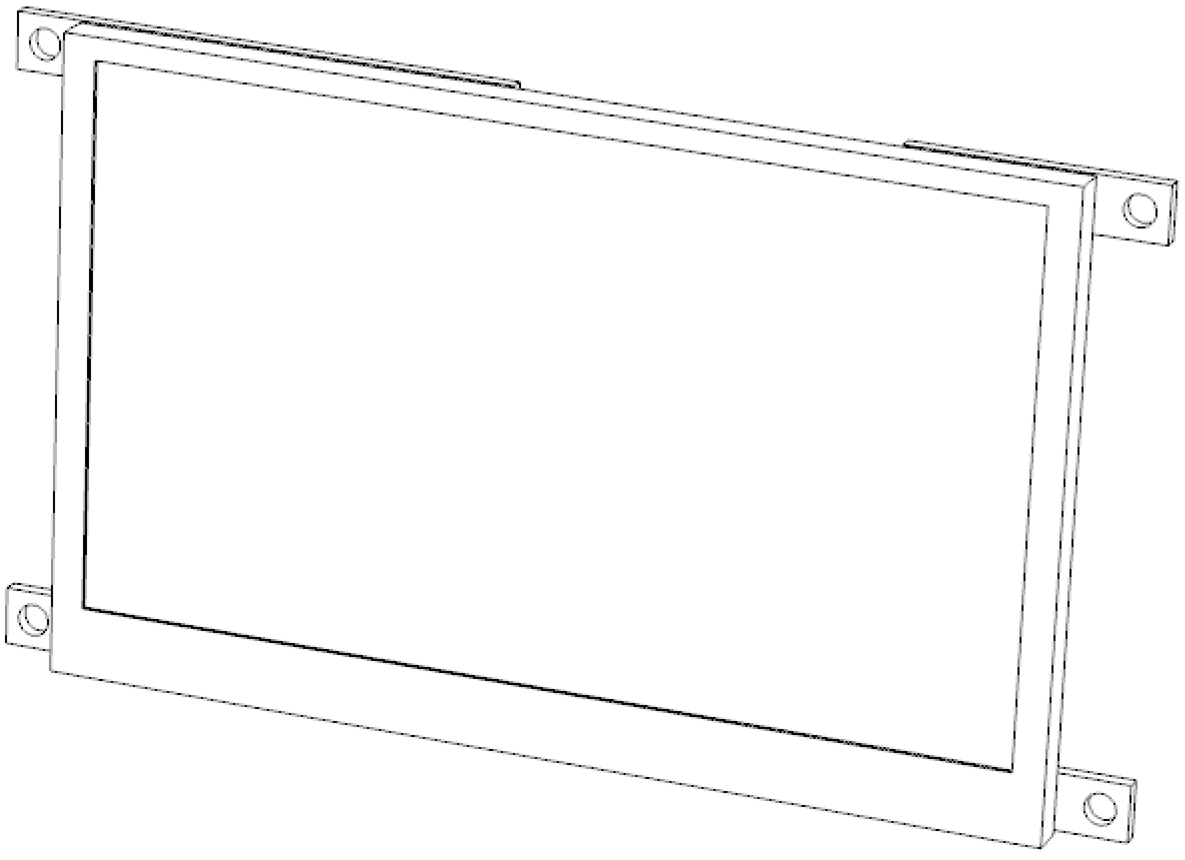
USBD480-LQ043 User Guide



Preliminary

Version 0.1

www.lcdinfo.com

Table of Contents

1. Introduction

Overview

The USBD480-LQ043 is a controller board designed to be used with a 4.3” TFT LCD panel from Sharp. The controller board provides framebuffer functionality and offers a Hi-Speed USB 2.0 interface for connecting to the host system.

USBD480 provides a quick and easy to way to add a display to systems that have USB host ports. The host system can be an embedded device or a PC computer. For best performance a Hi-Speed (480 Mbps) USB host capability is recommended.

It is also possible to use the display as an input user interface with the optional touchscreen.

