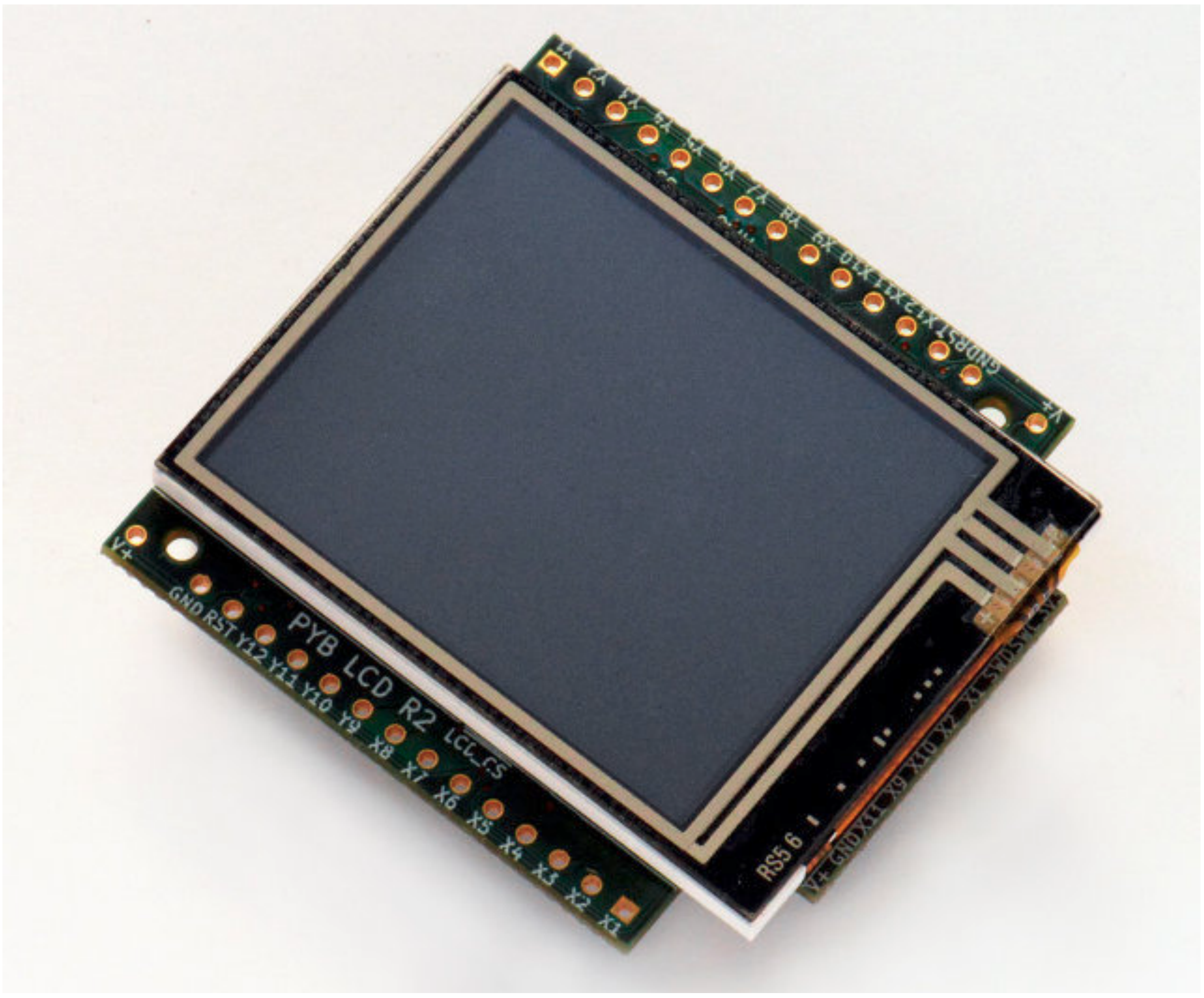
`lcd160cr` — control of LCD160CR display

This module provides control of the MicroPython LCD160CR display.



Further resources are available via the following links:

- LCD160CRv1.0 reference manual (100KiB PDF)
- LCD160CRv1.0 schematics (1.6MiB PDF)

class LCD160CR

The LCD160CR class provides an interface to the display. Create an instance of this class and use its methods to draw to the LCD and get the status of the touch panel.

