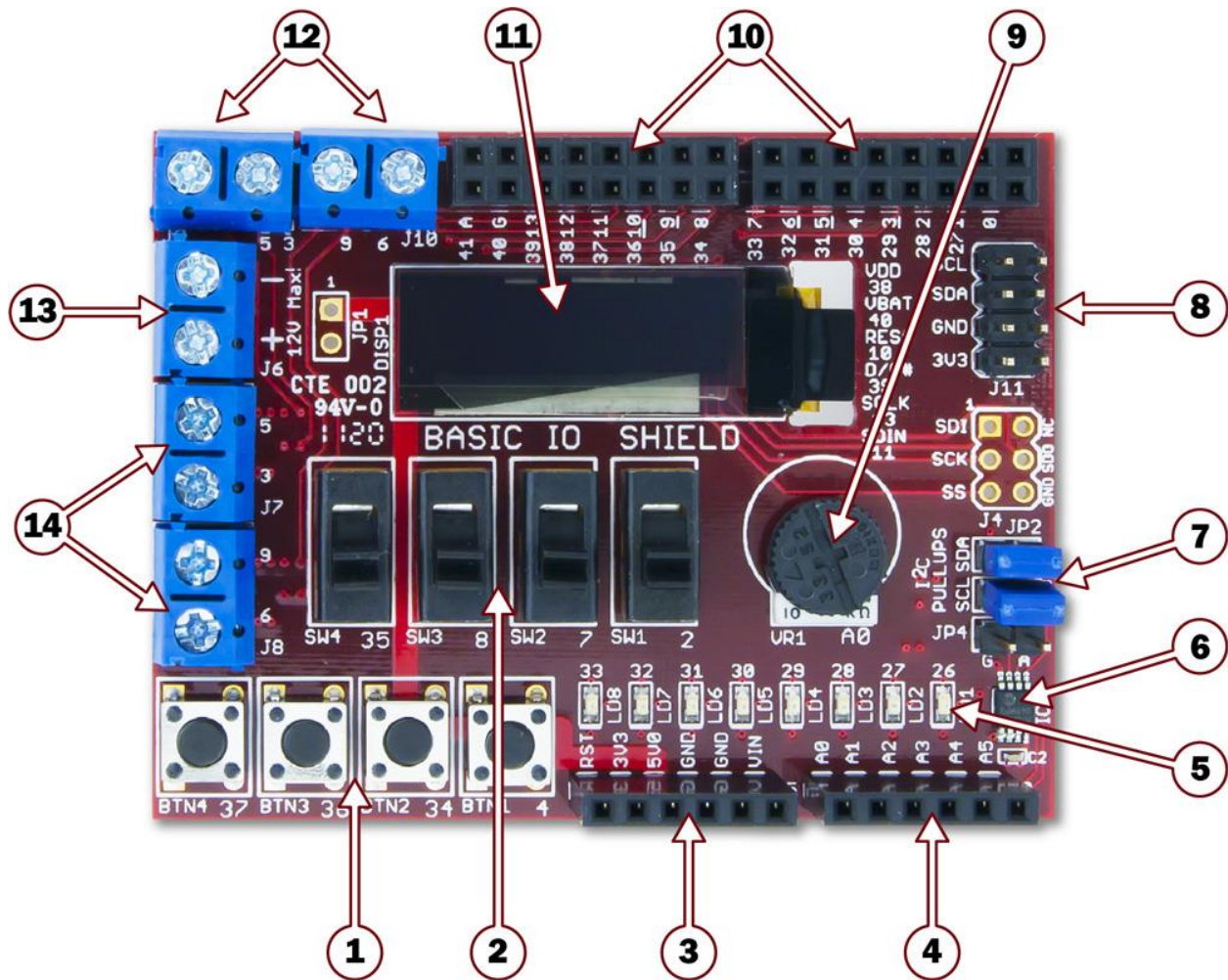
- A 128x32 pixel OLED graphic display.
- An I²C temperature sensor.
- A 256Kbit I²C EEPROM.
- An I²C daisy chain connector.
- Four push buttons.
- Four slide switches.
- Eight discrete LEDs.
- Four open-drain FET drivers.
- An analog potentiometer.



The Basic I/O Shield.

