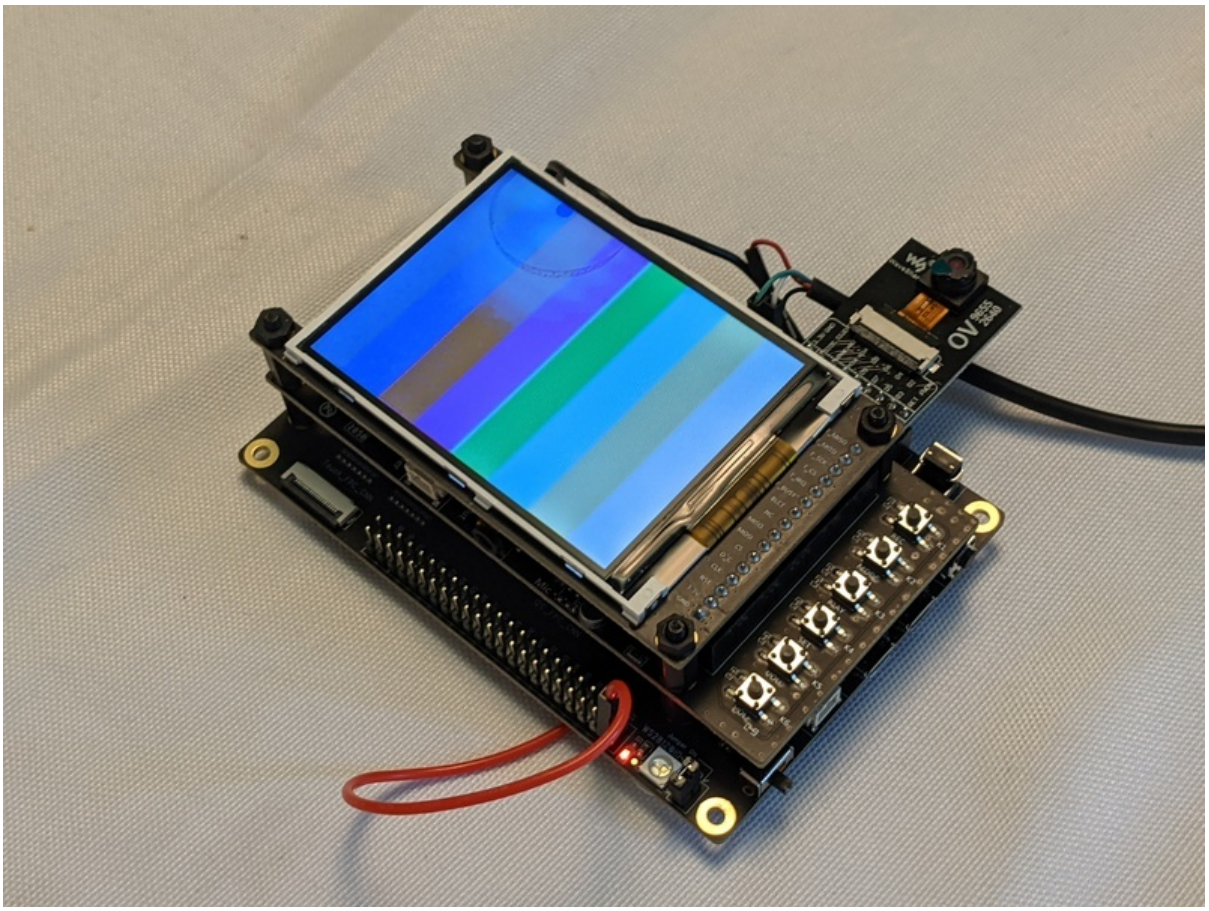
Capturing Camera Images with CircuitPython

Created by Jeff Epler



https://learn.adafruit.com/capturing-camera-images-with-circuitpython

Last updated on 2024-03-08 03:58:18 PM EST

Table of Contents

Overview 5

- Parts
- Parts

Cameras and Pinouts 9

- Pins

Working with espcamera 11

- Configure reserved PSRAM

Example: LCD Viewfinder 12

- How it works

Example: Webcam with Adafruit IO 16

- How it works

Example: QR Code Scanner 20

- QR Code Scanner
- Creating QR Codes

Documentation: espcamera 23

Install TinyUF2 on Espressif Kaluga 23

- Method 1: WebSerial ESPTool / esptool
- Step 1. Download the tinyuf2 combined.bin file here
- Step 2. Place your board in bootloader mode
- Step 3 Option A. Use the Web Serial ESPTool to upload
- Step 3. Option B. Use esptool.py to upload (for advanced users)
- Step 4. Reset the board
- Method 2: Flash an Arduino Sketch
- Arduino IDE Setup
- Load the Blink Sketch

Espressif Kaluga Setup 34

- Kaluga 1.3 with OV2640 and ili9341 display (CircuitPython 7)
- Kaluga 1.3 with OV2640 and st7789 display (CircuitPython 7)
- Kaluga 1.3 with OV7670 and ili9341 display
- Adapting to other ESP32-S2 boards

Raspberry Pi Pico Wiring 43

- Wiring
- For the OV2640 camera
- For the OV7670 camera
- Adapting to other RP2040 boards

Grand Central M4 Wiring 52

- Adapting to other SAM D5x/E5x boards

Working with Image Data 55

- RGB Data
- YUV Data

• Test modes

Capturing JPEG data (OV2640) 61

• Code
• Parts

OV2640 Webcam with Adafruit IO 65

• Set up the IO Feed
• Set up the IO Dashboard
• Secrets File Setup for Adafruit IO
• Upload the code

Working with BMP format data 69

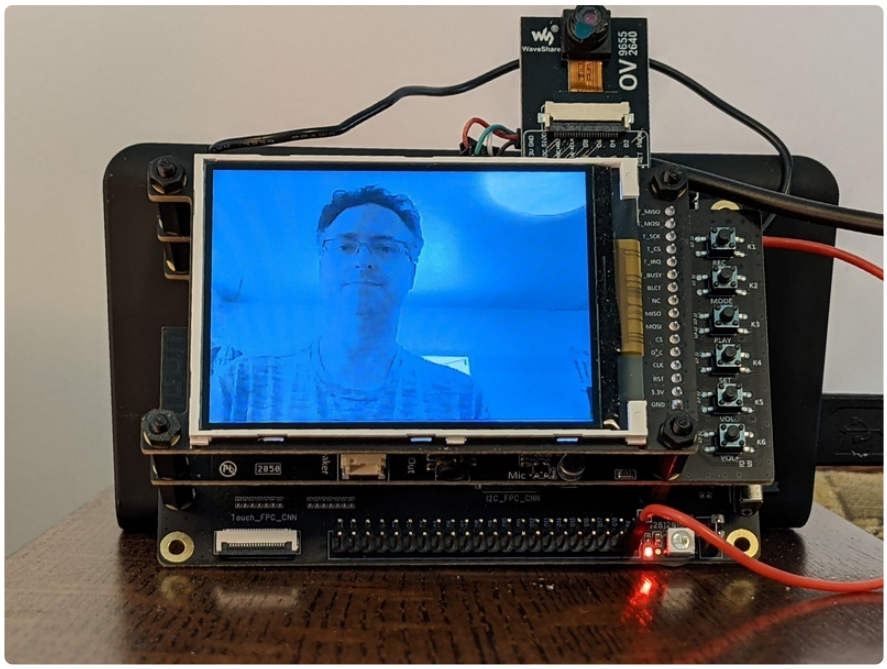
Coding Image Filters in C 73

Documentation: adafruit_ov2640 77

Documentation: adafruit_ov7670 77

Documentation: adafruit_ov5640 77

Overview



CircuitPython 7 adds support for capturing images from "parallel cameras" on select boards, and libraries are available to configure the popular OV7670, OV5640, and OV2640 cameras.

CircuitPython 8 adds a new library exclusively for Espressif ESP32 microcontrollers that supports a wider range of cameras, though the guide author has only tested with the OV5640 and OV2640 cameras.

While not up to standards we're used to from a current smartphone or laptop, it's nicely balanced to the capabilities of recent 32-bit microcontrollers.

CircuitPython on the Grand Central M4 is prone to locking up when the camera function is used. The Arduino implementation is much more reliable on that board. We recommend using ESP32-S2, ESP32-S3 or RP2040 instead.

The Arduino library for OV7670 cameras on the Grand Central M4 has its own dedicated guide (https://adafru.it/TAG).

Parts

Items needed

- Compatible microcontroller board with appropriate pins exposed:
    - any CircuitPython-compatible ESP32-S2 or S3 with PSRAM, including Kaluga and ESP32-S3-EYE
    - any RP2040 supported by CircuitPython including Raspberry Pi Pico
    - Adafruit's Grand Central M4 (not recommended)
- SPI TFT module
- compatible parallel camera module — **the board pinout must match exactly**
- Two 2.2k resistors, if pull-up resistors are necessary
- Soldering iron and supplies
- Appropriate USB data + power cable
- Prototyping supplies such as breadboards and jumper wires

The Espressif ESP32-S3 Eye development kit has all you need (including an OV2640 camera module and 240x240 LCD), and no soldering is required, so it's probably the best way to get started!

It is not possible to access the CIRCUITPY drive or the REPL using the USB Micro B connections on the Kaluga board so you will also need a USB Plug Breakout cable (see below). Also, you must have v1.3 of the Kaluga kit, v1.2 has some incompatible wiring that makes it unusable in this project!

OV7670, OV5640, and OV2640 camera modules with the 18 pin, 2-row header can be found on Amazon, eBay, etc. Make sure the pinout matches the camera shown above, as occasionally there are incompatible variants. The cameras are sometimes sold in **sets** which is a good idea, as they're easily damaged with the wrong voltage or rough handling, especially if you need to modify it for use with the Grand Central M4.

Other sensor models (such as OV3660) exist, but they require different initialization code and cannot currently be used.

When using `esp32_camera`, a wider range of camera boards are supported, but note that the guide author has only tried OV2640 and OV5640. A list of cameras supported by the underlying `esp32_camera` library can be seen on GitHub (https://adafru.it/10E7).

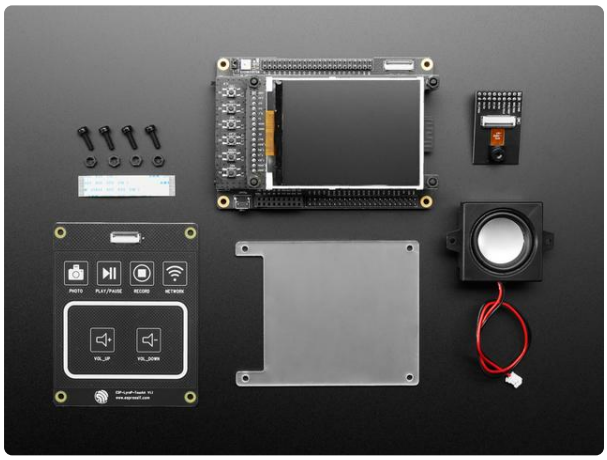
Parts

Development kit from Espressif with ESP32-S3 and 2-megapixel camera

1 x ESP32-S3-EYE

Development kit from Espressif with ESP32-S3 and 2-megapixel camera

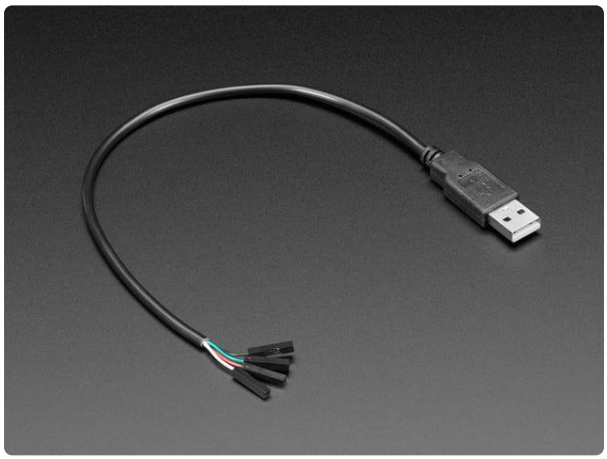
https://www.mouser.com/ProductDetail/Espressif-Systems/ESP32-S3-EYE?qs=Rp5uXu7WBW8jiAfcrui1pQ%3D%3D



ESP32-S2 Kaluga Dev Kit featuring ESP32-S2 WROVER

The ESP32-S2-Kaluga-1 kit is a full featured development kit by Espressif for the ESP32-S2 that comes with everything but the kitchen sink! From TFTs to touch panels,...

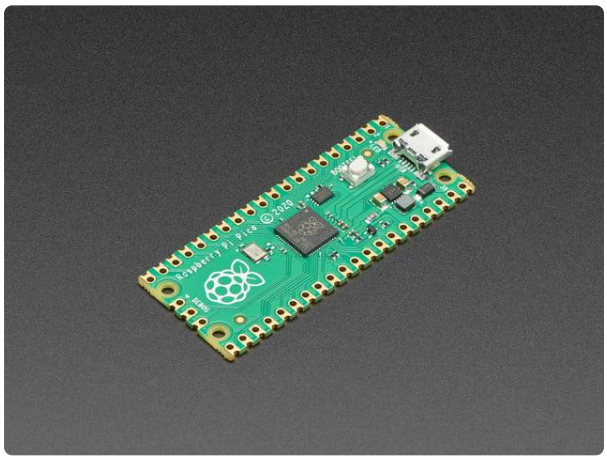
https://www.adafruit.com/product/4729



USB Type A Plug Breakout Cable with Premium Female Jumpers

If you'd like to connect a USB-capable chip to your USB host, this cable will make the task very simple. There is no converter chip in this cable! Its basically a...

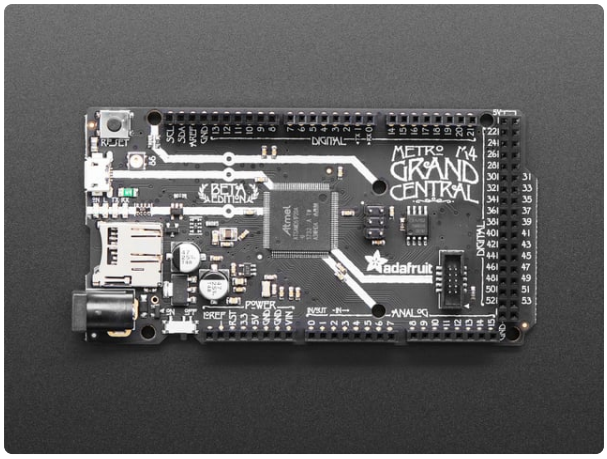
https://www.adafruit.com/product/4448



Raspberry Pi Pico RP2040

The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...

https://www.adafruit.com/product/4864



Adafruit Grand Central M4 Express featuring the SAMD51

Are you ready? Really ready? Cause here comes the Adafruit Grand Central featuring the Microchip ATSAMD51. This dev board is so big, it's not...

https://www.adafruit.com/product/4064

USB C to Micro B Cable - 3 ft 1 meter

As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to Micro B plugs!USB C is the latest...

https://www.adafruit.com/product/3878

USB cable - USB A to Micro-B

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

https://www.adafruit.com/product/592

1 x OV7670 Camera Boards

Lot of 5 camera boards

https://www.amazon.com/AITRIP-OV7670-640x480-0-3Mega-Arduino/dp/B09126R875

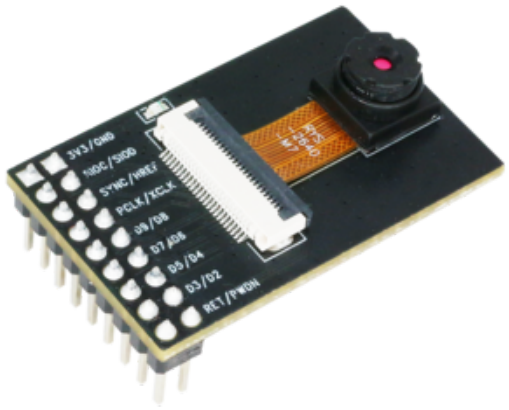
---

1 x OV2640 Camera Board

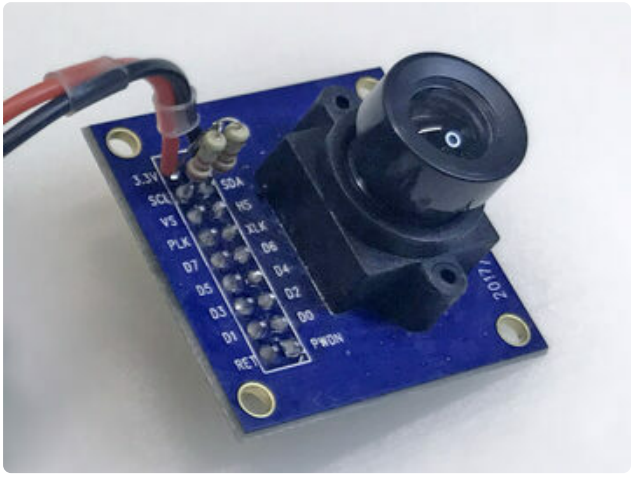
Camera module, 2 Megapixel

https://www.waveshare.com/OV2640-Camera-Board.htm

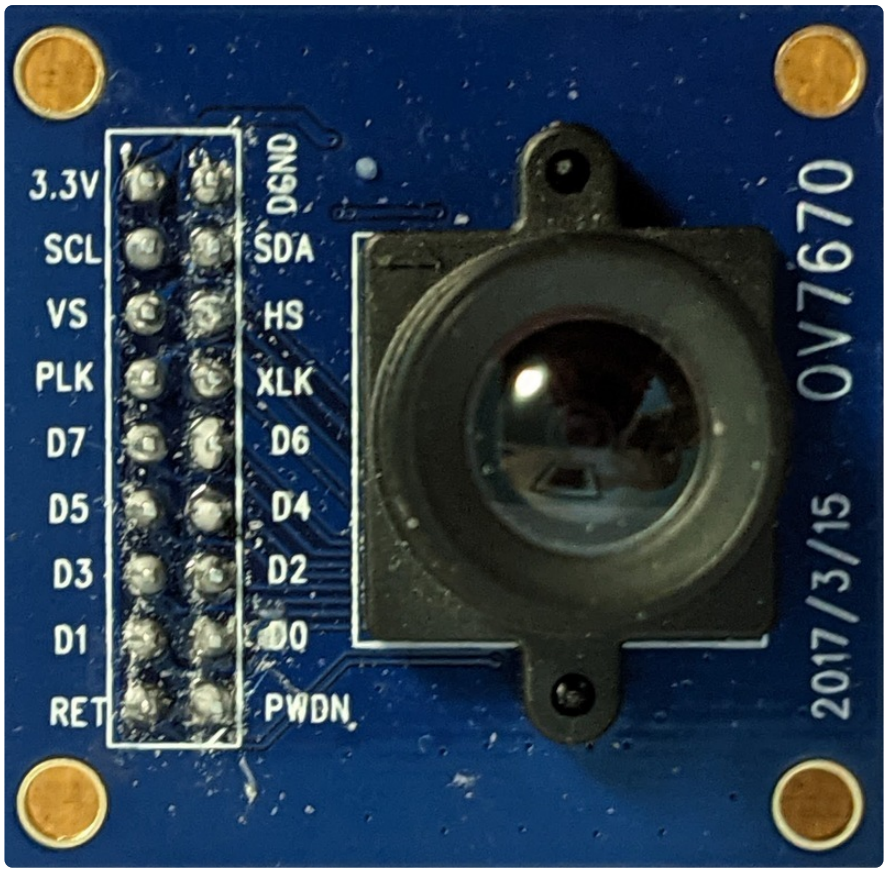
Cameras and Pinouts



The ESP-LyraP-CAM (https://adafru.it/TAH) v1.1 module included with the Kaluga development kit is ready to go: it includes the pull-up resistors on the I2C communication lines, here labeled SIOC and SIOD (equivalent to SCL and SDA).