For example:

```
import lcd160cr

lcd = lcd160cr.LCD160CR('X')
lcd.set_orient(lcd160cr.PORTRAIT)
lcd.set_pos(0, 0)
lcd.set_text_color(lcd.rgb(255, 0, 0), lcd.rgb(0, 0, 0))
lcd.set_font(1)
lcd.write('Hello MicroPython!')
print('touch:', lcd.get_touch())
```

Constructors

class **`lcd160cr.LCD160CR`**(*connect=None*, ***, *pwr=None*, *i2c=None*, *spi=None*, *i2c_addr=98*)

Construct an LCD160CR object. The parameters are:

- *connect* is a string specifying the physical connection of the LCD display to the board; valid values are "X", "Y", "XY", "YX". Use "X" when the display is connected to a pyboard in the X-skin position, and "Y" when connected in the Y-skin position. "XY" and "YX" are used when the display is connected to the right or left side of the pyboard, respectively.
- *pwr* is a Pin object connected to the LCD's power/enabled pin.
- *i2c* is an I2C object connected to the LCD's I2C interface.
- *spi* is an SPI object connected to the LCD's SPI interface.
- *i2c_addr* is the I2C address of the display.

One must specify either a valid *connect* or all of *pwr*, *i2c* and *spi*. If a valid *connect* is given then any of *pwr*, *i2c* or *spi* which are not passed as parameters (i.e. they are `None`) will be created based on the value of *connect*. This allows to override the default interface to the display if needed.

The default values are:

- "X" is for the X-skin and uses: `pwr=Pin("X4")`, `i2c=I2C("X")`, `spi=SPI("X")`
- "Y" is for the Y-skin and uses: `pwr=Pin("Y4")`, `i2c=I2C("Y")`, `spi=SPI("Y")`
- "XY" is for the right-side and uses: `pwr=Pin("X4")`, `i2c=I2C("Y")`, `spi=SPI("X")`
- "YX" is for the left-side and uses: `pwr=Pin("Y4")`, `i2c=I2C("X")`, `spi=SPI("Y")`

See this image for how the display can be connected to the pyboard.

Static methods

static **`LCD160CR.rgb`**(*r*, *g*, *b*)

Return a 16-bit integer representing the given rgb color values. The 16-bit value can be used to set the font color (see `LCD160CR.set_text_color()`) pen color (see `LCD160CR.set_pen()`) and draw individual pixels.

`LCD160CR.clip_line(data, w, h):`

Clip the given line data. This is for internal use.

Instance members

The following instance members are publicly accessible.

`LCD160CR.w`

`LCD160CR.h`

The width and height of the display, respectively, in pixels. These members are updated when calling `LCD160CR.set_orient()` and should be considered read-only.

Setup commands

`LCD160CR.set_power`(*on*)

Turn the display on or off, depending on the given value of *on*: 0 or `False` will turn the display off, and 1 or `True` will turn it on.

`LCD160CR.set_orient`(*orient*)

Set the orientation of the display. The *orient* parameter can be one of `PORTRAIT`, `LANDSCAPE`, `PORTRAIT_UPSIDEDOWN`, `LANDSCAPE_UPSIDEDOWN`.

`LCD160CR.set_brightness`(*value*)

Set the brightness of the display, between 0 and 31.

`LCD160CR.set_i2c_addr`(*addr*)

Set the I2C address of the display. The *addr* value must have the lower 2 bits cleared.

`LCD160CR.set_uart_baudrate`(*baudrate*)

Set the baudrate of the UART interface.

`LCD160CR.set_startup_deco`(*value*)

Set the start-up decoration of the display. The *value* parameter can be a logical or of `STARTUP_DECO_NONE`, `STARTUP_DECO_MLOGO`, `STARTUP_DECO_INFO`.

`LCD160CR.save_to_flash`()

Save the following parameters to flash so they persist on restart and power up: initial decoration, orientation, brightness, UART baud rate, I2C address.

Pixel access methods

The following methods manipulate individual pixels on the display.

`LCD160CR.set_pixel`(*x, y, c*)

Set the specified pixel to the given color. The color should be a 16-bit integer and can be created by `LCD160CR.rgb()`.

`LCD160CR.get_pixel`(*x, y*)

Get the 16-bit value of the specified pixel.

