PmodOLED Library Reference Manual

Revision: February 16, 2012

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Introduction

The Digilent PmodOLED contains a 128x32 pixel monochrome organic LED (OLED) graphics display. Digilent provides a driver library for this display. This document provides an overview of the operation of this driver library and describes the functions that make up its programming interface.

Overview

The OLED display uses a Solomon SSD1306 display controller. This controller is capable of working with displays up to 128x64 pixel resolution, although the display on the PmodOLED has a 128x32 pixel resolution. The display is monochrome. Each individual pixel can only be on (illuminated) or off (not illuminated). For more information about the hardware interface of the PmodOLED, refer to the PmodOLED Reference Manual available for download from the Digilent web site. (www.digilentinc.com).

The display is a serial device that is accessed via an SPI interface. It is, however, a write-only device. It is not possible to read back status or display data from the display. For this reason, this driver library maintains a frame buffer in memory. All drawing and read-back operations are performed using this memory frame buffer. The display is updated with new image data by copying the frame buffer to the display.

Note: The included demo project sets the configuration bits assuming that the PmodOLED is plugged into JB of the Cerebot MX4cK. The project was created using MPLAB v.8.83

Library Operation

Library Interface

The header file PmodOLED.h defines the interfaces to the OLED driver.

Display Initialization

The OLED display has a power on/power off sequence that should be followed. Before making calls to any other library functions. The OledInit() function must be called. This function initializes the PIC32 resources used by the library, and then turns on power to the display and initializes it.

Character Mode Operation

The display hardware is a graphical display. The library, however, supports a character oriented display where the display is treated as if it were a character display. The character mode functions use character row and character column numbers for cursor position.

Graphic Mode Operation

The graphics functions support reading and writing pixels, drawing lines and rectangles and other graphical operations. The graphics functions also support drawing characters at any position on the display.

The character ‘mode’ and graphics ‘mode’ are not really modes of operation of the library, they are simply sets of functions. Calls to character functions and graphics functions can be interleaved without restriction.

Drawing Mode

The library supports four drawing modes for the graphic drawing operations. These modes specify the operation to be performed between the current drawing color and the current state of the pixel to determine the final pixel value. The following modes are supported:

modeSet	- set the pixel to the current drawing color
modeOr	- OR the current pixel value with the current drawing color
modeAnd	- AND the current pixel value with the current drawing color
modeXor	- XOR (exclusive or) the current pixel color with the current drawing color.

Display Organization

The display memory is organized such that the first byte (byte 0) corresponds to a vertical column of eight pixels at the left side of the display, with the least significant bit the uppermost pixel in the column. The next byte corresponds to the next column of eight pixels to the right of the first. This continues across the display to byte 127, which is the rightmost column of eight pixels. Byte 128 corresponds to the next column of eight pixels at the left of the display.

OLED Library Functions

Display Management Functions

void begin(void)

Parameters:
none

This function initializes the driver, turns on power to the display and initializes it. This function must be called before any other functions in the library are called.

void end(void)

Parameters:
none

This function turns off power to the display and then releases the PIC32 SPI controller and the pins used for the OLED interface.

void displayOn(void)

Parameters:
none

This enables the un-blanks the display.

void displayOffvoid)

Parameters:
none

This blanks the display.

void clear(void)

Parameters:
none

Clears the memory frame buffer and then updates the display thus clearing it

void clearBuffer(void)

Parameters:
none

Clear the display memory buffer without updating the display.

void updateDisplay(void)

Parameters:
none

Update the OLED display with the contents of the memory buffer.

Character Functions

The following functions treat the display as if it were a character mode display. They use the character column number and row number for the cursor position. The row and column numbers are 0 based. Therefore the column numbers will be in the range 0 – (N-1) and the row numbers will be in the range 0 – (M-1), where N and M are the number of columns and the number of rows.

void setCursor(int xch, int ych)

Parameters:

xch	- horizontal character position (column)
ych	- vertical character position (row)

Set the character cursor position to the specified location. If either the specified X or Y location is off the display, it is clamped to be on the display.

void getCursor(int *pxcy, int *pych)

Parameters:

pxch	- pointer to variable to receive horizontal position
pych	- pointer to variable to receive vertical position

Return the current cursor position.

int defineUserChar(char ch, BYTE *pbDef)

Parameters:

ch	- character code to define
pdDef	- definition for character

Return Value:
True if successful, false if not

Give a definition for the glyph for the specified user character code. User definable character codes are in the range 0x00 – 0x1F. If the code specified by ch is outside this range, the function returns false;

void setCharUpdate(int f)

Parameters:
f - enable/disable automatic update

Set the character update mode. This determines whether or not the display is automatically updated after a character or string is drawn. A zero value turns off automatic updating and a non-zero value turns automatic updating on.

int getCharUpdate(void)

Parameters:
none

Return Value
Returns current character update mode

Return the current character update mode

void putChar(char ch)

Parameters:
ch -character to write to display

Write the specified character to the display at the current cursor position and advance the cursor.

void putString(char *sz)

Parameters:
sz -pointer to the null terminated string

Write the specified null terminated character string to the display and advance the cursor.

