

How to use MatrixPortal M4

What is MatrixPortal M4

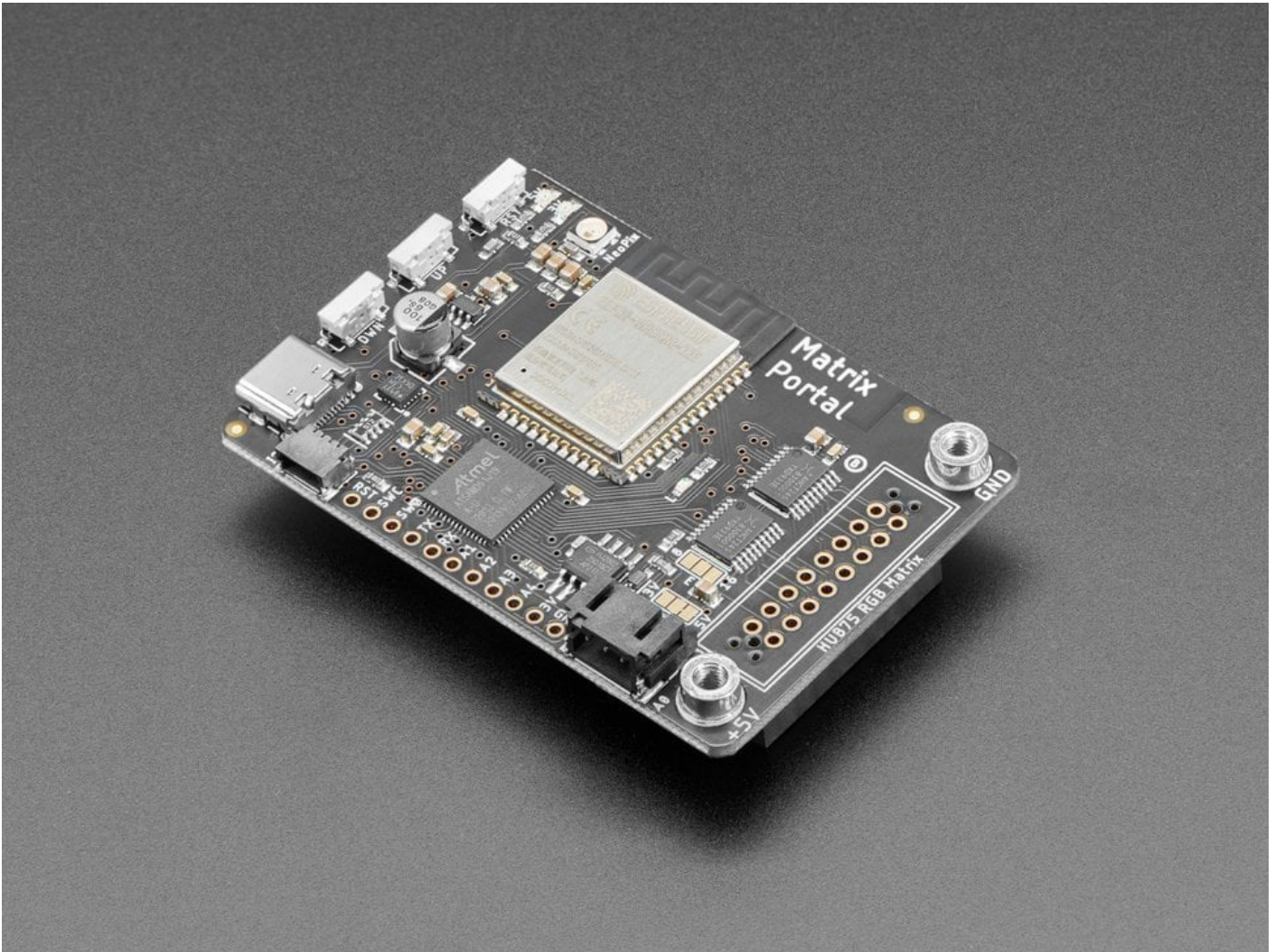
The MatrixPortal M4 is a development board created by Adafruit designed to control RGB LED matrices. It is equipped with an ATSAMD51 microcontroller (Cortex M4) and has built-in Wi-Fi support thanks to the ESP32 coprocessor.

Here are some key features of the MatrixPortal M4:

1. Microcontroller: ATSAMD51, Cortex M4 processor running at 120 MHz.
2. Coprocessor: ESP32 handles Wi-Fi and network communication.
3. Memory: 512KB of RAM, 8MB of QSPI flash storage.
4. Matrix Control: Dedicated connectors for RGB LED matrices (HUB75 interface), allowing direct control without needing additional hardware.
5. Power: Can be powered via USB-C or through the matrix's power supply.
6. Programming: Supports programming via CircuitPython and Arduino IDE.
7. Wi-Fi Connectivity: Useful for IoT projects that require network connectivity, such as weather stations, stock tickers, or message boards.