1 Hardware Overview

The Basic I/O Shield has the following hardware features:



1. Pushbuttons

Four pushbuttons provide discrete momentary-contact digital inputs. These can be accessed using the `digitalRead` function.

2. Slide Switches

Four slide switches provide discrete digital inputs. These can be accessed using the `digitalRead` function.

3. J2: Shield Power Connector

This connector receives power from the microcontroller board.

4. J3: Analog Signal Connector

Pin 1 (analog signal A0) is connected to a potentiometer. Pins 5 and 6 (labeled A4 and A5) provide access to the I²C signals SDA and SCL for the I²C bus. The I²C bus is used to access the temperature sensor and the EEPROM and for the I²C daisy chain connector.

Note: On the Uno32 and uC32 boards, you need to set jumpers JP6 & JP7 to the correct position to access the I²C signals on that board. On the Max32 board, you need to use jumper wires to bring the I²C signals onto the board, because the connector that provides the I²C bus on the Max32 board doesn't contact the Basic I/O Shield.

5. Discrete LEDs

The eight LEDs provide discrete digital outputs. These can be accessed individually using the digitalWrite function. The LEDs are connected to the low eight bits of microcontroller PORTE and all eight can be written at the same time by writing to PORTE.

6. IC2: Temperature Sensor

This is a Microchip TCN75A digital temperature sensor. It is accessed via the I²C bus.

7. JP2 & JP3: I²C Pull-up Resistor Enable Jumpers

These jumpers are used to enable or disable the I²C pull-up resistors on the I²C bus. Installing the shorting blocks enables the resistors. Removing the shorting blocks disables the resistors.

8. J11: I²C Daisy Chain Connector

This is a 2x4 pin header connector that provides access to the I²C signals SDA and SCL as well as power from the 3.3V power bus and ground. The connector can be used to extend the I²C bus to another board and to power an external I²C device. Digilent has cables and a selection of I²C peripheral modules that can be connected to this shield.

9. Potentiometer

This is a 10K-ohm potentiometer connected across VCC3V3 and ground. It provides an analog input voltage to analog input A0.

10. J1 & J2: Digital Signal Connectors

These connectors bring digital signals from the microcontroller board to the Basic I/O shield.

11. Organic LED Graphic Display

This is a 128x32 pixel monochrome OLED graphic display panel is accessed using the SPI interface.

12. J9 & J10: Digital I/O Signal Connectors

These are screw terminal connectors that provide access to digital I/O signals 3, 5, 6, and 9. These are four of the PWM signals from the microcontroller board.

13. J6: Open Drain FET Power Connector

This provides access to the power connections for the open drain FETs.

14. J7 & J8: Open Drain FET Output

These provide access to the outputs of the open drain FETs.

2 Hardware Description

The following is a description of the Basic I/O Shield and how to use it. See Appendix A for pin definitions and Appendix B for sample code to use the OLED graphic display.

2.1 OLED Graphic Display

The Basic I/O Shield provides a 128x32 pixel organic LED (OLED) graphic display panel, the WiseChip/Univision UG-23832HSWEG04. The UG2832 uses the Solomon Systech SSD1306 display controller.

Note: The UG2832 has a power on/power off sequence that should be followed. Failure to follow the power on/power off sequence can shorten the life of the display.

The Basic I/O provides two FETs for software control of the two power supplies for the display. The VDD_EN control is used to toggle the power to the logic of the display. The VBAT_EN control is used to toggle power to the OLED display itself. These two pins have pull-up resistors to turn off their respective power supplies when not being driven. The pin is made an output and driven low to turn on the power supply.

Power on sequence:

1. Apply power to VDD.
2. Send Display Off command.
3. Initialize display to desired operating mode.
4. Clear screen.
5. Apply power to VBAT.
6. Delay 100ms.
7. Send Display On command.

Power off sequence:

1. Send Display Off command.
2. Power off VBAT.
3. Delay 100ms.
4. Power off VDD.

The display has a D/C pin (display or command select) that is used to determine whether bytes sent to the display are interpreted as commands or as display data. The D/C pin is set high for display buffer access and low for command access.

The RES pin is used to reset the SG1306 display controller. The RES pin is driven low for reset and driven high for normal operation. The low-going reset pulse must be a minimum of 3 μ s (microseconds) for the display controller to reset correctly.