The typical OV7670 module doesn't include the I2C pull-up resistors, so you may have to add them. This one was also modified so that it would fit directly on the header of a Grand Central M4 board.

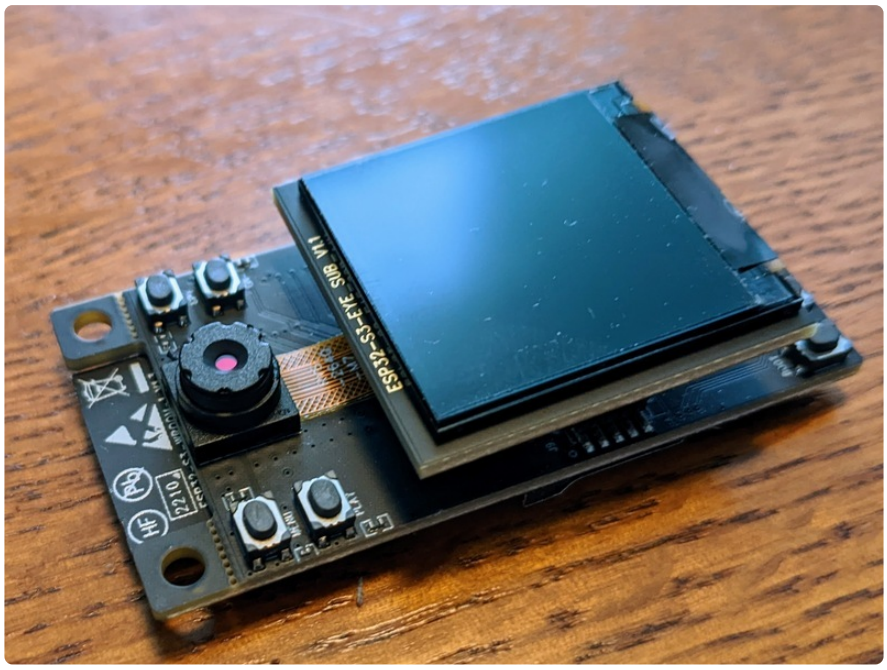


Pins

See the specific board pages for details on how to connect the pins to a microcontroller.

- 3.3V: Connect to a regulated 3.3V supply
- GND (sometimes labeled DGND): Connect to GND
- SDA, SCL (sometimes labeled SIOC, SIOD): I2C bus used to configure the camera.
- XLK (sometimes labeled XCLK): Clock signal from the microcontroller to the camera.
- VS, HS, PLK (sometimes labeled SYNC, HREF, PCLK):
  Vertical & Horizontal synchronization pulses, and pixel clock from the camera to the microcontroller
- D0..D7: Pixel data from the camera to the microcontroller
- RET, PWDN: Reset and Power Down pins from the microcontroller are used during initialization to reset the camera into a known state.

Working with espcamera



The esp32_camera module is new in CircuitPython 8.

CircuitPython 8 features a new importable camera module for Espressif ESP32 microcontrollers including the ESP32, the ESP32-S2, and the ESP32-S3. It's just different enough from the **adafruit_ov####** modules that different code is needed.

Requirements:

- A compatible board and microcontroller with PSRAM
- A build of CircuitPython 8 or newer with the **espcamera** built in module enabled
- A supported camera, connected properly

At the moment, we really like the esp32-s3-eye from Espressif (https://adafru.it/19bV) since it packs a 2-megapixel camera and LCD into a cute little development kit. The following examples are all coded for this board.

1 x ESP32-S3-EYE
Development kit from Espressif with ESP32-S3 and 2-megapixel camera

https://www.mouser.com/ProductDetail/Espressif-Systems/ESP32-S3-EYE?qs=Rp5uXu7WBW8jiAfcrui1pQ%3D%3D

CircuitPython 9 no longer requires the use of reserved PSRAM, and the CIRCUITPY_RESERVED_PSRAM= directive is ignored.

Configure reserved PSRAM

The camera captures images to a special region of RAM called "reserved PSRAM". This is a portion of RAM that is set aside and cannot be used for regular Python objects, but can be used by **espcamera** (as well as other behind the scenes activity such as sockets and networking that are managed by **esp-idf**). The large size of images, especially bitmap images, are why psram is required for **espcamera**.

Boards like the esp32-s3-eye that include a camera module built in already reserve a fixed amount of PSRAM, usually 1MiB (1048576 bytes). For other boards, or if you want to fine-tune the default reserved value, you must place a line in the **CIRCUITPY/.env** file with the amount, such as:

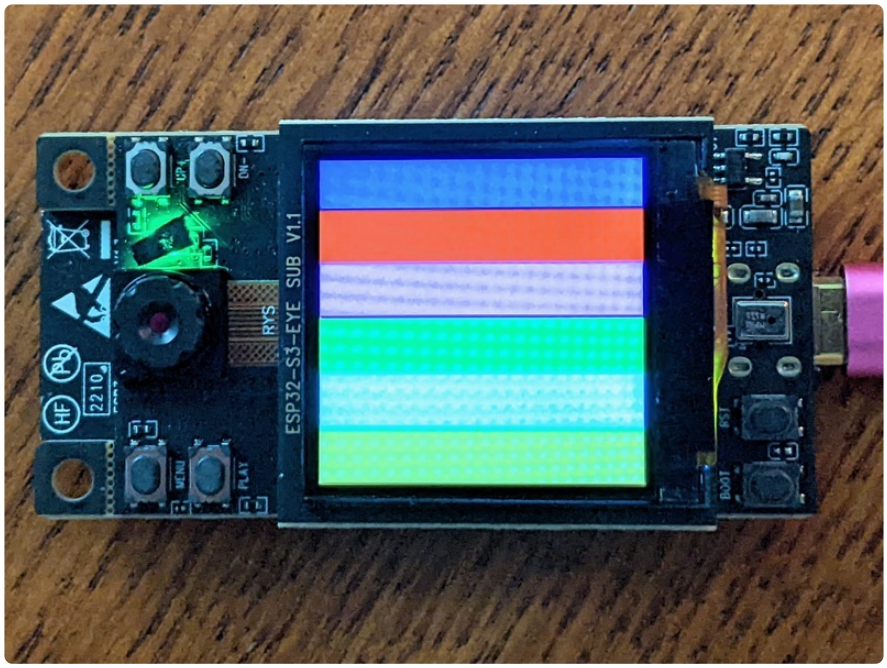
```
CIRCUITPY_RESERVED_PSRAM=1048576
```

After updating the **.env** file, you must hard-reset the board for the change to take effect. You can verify the setting in the repl:

```
&amp;gt;&amp;gt;&amp;gt; import espidf
&amp;gt;&amp;gt;&amp;gt; espidf.get_reserved_psram()
1048576
```

Continue to the next pages for some examples! First up: LCD Viewfinder (https://adafru.it/19bW).

Example: LCD Viewfinder



This example is for CircuitPython 8 and newer.

This program is designed for the ESP32-S3-EYE development kit.

Your project will use a specific set of CircuitPython libraries and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your Feather board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

When CircuitPython restarts, the live camera view will be shown on the LCD. You can click the button marked "BOOT" to switch between live camera view and a color-bar test pattern. The test pattern is shown here. A small black sticker has been placed over the **extremely** bright green power LED.


```
# SPDX-FileCopyrightText: Copyright (c) 2022 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Use the built-in LCD as a viewfinder for the camera

This example requires:
 * ESP32-S3-EYE development kit from Espressif

To use:

Copy the project bundle to CIRCUITPY.
"""

import struct

import adafruit_ticks
import board
import displayio
import espcamera
import keypad

button = keypad.Keys((board.BOOT,), value_when_pressed=False)

cam = espcamera.Camera(
    data_pins=board.CAMERA_DATA,
    external_clock_pin=board.CAMERA_XCLK,
```

```
    pixel_clock_pin=board.CAMERA_PCLK,
    vsync_pin=board.CAMERA_VSYNC,
    href_pin=board.CAMERA_HREF,
    pixel_format=espcamera.PixelFormat.RGB565,
    frame_size=espcamera.FrameSize.R240X240,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2,
    grab_mode=espcamera.GrabMode.WHEN_EMPTY)

cam.vflip = True

board.DISPLAY.auto_refresh = False
display_bus = board.DISPLAY.bus

display_bus.send(36, struct.pack(">hh", 0, 239))
display_bus.send(42, struct.pack(">hh", 0, 239))
display_bus.send(43, struct.pack(">hh", 0, 80+239))
t0 = adafruit_ticks.ticks_ms()
while True:
    if (event := button.events.get()) and event.pressed:
        cam.colorbar = not cam.colorbar
    frame = cam.take(1)
    if isinstance(frame, displayio.Bitmap):
        display_bus.send(44, frame)
        t1 = adafruit_ticks.ticks_ms()
        fps = 1000 / adafruit_ticks.ticks_diff(t1, t0)
        print(f"{fps:3.1f}fps")  # typically runs at about 25fps
        t0 = t1
```

How it works

Libraries

First, the code imports the necessary libraries. In particular:

- **espcamera** to interface with the camera
- **displayio** to use Bitmap objects

```
import struct

import adafruit_ticks
import board
import displayio
import espcamera
import keypad
```

Test-pattern Toggle Button

The "Boot" button on this board can be used as a regular button with `keypad.Keys`. The other 4 buttons share a single pin, so using them is more complicated.

```
button = keypad.Keys((board.BOOT,), value_when_pressed=False)
```

Camera

Next, the camera object is created. Because the LCD's size is 240x240, we use the 240x240 resolution mode, `FrameSize.R240X240` when setting up the camera. Setting the `vflip` property makes the LCD show the image right side up. Depending whether you want a mirror mode or not you can set the `hmirror` property to `True`.

```
cam = espcamera.Camera(
    data_pins=board.CAMERA_DATA,
    external_clock_pin=board.CAMERA_XCLK,
    pixel_clock_pin=board.CAMERA_PCLK,
    vsync_pin=board.CAMERA_VSYNC,
    href_pin=board.CAMERA_HREF,
    pixel_format=esp32_camera.PixelFormat.RGB565,
    frame_size=esp32_camera.FrameSize.R240X240,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2,
    grab_mode=esp32_camera.GrabMode.WHEN_EMPTY)

cam.vflip = True
```

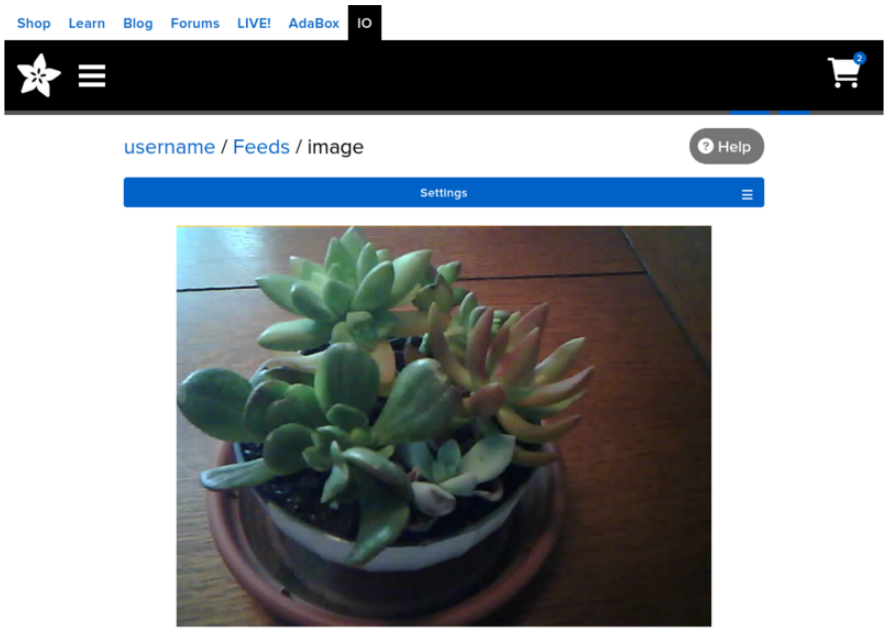
Forever Loop to Display and check button

Whenever a "pressed" event arrives, toggle the "colorbar" property of the camera. It defaults to `False` (off), so pressing BOOT once toggles it on and pressing BOOT again toggles it back off.

To make the display refresh as quickly as possible, we bypass displayio and send out our bitmap directly to the LCD. Just for bragging rights, we calculate the Frames Per Second (FPS) that we update the LCD. Refresh rates of 25FPS can be obtained with this code.

```
while True:
    if (event := button.events.get()) and event.pressed:
        cam.colorbar = not cam.colorbar
    frame = cam.take(1)
    if isinstance(frame, displayio.Bitmap):
        display_bus.send(44, frame)
        t1 = adafruit_ticks.ticks_ms()
        fps = 1000 / adafruit_ticks.ticks_diff(t1, t0)
        print(f"{fps:3.1f}fps")  # typically runs at about 25fps
        t0 = t1
```

Example: Webcam with Adafruit IO



This example is for CircuitPython 8 and newer.

This program is designed for the ESP32-S3-EYE development kit.

Upload a jpeg image to Adafruit IO at regular intervals

This example requires:

- ESP32-S3-EYE development kit from Espressif

To use, you must set up WiFi and Adafruit IO:

- On io.adafruit.com (https://adafru.it/eZ8), create a feed named "image" and turn **OFF** history
- On io.adafruit.com, create a dashboard and add an "image" block using the feed "image" as its data
- Set up **CIRCUITPY/.env** with WiFI and Adafruit IO credentials

Your project will use a specific set of CircuitPython libraries and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your Feather board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

When the new code starts, it will upload an image to Adafruit IO approximately every 10 seconds.


```
# SPDX-FileCopyrightText: Copyright (c) 2022 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Upload a jpeg image to Adafruit IO at regular intervals

This example requires:
 * ESP32-S3-EYE development kit from Espressif

To use:
 * On io.adafruit.com, create a feed named "image" and turn OFF history
 * On io.adafruit.com, create a dashboard and add an "image" block
   using the feed "image" as its data
 * Set up CIRCUITPY/.env with WiFI and Adafruit IO credentials
 * Copy the project bundle to CIRCUITPY
"""

import binascii
import io
import os
import ssl
import time
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
import board
import espcamera
import socketpool
import wifi

aio_username = os.getenv('AIO_USERNAME')
aio_key = os.getenv('AIO_KEY')

image_feed = "image"

cam = espcamera.Camera(
    data_pins=board.CAMERA_DATA,
    external_clock_pin=board.CAMERA_XCLK,
    pixel_clock_pin=board.CAMERA_PCLK,
    vsync_pin=board.CAMERA_VSYNC,
    href_pin=board.CAMERA_HREF,
    pixel_format=espcamera.PixelFormat.JPEG,
```

```
    frame_size=espcamera.FrameSize.SVGA,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    grab_mode=espcamera.GrabMode.WHEN_EMPTY)
cam.vflip = True


pool = socketpool.SocketPool(wifi.radio)

print("Connecting to Adafruit IO")
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=aio_username,
    password=aio_key,
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)
mqtt_client.connect()
io = IO_MQTT(mqtt_client)

while True:
    frame = cam.take(1)
    if isinstance(frame, memoryview):
        jpeg = frame
        print(f"Captured {len(jpeg)} bytes of jpeg data")

        # b2a_base64() appends a trailing newline, which IO does not like
        encoded_data = binascii.b2a_base64(jpeg).strip()
        print(f"Expanded to {len(encoded_data)} for IO upload")

        io.publish("image", encoded_data)

        time.sleep(10)
```

How it works

Many parts of this example are similar to the previous one, but the explanations are not repeated.