Features

- 4.3" 480x272 pixel resolution color TFT LCD
- Optional touchscreen
- Hi-Speed USB interface 480 Mbps
- 16 bit colors (RGB565)
- 8 MB framebuffer memory (available for double buffering etc.)
- Display can be updated at 50 frames per second over USB [1)]
- USB powered
- User configurable startup image
- Custom versions of the controller board possible for different display panels

Note:

1. If the application requires it would be possible to increase the speed at the expense of some increased power consumption

2. Physical Characteristics

Item	Specifications	Unit
Display resolution	480 x 272	dot
Screen size	10.9 (4.3" type) diagonal	cm
Active area	95.04(H) x 53.856(V)	mm
Pixel pitch	0.198 x 0.198	mm
Unit outline dimensions [1)]	120(W) x 67.2(H) x ___(D)	mm
Weight	TBD	g
Backlight type	LED	

Note:

1. Includes mounting tabs

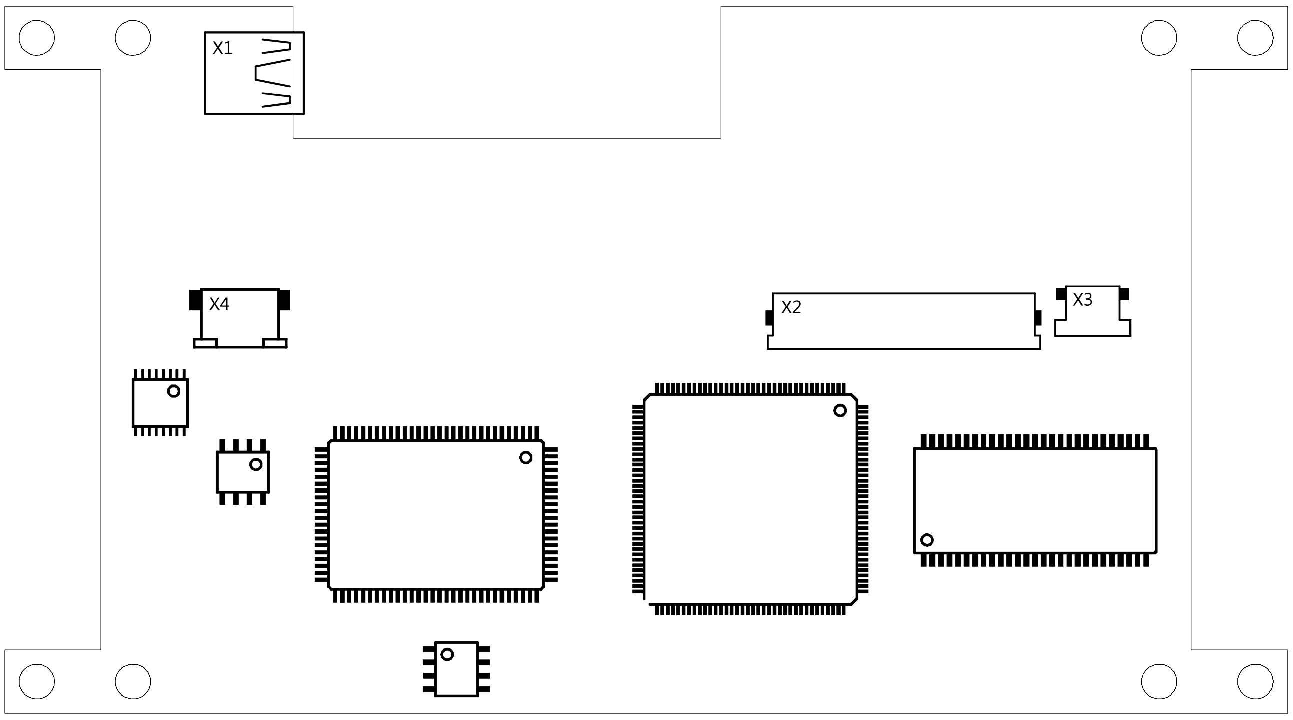
3. Electrical Characteristics

Parameter	Symbol	Min.	Typ.	Max.	Unit
Supply Voltage [1)]	V_{CC}	4.8	5.0	5.2	V
Current Consumption without backlight [2)]			110		mA
Backlight Current Consumption full brightness			110		mA

Note:

1. Sharp LCD panel specification
2. Including LCD panel, USB data transfer in progress

4. USBD480-LQ043 board



For more details see dimensional drawings in Appendix

5. Pin descriptions

USB connector - X1

The USB connector is used as the communication interface and the power supply for the display module. The connector used is a mini USB B type connector. When connecting the display some care should be taken to use a good quality USB cable. Some cables and bus powered hubs might cause too much voltage drop which is seen as inferior image quality on the display.

Pin	Name	Function
1	VBUS	+5 VDC
2	D-	Data -
3	D+	Data +
4	NC	Unconnected
5	GND	Ground

Display connector - X2

Backlight connector - X3

Touchscreen connector - X4

4 way 1 mm pitch FFC/FPC connector with bottom contacts.