The UG2832 is a serial device that is accessed using SPI. It is however, a write-only device. It is not possible to read back either the display buffer contents or any kind of status from the panel. The maximum SPI clock frequency supported by the UG2832 is 10MHz. Due to pin limitations between the Basic I/O Shield and the Uno32 board, the select pin (SS) is wired low on the Basic I/O Shield and the display is always enabled to receive data over the SPI interface.

Digilent has a set of libraries for using the Basic I/O Shield with the MPIDE. The OLED library can be used as-is or as a starting point for a more sophisticated graphics library.

It is contained in document # DSD-0000311 (chipKIT IOShield Library.zip) which can be downloaded from the Basic I/O Shield product page at www.digilentinc.com. The OLED library is IOShieldOled.

Appendix B provides sample driver code for initializing the display and writing to it.

2.2 Discrete Digital I/O Devices

The Basic I/O Shield provides various discrete digital I/O devices. These can be accessed using the `digitalRead` and `digitalWrite` functions. The `pinMode` function is used to set the pin to input or output.

When the buttons and switches on the Basic I/O Shield are not being used, the pins are available on the pass-through shield connectors for use by other shields in the stack. The pins used by the LEDs are also available, however the presence of the LEDs on the lines will load them down, possibly causing some devices to not work. It is safe to use any of these pins as either inputs or outputs.

Pushbuttons: There are four pushbuttons labeled BTN1 (pin 4), BTN2 (pin 34), BTN3 (pin 36), and BTN4 (pin 37). The `digitalRead` function will return low if the button is not pressed and high when the button is pressed.

Slide Switches: There are four slide switches labeled SW1 (pin 2), SW2 (pin 7), SW3 (pin 8) and SW4 (pin 35). The `digitalRead` function will return low when the switch is down (toward the pushbuttons) and high when the switch is up (toward the OLED display).

LEDs: There are eight LEDs labeled LD1 –LD8 that are accessed through digital pins 33 – 26. An LED is illuminated when the corresponding pin is set to high using the `digitalWrite` function and off when set to low.

The LEDs are attached to the low eight bits of PORTE, with LD1 connected to PORTE bit 0, and LD8 connected to PORTE bit 7. An 8-bit value written to PORTE (or LATE) will display the corresponding binary value on the LEDs.

2.3 Open Drain FET Outputs

The Basic I/O Shield has four open-drain FET outputs. These are low-side N-channel devices and can be used to provide a digital switch closure to ground. These can be used to switch external loads such as relay coils, solenoids, stepper motors, and so on.

An FET is used to switch an external load, such as a relay coil, on and off. The load is wired between the positive side of an external power supply and the output of the FET. When the FET is switched on, current flows from the external power supply through the load and the FET to ground.

FETs are often used to switch highly inductive loads, such as relay coils, solenoids, and motors. When the current through an inductive load is switched off, a voltage spike will occur that can damage the FET. The Basic I/O Shield has clamp diodes (also called snubber or flyback diodes) to clamp the voltage spike and feed the current back to the external supply.

The NTHD4508N FETs are rated for a maximum V_{DS} (voltage from drain to source) of 20V. They are rated for a maximum continuous current of 3.0A at 25°C and 2.2A at 85°C. For detailed specifications, see the data sheet available at www.onsemi.com.

The FETs are labeled Q1A (pin 9), Q1B (pin 6), Q2A (pin 5), and Q2B (pin 3). These four pins are also four of the pulse width modulation (PWM) outputs supported by the boards. The FETs can be switched using PWM via the `analogWrite` function.

An FET is switched on by driving its gate high. When the FET is on, it provides a low impedance path to ground (similar to a closed switch to ground). When the FET is switched off by driving its gate low, it becomes a high impedance path to ground (similar to an open switch to ground). Pull-down resistors are connected to the gate of each FET to ensure that it is off unless being actively driven high by the microcontroller on the board.

The FET outputs are accessed via screw terminal connectors J7 and J8 on the left side of the shield. The digital signals used to switch the FETs on and off are also available on screw terminal connectors J9 and J10 at the left side of the upper edge of the shield.

Screw terminal connector J6 is used to make the power and ground connections back to the external power supply. The negative (-) pin of J6 connects the external supply ground with the Basic I/O Shield ground, and the positive (+) terminal connects the Basic I/O Shield to the external supply voltage.