For this example you'll need to configure your **CIRCUITPY/.env** file properly. It will need to contain the following keys. The values depend on your WiFi and adafruit io settings.

```
AIO_KEY=YourAioKey
AIO_USERNAME=YourAioUsername
CIRCUITPY_WEB_API_PASSWORD=YourWebApiPassword
CIRCUITPY_WIFI_PASSWORD=YourWifiPassword
CIRCUITPY_WIFI_SSID=YourWifiName
```

Adafruit IO Credentials

Adafruit IO Credentials are stored in the **CIRCUITPY/.env** file and retrieved using dotenv:

```
aio_username = os.getenv('/.env', 'AIO_USERNAME')
aio_key = os.getenv('/.env', 'AIO_KEY')
```

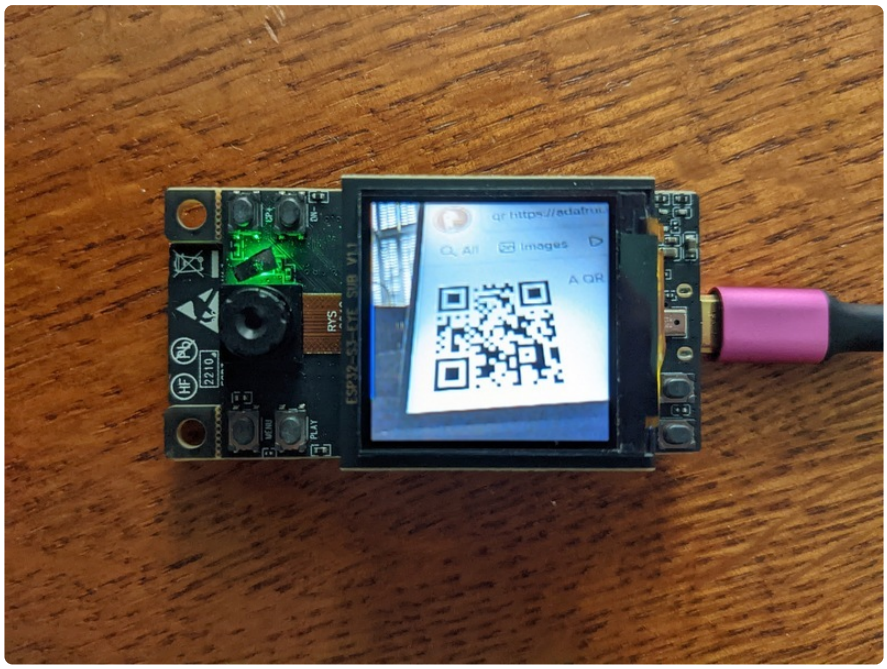
The WiFi interface is assumed to be configured for the web workflow, so we skip making a connection ourselves and skip directly to creating the connection to Adafruit IO:

```
pool = socketpool.SocketPool(wifi.radio)

print("Connecting to Adafruit IO")
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=aio_username,
    password=aio_key,
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)
mqtt_client.connect()
io = IO_MQTT(mqtt_client)
```

We wait for a JPEG image capture to be ready from the camera. Then, we encode it in a specific way that Adafruit IO understands, and upload it. This repeats forever, waiting 10 seconds between image captures.

```
while True:
    frame = cam.take(1)
    if isinstance(frame, memoryview):
        jpeg = frame
        print(f"Captured {len(jpeg)} bytes of jpeg data")

        # b2a_base64() appends a trailing newline, which IO does not like
        encoded_data = binascii.b2a_base64(jpeg).strip()
        print(f"Expanded to {len(encoded_data)} for IO upload")

        io.publish("image", encoded_data)

        time.sleep(10)
```

Example: QR Code Scanner



This example is for CircuitPython 8 and newer.

QR Code Scanner

This program is designed for the ESP32-S3-EYE development kit.

`qrio`, the built-in module for decoding QR codes, is discussed in more detail in its dedicated guide (https://adafru.it/19bX).

Whenever it "sees" a QR code, it will print the contained data on the REPL. It also shows the live view on the LCD.

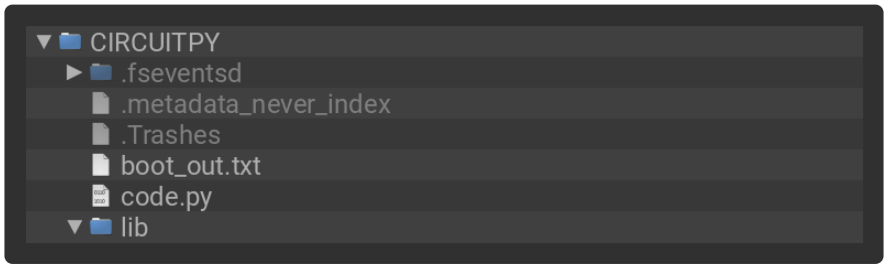
Your project will use a specific set of CircuitPython libraries and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your Feather board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

Once the demo starts, it will look for a QR code and print it on the REPL. While it works, it'll show the camera image on the viewfinder so you can line up the code within the frame.

This code works great with a standard or telephoto lens, but a wide-angle or fish-eye lens distorts the QR code and prevents CircuitPython from decoding it.

Scanning from a computer LCD or phone screen works sometimes, but scanning from paper is more reliable. It helps if the image is large (several inches or 75 to 100mm), since most cameras don't have good close focus for a small (two inches or 50mm or smaller) codes.




```
# SPDX-FileCopyrightText: Copyright (c) 2022 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
This demo is designed for the Kaluga development kit version 1.3 with the
ILI9341 display.
"""

import struct
import board
import espcamera
import qrio

print("Initializing camera")
cam = espcamera.Camera(
    data_pins=board.CAMERA_DATA,
    external_clock_pin=board.CAMERA_XCLK,
    pixel_clock_pin=board.CAMERA_PCLK,
    vsync_pin=board.CAMERA_VSYNC,
    href_pin=board.CAMERA_HREF,
    pixel_format=espcamera.PixelFormat.RGB565,
    frame_size=espcamera.FrameSize.R240X240,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2)
cam.vflip = True
cam.hmirror = True

board.DISPLAY.auto_refresh = False
display_bus = board.DISPLAY.bus

print(cam.width, cam.height)
qrdecoder = qrio.QRDecoder(cam.width, cam.height)

print(qrdecoder.width, qrdecoder.height)
#raise SystemExit

ow = (board.DISPLAY.width - cam.width) // 2
oh = (board.DISPLAY.height - cam.height) // 2
display_bus.send(42, struct.pack(">hh", ow, cam.width + ow - 1))
display_bus.send(43, struct.pack(">hh", oh, cam.height + ow - 1))

while True:
    frame = cam.take(1)
```

```
    display_bus.send(44, frame)
    for row in qrdecoder.decode(frame, qrio.PixelPolicy.RGB565_SWAPPED):
        payload = row.payload
        try:
            payload = payload.decode("utf-8")
        except UnicodeError:
            payload = str(payload)
        print(payload)
    print(end=".")
```

The QR decoder is specific to the size of image, so it needs to be created to match the camera's image dimensions:

```
qrdecoder = qrio.QRDecoder(cam.width, cam.height)
```

Each frame is simply passed on to the QR decoder, which returns zero or more pieces of data. Because the data might or might not be valid UTF-8, we try two ways of decoding it into a printable value.

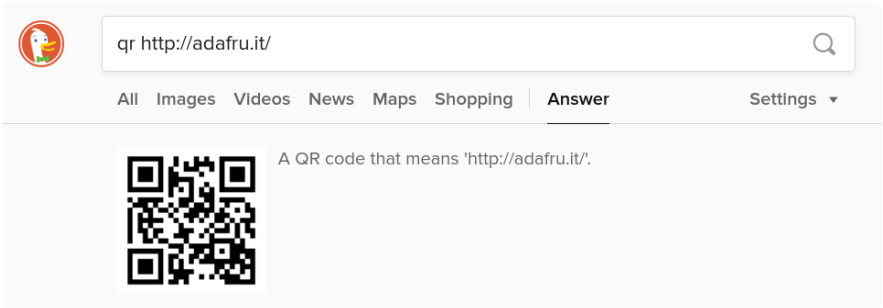
```
# (inside the forever-loop)
    for row in qrdecoder.decode(frame, qrio.PixelPolicy.RGB565_SWAPPED):
        payload = row.payload
        try:
            payload = payload.decode("utf-8")
        except UnicodeError:
            payload = str(payload)
        print(payload)
```

Creating QR Codes

There are lots of sites & services to generate QR codes. This one (https://adafru.it/UeW) seems pretty no-nonsense.

If you have Python installed on your host computer, you can use the Adafruit miniQR Library (https://adafru.it/UeX)to show QR codes in a terminal window.

If you use the Duck Duck Go search engine, you can search for "qr" + your terms, like so (https://adafru.it/UeY). It can help to use the web browser's magnification function (ctrl-+ or command-+) to adjust the image to be bigger on screen.



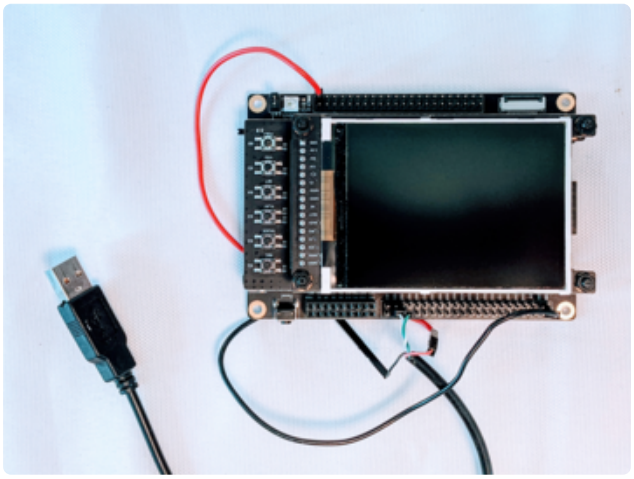
Documentation: espcamera

Documentation: espcamera (https://adafru.it/18tC)

Install TinyUF2 on Espressif Kaluga

There are two versions of the Kaluga board, v1.2 and v1.3. Check which version you have, and install the correct build of CircuitPython. The board revisions change the pinout of the camera connector slightly.

Now, use the breakout USB connection in lieu of either of the built-in USB Micro B ports to install and use CircuitPython.



Start by connecting the USB Breakout Cable to the Kaluga board.

Black: Use a Male/Female Extension Jumper Wire to connect to **GND**
White: Connect to **IO19**
Green: Connect to **IO20**
Red: Use a Male/Female Extension Jumper Wire to connect to **5V**

Do not connect the Red wire to 3V3, it will irreversibly damage the Kaluga.

If you're familiar with our other products and chipsets you may be famliar with our drag-n-drop bootloader, a.k.a UF2. We have a UF2 bootloader for the ESP32-S2, that will let you drag firmware on/off a USB disk drive.

Unlike the M0 (SAMD21) and M4 (SAMD51) boards, there is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the bootloader, especially if you upload Arduino sketches to ESP32S2 boards that doesn't "know" there's a bootloader it should not overwrite!

However, thanks to the ROM bootloader, you don't have to worry about it if the UF2 bootloader is damaged. The ROM bootloader can never be disabled or erased, so its always there if you need it! You can simply re-load the UF2 bootloader (USB-disk-style) with the ROM bootloader (non-USB-drive)

You can use the TinyUF2 bootloader to load code directly, say CircuitPython or the binary output of an Arduino compilation or you can use it to load a second bootloader on, like UF2 which has a drag-n-drop interface.

Installing the UF2 bootloader will erase your board's firmware which is also used for storing CircuitPython/Arduino/Files! Be sure to back up your data first.

Method 1: WebSerial ESPTool / `esptool`

This section outlines using WebSerial ESPTool or `esptool` to flash the UF2 bootloader onto your ESP32-S2 board.

Step 1. Download the tinyuf2 combined.bin file here

Note that this file is 3MB but that's because the bootloader is near the end of the available flash. It's not actually 3MB large, most of the file is empty but its easier to program if we give you one combined 'swiss cheese' file. Save this file to your desktop or wherever you plan to run esptool from

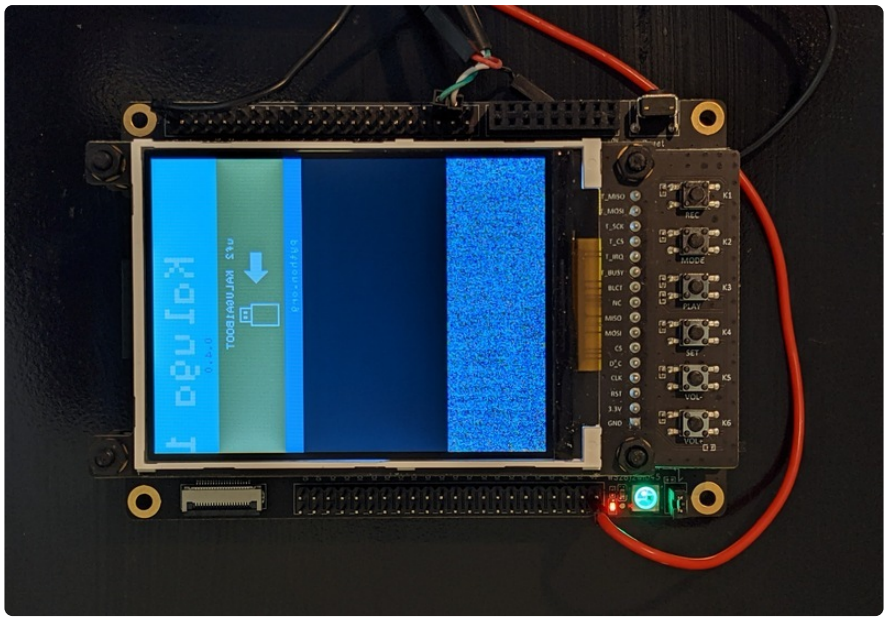
combined.bin

https://adafru.it/TAI

Step 2. Place your board in bootloader mode

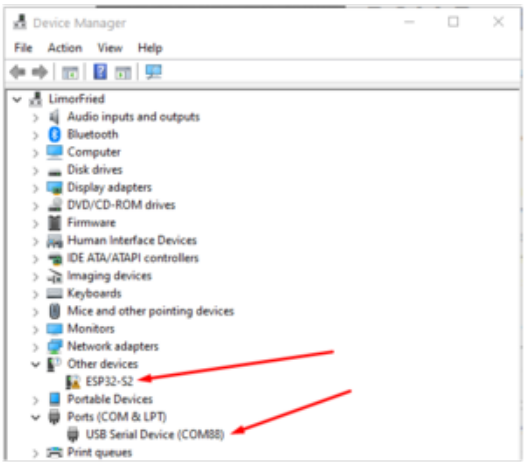
Entering the bootloader is easy. Complete the following steps.

1. **Make sure your ESP32-S2 is plugged into USB port to your computer using a data/sync cable.** Charge-only cables will not work!
2. **Turn on the On/Off switch** - If your board has a power switch, check that you see the OK light on so you know the board is powered, a prerequisite!
3. **Press and hold the DFU / Boot0 button down.** Don't let go of it yet!
4. **Press and release the Reset button.** You should have the DFU/Boot0 button pressed while you do this.
5. **Now you can release the DFU / Boot0 button**



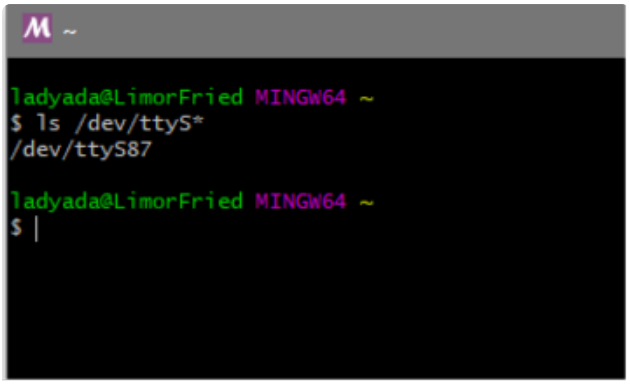
Because there are several incompatible versions of the Kaluga TFT display, the bootloader's screen may appear incorrectly or not at all. This does not affect its operation.

Check for a new serial / COM port



On Windows check the Device manager - you will see a COM port, for example here its COM88. You may also see another "Other device" called ESP32-S2

It's best to do this with no other dev boards plugged in so you don't get confused about which COM port is the ESP32-S2



On Mac/Linux you will need to find the tty name which lives under /dev

On Linux, try **ls /dev/ttyS*** for example, to find the matching serial port name. In this case it shows up as **/dev/ttyS87**. If you don't see it listed try **ls /dev/ttyA*** on some Linux systems it might show up like **/dev/ttyACM0**

On Mac, try **ls /dev/cu.usbmodem*** for example, to find the matching serial port name. In this case, it shows up as **/dev/cu.usbmodem01**

It's best to do this with no other dev boards plugged in so you don't get confused about which serial port is the ESP32-S2

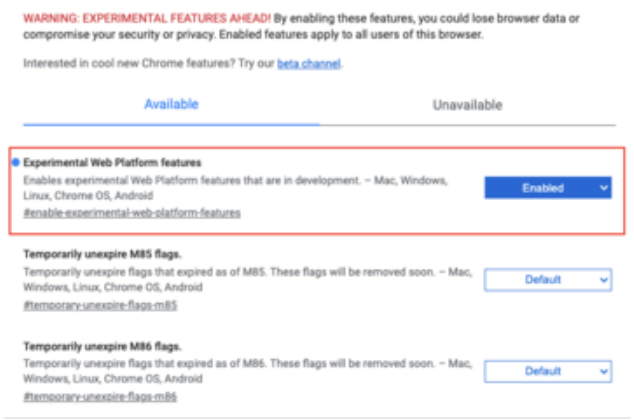
Step 3 Option A. Use the Web Serial ESPTool to upload

The WebSerial ESPTool was designed to be a web-capable option for programming ESP32-S2 boards. It allows you to erase the contents of the microcontroller and program up to 4 files at different offsets.

You will have to use the Chrome browser for this to work, Safari and Firefox, etc are not supported because we need Web Serial and only Chrome is supporting it to the level needed.

Enable Web Serial (For older chrome)

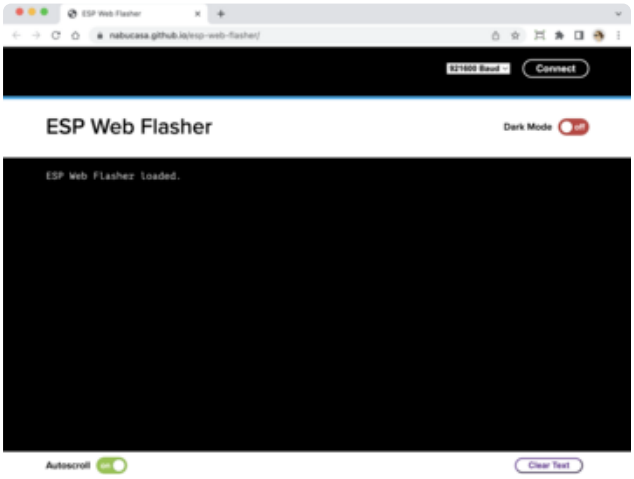
As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.