Pin	Name	Function
1	X1	
2	Y2	
3	X2	
4	Y1	

6. Getting started

LCD panel mounting

The USBD480-LQ043 controller board is designed to be mechanically compatible with the Sharp LQ043T3DX02 and compatible LCD panels. These LCD panels do not provide built-in mounting tabs but the USBD480-LQ043 mounting holes can be used to mount the assembly that includes both the USBD480-LQ043 controller board and the LCD panel. The LCD panel can be folded against the bottom side of the controller board and attached using suitable adhesive. Even if not attaching the LCD panel to the controller board some insulation should be used between the LCD panel metal frame and the circuit board to prevent a short circuit between them.

Connecting the display module

The USB connector is used as the communication interface and as the power supply for the display module. The connector used is the common mini USB B type connector. When connecting the display some care should be taken to use a good quality USB cable. Some cables and bus powered hubs might cause too much voltage drop which is seen as inferior image quality on the display.

7. Programming interface

Overview

USBD480 is a composite USB device with two interfaces. First interface is for the main display functionality and the second interface is for touchscreen input.

Framebuffer memory addressing

The controller has 8 MB of memory available for storing the image data. The memory size of 4096 x 256 x 16 bits makes for a total of 1048576 different addresses, 16 bits of data per address. One frame requires 480 x 272 x 2 = 261120 bytes of memory. This means that it is possible to store 32 separate full frames of image data to the controller memory. Aligning the start address for each frame to the 512 byte boundaries results in a bit more efficient memory bandwidth use.

Double buffering can be implemented using two alternating frames in the memory. The actual address change for the SET_FRAME_START_ADDRESS command is synchronized with VSYNC in the controller hardware.

It is also possible to implement hardware accelerated vertical scrolling by first filling the controller memory with video data and then just changing the frame start address to scroll the image.

Pixel format

The USBD480 controller uses the RGB565 pixel format. One pixel consists of 16 bits of data – 5 bits red, 6 bits green and 5 bits blue. This results in 65536 different colors.

R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Windows API DLL

For Windows there exists a DLL interface that provides an API that can be used for accessing the display. When using the DLL it is not necessary to know the low level details of accessing the USB device. The DLL also automatically detects and uses the available USB driver installed in the system. The currently available options for the USB driver are Libusb or the UMDF USB driver. Two alternative drivers make it possible to support a wider range of Windows versions and with the DLL automatically detecting the driver the user doesn't need to know which one is installed in the system.

The API interface is described in a separate document.

Linux framebuffer driver

When Linux is used one possible way to access the display is to use the USBD480 framebuffer kernel driver. This driver makes the USBD480 display appear as a normal Linux framebuffer device (/dev/fb) which is used by many existing libraries for accessing display hardware. Qt Embedded and SDL included for example. The driver also connects the touchscreen to the Linux input API.

Direct USB device access

Direct access to the USB device is also possible if it is preferred instead of the existing library or if the platform used doesn't have existing support libraries.

Other interfaces

There are also other libraries and drivers being developed that support using the USBD480 controller. See the website for more details about what is currently available. Also feel free to give feedback if your application would need something different from the currently available offerings.

8. Direct USB command interface

Overview

The display is controlled by sending USB vendor requests to the control endpoint. For image data transfer bulk endpoints are used.

Vendor requests

SET_ADDRESS	0xC0	
SET_FRAME_START_ADDRESS	0xC4	
GET_DEVICE_DETAILS	0x80	
SET_BRIGHTNESS	0x81	
SET_CONFIG_VALUE	0x82	
GET_CONFIG_VALUE	0x83	
GET_SAVED_CONFIG_VALUE	0x86	
SAVE_CONFIGURATION	0x84	
SET_TOUCH_MODE	0xE2	

SET_ADDRESS 0xC0

This request sets the write address to the framebuffer.

bRequest	wValue	wIndex	wLength	Data
SET_ADDRESS	Bits 15 to 0 of the address	Bits 21 to 16 of the address	0	None

SET_FRAME_START_ADDRESS 0xC4

This request sets the start address of the visible frame.

bRequest	wValue	wIndex	wLength	Data
SET_FRAME_START_ADDRESS	Bits 15 to 0 of the address	Bits 21 to 16 of the address	0	None

GET_DEVICE_DETAILS 0x80

This request gets the device information structure.

bRequest	wValue	wIndex	wLength	Data
GET_DEVICE_DETAILS	0	0	0	Device information

The device returns 64 bytes and of those the following are currently defined

Bytes	Data
0-19	Device name
20-21	Display width in pixels
22-23	Display height in pixels
24-25	Version
26-36	Serial number

SET_BRIGHTNESS 0x81

This request sets the display backlight brightness.

bRequest	wValue	wIndex	wLength	Data
SET_BRIGHTNESS	Brightness	0	0	None

Brightness accepts values from 0 to 255.
Internally uses 128 PWM levels.