The unloaded 2-pin header JP1 can be used to connect the shield's VCC5V0 power supply bus to the external power supply bus so that the shield's 5V supply can be used instead of an external supply. In this case, J6 is used to connect the shield's supply to the external loads. If this is done, the total current consumed by all external loads must not exceed 2A or the current rating of the shield's power supply (whichever is less).

2.4 I²C Bus

The I²C bus from the microcontroller board is brought onto the Basic I/O Shield. There are two I²C devices on the board and a connector for taking the I²C bus off the shield to connect to additional external I²C devices.

I²C Connector: Connector J11 can be used to extend the I²C bus off of the shield to connect to additional external I²C devices. J11 is a standard 2x4 pin header connector with 0.100" spaced pins. It provides access to the I²C signals, SCL and SDA, plus VCC3V3 and ground. The VCC3V3 can be used to power external I²C devices.

The I²C bus uses open collector drivers to allow multiple devices to drive the bus signals. This means that pull-up resistors must be provided to supply the logic high state for the signals. The Basic I/O Shield provides 2.2K-ohm pull-up resistors. Generally, only one set of pull-up resistors is used on the bus. Jumpers JP2 and JP3 can be used to disable the on-board pull-up resistors if a different value is needed or some other device on the bus is providing the pull-ups. The on-board pull-up resistors are enabled by install shorting blocks on JP2 and JP3. Removing the shorting blocks disables the pull-up resistors.

Digilent has several small I/O peripheral modules (Pmods) that can be connected using the I²C connector. These include a 3-axis accelerometer, a 4-channel 12-bit A/D converter, a serial character LCD panel, a 3-axis gyroscope, a real-time clock/calendar, and an I/O expander.

EEPROM: A 256Kbit (32Kbyte), I²C EEPROM is provided using a Microchip 24LC256. This EEPROM, IC1, is located on the bottom left of the shield, just below the logo.

The 7-bit I²C device address for the EEPROM is '1010000'.

Digilent has a library for using the EEPROM. It is contained in document # DSD-0000311 (chipKIT IOShield Library.zip) which can be downloaded from the Basic I/O Shield product page at www.digilentinc.com. The EEPROM library is IOShieldEEPROM.

For details on the 24LC256, see the data sheet at www.microchip.com.

Temperature Sensor: A digital temperature sensor is provided using a Microchip TCN75A 2-Wire Serial Temperature Sensor. The temperature sensor, IC2, is an I²C device, and is located in the lower right corner of the shield.

The TCN75A is rated for an accuracy of +/-1°C and has selectable resolution from 0.5°C down to 0.0625°C. The 7-bit device address is '1001000'.

Digilent has a library for using the temperature sensor. It is contained in document # DSD-0000311 (chipKIT I/OShield Library.zip) which can be downloaded from the Basic I/O Shield product page at www.digilentinc.com. The temperature sensor library is I/OShieldTemp.

The TCN75A provides an alert output that can be programmed for various functions. This output can be accessed using JP4, which is adjacent to IC2.

For complete technical documentation on the TCN75A, see the data sheet available at www.microchip.com.

2.5 Potentiometer

A potentiometer (pot) is provided on the shield for use as an analog signal source or analog control input. The pot is a 10K-ohm trimmer pot connected between the VCC3V3 supply and ground. The wiper of the pot is connected to analog input A0.

The pot is read using the analogRead function.

Appendix A: Basic I/O Shield Pinout Table

Uno32 Pin #	Max32 Pin #	Function	Description	Notes
10	10	RES	OLED reset	JP4 on Uno32
39	83	DC	OLED data/command select	
13	13	SCLK	OLED serial clock	
11	11	SDIN	OLED serial data in	JP7 on Uno32/JP4 on Max32
40	84	VBAT_EN	OLED VBAT enable	
38	82	VDD_EN	OLED VDD enable	
33	77	LD1	User LED	
32	76	LD2	User LED	
31	75	LD3	User LED	
30	74	LD4	User LED	
29	73	LD5	User LED	
28	72	LD6	User LED	
27	71	LD7	User LED	
26	70	LD8	User LED	
4	4	BTN1	Pushbutton	
34	78	BTN2	Pushbutton	
36	80	BTN3	Pushbutton	
37	81	BTN4	Pushbutton	
2	2	SW1	Slide switch	
7	7	SW2	Slide switch	
8	8	SW3	Slide switch	
35	79	SW4	Slide switch	
3	3	OC1	Open drain/PWM output	
5	5	OC2	Open drain/PWM output	
6	6	OC3	Open drain/PWM output	
9	9	OC4	Open drain/PWM output	
19	59	SCL	I2C clock	JP8 on Uno32/jumper wire on Max32
18	58	SDA	I2C data	JP6 on Uno32/jumper wire on Max32
14	54	A0	Potentiometer	