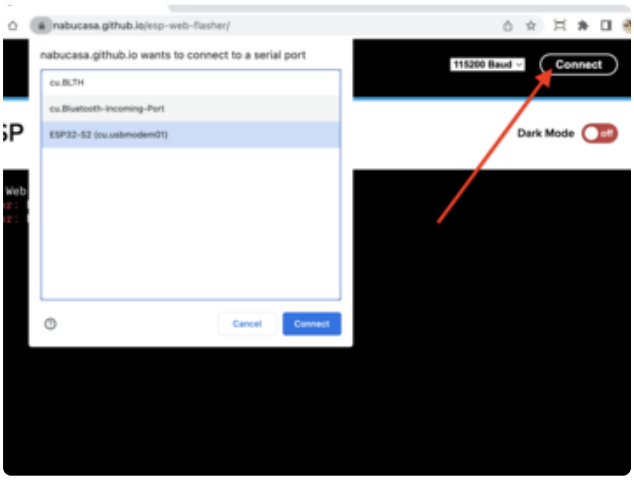
Visit **chrome://flags** from within Chrome. Find and enable the **Experimental Web Platform features**

Restart Chrome

Connecting



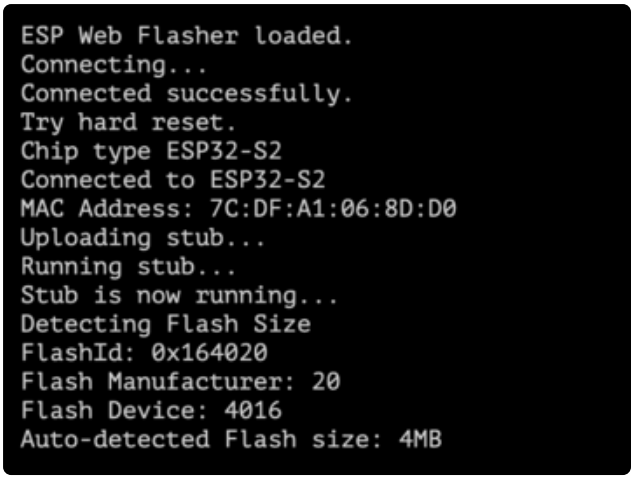
In the **Chrome browser** visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru.it/PMB). It should look like the image to the left.



Press the **Connect** button in the top right of the web browser. You will get a pop up asking you to select the COM or Serial port.

Remember, you should remove all other USB devices so only the ESP32-S2 board is attached, that way there's no confusion over multiple ports!

On some systems, such as MacOS, there may be additional system ports that appear in the list.



The Javascript code will now try to connect to the ROM bootloader. It may timeout for a bit until it succeeds. On success, you will see that it is **Connected** and will print out a unique **MAC address** identifying the board.

Once you have successfully connected, the command toolbar will appear.

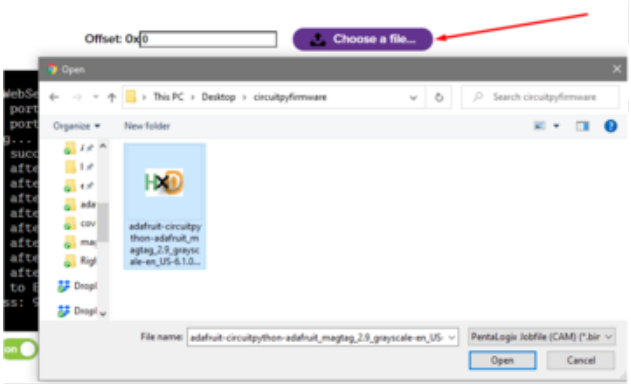
Erasing the Contents

If you would like to erase the entire flash area so that you can start with a clean slate, you can use the erase feature. We recommend doing this if you are having issues.

To erase the contents, click the Erase button. You will be prompted whether you want to continue. Click OK to continue or if you changed your mind, just click cancel.

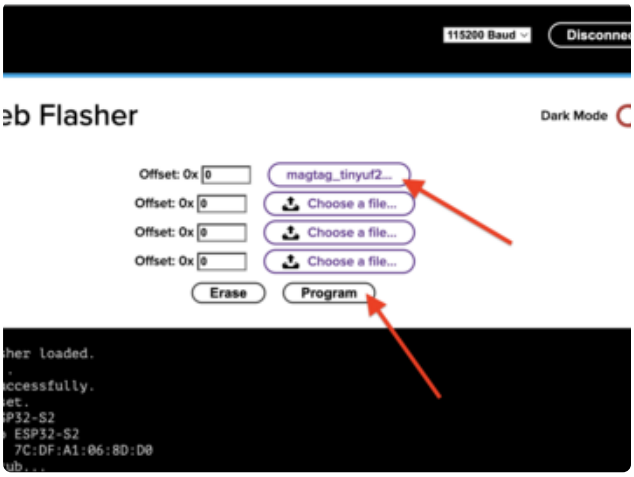
Programming the Microcontroller

Programming the microcontroller can be done with up to 4 files at different locations, but with the **tinyuf2combo BIN** file, which you should have downloaded under **Step 1** on this page, you only need to use 1 file.



You can click on **Choose a file...** from any of the available buttons. It will only attempt to program buttons with a file and a unique location. Then select the Adafruit CircuitPython **BIN** files (not the UF2 file!)

Verify that the **Offset** box next to the file location you used is 0x0.



Once you choose a file, the button text will change to match your filename. You can then select the Program button to start flashing.

A progress bar will appear and after a minute or two, you will have written the firmware.

After using the tool, press the reset button to get out of bootloader mode and launch the new firmware!

Step 3. Option B. Use esptool.py to upload (for advanced users)

Once you have entered ROM bootloader mode, you can then use Espressif's esptool program (https://adafru.it/E9p) to communicate with the chip! `esptool` is the 'official' programming tool and is the most common/complete way to program an ESP chip.

Install ESPTool.py

You will need to use the command line / Terminal to install and run `esptool`.

You will also need to have pip and Python installed (any version!)

Install the latest version using pip (you may be able to run `pip` without the `3` depending on your setup):

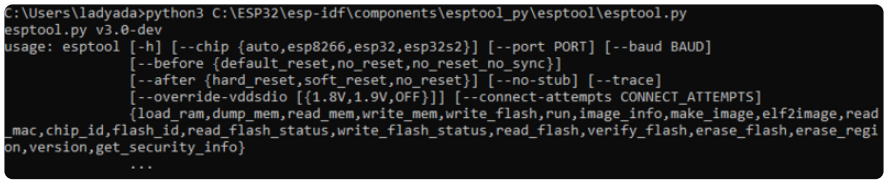
```
pip3 install --upgrade esptool
```

Then, you can run:

```
esptool.py
```

Test the Installation

Run `esptool.py` in a new terminal/command line and verify you get something like the below:

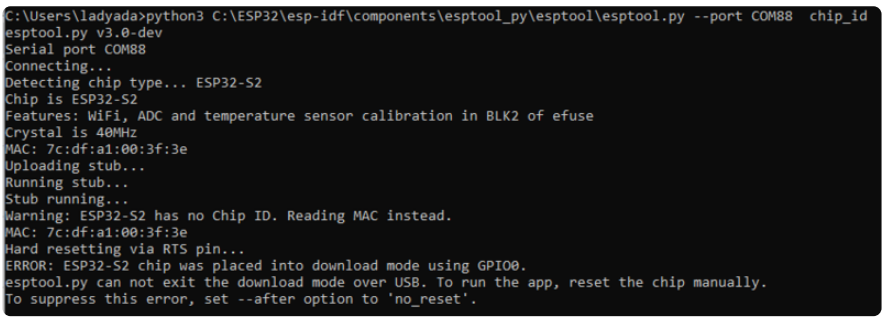


Make sure you are running esptool v3.0 or higher, which adds ESP32-S2 support

Run the following command, replacing the identifier after `--port` with the `COMxx`, `/dev/cu.usbmodemxx` or `/dev/ttySxx` you found above.

```
esptool.py --port COM88 chip_id
```

You should get a notice that it connected over that port and found an ESP32-S2



Installing the Bootloader

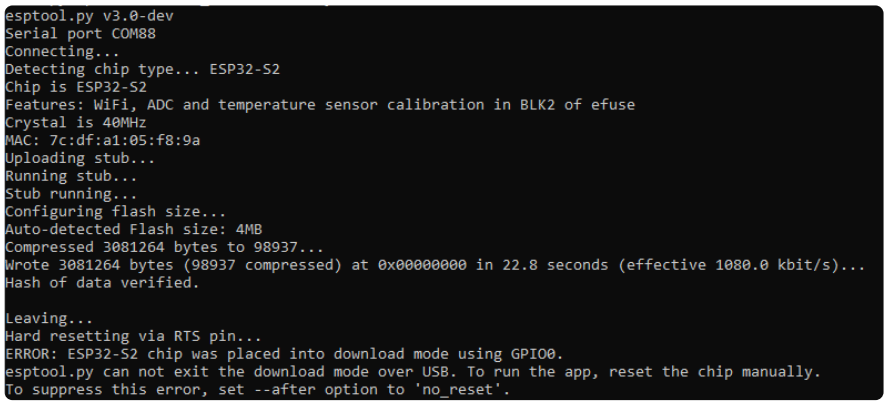
Run this command and replace the serial port name with your matching port and the file you just downloaded

```
esptool.py --port COM88 write_flash 0x0 tinyuf2_combo.bin
```

Don't forget to change the `--port` name to match.

There might be a bit of a 'wait' when programming, where it doesn't seem like it's working. Give it a minute, it has to erase the old flash code which can cause it to seem like it's not running.

You'll finally get an output like this:



Step 4. Reset the board

Click the RESET button to launch the bootloader. You'll see a new disk drive on your computer with the name **KALUGA1BOOT**.

You're now ready to copy the CircuitPython UF2 on to the drive which will set up CircuitPython!

Method 2: Flash an Arduino Sketch

This section outlines flashing an Arduino sketch onto your ESP32-S2 board, which automatically installs the UF2 bootloader as well.

Arduino IDE Setup

If you don't already have the Arduino IDE installed, the first thing you will need to do is to download the latest release of the Arduino IDE. ESP32-S2 requires **version 1.8** or higher. Click the link to download the latest.

Arduino IDE Download

https://adafru.it/Pd5

After you have downloaded and installed the latest version of Arduino IDE, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the **File** > **Preferences** menu in Windows or Linux, or the **Arduino** > **Preferences** menu on OS X.

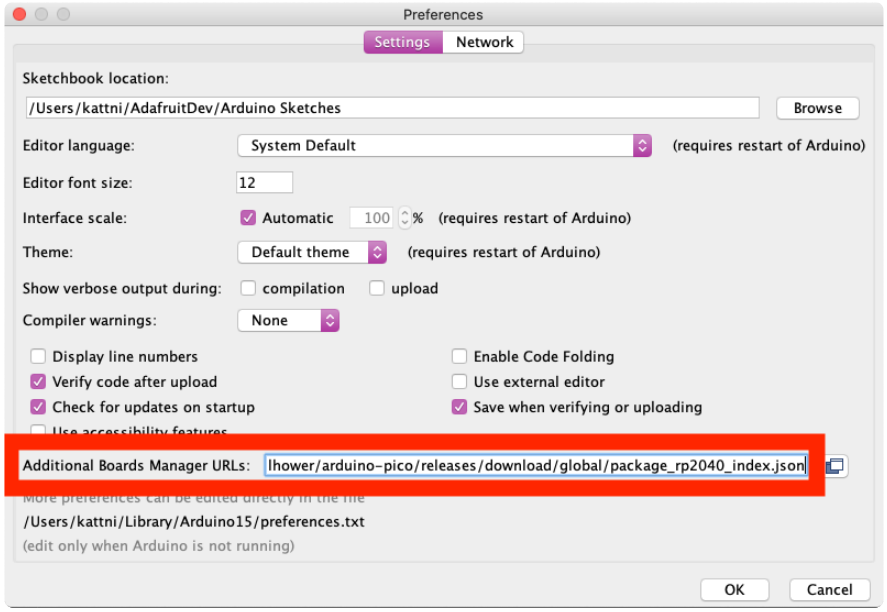
The **Preferences** window will open.

In the **Additional Boards Manager URLs** field, you'll want to add a new URL. The list of URLs is comma separated, and you will only have to add each URL once. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

Copy the following URL.

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```

Add the URL to the the **Additional Boards Manager URLs** field (highlighted in red below).



Click **OK** to save and close **Preferences**.

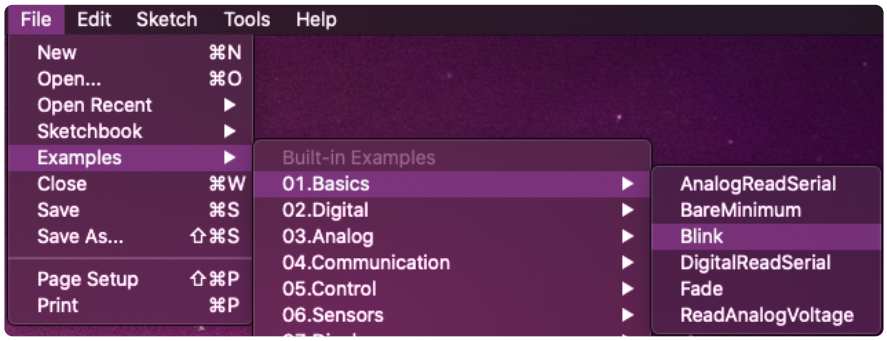
In the **Tools** > **Boards** menu you should see the **ESP32 Arduino** menu. In the expanded menu, it should contain the ESP32 boards along with all the latest ESP32-S2 boards.

Now that your IDE is setup, you can continue on to loading the sketch.

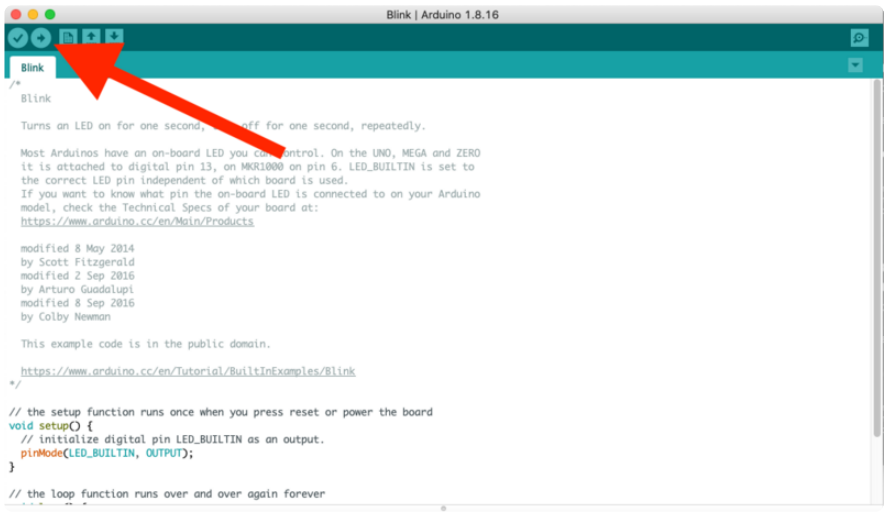
Load the Blink Sketch

In the **Tools** > **Boards** menu you should see the **ESP32 Arduino** menu. In the expanded menu, look for the menu option for the **ESP32S2 Dev Module**, and click on it to choose it.

Open the Blink sketch by clicking through **File** > **Examples** > **01.Basics** > **Blink**.



Once open, click Upload from the sketch window.



Once successfully uploaded, the little red LED will begin blinking once every second. At that point, you can now enter the bootloader.

Espressif Kaluga Setup

The Kaluga board v1.2's reset button will not work when the camera is attached. Instead, the board must be power cycled. This hardware bug was corrected for v1.3 of the board.

On the Kaluga, the camera connector shares pins with the JTAG debugging facility. It is not possible to use a JTAG debugger together with the camera on this board.

The Kaluga development kit from Espressif includes almost everything you need: The microcontroller, a camera, and an LCD.

Take the assembled Kaluga board stack (all three boards) and attach the camera at the dedicated header, making sure the pins are inserted properly.

You **do not** need to add any pull-up resistors; they are already provided on the Kaluga's audio daughterboard.

There are at least 3 variants of the LCD board that ship with the Kaluga:

- st7789
- ili9341
- ili9341 with rotation=90

There are no markings to distinguish the three, so you will need to try each variant until you find the one that works.

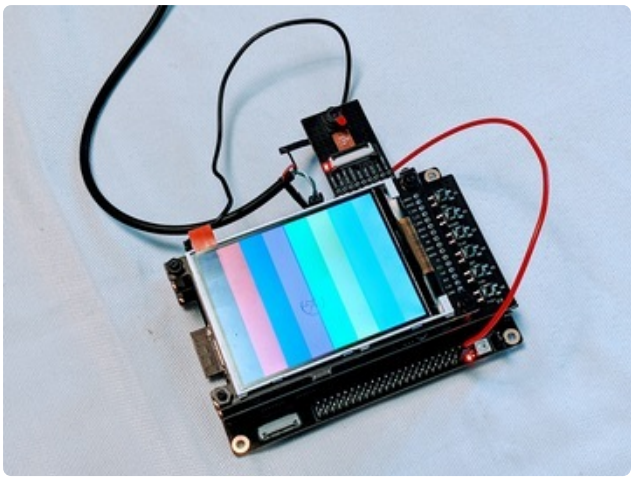
First, make sure you can see the Kaluga's **CIRCUITPY** drive and connect to the REPL. Open the REPL and double check that `import imagecapture` works without showing an error. (note: if it does, the most likely reason is that you are using CircuitPython 8 or newer, which is incompatible with the code in this guide)

Then, copy the correct bundle to your device. It will automatically reload and start displaying the image from the camera on the built-in LCD.