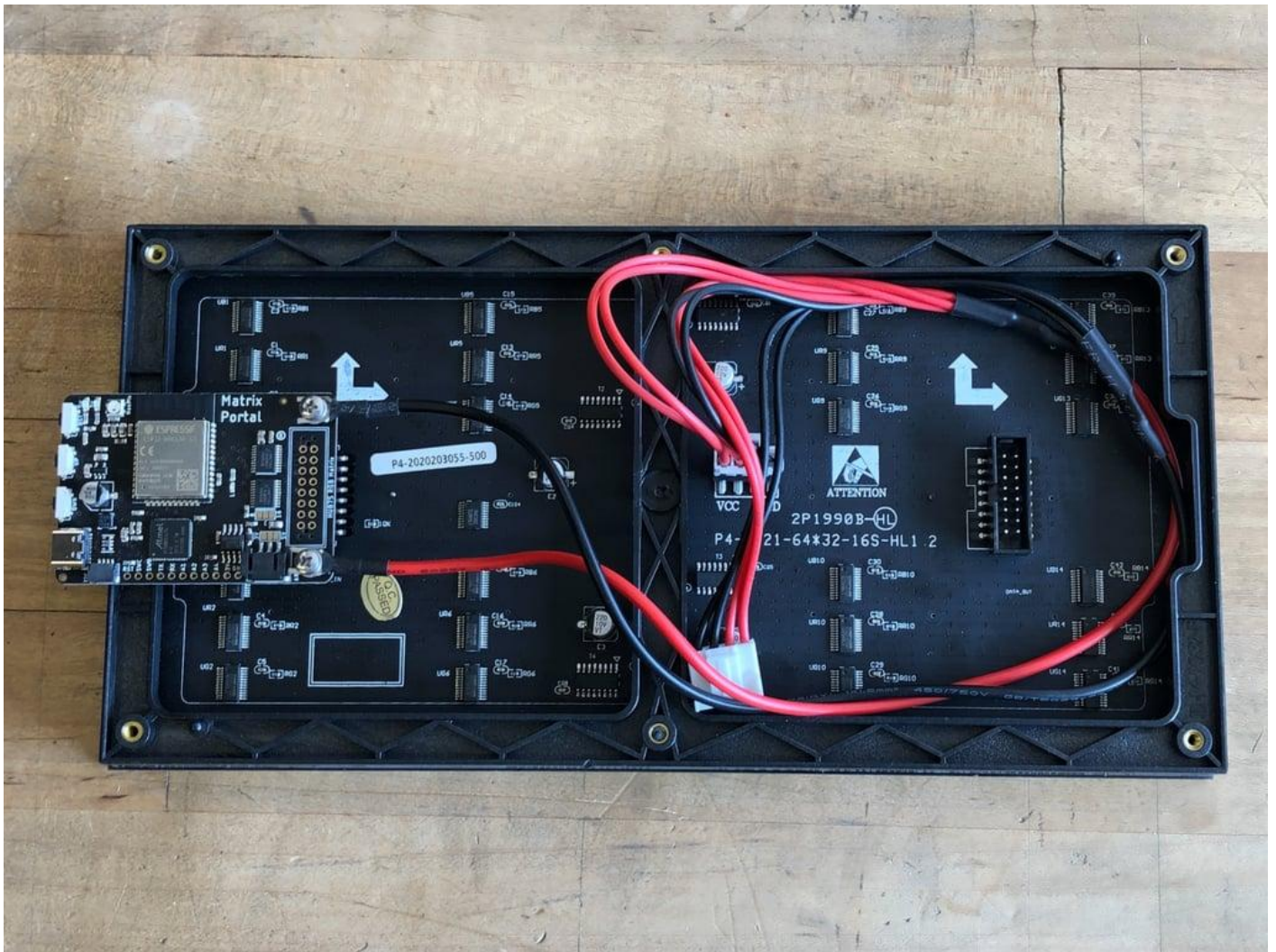
It's great for building projects that involve large, colourful LED displays, such as scrolling text, interactive dashboards, or internet-connected signs. In this tutorial, we will use **CircuitPython** to program a **64X32 matrix** as CircuitPython libraries make it easier to display images, animations, text, etc than Arduino IDE.

You can read more about this component [here](#).