Appendix B: Example Driver Code

The following gives an example of code that initializes the display controller and writes the contents of a memory buffer into the display. This example is written for the Uno32 board.

Symbol and Variable Declarations

```

/* ----- */
/* Pin definitions for access to OLED control signals on Uno32
*/

#define prtVddCtrl      IOPORT_F
#define prtVbatCtrl    IOPORT_F
#define prtDataCmd     IOPORT_F
#define prtReset       IOPORT_G
#define bitVddCtrl     BIT_6
#define bitVbatCtrl    BIT_5
#define bitDataCmd     BIT_4
#define bitReset       BIT_9

/* ----- */
/* Symbols describing the geometry of the display.

#define cbOledDispMax  512      //max number of bytes in display buffer

#define ccolOledMax    128     //number of display columns
#define crowOledMax    32     //number of display rows
#define cpagOledMax    4      //number of display memory pages

/* ----- */

/* This array is the off-screen frame buffer used for rendering.
** It isn't possible to read back from the OLED display device,
** so display data is rendered into this off-screen buffer and then
** copied to the display.
*/
BYTE rgbOledBmp[cbOledDispMax];

```

PIC32 Hardware Initialization

The following function initializes the PIC32 hardware for talking to the display. It initializes the SPI controller and sets the control pins to be outputs.

```

/* ----- */
/** OledHostInit
**
** Parameters:
**     none
**
** Return Value:
**     none
**
** Errors:
**     none
**
** Description:
**     Perform PIC32 device initialization to prepare for use
**     of the OLED display.
**     This example is hard coded for the Uno32 and
**     SPI2.
*/

```

```

void
OledHostInit()
{
    unsigned int    tcfg;

    /* Initialize SPI port 2.
    */
    SPI2CON = 0;
    SPI2BRG = 15;                //8Mhz, with 80Mhz PB clock
    SPI2STATbits.SPIROV = 0;
    SPI2CONbits.CKP = 1;
    SPI2CONbits.MSTEN = 1;
    SPI2CONbits.ON = 1;

    /* Make pins RF4, RF5, and RF6 be outputs.
    */
    PORTSetBits(IOPORT_F, bitVddCtrl|bitVbatCtrl|bitDataCmd);
    PORTSetPinsDigitalOut(prtDataCmd, bitDataCmd);        //Data/Command# select
    PORTSetPinsDigitalOut(prtVddCtrl, bitVddCtrl);        //VDD power control
(1=off)
    PORTSetPinsDigitalOut(prtVbatCtrl, bitVbatCtrl);        //VBAT power control
(1=off)

    /* Make the RG9 pin be an output. On the Basic I/O Shield, this pin
    ** is tied to reset.
    */
    PORTSetBits(prtReset, bitReset);
    PORTSetPinsDigitalOut(prtReset, bitReset);
}

```

Display Controller Initialization

The following function performs initialization of the display controller on the display panel. This performs the power up sequence on the display and initializes it for a non-interleaved display buffer with the origin in the upper left corner.

```

/* ----- */
/** OledDspInit
**
** Parameters:
**     none
**
** Return Value:
**     none
**
** Errors:
**     none
**
** Description:
**     Initialize the OLED display controller and turn the display on.
*/

void
OledDspInit()
{
    /* We're going to be sending commands, so clear the Data/Cmd bit
    */
    PORTClearBits(prtDataCmd, bitDataCmd);

    /* Start by turning VDD on and wait a while for the power to come up.