Graphic Functions

void setDrawColor(Byte clr)

Parameters:
clr -drawing color to set

Set the foreground color used for pixel draw operations

void setDrawMode(int mod)

Parameters:
mod - drawing mode select

Set the specified mode as the current drawing mode. The following values are the drawing mode options.

modeSet	Set the pixel to the current drawing color
modeOr	OR the pixel value with the current drawing color
modeAnd	AND the pixel value with the current drawing color
modeXor	XOR the pixel value with the current drawing color

int getDrawMode(void)

Parameters:
none

Return Value:
Returns current drawing mode

Return the current drawing mode

BYTE* getStdPattern(int ipat)

Parameters:
ipat -index to standard fill pattern (0-7)

Return Value:
Returns a pointer to the standard fill pattern

Return a pointer to the byte array for the specified standard fill pattern. The following patterns are available (each pattern fills an 8x8 pixel square)

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x00
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // 0x01
0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, // 0x02
0x11, 0x44, 0x00, 0x11, 0x44, 0x00, 0x11, 0x44, // 0x03
0x92, 0x45, 0x24, 0x92, 0x45, 0x24, 0x92, 0x45, // 0x04
0x49, 0x92, 0x24, 0x49, 0x92, 0x24, 0x49, 0x92, // 0x05
0x22, 0x11, 0x22, 0x00, 0x22, 0x11, 0x22, 0x00, // 0x06
0x11, 0x22, 0x11, 0x00, 0x11, 0x22, 0x11, 0x00  // 0x07
```

Standard fill pattern 0, is all pixels off (black). Fill pattern 1 is all pixels on (white).

void setFillPattern(BYTE *pbPat)

Parameters:
pbPat - pointer to the fill pattern

Set a pointer to the current fill pattern to use. A fill pattern is an array of 8 bytes. This pattern will be used by the drawFillRect function.

void moveTo(int xco, int yco)

Parameters:
- xco - x coordinate
- yco - y coordinate

Set the current graphics drawing position.

void getPos(int *pxco, int *pyco)

Parameters:
- pxco - variable to receive x coordinate
- pyco - variable to receive y coordinate

Return the current graphics drawing positions

void drawPixel(void)

Parameters:
- none

Set the pixel at the current drawing location to the specified value

BYTE getPixel(void)

Parameters:
- none

Return Value:
- Returns pixel value at current drawing location

Return the value of the pixel at the current drawing location

void drawLine(int xco, int yco)

Parameters:
- xco - x coordinate of the other corner
- yco - y coordinate of the other corner

Draw a line from the current position to the specified position. The specified position becomes the new current position.

void drawRect(int xco, int yco)

Parameters:

xco	- x coordinates of other corner
yco	- y coordinates of other corner

Draw a rectangle bounded by the current location and the specified location. The current location is not modified.

void drawFillRect(int xco, int yco)

Parameters:

xco	- x coordinates of other corner
yco	- y coordinates of other corner

Fill a rectangle bounded by the current location and the specified location. This does not draw an outline around the rectangle. The current position is not modified.

void getBmp(int dxco, int dyco, BYTE *pbBmp)

Parameters:

dxco	- width of bitmap
dyco	- height of bitmap
pbBits	- pointer to the bitmap bits

This routine will get the bits corresponding to the rectangle implied by the current location and the specified width and height. The buffer specified by pbBits must be large enough to hold the resulting bytes. The required buffer size in bytes is: dxco * ((dyco/8)+1)

void putBmp(int dxco, int dyco, BYTE *pbBmp)

Parameters:

dxco	- width of bitmap
dyco	- height of bitmap
pbBits	- pointer to the bitmap bits

This routine will draw the specified bitmap into the display buffer at the current location

void drawChar(char ch)

Parameters:

ch	- character to write to display

Draw the specified character to the display at the current cursor position and advance the cursor

void drawstring(char *sz)

Parameters

sz -pointer to the null terminated string

Write the specified null terminated character string to the display and advance the cursor