Kaluga 1.3 with OV2640 and ili9341 display (CircuitPython 7)

This code is for CircuitPython 7. Revised code will be required for CircuitPython 8.

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.



Espressif Kaluga ESP32-S2 with OV2640 display showing the test pattern. The test pattern's color bars appear heavily distorted due to the viewing angle.

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.  It probably won't work on v1.2 without modification.

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  Furthermore, there are at least 2 ILI9341 variants, one of which needs
rotation=90!  This demo is for the ili9341.  If the display is garbled, try adding
rotation=90, or try modifying it to use ST7799.

The audio board must be mounted between the Kaluga and the LCD, it provides the
```

```
I2C pull-ups(!)
"""

import board
import busio
import displayio
from adafruit_ili9341 import ILI9341
import adafruit_ov2640

# Pylint is unable to see that the "size" property of OV2640_GrandCentral exists
# pylint: disable=attribute-defined-outside-init

# Release any resources currently in use for the displays
displayio.release_displays()

spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi, command=board.LCD_D_C, chip_select=board.LCD_CS, reset=board.LCD_RST
)
display = ILI9341(display_bus, width=320, height=240, rotation=90)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

cam.flip_x = False
cam.flip_y = True
pid = cam.product_id
ver = cam.product_version
print(f"Detected pid={pid:x} ver={ver:x}")
# cam.test_pattern = True

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.BGR565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g

display.auto_refresh = False
while True:
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)

cam.deinit()
```

Your **CIRCUITPY** drive should resemble the screenshot below

You should have in / of the **CIRCUITPY** drive:

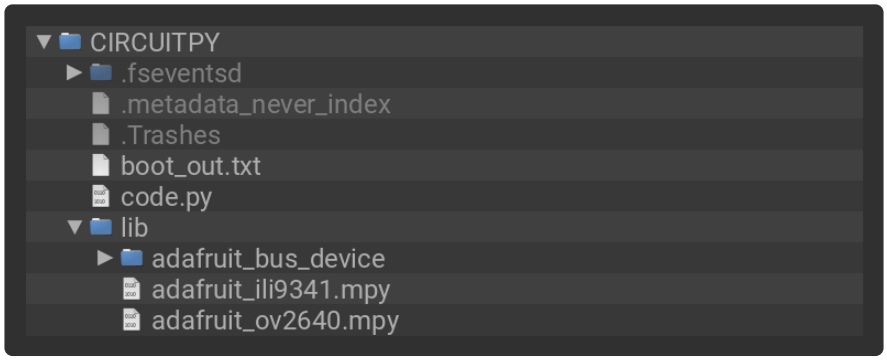
- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov2640.mpy**
- **adafruit_ili9341.mpy**

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.4 or newer.

If the image does not fill the whole display, try removing `rotation=90` from the line beginning `display = ILI9341`. If it does not appear at all or is in reverse video, try the example for the st7789 display.



Kaluga 1.3 with OV2640 and st7789 display (CircuitPython 7)

This code is for CircuitPython 7. Revised code will be required for CircuitPython 8.

If you have a Kaluga 1.3 board with an ili9341 LCD display, use the project below.

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

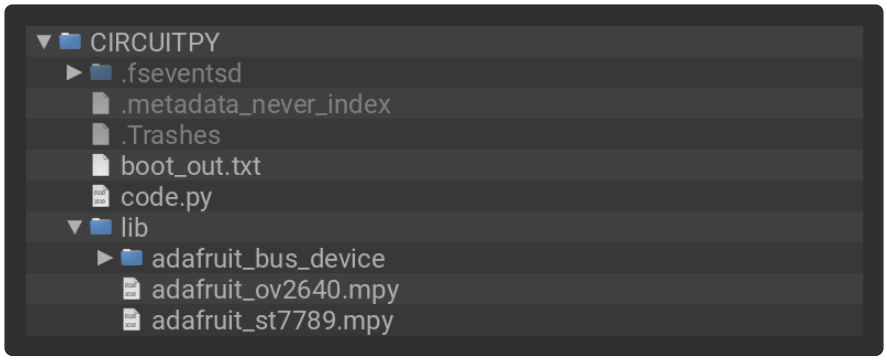
And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov2640.mpy**
- **adafruit_st7789.mpy**

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.4 or newer.

If the image does not appear at all or is in reverse video, try the example for the ili9341. display.

The author did not have a Kaluga with an st7789 display, so this example is untested.




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  This demo is for the ili9341.  There is no marking to distinguish the
two chips.  If the visible portion of the display's flexible cable has a bunch
of straight lines, it may be an ili9341.  If it has a bunch of wiggly traces,
it may be an st7789.  If in doubt, try both demos.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)
"""

import board
import busio
import displayio
from adafruit_st7789 import ST7789
import adafruit_ov2640

# Pylint is unable to see that the "size" property of OV2640_GrandCentral exists
# pylint: disable=attribute-defined-outside-init
```

```
# Release any resources currently in use for the displays
displayio.release_displays()

spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi, command=board.LCD_D_C, chip_select=board.LCD_CS, reset=board.LCD_RST
)
display = ST7789(
    display_bus, width=320, height=240, rotation=90, reverse_bytes_in_word=True
)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

# cam.flip_x = False
# cam.flip_y = True
pid = cam.product_id
ver = cam.product_version
print(f"Detected pid={pid:x} ver={ver:x}")
# cam.test_pattern = True

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.BGR565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g

display.auto_refresh = False
while True:
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)
    print(".")

cam.deinit()
```

Kaluga 1.3 with OV7670 and ili9341 display

This code is for CircuitPython 7. Revised code will be required for CircuitPython 8.

Take the assembled Kaluga board stack (all three boards) and attach the camera at the dedicated header, making sure the pins are inserted properly.

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

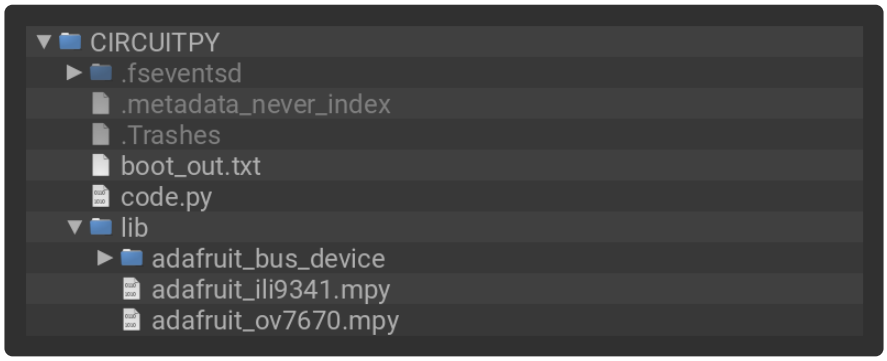
- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov7670.mpy**
- **adafruit_ili9341.mpy**

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.4 or newer.

If the image does not fill the whole display, try removing `rotation=90` from the line beginning `display = ILI9341`. If it does not appear at all or is in reverse video, try the example for the st7789 display.




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.  It probably won't work on v1.2 without modification.

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  Furthermore, there are at least 2 ILI9341 variants, one of which needs
rotation=90!  This demo is for the ili9341.  If the display is garbled, try adding
```

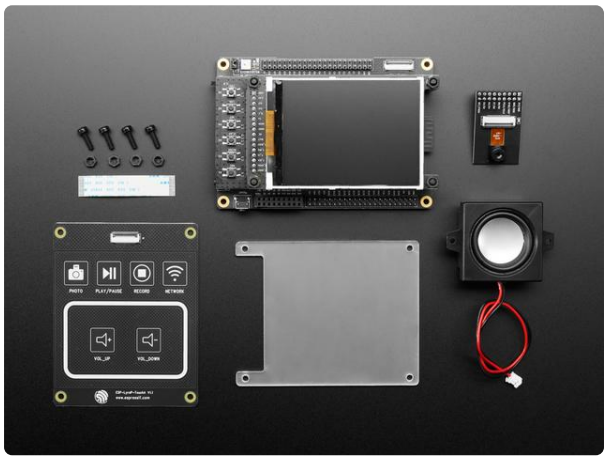
```
rotation=90, or try modifying it to use ST7799.

The camera included with the Kaluga development kit is the incompatible OV2640,
it won't work.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)
"""

import time
import board
import busio
import displayio
from adafruit_ili9341 import ILI9341
from adafruit_ov7670 import (  # pylint: disable=unused-import
    OV7670,
    OV7670_TEST_PATTERN_COLOR_BAR,
    OV7670_SIZE_DIV2,
    OV7670_NIGHT_MODE_2,
)

# Pylint is unable to see that the "size" property of OV7670_GrandCentral exists
# pylint: disable=attribute-defined-outside-init

# Release any resources currently in use for the displays
displayio.release_displays()

spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi, command=board.LCD_D_C, chip_select=board.LCD_CS, reset=board.LCD_RST
)
display = ILI9341(display_bus, width=320, height=240)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = OV7670(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
)

cam.size = OV7670_SIZE_DIV2
cam.flip_x = False
cam.flip_y = True
pid = cam.product_id
ver = cam.product_version
print(f"Detected pid={pid:x} ver={ver:x}")
# cam.test_pattern = OV7670_TEST_PATTERN_COLOR_BAR

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.RGB565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g

t0 = time.monotonic_ns()
display.auto_refresh = False
while True:
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)
```

```
    t1 = time.monotonic_ns()
    print("fps", 1e9 / (t1 - t0))
    t0 = t1

cam.deinit()
```

Adapting to other ESP32-S2 boards

By selecting appropriate pins, you can adapt the example to work on other RP2040 boards:

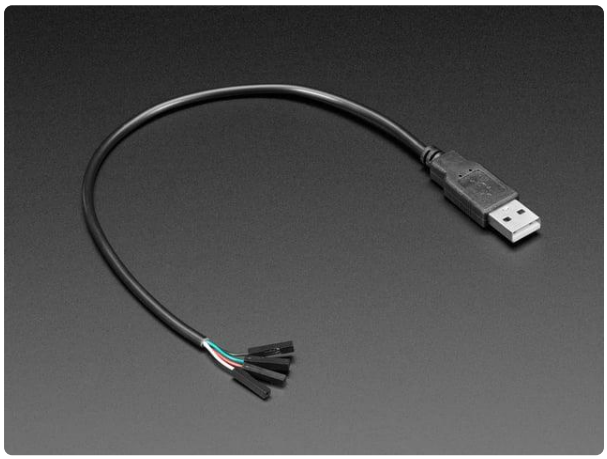
- **mclk**, **pclk**, **vsync**, **href**: Free choice of any pin
- **reset**, **shutdown**: Free choice of any pin. Can omit one or both, but the initialization sequence is less reliable.
- **d0**...**d7**: Free choice of any pin



ESP32-S2 Kaluga Dev Kit featuring ESP32-S2 WROVER

The ESP32-S2-Kaluga-1 kit is a full featured development kit by Espressif for the ESP32-S2 that comes with everything but the kitchen sink! From TFTs to touch panels,...

https://www.adafruit.com/product/4729



USB Type A Plug Breakout Cable with Premium Female Jumpers

If you'd like to connect a USB-capable chip to your USB host, this cable will make the task very simple. There is no converter chip in this cable! Its basically a...

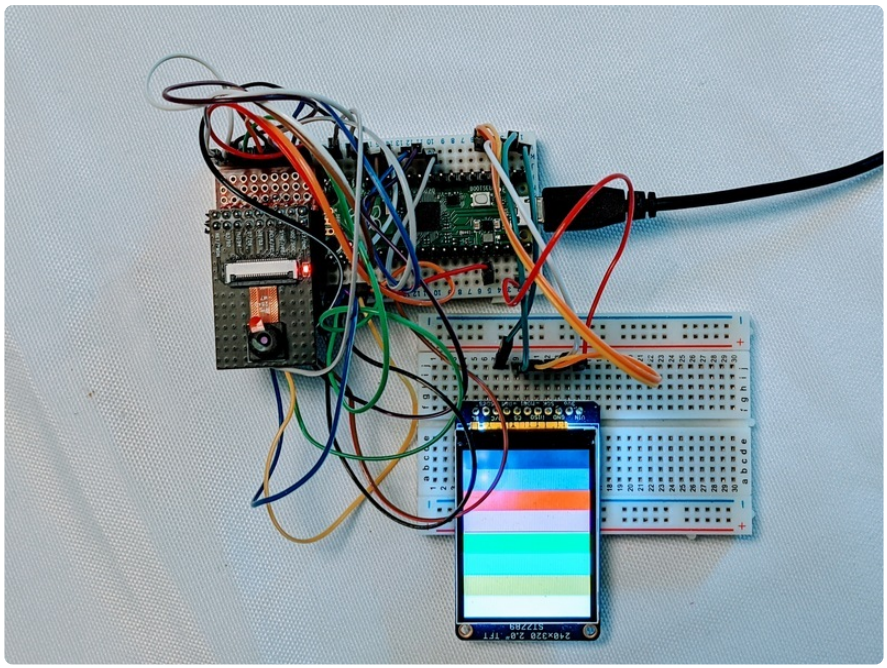
https://www.adafruit.com/product/4448

USB Extension Cable - 3 meters / 10 ft long

This handy USB extension cable will make it easy for you to extend your USB cable when it won't reach. The connectors are gold plated for years of reliability. We use these handy...

https://www.adafruit.com/product/993

Raspberry Pi Pico Wiring

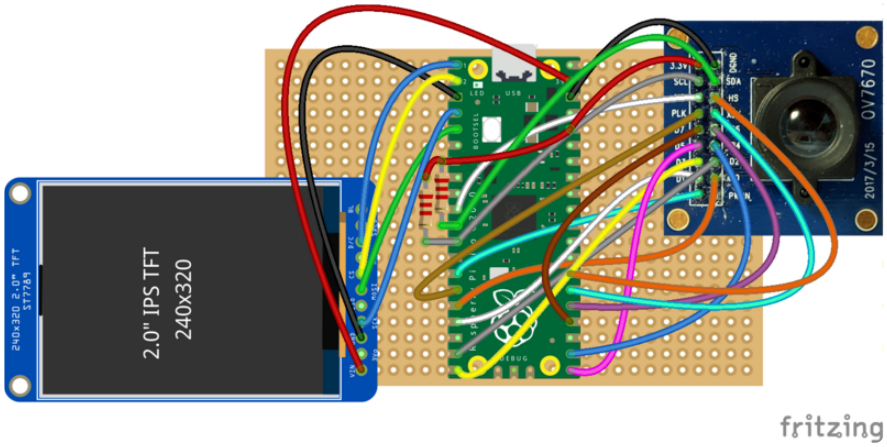


Wiring

There's no ready-made breakout board for the OV cameras and the Raspberry Pi, so get ready to do some wiring.

This diagram shows the many connections needed. Continue below for a list.

This technique is very advanced, requires a lot of wires, and is documented for curiousity-sake. We don't provide support for folks wiring stuff up this way.



Because the camera has two rows of connections that are 0.100 apart, a standard solderless breadboard doesn't work very well. Use a solderable breadboard or perfboard. If your solderable perfboard has a gap down the middle, you can use an IDC Breakout Helper, but these do not work well with solderless breadboards.

You can also use jumper wires (M-F to go from breadboard to camera, or F-F to go from pico pin header to camera), which is also nice because it gives you some flexibility to orient the camera differently than the LCD.

Power & Ground

- Connect **GND** of LCD, Pico, and Camera
- Connect **3V3** from Pico to Camera
- Connect **3V3** from Pico to I2C pull-up resistors (×2)
- Connect **VSYS** from Pico to LCD **VIN**

LCD Connections

- Connect Pico **GP2** to LCD **SCK**
- Connect Pico **GP3** to LCD **MOSI**
- Connect Pico **GP0** to LCD **D/C**
- Connect Pico **GP1** to LCD **CS**

I2C Connections

- Connect one I2C pull-up resistor to Pico **GP8**
- Connect the other I2C pull-up resistor to Pico **GP9**
- Connect Pico **GP8** to Camera **SDA**
- Connect Pico **GP9** to Camera **SCL**

Camera Control Connections

- Connect Pico **GP7** to Camera **VSYNC**
- Connect Pico **GP10** to Camera **RESET**
- Connect Pico **GP11** to Camera **CLOCK**
- Connect Pico **GP20** to Camera **MCLK**
- Connect Pico **GP21** to Camera **HREF**

Camera Data Connections

- Connect Pico **GP12** to Camera **D0**
- Connect Pico **GP13** to Camera **D1**
- Connect Pico **GP14** to Camera **D2**
- Connect Pico **GP15** to Camera **D3**
- Connect Pico **GP16** to Camera **D4**
- Connect Pico **GP17** to Camera **D5**
- Connect Pico **GP18** to Camera **D6**
- Connect Pico **GP19** to Camera **D7**

If your camera board includes pull-up resistors, you can omit them from the breadboard. Most OV2640 camera modules include them, while most OV7670 camera modules exclude them.

Make sure you can see the Pico's **CIRCUITPY** drive and connect to the REPL. Open the REPL and double check that `import imagecapture` works without showing an error. Then, copy the correct bundle to your device. It will automatically reload and start displaying the image from the camera on the built-in LCD.

For the OV2640 camera

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

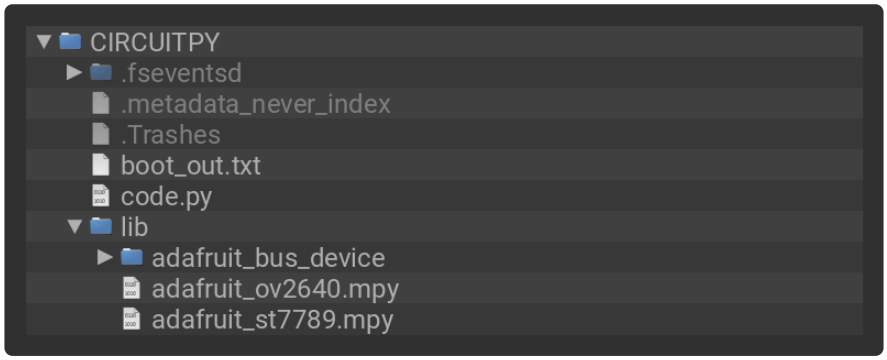
Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov2640.mpy**
- **adafruit_st7789.mpy**




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
Capture an image from the camera and display it on a supported LCD.
"""

import time
from displayio import (
    Bitmap,
    Group,
    TileGrid,
    FourWire,
    release_displays,
    ColorConverter,
    Colorspace,
)

from adafruit_st7789 import ST7789
import board
import busio
import digitalio
import adafruit_ov2640

release_displays()
# Set up the display (You must customize this block for your display!)
spi = busio.SPI(clock=board.GP2, MOSI=board.GP3)
display_bus = FourWire(spi, command=board.GP0, chip_select=board.GP1, reset=None)
display = ST7789(display_bus, width=320, height=240, rotation=270)
display.auto_refresh = False

# Ensure the camera is shut down, so that it releases the SDA/SCL lines,
# then create the configuration I2C bus

with digitalio.DigitalInOut(board.GP10) as reset:
    reset.switch_to_output(False)
    time.sleep(0.001)
    bus = busio.I2C(board.GP9, board.GP8)

# Set up the camera (you must customize this for your board!)
cam = adafruit_ov2640.OV2640(
```

```
    bus,
    data_pins=[
        board.GP12,
        board.GP13,
        board.GP14,
        board.GP15,
        board.GP16,
        board.GP17,
        board.GP18,
        board.GP19,
    ],  # [16]     [org] etc
    clock=board.GP11,  # [15]     [blk]
    vsync=board.GP7,  # [10]     [brn]
    href=board.GP21,  # [27/o14] [red]
    mclk=board.GP20,  # [16/o15]
    shutdown=None,
    reset=board.GP10,
)  # [14]

width = display.width
height = display.height

cam.size = adafruit_ov2640.OV2640_SIZE_QQVGA
# cam.test_pattern = True
bitmap = Bitmap(cam.width, cam.height, 65536)

print(width, height, cam.width, cam.height)
if bitmap is None:
    raise SystemExit("Could not allocate a bitmap")

g = Group(scale=1, x=(width - cam.width) // 2, y=(height - cam.height) // 2)
tg = TileGrid(
    bitmap, pixel_shader=ColorConverter(input_colorspace=Colorspace.BGR565_SWAPPED)
)
g.append(tg)
display.root_group = g

display.auto_refresh = False
while True:
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)
```

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.0 or newer.