SET_CONFIG_VALUE 0x82

This request sets a configuration parameter value.

bRequest	wValue	wIndex	wLength	Data
SET_CONFIG_VALUE	Parameter ID	0	1	Value

Configuration parameters

Parameter	ID	Notes
TOUCH_MODE	2	Default value is 0 (touch disabled)
TOUCH_DEBOUNCE_VALUE	3	How many samples are required for state change
TOUCH_SKIP_SAMPLES	4	How many samples to skip when pen goes down
TOUCH_PRESSURE_LIMIT_LO	5	Filters reported samples based on pressure. Low limit
TOUCH_PRESSURE_LIMIT_HI	6	Filters reported samples based on pressure. High limit
BACKLIGHT_BRIGHTNESS	20	Backlight brightness
USB_ENUMERATION_MODE	22	Enumerate touch interface as HID or custom

For more information about the different parameters see Configuration parameters section starting from page 14.

GET_CONFIG_VALUE 0x83

This request gets a configuration parameter value.

bRequest	wValue	wIndex	wLength	Data
GET_CONFIG_VALUE	Parameter ID	0	1	Value

Gets the current configuration parameter value being used.

GET_SAVED_CONFIG_VALUE 0x86

This request gets a configuration parameter saved as a default value.

bRequest	wValue	wIndex	wLength	Data
GET_SAVED_CONFIG_VALUE	Parameter ID	0	1	Value

Gets the configuration parameter value from non-volatile memory that is saved as the default value to be loaded on power up.

SAVE_CONFIGURATION 0x84

This request saves the configuration parameters.

bRequest	wValue	wIndex	wLength	Data
SAVE_CONFIGURATION	0x8877	0	0	None

Saves the current configuration parameters to non-volatile memory so that they are automatically loaded on the next power up.

SET_TOUCH_MODE 0xE2

This request sets the touchscreen operating mode.

bRequest	wValue	wIndex	wLength	Data
SET_TOUCH_MODE	Mode	0	0	None

Sets the touchscreen operation mode.

	Mode	Notes
0	Touchscreen disabled	(default)
1	Filtered output	Reports sent only when pen is down
2	Raw output	Raw touchscreen samples without any filtering
3	Scaling to screen coordinates	Approximate and not too accurate for now
4	Simple filter	Reports sent only when pen is down

TOUCH_DEBOUNCE_VALUE applies to modes 1 and 4

TOUCH_SKIP_SAMPLES applies to mode 1

TOUCH_PRESSURE_LIMIT applies to modes 1, 3 and 4

Configuration parameters

Configuration parameters can be changed with the SET_CONFIG_VALUE request and the currently used parameter can be read with the GET_CONFIG_VALUE request. The currently used parameters can be saved as new defaults to be loaded on power up using the SAVE_CONFIGURATION request.

TOUCH_MODE 2

Default mode is 0 (Touchscreen disabled). It is recommended for each software to set the mode it expects to use during initialisation and not assume any specific touchscreen sampling mode to be enabled. This setting is saved with SAVE_CONFIGURATION request so the user may have set any mode to be the power on setting.

	Mode	**Notes**
0	Touchscreen disabled	(default)
1	Filtered output	Reports sent only when pen is down
2	Raw output	Raw touchscreen samples without any filtering
3	Scaling to screen coordinates	Approximate and not too accurate for now
4	Simple filter	Reports sent only when pen is down

Equivalent functionality to using SET_CONFIG_VALUE request with TOUCH_MODE parameter can be achieved using the SET_TOUCH_MODE request.

TOUCH_DEBOUNCE_VALUE 3

0 – 255

This parameter defines how many samples are required until a change of pen state is recognised.

The parameter applies to touch modes 1 and 5.

TOUCH_SKIP_SAMPLES 4

0 – 255

This parameter defines how many of the initial samples are discarded when a pen down event is recognised. This can be useful to discard the very first samples when the pressure is low and the position sampled is not yet accurate.

The parameter applies to touch mode 1.

TOUCH_PRESSURE_LIMIT_LO 5

0 – 255

Default 30

This parameter can be used to fine tune the touchscreen pressure filtering.

TOUCH_PRESSURE_LIMIT_HI 6

0 – 255

Default 120

This parameter can be used to fine tune the touchscreen pressure filtering.

BACKLIGHT_BRIGHTNESS 20

0 – 255

Default 255

This parameter sets the backlight brightness.