`LCD160CR.get_line`(*x, y, buf*)

Low-level method to get a line of pixels into the given buffer. To read *n* pixels *buf* should be *2*n+1* bytes in length. The first byte is a dummy byte and should be ignored, and subsequent bytes represent the pixels in the line starting at coordinate *(x, y)*.

`LCD160CR.screen_dump`(*buf, x=0, y=0, w=None, h=None*)

Dump the contents of the screen to the given buffer. The parameters *x* and *y* specify the starting coordinate, and *w* and *h* the size of the region. If *w* or *h* are `None` then they will take on their maximum values, set by the size of the screen minus the given *x* and *y* values. *buf* should be large enough to hold `2*w*h` bytes. If it's smaller then only the initial horizontal lines will be stored.

`LCD160CR.screen_load`(*buf*)

Load the entire screen from the given buffer.

Drawing text

To draw text one sets the position, color and font, and then uses `LCD160CR.write` to draw the text.

`LCD160CR.set_pos`(*x, y*)

Set the position for text output using `LCD160CR.write()`. The position is the upper-left corner of the text.

`LCD160CR.set_text_color`(*fg, bg*)

Set the foreground and background color of the text.

`LCD160CR.set_font`(*font, scale=0, bold=0, trans=0, scroll=0*)

Set the font for the text. Subsequent calls to `write` will use the newly configured font. The parameters are:

- *font* is the font family to use. Valid values are 0, 1, 2, 3.
- *scale* is a scaling value for each character pixel, where the pixels are drawn as a square with side length equal to *scale + 1*. The value can be between 0 and 63.
- *bold* controls the number of pixels to overdraw each character pixel, making a bold effect. The lower 2 bits of *bold* are the number of pixels to overdraw in the horizontal direction, and the next 2 bits are for the vertical direction. For example, a *bold* value of 5 will overdraw 1 pixel in both the horizontal and vertical directions.
- *trans* can be either 0 or 1 and if set to 1 the characters will be drawn with a transparent background.
- *scroll* can be either 0 or 1 and if set to 1 the display will do a soft scroll if the text moves to the next line.

`LCD160CR.write`(*s*)

Write text to the display, using the current position, color and font. As text is written the position is automatically incremented. The display supports basic VT100 control codes such as newline and backspace.

Drawing primitive shapes

Primitive drawing commands use a foreground and background color set by the `set_pen` method.

`LCD160CR.set_pen`(*line, fill*)

Set the line and fill color for primitive shapes.

`LCD160CR.erase`()

Erase the entire display to the pen fill color.

`LCD160CR.dot`(*x, y*)

Draw a single pixel at the given location using the pen line color.

`LCD160CR.rect`(*x, y, w, h*)

`LCD160CR.rect_outline`(*x, y, w, h*)

`LCD160CR.rect_interior`(*x, y, w, h*)

Draw a rectangle at the given location and size using the pen line color for the outline, and the pen fill color for the interior. The `rect` method draws the outline and interior, while the other methods just draw one or the other.

`LCD160CR.line`(*x1, y1, x2, y2*)

Draw a line between the given coordinates using the pen line color.

LCD160CR.dot_no_clip(*x, y*)

LCD160CR.rect_no_clip(*x, y, w, h*)

LCD160CR.rect_outline_no_clip(*x, y, w, h*)

LCD160CR.rect_interior_no_clip(*x, y, w, h*)

LCD160CR.line_no_clip(*x1, y1, x2, y2*)

These methods are as above but don't do any clipping on the input coordinates. They are faster than the clipping versions and can be used when you know that the coordinates are within the display.

LCD160CR.poly_dot(*data*)

Draw a sequence of dots using the pen line color. The *data* should be a buffer of bytes, with each successive pair of bytes corresponding to coordinate pairs (x, y).

LCD160CR.poly_line(*data*)

Similar to `LCD160CR.poly_dot()` but draws lines between the dots.

Touch screen methods

LCD160CR.touch_config(*calib=False, save=False, irq=None*)

Configure the touch panel:

- If *calib* is `True` then the call will trigger a touch calibration of the resistive touch sensor. This requires the user to touch various parts of the screen.
- If *save* is `True` then the touch parameters will be saved to NVRAM to persist across reset/power up.
- If *irq* is `True` then the display will be configured to pull the IRQ line low when a touch force is detected. If *irq* is `False` then this feature is disabled. If *irq* is `None` (the default value) then no change is made to this setting.