For the OV7670 camera

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

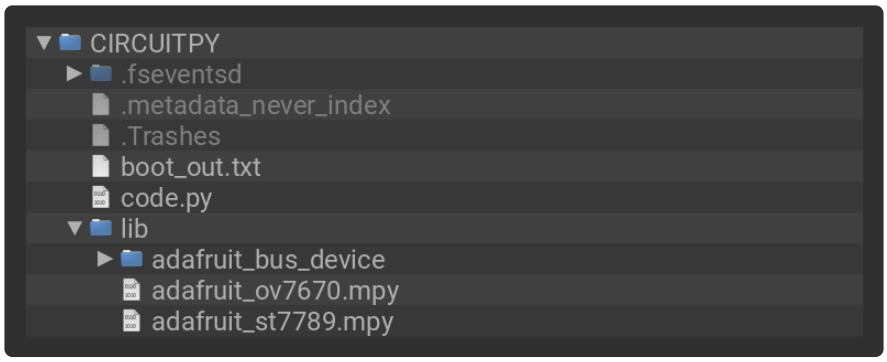
Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov7670.mpy**
- **adafruit_st7789.mpy**




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
Capture an image from the camera and display it on a supported LCD.
"""

import time
from displayio import (
    Bitmap,
    Group,
    TileGrid,
    FourWire,
    release_displays,
    ColorConverter,
    Colorspace,
)
from adafruit_st7789 import ST7789
import board
import busio
import digitalio
from adafruit_ov7670 import (
    OV7670,
    OV7670_SIZE_DIV1,
    OV7670_SIZE_DIV16,
)

# Set up the display (You must customize this block for your display!)
release_displays()
spi = busio.SPI(clock=board.GP2, MOSI=board.GP3)
display_bus = FourWire(spi, command=board.GP0, chip_select=board.GP1, reset=None)
display = ST7789(display_bus, width=320, height=240, rotation=270)
```

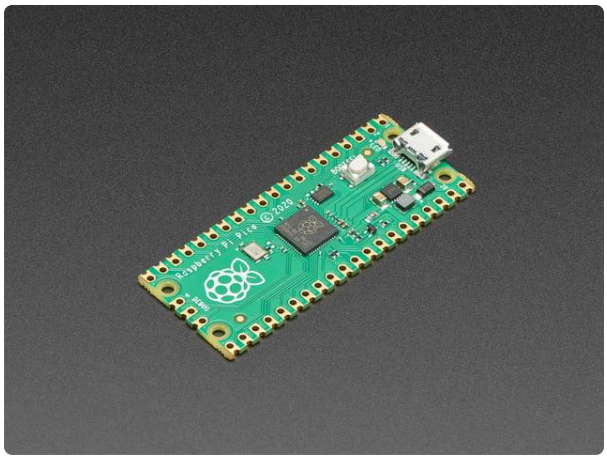
```
# Ensure the camera is shut down, so that it releases the SDA/SCL lines,
# then create the configuration I2C bus

with digitalio.DigitalInOut(board.GP10) as reset:
    reset.switch_to_output(False)
    time.sleep(0.001)
    bus = busio.I2C(board.GP9, board.GP8)

# Set up the camera (you must customize this for your board!)
cam = OV7670(
    bus,
    data_pins=[
        board.GP12,
        board.GP13,
        board.GP14,
        board.GP15,
        board.GP16,
        board.GP17,
        board.GP18,
        board.GP19,
    ],  # [16]     [org] etc
    clock=board.GP11,  # [15]     [blk]
    vsync=board.GP7,  # [10]     [brn]
    href=board.GP21,  # [27/o14] [red]
    mclk=board.GP20,  # [16/o15]
    shutdown=None,
    reset=board.GP10,
)  # [14]

width = display.width
height = display.height

# cam.test_pattern = OV7670_TEST_PATTERN_COLOR_BAR

bitmap = None
# Select the biggest size for which we can allocate a bitmap successfully, and
# which is not bigger than the display
for size in range(OV7670_SIZE_DIV1, OV7670_SIZE_DIV16 + 1):
    cam.size = size
    if cam.width > width:
        continue
    if cam.height > height:
        continue
    try:
        bitmap = Bitmap(cam.width, cam.height, 65536)
        break
    except MemoryError:
        continue

print(width, height, cam.width, cam.height)
if bitmap is None:
    raise SystemExit("Could not allocate a bitmap")

g = Group(scale=1, x=(width - cam.width) // 2, y=(height - cam.height) // 2)
tg = TileGrid(
    bitmap, pixel_shader=ColorConverter(input_colorspace=Colorspace.RGB565_SWAPPED)
)
g.append(tg)
display.root_group = g

t0 = time.monotonic_ns()
display.auto_refresh = False
while True:
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)
    t1 = time.monotonic_ns()
    print("fps", 1e9 / (t1 - t0))
    t0 = t1
```

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.0 or newer.

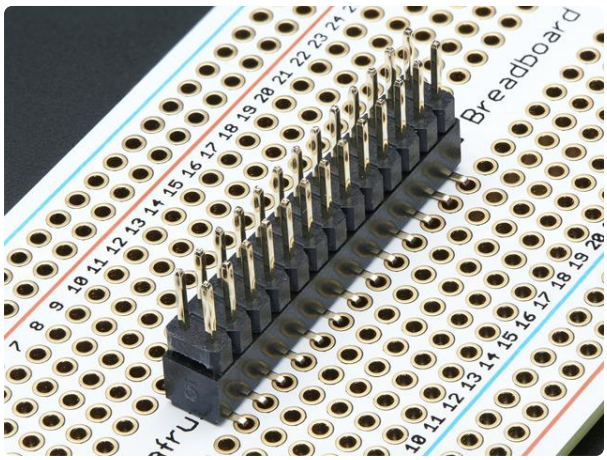
Adapting to other RP2040 boards

By selecting appropriate pins, you can adapt the example to work on other RP2040 boards which are supported by CircuitPython and have enough pins exposed for the connections below:

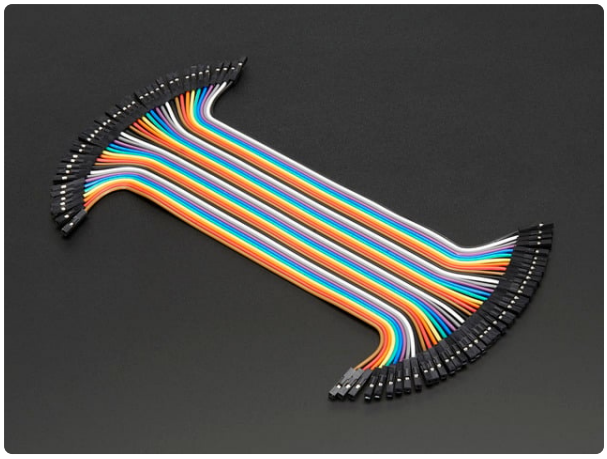
- **mclk**, **pclk**, **vsync**, **href**: Free choice of any pin
- **reset**, **shutdown**: Free choice of any pin. Can omit one or both, but the initialization sequence is less reliable.
- **d0**...**d7**: These 8 pins must be consecutive in the "**IO##**" ordering, so you could use **IO3**...**IO10**, **IO9**...**IO16**, etc.



Raspberry Pi Pico RP2040
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...
https://www.adafruit.com/product/4864



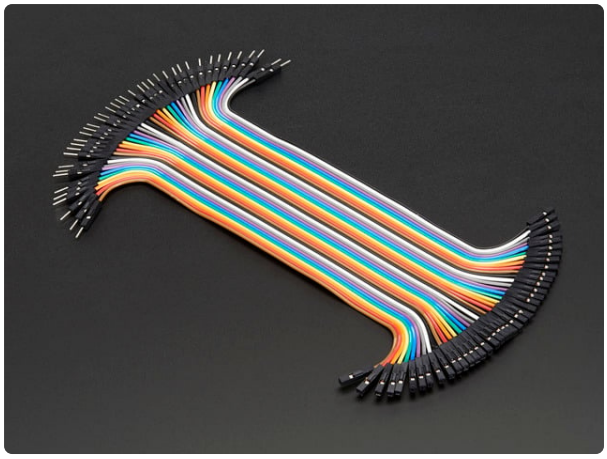
IDC Breakout Helper - 2x13 (26 pin)
The "poor woman's" Raspberry Pi cobbler! This combo of 2x13 pin (0.1 spaced) header and socket is for our popular GPIO...
https://www.adafruit.com/product/2101



Premium Female/Female Jumper Wires - 40 x 6"

Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires approximately 6" (150mm) long and come in a 'strip' of 40 (4 pieces...

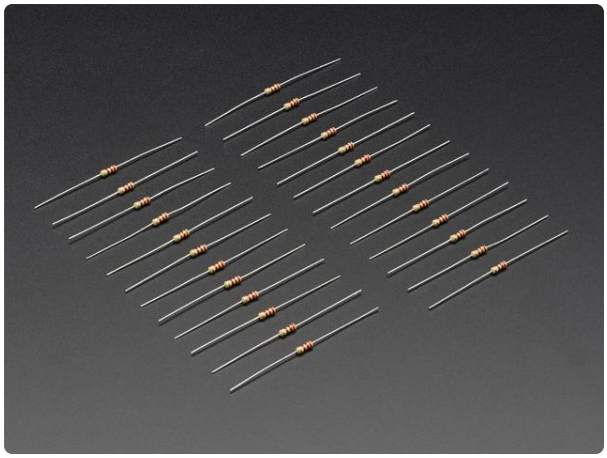
https://www.adafruit.com/product/266



Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...

https://www.adafruit.com/product/826

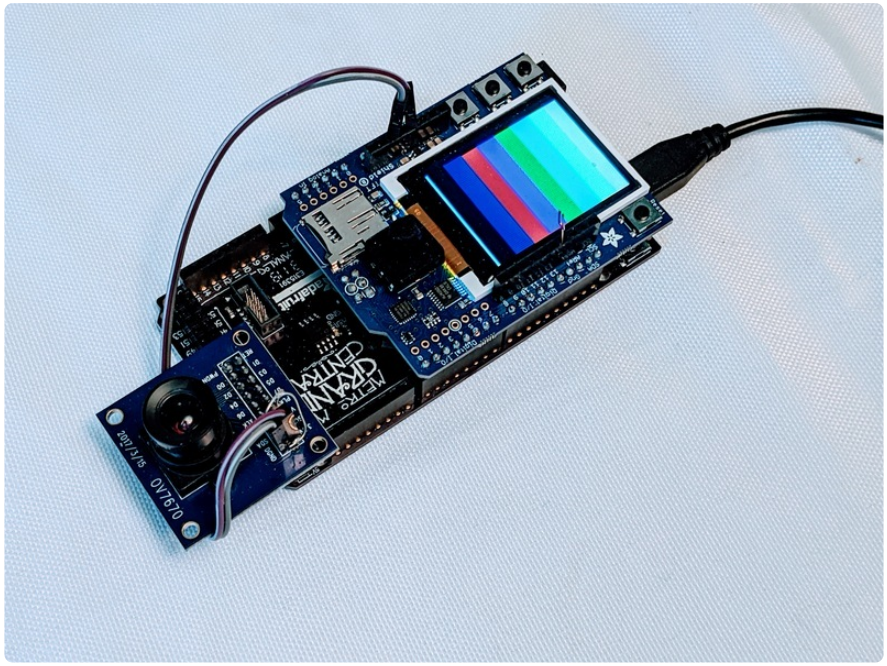


Through-Hole Resistors - 2.2K ohm 5% 1/4W - Pack of 25

ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you!This is a 25 Pack of 2.2K...

https://www.adafruit.com/product/2782

Grand Central M4 Wiring



We've found the ParallelImageCapture function to be very unstable under CircuitPython. It works sometimes with OV7670 and never or almost never with OV2640. If you need to use the SAM D5x/E5x microcontroller, consider using the Arduino version instead of CircuitPython.

Before you can use an OV7670 camera with the Grand Central M4, you have to perform some hardware modifications which are detailed on their own page. (https://adafru.it/TAJ) These modifications are not easily reversible and will make it difficult to use the camera modules on other boards.

Then, position the camera so that it sticks off the right side of the Grand Central PCB and the SDA/SCL pins insert into the third row of the header and insert it carefully.

Align the pins of the TFT Shield and insert it into the Grand Central too.

Connect GND and 3V3 from the TFT Shield to the modified pins of the OV7670.

Now, make sure you can see the Grand Central's **CIRCUITPY** drive and connect to the REPL. Open the REPL and double check that `import imagecapture` works without showing an error. Then, copy the correct bundle to your device. It will automatically reload and start displaying the image from the camera on the built-in LCD.

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov7670.mpy**
- **adafruit_st7789.mpy**

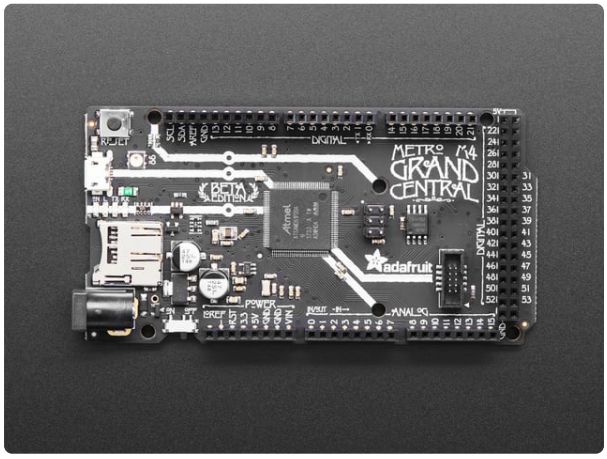
CircuitPython will automatically reload and begin showing the image on the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.0 or newer.

Adapting to other SAM D5x/E5x boards

By selecting appropriate pins, you can adapt the example to work on other SAM D5x/E5x boards supported by CircuitPython which expose enough pins for the connections below:

- **mclk:** Free choice of any PWM pin
- **pclk**, **vsync**, **href**: Only the specific PCC pins may be used.

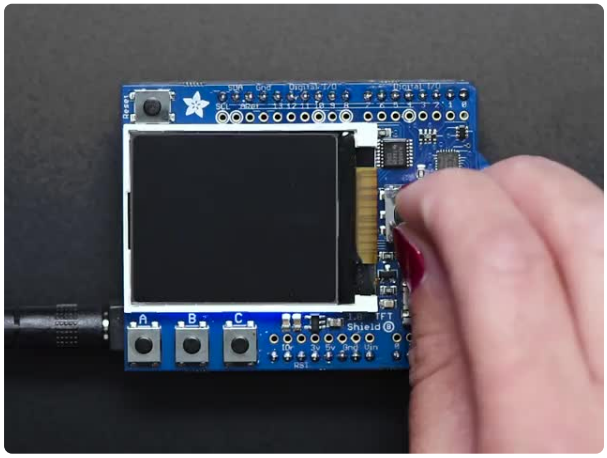
- **reset**, **shutdown**: Free choice of any pin. Can omit one or both, but the initialization sequence is less reliable.
- **d0**...**d7**: Only the specific PCC pins may be used.



Adafruit Grand Central M4 Express featuring the SAMD51

Are you ready? Really ready? Cause here comes the Adafruit Grand Central featuring the Microchip ATSAMD51. This dev board is so big, it's not...

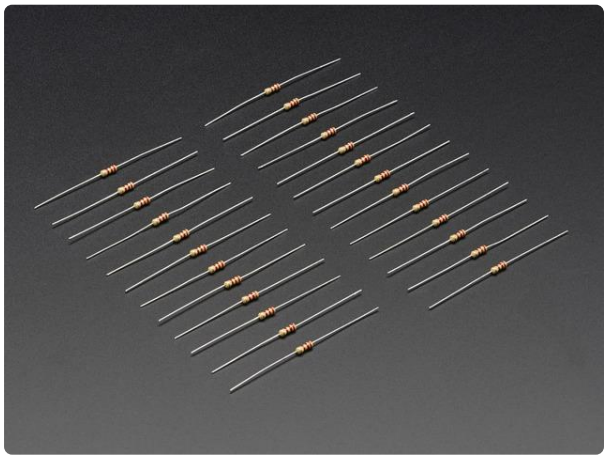
https://www.adafruit.com/product/4064



Adafruit 1.8" Color TFT Shield w/microSD and Joystick

This lovely little shield is the best way to add a small, colorful and bright display to any project. We took our popular 1.8" TFT breakout board and remixed it into an Arduino...

https://www.adafruit.com/product/802



Through-Hole Resistors - 2.2K ohm 5% 1/4W - Pack of 25

ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you!This is a 25 Pack of 2.2K...

https://www.adafruit.com/product/2782

[Hook-up Wire Spool Set - 22AWG Solid Core - 6 x 25 ft]
Perfect for bread-boarding, free wiring, etc. This box contains 6 spools of solid-core wire. The wire is easy to solder to and when bent it keeps its shape pretty well. We like to have...
https://www.adafruit.com/product/1311

Working with Image Data

RGB Data

When the OV7670 is working in RGB mode (the default), it is in a 16-bit format called "RGB565-swapped". This means that every pixel is treated as a 16-bit number, with the left and right 8 bits "swapped":

The OV2640 uses a 16-bit format called "BGR565-swapped", which switches the positions of the red and blue values within the pixel.

Happily, displayio's ColorConverter is able to deal with this format natively, but it is tough to write efficient Python code to work with it. In some circumstances, the CircuitPython version of the `ulab` library can help.

In the examples before now, the "capture" operation worked with a bitmap object, but it can also work with a `ulab` array:

