Instruction how to connect LM6020 using micropython

**Table of Contents**

[Preparation 2](#__RefHeading___Toc1391_520338010)

[The LM6020 2](#__RefHeading___Toc1393_520338010)

[Step 1 3](#__RefHeading___Toc1395_520338010)

[Step 2 4](#__RefHeading___Toc1397_520338010)

[Step 3 4](#__RefHeading___Toc1280_4163719918)

[Step 4 4](#__RefHeading___Toc1282_4163719918)

[Annex 1 Demonstration program 5](#__RefHeading___Toc1407_520338010)

[Annex 2 Framebuffer Commands 7](#__RefHeading___Toc1399_520338010)

[Drawing primitive shapes 7](#__RefHeading___Toc1292_4163719918)

[Drawing text 7](#__RefHeading___Toc1401_520338010)

[Other methods 8](#__RefHeading___Toc1403_520338010)

[Constants 8](#__RefHeading___Toc1405_520338010)

# Preparation

In this example a Raspberry Pi Pico with MicroPython is used, but with some slight alterations it should work on other MCU’s like an ESP32.

It is assumed the RP Pico is connected to the PC through USB with the Thonny IDE application installed on the PC and initiated through the Thonny IDE. Other IDE’s may work but have not been tested.

# The LM6020

Figure 1: The connected display with the RP Pico

The LM6020 is a simple black and white graphics LCD display with according to the specs a layout of 128x 64 dots. The complete [datasheet](https://datasheetspdf.com/pdf-file/1397601/TOPWAY/LM6020CCW/1) [[1]](#footnote-2).with more information can be found on the internetnet.

The LM6020 is fitted with a ST7565 display driver which has a serial interface that can be used through a SPI interface on the MCU.

Setting up a working system is therefor rather easy and can be done in some simple steps. All you need further is an 8 pin adapter board for 1mm ffc.

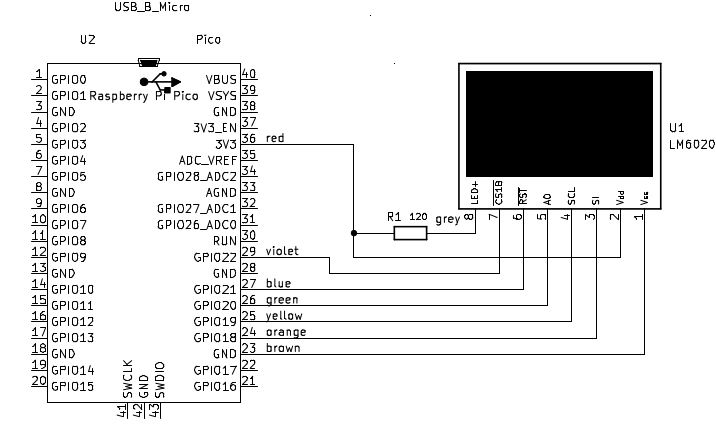
# Step 1

Connect the flat cable to the adapter board, which has the following layout and connect this board in turn to a RP Pico. Note that the numbering on this board is confusing and the correct numbering is on other side of the board!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LM6020C Pin Description** | | | | **Raspberry PI Pico** | |
| **PIN NO** | **PIN Name** | **I/O** | **Descriptions** | **PIN Name** | **PIN NO** |
| 1 | VSS | Supply | Negative power supply,0V | GND | 23 |
| 2 | VDD | Supply | Positive power supply | 3V3(OUT) | 36 |
| 3 | SI | I/O | Serial data input | GP19/SPI0\_SCK | 24 |
| 4 | SCL | I/O | Serial clock input | GP18/SPI0\_TX | 25 |
| 5 | A0 | Input | Register Select  A0 = H, Transferring the Display Data  A0 = L, Transferring the Control Data | GP20 | 26 |
| 6 | /RES | Input | Reset signal  /RES = L, Initialization is executed  /RES = H, Normal running | GP21 | 27 |
| 7 | /CS1 | Input | Chip Select  /CS1=L, enable access to the LCD module  /CS1=H, disable access to the LCD module | GP22 | 29 |
| 8 | BLA | Power | Backlight Positive Supply | 3V3 via resistor | 36 |

This is the schematics

# Step 2

Figure 2: How to wire up the display to the RP Pico 

Download the driver from [GITHUB[[2]](#footnote-3)](https://github.com/nquest/micropython-st7565/blob/master/st7565.py), extract and save it.

# Step 3

It is recommended to use Thonny as your IDE connect this to your RP Pico.

On the bottom line at the right there is a remark showing which environment it is using. It should read Micropython (Raspberry Pi Pico), if not click on this line and select the correct environment.

Click on the open icon. It should give you an option ‘Where to open from’ with two choices ‘This Computer’ or ‘ Raspberry Pi Pico’. Use this computer and get the downloaded st7565.py software driver. Save a copy on the RP Pico.

Open a new file and copy the given demonstation program from Annex 1 and paste it into Thonny,save this also on the RP Pico as demo.py. It is recommended to save a copy on your computer.

# Step 4

Copy the MicroPython demonstration program to Thonny and save it on the RP Pico as Display\_test.py or whatever name you can think of and run it.

This demonstration program also demonstrates some framebuffer commands!