LCD160CR.is_touched()

Returns a boolean: `True` if there is currently a touch force on the screen, `False` otherwise.

LCD160CR.get_touch()

Returns a 3-tuple of: *(active, x, y)*. If there is currently a touch force on the screen then *active* is 1, otherwise it is 0. The *x* and *y* values indicate the position of the current or most recent touch.

Advanced commands

`LCD160CR.set_spi_win`(*x, y, w, h*)

Set the window that SPI data is written to.

`LCD160CR.fast_spi`(*flush=True*)

Ready the display to accept RGB pixel data on the SPI bus, resetting the location of the first byte to go to the top-left corner of the window set by `LCD160CR.set_spi_win()`. The method returns an SPI object which can be used to write the pixel data.

Pixels should be sent as 16-bit RGB values in the 5-6-5 format. The destination counter will increase as data is sent, and data can be sent in arbitrary sized chunks. Once the destination counter reaches the end of the window specified by `LCD160CR.set_spi_win()` it will wrap around to the top-left corner of that window.

`LCD160CR.show_framebuf`(*buf*)

Show the given buffer on the display. *buf* should be an array of bytes containing the 16-bit RGB values for the pixels, and they will be written to the area specified by `LCD160CR.set_spi_win()`, starting from the top-left corner.

The framebuf module can be used to construct frame buffers and provides drawing primitives. Using a frame buffer will improve performance of animations when compared to drawing directly to the screen.

`LCD160CR.set_scroll`(*on*)

Turn scrolling on or off. This controls globally whether any window regions will scroll.

`LCD160CR.set_scroll_win`(*win, x=-1, y=0, w=0, h=0, vec=0, pat=0, fill=0x07e0, color=0*)

Configure a window region for scrolling:

- *win* is the window id to configure. There are 0..7 standard windows for general purpose use. Window 8 is the text scroll window (the ticker).
- *x*, *y*, *w*, *h* specify the location of the window in the display.
- *vec* specifies the direction and speed of scroll: it is a 16-bit value of the form `0bF.ddSSSSSSSSSSSS`. *dd* is 0, 1, 2, 3 for +x, +y, -x, -y scrolling. *F* sets the speed format, with 0 meaning that the window is shifted *S % 256* pixel every frame, and 1 meaning that the window is shifted 1 pixel every *S* frames.
- *pat* is a 16-bit pattern mask for the background.
- *fill* is the fill color.
- *color* is the extra color, either of the text or pattern foreground.

`LCD160CR.set_scroll_win_param`(*win, param, value*)

Set a single parameter of a scrolling window region:

- *win* is the window id, 0..8.
- *param* is the parameter number to configure, 0..7, and corresponds to the parameters in the `set_scroll_win` method.
- *value* is the value to set.

`LCD160CR.set_scroll_buf`(*s*)

Set the string for scrolling in window 8. The parameter *s* must be a string with length 32 or less.

`LCD160CR.jpeg`(*buf*)

Display a JPEG. *buf* should contain the entire JPEG data. JPEG data should not include EXIF information. The following encodings are supported: Baseline DCT, Huffman coding, 8 bits per sample, 3 color components, YCbCr4:2:2. The origin of the JPEG is set by `LCD160CR.set_pos()`.

`LCD160CR.jpeg_start`(*total_len*)

`LCD160CR.jpeg_data`(*buf*)

Display a JPEG with the data split across multiple buffers. There must be a single call to `jpeg_start` to begin with, specifying the total number of bytes in the JPEG. Then this number of bytes must be transferred to the display using one or more calls to the `jpeg_data` command.

`LCD160CR.feed_wdt`()

The first call to this method will start the display's internal watchdog timer. Subsequent calls will feed the watchdog. The timeout is roughly 30 seconds.

`LCD160CR.reset`()

Reset the display.

Constants

`lcd160cr.PORTRAIT` **`lcd160cr.LANDSCAPE`** **`lcd160cr.PORTRAIT_UPSIDEDOWN`**

`lcd160cr.LANDSCAPE_UPSIDEDOWN`

Orientations of the display, used by `LCD160CR.set_orient()`.

`lcd160cr.STARTUP_DECO_NONE` **`lcd160cr.STARTUP_DECO_MLOGO`**

`lcd160cr.STARTUP_DECO_INFO`

Types of start-up decoration, can be OR'ed together, used by `LCD160CR.set_startup_deco()`.