```
from ulab import numpy as np

arr = np.zeros((80, 60), dtype=np.uint16)
camera.capture(arr)
arr.byteswap(inplace=True)
```

After using **byteswap**, the order of the values within the pixel is now:

making it possible to extract an individual red, green, or blue value with bit shifts (this code is for RGB565):

```
def R(pixel):
    return pixel >> 11

def G(pixel):
    return (pixel & 0b11111100000) >> 5

def B(pixel):
    return pixel & 0b11111

print("The green value of the pixel at (0,0) is", G(arr[0,0]))
```

Wholesale modifications of the pixel data can be done with ulab. For instance, to invert colors in the whole image,

```
arr[:] = ~arr
```

That's exactly what the following program does on the Espressif Kaluga with OV2640.

Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov2640.mpy**
- **adafruit_ili9341.mpy**

CircuitPython will automatically reload and begin showing the image from the camera on the LCD. If it doesn't, you can open up the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.0 or newer.

If the image does not fill the whole display, try removing `rotation=90` from the line beginning `display = ILI9341`. If it does not appear at all or is in reverse video, try adapting the example to use the st7789 display.

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
```

```
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.  It probably won't work on v1.2 without modification.

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  Furthermore, there are at least 2 ILI9341 variants, one of which needs
rotation=90!  This demo is for the ili9341.  If the display is garbled, try adding
rotation=90, or try modifying it to use ST7799.

The camera included with the Kaluga development kit is the incompatible OV2640,
it won't work.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)
"""

import board
import busio
import displayio
from adafruit_ili9341 import ILI9341
import ulab.numpy as np
import adafruit_ov2640

# Pylint is unable to see that the "size" property of OV2640_GrandCentral exists
# pylint: disable=attribute-defined-outside-init

# Release any resources currently in use for the displays
displayio.release_displays()

spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi, command=board.LCD_D_C, chip_select=board.LCD_CS, reset=board.LCD_RST
)
display = ILI9341(display_bus, width=320, height=240, rotation=90)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

cam.flip_x = False
cam.flip_y = True
pid = cam.product_id
ver = cam.product_version
print(f"Detected pid={pid:x} ver={ver:x}")
cam.test_pattern = True

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
arr = np.frombuffer(bitmap, dtype=np.uint16)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.RGB565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g
```

```
display.auto_refresh = False
while True:
    cam.capture(bitmap)
    arr[:] = ~arr  # Invert every pixel in the bitmap, via the array
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)

cam.deinit()
```

YUV Data

YUV is another representation of image data. Y represents the luminance (brightness) of a pixel, while U and V represent the color information. Wikipedia has an article about YUV (https://adafru.it/TAK), if you'd like to know more.

The most useful thing about YUV data is that it allows easily extracting a greyscale image, by using the data from every other byte. That is how this "image in a terminal window" example works on the Espressif Kaluga with OV2640.

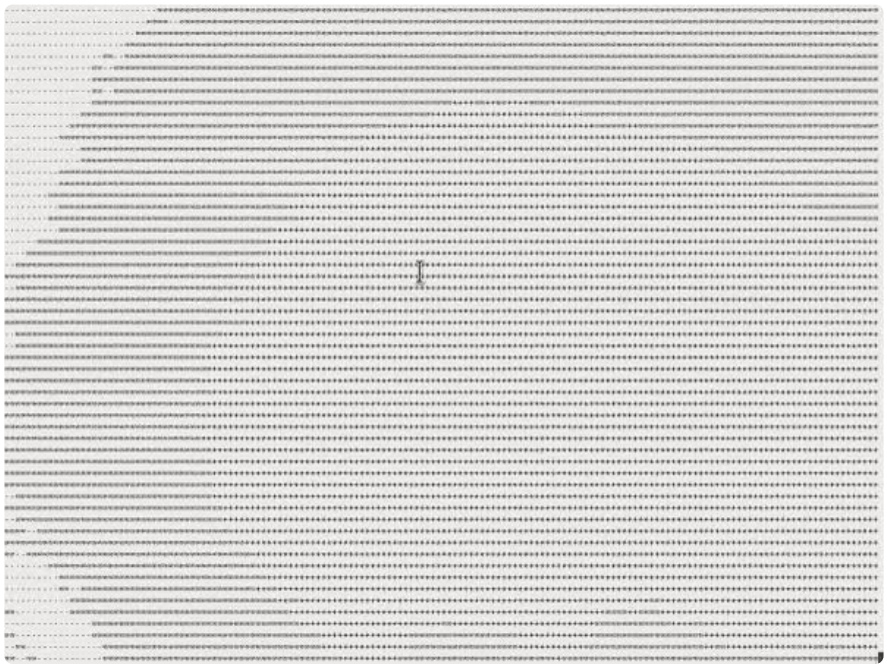
The OV2640 can be placed in YUV mode by assigning a property:

```
cam.colorspace = adafruit_ov2640.OV2640_COLOR_YUV
```

The OV7670 is similar:

```
cam.colorspace = adafruit_ov7670.OV7670_COLOR_YUV
```

To demonstrate the YUV mode, the simpletest demo converts a low resolution camera image into lo-fi ASCII art.



Click the **Download Project Bundle** button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should resemble the image.

You should have in / of the **CIRCUITPY** drive:

- **code.py**

And in the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_bus_device**
- **adafruit_ov2640.mpy**

CircuitPython will automatically reload. Connect to the REPL and the image will be shown in the finest 3-bit ASCII art. If it doesn't, use the REPL to diagnose what went wrong. Double check that you copied all the files from the bundle, and that you have a compatible build of CircuitPython installed, 7.0.0-beta.0 or newer.

If the REPL updates very slowly, one trick is to reset the Kaluga so that it forgets about the LCD display if you ran one of the demos that uses the LCD. Updating the LCD display is much slower than sending data over the USB CDC connection.

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""Capture an image from the camera and display it as ASCII art.

This demo is designed to run on the Kaluga, but you can adapt it
to other boards by changing the constructors for `bus` and `cam`
appropriately.

The camera is placed in YUV mode, so the top 8 bits of each color
value can be treated as "greyscale".

It's important that you use a terminal program that can interpret
"ANSI" escape sequences.  The demo uses them to "paint" each frame
on top of the prevous one, rather than scrolling.

Remember to take the lens cap off, or un-comment the line setting
the test pattern!
"""

import sys
import time

import busio
import board
```

```
import adafruit_ov2640

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QQVGA,
)
cam.colorspace = adafruit_ov2640.OV2640_COLOR_YUV
cam.flip_y = True
# cam.test_pattern = True

buf = bytearray(2 * cam.width * cam.height)
chars = b" .:-=+*#%@"
remap = [chars[i * (len(chars) - 1) // 255] for i in range(256)]

width = cam.width
row = bytearray(2 * width)

sys.stdout.write("\033[2J")
while True:
    cam.capture(buf)
    for j in range(cam.height // 2):
        sys.stdout.write(f"\033[{j}H")
        for i in range(cam.width // 2):
            row[i * 2] = row[i * 2 + 1] = remap[buf[4 * (width * j + i)]]
        sys.stdout.write(row)
        sys.stdout.write("\033[K")
    sys.stdout.write("\033[J")
    time.sleep(0.05)
```

Test modes

These cameras have a test mode which shows color bars.

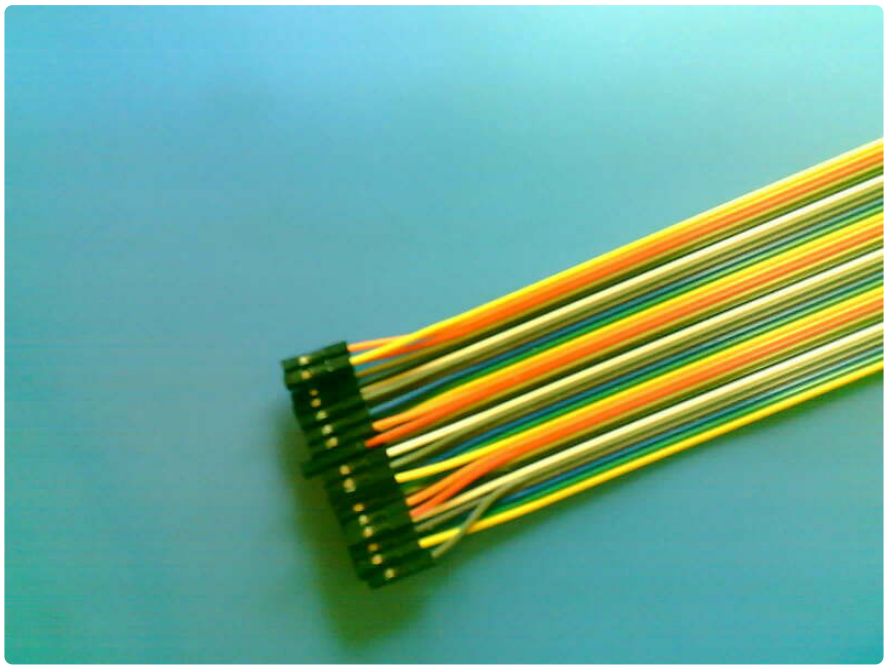
You can activate the test pattern mode on the OV2640 camera like so:

```
cam.test_pattern = True
```

It's a little different on the OV7670:

```
cam.test_pattern = adafruit_ov7670.OV7670_TEST_PATTERN_COLOR_BAR
```

Capturing JPEG data (OV2640)



The OV2640 camera module can also capture JPEG images up to 2 megapixels (1600x1200 pixels). This still requires a large buffer (of width×height÷5 bytes, or 384,000 bytes for a 2-megapixel image), so the demo below is coded for the Kaluga development board together with the MicroSD card breakout board+.

CircuitPython doesn't encode or decode the JPEG image itself, it just uses a JPEG-encoded file produced by the camera and stores it on the SD card.

Make the following connections for the SD card breakout:

- **IO18** to **CLK**
- **IO14** to **DI**
- **IO17** to **DO**
- **IO12** to **CS**
- **GND** to **GND**
- **5V** to **5V**

While the demo runs, it will show a live image on the LCD. When you hold the REC button, it will save the picture as a jpeg image to the inserted SD card. Note that because the REC button is only polled when the screen is not updating, you have to **hold** it, not just quickly **press** it.

The exposure in JPEG mode doesn't match what's shown on the LCD. We hope a future enhancement of the OV2640 library will improve this.

Code

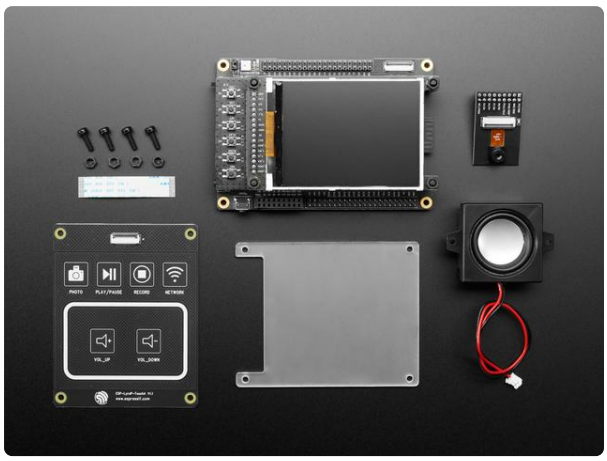
This code is for CircuitPython 7. Revised code will be required for CircuitPython 8.

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
Display an image on the LCD, then record an image when the REC button is pressed/
held.

The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  Furthermore, there are at least 2 ILI9341 variants, one of which needs
rotation=90!  This demo is for the ili9341.  If the display is garbled, try adding
rotation=90, or try modifying it to use ST7799.

This example also requires an SD card breakout wired as follows:
 * IO18: SD Clock Input
 * IO17: SD Serial Output (MISO)
 * IO14: SD Serial Input (MOSI)
 * IO12: SD Chip Select

Insert a CircuitPython-compatible SD card before powering on the Kaluga.
Press the "Record" button on the audio daughterboard to take a photo.
"""

import os

import analogio
import board
import busio
import displayio
import sdcardio
import storage
from adafruit_ili9341 import ILI9341
import adafruit_ov2640

V_MODE = 1.98
V_RECORD = 2.41

a = analogio.AnalogIn(board.IO6)

# Release any resources currently in use for the displays
displayio.release_displays()

spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi, command=board.LCD_D_C, chip_select=board.LCD_CS, reset=board.LCD_RST
)
display = ILI9341(display_bus, width=320, height=240, rotation=90)
```

```
bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

cam.flip_x = False
cam.flip_y = True
pid = cam.product_id
ver = cam.product_version
print(f"Detected pid={pid:x} ver={ver:x}")
# cam.test_pattern = True

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.BGR565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g

display.auto_refresh = False

sd_spi = busio.SPI(clock=board.IO18, MOSI=board.IO14, MISO=board.IO17)
sd_cs = board.IO12
sdcard = sdcardio.SDCard(sd_spi, sd_cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")


def exists(filename):
    try:
        os.stat(filename)
        return True
    except OSError:
        return False


_image_counter = 0


def open_next_image():
    global _image_counter  # pylint: disable=global-statement
    while True:
        filename = f"/sd/img{_image_counter:04d}.jpg"
        _image_counter += 1
        if exists(filename):
            continue
        print("#", filename)
        return open(filename, "wb")  # pylint: disable=consider-using-with


def capture_image():
    old_size = cam.size
    old_colorspace = cam.colorspace

    try:
        cam.size = adafruit_ov2640.OV2640_SIZE_UXGA
        cam.colorspace = adafruit_ov2640.OV2640_COLOR_JPEG
```

```
        b = bytearray(cam.capture_buffer_size)
        jpeg = cam.capture(b)

        print(f"Captured {len(jpeg)} bytes of jpeg data")
        with open_next_image() as f:
            f.write(jpeg)
    finally:
        cam.size = old_size
        cam.colorspace = old_colorspace


display.auto_refresh = False
while True:
    a_voltage = a.value * a.reference_voltage / 65535  # pylint: disable=no-member
    record_pressed = abs(a_voltage - V_RECORD) < 0.05
    if record_pressed:
        capture_image()
    cam.capture(bitmap)
    bitmap.dirty()
    display.refresh(minimum_frames_per_second=0)
```

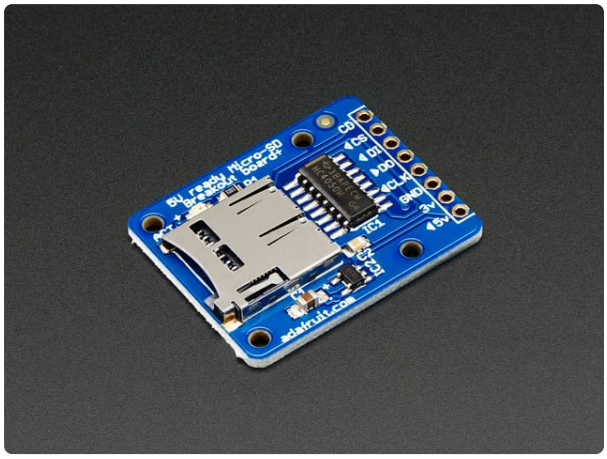
Parts



ESP32-S2 Kaluga Dev Kit featuring ESP32-S2 WROVER

The ESP32-S2-Kaluga-1 kit is a full featured development kit by Espressif for the ESP32-S2 that comes with everything but the kitchen sink! From TFTs to touch panels,...

https://www.adafruit.com/product/4729



MicroSD card breakout board+

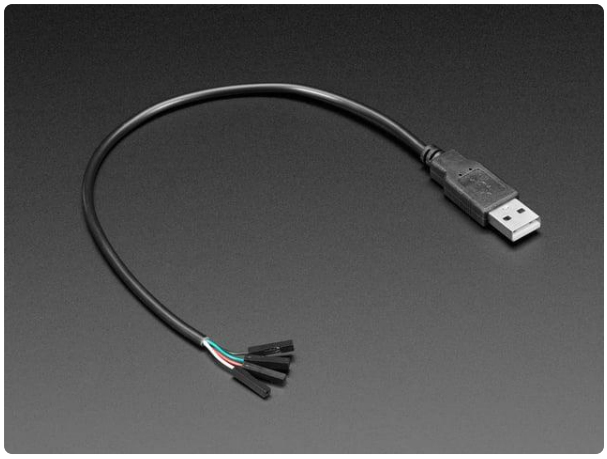
Not just a simple breakout board, this microSD adapter goes the extra mile - designed for ease of use.Onboard 5v->3v regulator provides 150mA for power-hungry...