The available commands for drawing in the FrameBuffer can be found on the official micropython site, but for your convienance are also described in the following Annex 1.

# Annex 1 Framebuffer Commands

## Drawing primitive shapes

The following methods draw shapes onto the FrameBuffer.

|  |  |
| --- | --- |
| FrameBuffer function  eg display.fill(c) | Description |
| f.fill(c)[¶](https://docs.micropython.org/en/latest/library/framebuf.html" \l "framebuf.FrameBuffer.fill) | Fill the entire FrameBuffer with the specified color. |
| f.pixel(x, y[, c]) | If c is not given, get the color value of the specified pixel. If c is given, set the specified pixel to the given color. |
| f.hline(x, y, w, c) | Draw a line from a set of coordinates using the given color and a thickness of 1 pixel. The [line](https://docs.micropython.org/en/latest/library/framebuf.html" \l "framebuf.FrameBuffer.line) method draws the line up to a second set of coordinates whereas the [hline](https://docs.micropython.org/en/latest/library/framebuf.html" \l "framebuf.FrameBuffer.hline) and [vline](https://docs.micropython.org/en/latest/library/framebuf.html" \l "framebuf.FrameBuffer.vline) methods draw horizontal and vertical lines respectively up to a given length. |
| f.vline(x, y, h, c)[¶](https://docs.micropython.org/en/latest/library/framebuf.html" \l "framebuf.FrameBuffer.vline) |
| f.line(x1, y1, x2, y2, c) |
| f.rect(x, y, w, h, c[, f]) | Draw a rectangle at the given location, size and color.  The optional f parameter can be set to True to fill the rectangle. Otherwise just a one pixel outline is drawn. |
| f.ellipse(x, y, xr, yr, c[, f, m]) | Draw an ellipse at the given location. Radii xr and yr define the geometry; equal values cause a circle to be drawn. The c parameter defines the color.  The optional f parameter can be set to True to fill the ellipse. Otherwise just a one pixel outline is drawn.  The optional m parameter enables drawing to be restricted to certain quadrants of the ellipse. The LS four bits determine which quadrants are to be drawn, with bit 0 specifying Q1, b1 Q2, b2 Q3 and b3 Q4. Quadrants are numbered counterclockwise with Q1 being top right. |
| f.poly(x, y, coords, c[, f]) | Given a list of coordinates, draw an arbitrary (convex or concave) closed polygon at the given x, y location using the given color.  The coords must be specified as a [array](https://docs.micropython.org/en/latest/library/array.html" \l "module-array) of integers, e.g. array('h', [x0, y0, x1, y1, ... xn, yn]).  The optional f parameter can be set to True to fill the polygon. Otherwise just a one pixel outline is drawn. |

## Drawing text

|  |  |
| --- | --- |
| FrameBuffer function  eg display.fill(c) | Description |
| f.text(s, x, y[, c]) | Write text to the FrameBuffer using the the coordinates as the upper-left corner of the text. The color of the text can be defined by the optional argument but is otherwise a default value of 1. All characters have dimensions of 8x8 pixels and there is currently no way to change the font. |

## Other methods

|  |  |
| --- | --- |
| FrameBuffer function (f)  eg display.fill(c) | Description |
| f.scroll(xstep, ystep) | Shift the contents of the FrameBuffer by the given vector. This may leave a footprint of the previous colors in the FrameBuffer. |
| f.blit(fbuf, x, y, key=-1, palette=None) | Draw another FrameBuffer on top of the current one at the given coordinates. If key is specified then it should be a color integer and the corresponding color will be considered transparent: all pixels with that color value will not be drawn. (If the palette is specified then the key is compared to the value from palette, not to the value directly from fbuf.)  The palette argument enables blitting between FrameBuffers with differing formats. Typical usage is to render a monochrome or grayscale glyph/icon to a color display. The palette is a FrameBuffer instance whose format is that of the current FrameBuffer. The palette height is one pixel and its pixel width is the number of colors in the source FrameBuffer. The palette for an N-bit source needs 2\*\*N pixels; the palette for a monochrome source would have 2 pixels representing background and foreground colors. The application assigns a color to each pixel in the palette. The color of the current pixel will be that of that palette pixel whose x position is the color of the corresponding source pixel. |

## Constants

|  |  |
| --- | --- |
| FrameBuffer function (f)  eg display.fill(c) | Description |
| f.MONO\_VLSB | Monochrome (1-bit) color format This defines a mapping where the bits in a byte are vertically mapped with bit 0 being nearest the top of the screen. Consequently each byte occupies 8 vertical pixels. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered at locations starting at the leftmost edge, 8 pixels lower. |
| f.MONO\_HLSB | Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 7 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower. |
| f.MONO\_HMSB | Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 0 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower. |
| f.RGB565 | Red Green Blue (16-bit, 5+6+5) color format |
| f.GS2\_HMSB | Grayscale (2-bit) color format |
| f.GS4\_HMSB | Grayscale (4-bit) color format |
| f..GS8 | Grayscale (8-bit) color format |

1. <https://datasheetspdf.com/pdf-file/1397601/TOPWAY/LM6020CCW/1> [↑](#footnote-ref-2)
2. https://github.com/nquest/micropython-st7565/blob/master/st7565.py [↑](#footnote-ref-3)