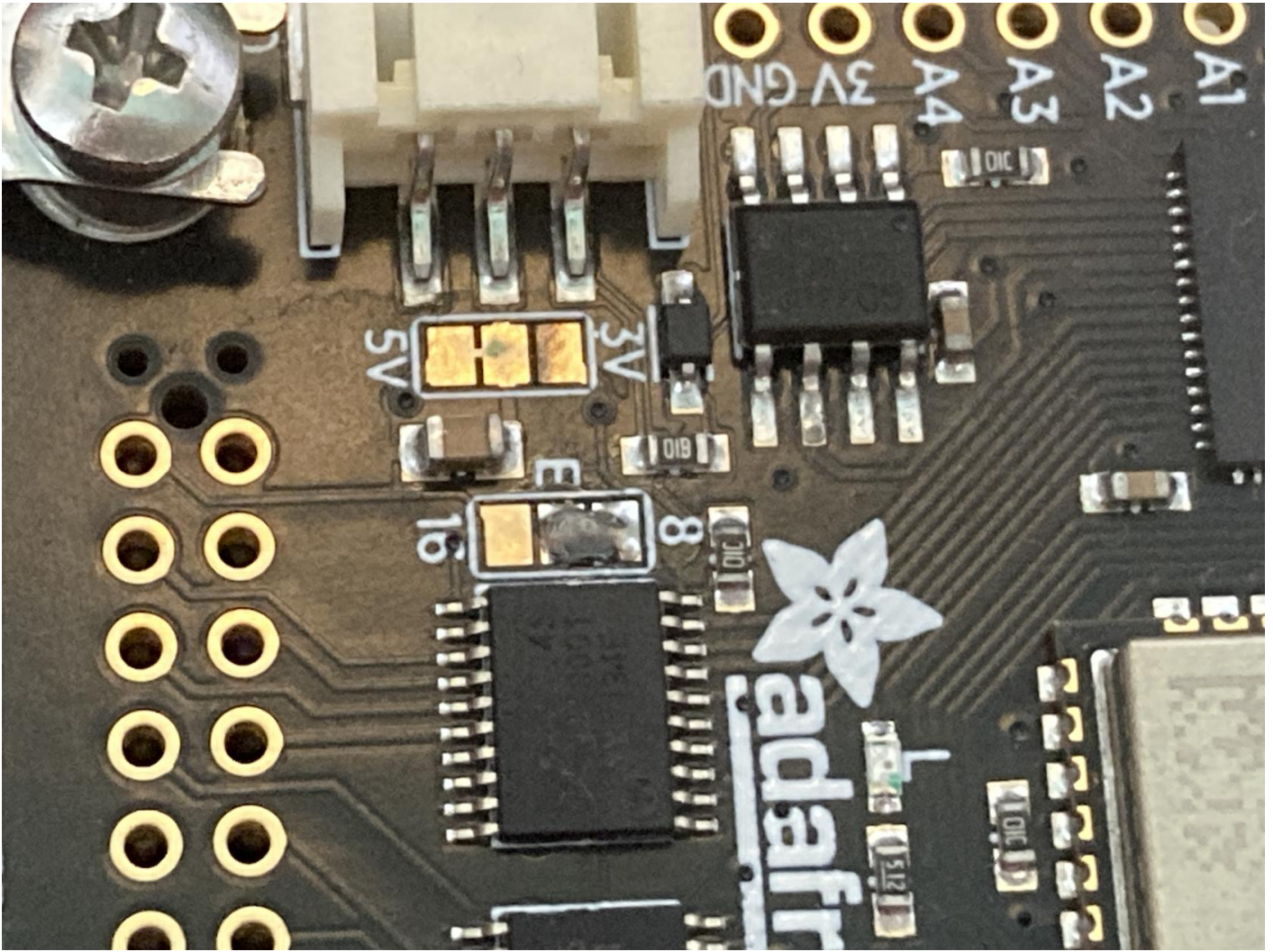


Wiring

There is no wiring needed for basic setup, it's literally plug and play. Please refer to this [**page**](#) to see how to set it up.



Using a 64X64 Matrix



You can read more about Address E Line Jumper [here](#).

Install CircuitPython

CircuitPython and libraries versions

In this tutorial, we are using CircuitPython 9.0.5 (11/10/2024), all libraries and code used are compatible with this version. Please double-check the latest version of CircuitPython you have installed and use updated and compatible libraries.

CircuitPython is an open-source programming language designed for microcontrollers. It's a beginner-friendly version of Python developed by Adafruit, optimized for hardware projects like controlling sensors, displays, and other electronics.

1. Download CircuitPython

Download the latest version for MatrixPortal UF2 file for your board [here](#).