USB_ENUMERATION_MODE 22

0 or 1

Default 0

This parameter defines if the touchscreen device enumerates as a HID device or a vendor specific device.

0 = enumerate as a HID device

1 = enumerate as a vendor specific device

See chapter 9 for more information about the touchscreen interface.

Bulk endpoint

For actual image data bulk out endpoint 2 is used. Data size in bytes needs to be dividable with 512. Image data is in RGB565 format.

Embedded control header

Alternative mode of operation for controlling the addresses is to embed them into the data stream. If embedded control header is used then the SET_ADDRESS and SET_FRAME_START_ADDRESS control requests don't need to be used and everything is controlled by just the data stream to the bulk endpoint.

The embedded control header needs to be the very last bytes of the buffer to work properly. The embedded control header has two address fields, FRAME_START_ADDRESS and ADDRESS.

FRAME_START_ADDRESS is the address of the framebuffer from where the visible frame is set to start after processing this control header. This address change is synchronised with the display refresh VSYNC.

ADDRESS sets the new framebuffer write address. Next data written to the display goes here.

The control header is 128 bits long and is made of a 64 bit identifier in the beginning and then of four 16 bit words of data.

Control header structure

	DATA - 16 bits	**Notes**
1	0xB3D9	Identifier bits 15 to 0
2	0xB494	Identifier bits 31 to 16
3	0xE81E	Identifier bits 47 to 32
4	0xC2A9	Identifier bits 63 to 48
5	FRAME_START_ADDRESS bits 15..0	
6	FRAME_START_ADDRESS bits 21..16	
7	ADDRESS bits 15..0	
8	ADDRESS bits 21..16	

Example buffer contents

```
data[0] // begin pixel data for frame
.
.
.
.
.
.
data[261119] // last pixel data
data[261120] // just fill don't care bytes here so that we get a full 512 byte
block
.
.
.
.
.
.
data[261615] // last fill byte, next add the control data to the end of the
last 512 byte block of the buffer
data[261616] = 0xB3D9 & 0xff; // embedded header starts with a 64-bit
identifier
data[261617] = 0xB3D9 >> 8;
data[261618] = 0xB494 & 0xff;
data[261619] = 0xB494 >> 8;
data[261620] = 0xE81E & 0xff;
data[261621] = 0xE81E >> 8;
data[261622] = 0xC2A9 & 0xff;
data[261623] = 0xC2A9 >> 8;
data[261624] = 0x0000 & 0xff; // FRAME_START_ADDRESS bits 15..0 - lo byte
data[261625] = 0x0000 >> 8;   // FRAME_START_ADDRESS bits 15..0 - hi byte
data[261626] = 0x0000 & 0xff; // FRAME_START_ADDRESS bits 21..16 - lo byte
data[261627] = 0x0000 >> 8;   // FRAME_START_ADDRESS bits 21..16 - hi byte
data[261628] = 0x0000 & 0xff; // ADDRESS bits 15..0 - lo byte
data[261629] = 0x0000 >> 8;   // ADDRESS bits 15..0 - hi byte
data[261630] = 0x0000 & 0xff; // ADDRESS bits 21..16 - lo byte
data[261631] = 0x0000 >> 8;   // ADDRESS bits 21..16 - hi byte
```

When working with the framebuffer addresses it is good to remember that one address points to a 16 bit word. So two bytes of image data per address.

Below is an example how double buffering could be implemented using the embedded header:

<u>For the first frame embedded control data</u>

framestart = 0

address = 130816

<u>For the second frame embedded control data</u>

framestart = 130816

address = 0

and then just alternate these for the following frames.

9. Touchscreen interface

Second interface of the composite USB device.

The default for the touchscreen interface is to enumerate as a HID device but it can be also set to enumerate as a vendor specific device. This can be helpful in some situations to prevent the system HID driver automatically claiming the interface and allowing to use custom drivers more easily.

Endpoint 1 is used as an interrupt endpoint to receive the touch reports. One report is 16 bytes in size.

16 byte touch report

Byte(s)	Content	Notes
[0][1]	X	12 bit x sample LSB first
[2][3]	Y	12 bit y sample
[4][5]	Z1	12 bit z1 sample
[6][7]	Z2	12 bit z2 sample
[8]	pen	0=pen down, 1=pen up
[9]	pressure	firmware calculated pressure value
[10]-[15]	reserved	

10. Appendix A – Dimensional drawings

See:

http://www.lcdinfo.com/usbd480/documentation/USBD480-LQ043_dimension_drawing.pdf