```

```

*/
PORTClearBits(prtVddCtrl, bitVddCtrl);
DelayMs(1);

/* Display off command
*/
Spi2PutByte(0xAE);

/* Bring Reset low and then high
*/
PORTClearBits(prtReset, bitReset);
DelayMs(1);
PORTSetBits(prtReset, bitReset);

/* Send the Set Charge Pump and Set Pre-Charge Period commands
*/
Spi2PutByte(0x8D);
Spi2PutByte(0x14);

Spi2PutByte(0xD9);
Spi2PutByte(0xF1);

/* Turn on VCC and wait 100ms
*/
PORTClearBits(prtVbatCtrl, bitVbatCtrl);
DelayMs(100);

/* Send the commands to invert the display. This puts the display origin
** in the upper left corner.
*/
Spi2PutByte(0xA1);           //remap columns
Spi2PutByte(0xC8);           //remap the rows

/* Send the commands to select sequential COM configuration. This makes the
** display memory non-interleaved.
*/
Spi2PutByte(0xDA);           //set COM configuration command
Spi2PutByte(0x20);           //sequential COM, left/right remap enabled

/* Send Display On command
*/
Spi2PutByte(0xAF);
}

```

Display Memory Update

This function copies the contents of a 512-byte buffer from PIC32 memory to the display. The display memory is organized as four pages of 128 bytes each. Each memory page corresponds to an 8-pixel-high stripe across the display. Each byte in the memory page corresponds to an 8-pixel-high column on the display. The least significant bit in a display byte is the top most pixel, and the most significant bit the bottom most pixel. The first byte in the page corresponds to the left most pixels on the display and the last byte the right most pixels.

This function assumes that the display buffer to be copied is the global variable `rgbOledBmp`

```

/* ----- */
/** OledUpdate
**
** Parameters:
**     none
**
** Return Value:

```

```

**      none
**
**  Errors:
**      none
**
**  Description:
**      Update the OLED display with the contents of the memory buffer
*/

void
OledUpdate()
{
    int      ipag;
    int      icol;
    BYTE *   pb;

    pb = rgbOledBmp;

    for (ipag = 0; ipag < cpagOledMax; ipag++) {

        PORTClearBits(prtDataCmd, bitDataCmd);

        /* Set the page address
        */
        Spi2PutByte(0x22);      //Set page command
        Spi2PutByte(ipag);     //page number

        /* Start at the left column
        */
        Spi2PutByte(0x00);     //set low nybble of column
        Spi2PutByte(0x10);     //set high nybble of column

        PORTSetBits(prtDataCmd, bitDataCmd);

        /* Copy this memory page of display data.
        */
        OledPutBuffer(ccolOledMax, pb);
        pb += ccolOledMax;

    }

}

```

Low-Level SPI Functions

The following functions are used to write data to the display panel using the SPI controller.

```

/* ----- */
/** OledPutBuffer
**
**  Parameters:
**      cb      - number of bytes to send/receive
**      rgbTx   - pointer to the buffer to send
**
**  Return Value:
**      none
**
**  Errors:
**      none
**
**  Description:
**      Send the bytes specified in rgbTx to the slave.
*/

```

```

void
OledPutBuffer(int cb, BYTE * rgbTx)
{
    int      ib;
    BYTE bTmp;

    /* Write/Read the data
    */
    for (ib = 0; ib < cb; ib++) {
        /* Wait for transmitter to be ready
        */
        while (SPI2STATbits.SPITBE == 0);

        /* Write the next transmit byte.
        */
        SPI2BUF = *rgbTx++;

        /* Wait for receive byte.
        */
        while (SPI2STATbits.SPIRBF == 0);
        bTmp = SPI2BUF;
    }
}

/* ----- */
/** Spi2PutByte
**
** Parameters:
**     bVal      - byte value to write
**
** Return Value:
**     Returns byte read
**
** Errors:
**     none
**
** Description:
**     Write/Read a byte on SPI port 2
**
*/

BYTE
Spi2PutByte(BYTE bVal)
{
    BYTE bRx;

    /* Wait for transmitter to be ready
    */
    while (SPI2STATbits.SPITBE == 0);

    /* Write the next transmit byte.
    */
    SPI2BUF = bVal;

    /* Wait for receive byte.
    */
    while (SPI2STATbits.SPIRBF == 0);

    /* Put the received byte in the buffer.
    */
    bRx = SPI2BUF;

    return bRx;
}

```