2. Put the Board into Bootloader Mode

To install CircuitPython, you need to place the board into bootloader mode.

- Press the reset button on your board twice quickly.
- The board's LED will change to a different colour (usually pulsing red or green)
- A new USB drive will appear on your computer named something like "**BOOT**".

3. Copy the CircuitPython UF2 File

- Drag and drop the downloaded **UF2 file** onto the new drive.
- The board will reboot, and a new drive named **CIRCUITPY** will appear. This drive is where you'll place your Python code and libraries.

Libraries

- Download the libraries bundle [here](#), choose the version you need.
- Copy the libraries you need to the `lib` folder of the **CIRCUITPY** drive.

Common libraries you need:

1. adafruit_matrixportal
2. adafruit_debouncer.mpy
3. adafruit_portalbase
4. adafruit_esp32spi
5. neopixel.mpy
6. adafruit_bus_device
7. adafruit_requests.mpy
8. adafruit_fakerequests.mpy
9. adafruit_io
10. adafruit_bitmap_font
11. adafruit_display_text
12. adafruit_lis3dh.mpy
13. adafruit_minimqtt
14. adafruit_ticks.py
15. adafruit_rgb_display
16. adafruit_imageload
17. adafruit_display_shapes

Code - Scrolling Text

```
# SPDX-FileCopyrightText: 2020 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# This example implements a simple two line scroller using
# Adafruit_CircuitPython_Display_Text. Each line has its own color
```

```
# and it is possible to modify the example to use other fonts and non-standard
# characters.
```

```
import adafruit_display_text.label
import board
import displayio
import framebufferio
import rgbmatrix
import terminalio

from adafruit_bitmap_font import bitmap_font
from displayio import Bitmap

# If there was a display before (protomatter, LCD, or E-paper), release it so
# we can create ours
displayio.release_displays()

matrix = rgbmatrix.RGBMatrix(
    width=64, bit_depth=6,
    rgb_pins=[
        board.MTX_R1,
        board.MTX_G1,
        board.MTX_B1,
        board.MTX_R2,
        board.MTX_G2,
        board.MTX_B2
    ],
    addr_pins=[
        board.MTX_ADDRA,
        board.MTX_ADDRB,
        board.MTX_ADDRD,
        board.MTX_ADDRD
    ],
    clock_pin=board.MTX_CLK,
    latch_pin=board.MTX_LAT,
    output_enable_pin=board.MTX_OE
)

display = framebufferio.FramebufferDisplay(matrix, auto_refresh=False)

keycolour = 0X7BFF4A

# Create two lines of text to scroll. Besides changing the text, you can also
# customize the color and font (using Adafruit_CircuitPython_Bitmap_Font).
# To keep this demo simple, we just used the built-in font.
# The Y coordinates of the two lines were chosen so that they looked good
```

but if you change the font you might find that other values work better.

```
line1 = adafruit_display_text.label.Label(
    terminalio.FONT,
    color=keycolour, #white
    text="This is Creative Technology Hub")
line1.x = display.width
line1.y = 5
```

```
line2 = adafruit_display_text.label.Label(
    terminalio.FONT,
    color=0x000000,
    background_color=keycolour,
    background_tight = True,
    text="Hello Hello Hello Hello Hello Hellooooooooo")
line2.x = display.width
line2.y = 15
```

```
line3 = adafruit_display_text.label.Label(
    terminalio.FONT,
    color=keycolour,

    text="Stop peeking come in")
line3.x = display.width
line3.y = 26
```

Put each line of text into a Group, then show that group.

```
g = displayio.Group()
g.append(line1)
g.append(line2)
g.append(line3)
display.root_group = g
```

This function will scoot one label a pixel to the left and send it back to
the far right if it's gone all the way off screen. This goes in a function
because we'll do exactly the same thing with line1 and line2 below.

```
def scroll(line):
    line.x = line.x - 1
    line_width = line.bounding_box[2]
    if line.x < -line_width:
        line.x = display.width
```



```
# This function scrolls lines backwards. Try switching which function is
# called for line2 below!
def reverse_scroll(line):
    line.x = line.x + 1
    line_width = line.bounding_box[2]
    if line.x >= display.width:
        line.x = -line_width

# You can add more effects in this loop. For instance, maybe you want to set the
# color of each label to a different value.
while True:
    scroll(line1)
    #scroll(line2)
    scroll(line3)
    reverse_scroll(line2)
    display.refresh(minimum_frames_per_second=1)
```

Revision #3

Created 11 October 2024 10:05:39 by Joanne Leung

Updated 16 December 2024 10:30:07 by Joanne Leung