https://www.adafruit.com/product/254

8GB Class 10 SD/MicroSD Memory Card - SD Adapter Included

Add mega-storage in a jiffy using this 8 GB micro-SD card. It comes with a SD adapter so you can use it with any of our shields or adapters! Preformatted to FAT so it works out of the...

https://www.adafruit.com/product/2692



USB Type A Plug Breakout Cable with Premium Female Jumpers

If you'd like to connect a USB-capable chip to your USB host, this cable will make the task very simple. There is no converter chip in this cable! Its basically a...

https://www.adafruit.com/product/4448

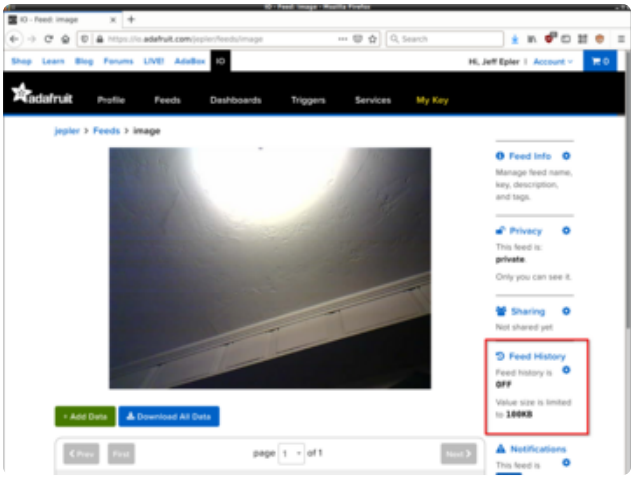
USB Extension Cable - 3 meters / 10 ft long

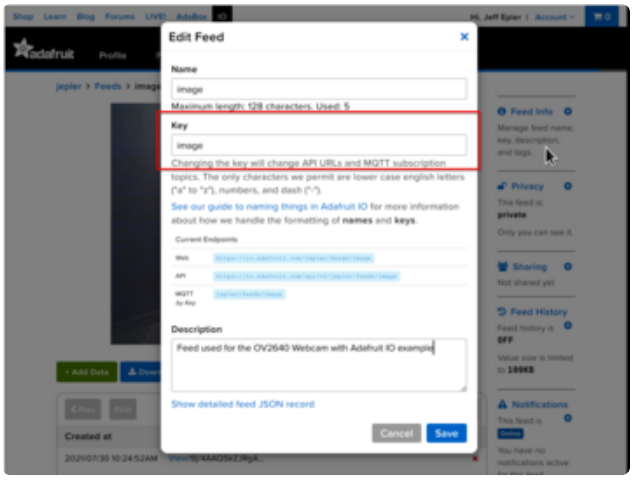
This handy USB extension cable will make it easy for you to extend your USB cable when it won't reach. The connectors are gold plated for years of reliability. We use these handy...

https://www.adafruit.com/product/993

OV2640 Webcam with Adafruit IO

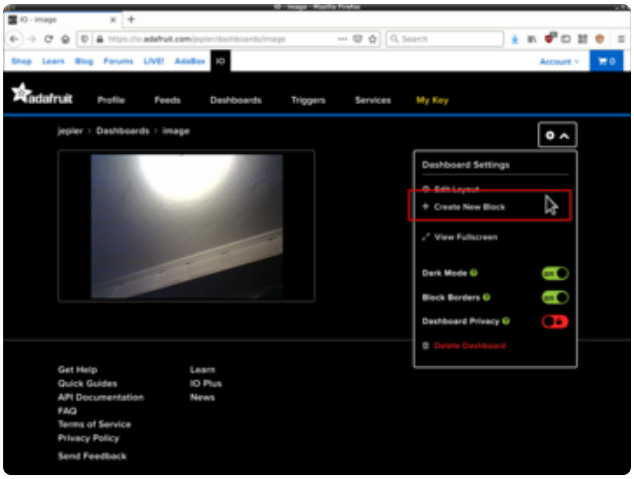
With a WiFi-enabled board like the Espressif Kaluga, you can upload your image data to Adafruit IO (https://adafru.it/fsU). We made sure that this example works with the free version, so you can try it out even if you haven't upgraded to a Plus subscription yet. New to Adafruit IO? Start with this guide (https://adafru.it/EYj) to learn the basics.

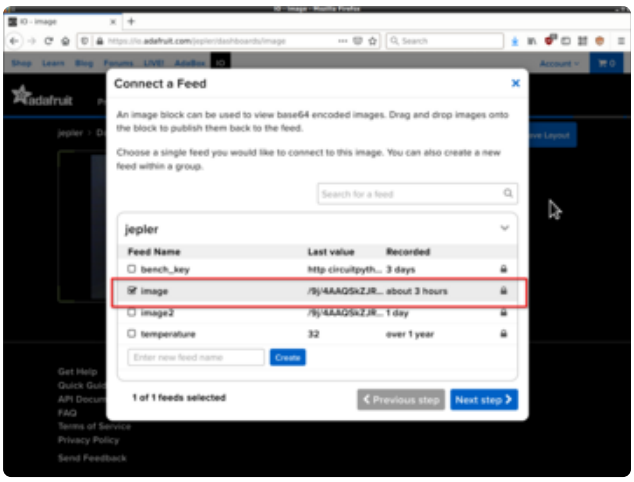




Set up the IO Feed

Create a feed called "image" and then set "Feed History" to "OFF". This allows storage of data up to 100kB, which is plenty to upload JPEGs at 640x480 resolution. (You can choose another feed name but you'll need to make sure that Adafruit IO's "key" for the feed matches what you use in your CircuitPython program!)





Set up the IO Dashboard

Create a new dashboard. Click the gear icon and then "Create New Block". Choose the camera icon ("image") and then select your feed named "image". Enter a block title if you like, and click "Create Block".

Secrets File Setup for Adafruit IO

If you don't have a **secrets.py** file in your **CIRCUITPY** drive yet, create one and add the information about your WiFi connection.

Then, add the following code to your **secrets.py** file, **replacing `_your_adafruit_io_username` with your Adafruit IO username.**

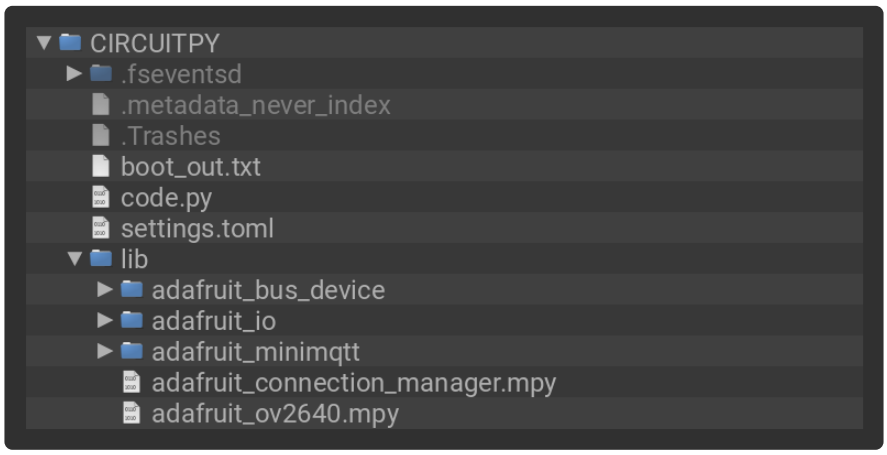
Then, **replace `_your_big_huge_super_long_aio_key_` with your Adafruit IO Active Key**.

```
secrets = {
    'ssid' : '_your_wifi_ssid_',
    'password' : '_your_wifi_password_',
    'aio_username' : '_your_adafruit_io_username_',
    'aio_key' : '_your_big_huge_super_long_aio_key_',
}
```

Make sure you **save this file** before proceeding as **secrets.py** in the root directory of your board **CIRCUITPY** drive.

Upload the code

Grab the Bundle below and unzip it on your Espressif Kaluga's **CIRCUITPY** drive. It will automatically start the code and upload a 640x480 JPEG to Adafruit IO every 3 seconds or so.




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)

This example requires that your WIFI and Adafruit IO credentials be configured
in CIRCUITPY/secrets.py, and that you have created a feed called "image" with
history disabled.

The maximum image size is 100kB after base64 encoding, or about 65kB before
base64 encoding.  In practice, "SVGA" (800x600) images are typically around
40kB even though the "capture_buffer_size" (theoretical maximum size) is
(width*height/5) bytes or 96kB.
"""

import binascii
import ssl
import time
from secrets import secrets  # pylint: disable=no-name-in-module

import board
import busio
import wifi
import socketpool
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
import adafruit_ov2640

feed_name = "image"

print("Connecting to WIFI")
wifi.radio.connect(secrets["ssid"], secrets["password"])
pool = socketpool.SocketPool(wifi.radio)
```

```
print("Connecting to Adafruit IO")
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=secrets["aio_username"],
    password=secrets["aio_key"],
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)
mqtt_client.connect()
io = IO_MQTT(mqtt_client)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

cam.flip_x = False
cam.flip_y = False
cam.test_pattern = False

cam.size = adafruit_ov2640.OV2640_SIZE_SVGA
cam.colorspace = adafruit_ov2640.OV2640_COLOR_JPEG
jpeg_buffer = bytearray(cam.capture_buffer_size)
while True:
    jpeg = cam.capture(jpeg_buffer)
    print(f"Captured {len(jpeg)} bytes of jpeg data")

    # b2a_base64() appends a trailing newline, which IO does not like
    encoded_data = binascii.b2a_base64(jpeg).strip()
    print(f"Expanded to {len(encoded_data)} for IO upload")

    io.publish("image", encoded_data)

    print("Waiting 3s")
    time.sleep(3)
```

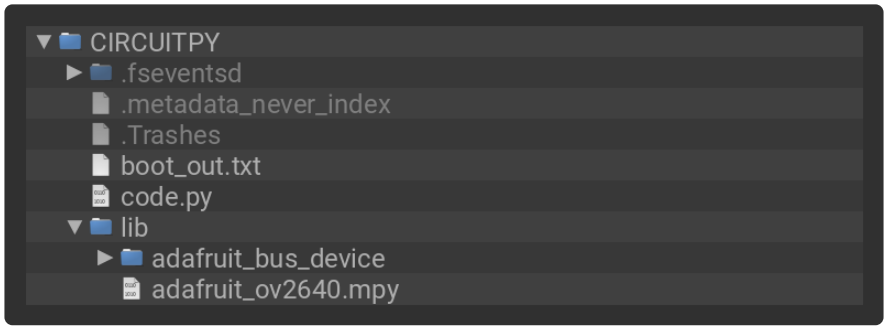
Working with BMP format data

You can also work with and save BMP format data from an OV2640 or OV7670 camera. Use this format if you need to do image processing within CircuitPython, or if your camera doesn't have a JPEG mode.

Make the following connections for the SD card breakout:

- **IO18** to **CLK**
- **IO14** to **DI**
- **IO17** to **DO**
- **IO12** to **CS**
- **GND** to **GND**
- **5V** to **5V**

While the demo runs, it will show a live image on the LCD. When you hold the REC button, it will save the picture as a BMP image to the inserted SD card. Note that because the REC button is only polled when the screen is not updating, you have to **hold** it, not just quickly **press** it.




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""
The Kaluga development kit comes in two versions (v1.2 and v1.3); this demo is
tested on v1.3.

The audio board must be mounted between the Kaluga and the LCD, it provides the
I2C pull-ups(!)

The v1.3 development kit's LCD can have one of two chips, the ili9341 or
st7789.  Furthermore, there are at least 2 ILI9341 variants, one of which needs
rotation=90!  This demo is for the ili9341.  If the display is garbled, try adding
rotation=90, or try modifying it to use ST7799.

This example also requires an SD card breakout wired as follows:
 * IO18: SD Clock Input
 * IO17: SD Serial Output (MISO)
 * IO14: SD Serial Input (MOSI)
 * IO12: SD Chip Select

Insert a CircuitPython-compatible SD card before powering on the Kaluga.
Press the "Record" button on the audio daughterboard to take a photo in BMP format.
"""

import os
import struct
import ulab.numpy as np

import analogio
import board
import busio
import displayio
import sdcardio
import storage
import adafruit_ov2640

# Nominal voltages of several of the buttons on the audio daughterboard
V_MODE = 1.98
V_RECORD = 2.41

a = analogio.AnalogIn(board.IO6)

# Release any resources currently in use for the displays
displayio.release_displays()
```

```
spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = displayio.FourWire(
    spi,
    command=board.LCD_D_C,
    chip_select=board.LCD_CS,
    reset=board.LCD_RST,
    baudrate=80_000_000,
)
_INIT_SEQUENCE = (
    b"\x01\x80\x80"  # Software reset then delay 0x80 (128ms)
    b"\xEF\x03\x03\x80\x02"
    b"\xCF\x03\x00\xC1\x30"
    b"\xED\x04\x64\x03\x12\x81"
    b"\xE8\x03\x85\x00\x78"
    b"\xCB\x05\x39\x2C\x00\x34\x02"
    b"\xF7\x01\x20"
    b"\xEA\x02\x00\x00"
    b"\xc0\x01\x23"  # Power control VRH[5:0]
    b"\xc1\x01\x10"  # Power control SAP[2:0];BT[3:0]
    b"\xc5\x02\x3e\x28"  # VCM control
    b"\xc7\x01\x86"  # VCM control2
    b"\x36\x01\x90"  # Memory Access Control
    b"\x37\x01\x00"  # Vertical scroll zero
    b"\x3a\x01\x55"  # COLMOD: Pixel Format Set
    b"\xb1\x02\x00\x18"  # Frame Rate Control (In Normal Mode/Full Colors)
    b"\xb6\x03\x08\x82\x27"  # Display Function Control
    b"\xF2\x01\x00"  # 3Gamma Function Disable
    b"\x26\x01\x01"  # Gamma curve selected
    b"\xe0\x0f\x0F\x31\x2B\x0C\x0E\x08\x4E\xF1\x37\x07\x10\x03\x0E\x09\x00"  # Set
Gamma
    b"\xe1\x0f\x00\x0E\x14\x03\x11\x07\x31\xC1\x48\x08\x0F\x0C\x31\x36\x0F"  # Set
Gamma
    b"\x11\x80\x78"  # Exit Sleep then delay 0x78 (120ms)
    b"\x29\x80\x78"  # Display on then delay 0x78 (120ms)
)

display = displayio.Display(
    display_bus, _INIT_SEQUENCE, width=320, height=240, auto_refresh=False
)

bus = busio.I2C(scl=board.CAMERA_SIOC, sda=board.CAMERA_SIOD)
cam = adafruit_ov2640.OV2640(
    bus,
    data_pins=board.CAMERA_DATA,
    clock=board.CAMERA_PCLK,
    vsync=board.CAMERA_VSYNC,
    href=board.CAMERA_HREF,
    mclk=board.CAMERA_XCLK,
    mclk_frequency=20_000_000,
    size=adafruit_ov2640.OV2640_SIZE_QVGA,
)

cam.flip_x = False
cam.flip_y = False
cam.test_pattern = False

g = displayio.Group(scale=1)
bitmap = displayio.Bitmap(320, 240, 65536)
tg = displayio.TileGrid(
    bitmap,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=displayio.Colorspace.RGB565_SWAPPED
    ),
)
g.append(tg)
display.root_group = g
```

```
sd_spi = busio.SPI(clock=board.IO18, MOSI=board.IO14, MISO=board.IO17)
sd_cs = board.IO12
sdcard = sdcardio.SDCard(sd_spi, sd_cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")


def exists(filename):
    try:
        os.stat(filename)
        return True
    except OSError:
        return False


_image_counter = 0


def open_next_image(extension="jpg"):
    global _image_counter  # pylint: disable=global-statement
    while True:
        filename = f"/sd/img{_image_counter:04d}.{extension}"
        _image_counter += 1
        if exists(filename):
            continue
        print("#", filename)
        return open(filename, "wb")  # pylint: disable=consider-using-with


### These routines are for writing BMP files in the RGB565 or BGR565 formats.
_BI_BITFIELDS = 3

_bitmask_rgb565 = (0xF800, 0x7E0, 0x1F)
_bitmask_bgr565 = (0x1F, 0x7E0, 0xF800)


def write_header(output_file, width, height, masks):
    def put_word(value):
        output_file.write(struct.pack("<H", value))

    def put_dword(value):
        output_file.write(struct.pack("<I", value))

    def put_long(value):
        output_file.write(struct.pack("<i", value))

    def put_padding(length):
        output_file.write(b"\0" * length)

    filesize = 14 + 108 + height * width * 2

    # BMP header
    output_file.write(b"BM")
    put_dword(filesize)
    put_word(0)  # Creator 1
    put_word(0)  # Creator 2
    put_dword(14 + 108)  # Offset of bitmap data

    # DIB header (BITMAPV4HEADER)
    put_dword(108)  # sizeof(BITMAPV4HEADER)
    put_long(width)
    put_long(-height)
    put_word(1)  # number of color planes (must be 1)
    put_word(16)  # number of bits per pixel
    put_dword(_BI_BITFIELDS)  # "compression"
    put_dword(2 * width * height)  # size of raw bitmap data
    put_long(11811)  # 72dpi -> pixels/meter
    put_long(11811)  # 72dpi -> pixels/meter
    put_dword(0)  # palette size
```

```
    put_dword(0)  # important color count
    put_dword(masks[0])  # red mask
    put_dword(masks[1])  # green mask
    put_dword(masks[2])  # blue mask
    put_dword(0)  # alpha mask
    put_dword(0)  # CS Type
    put_padding(3 * 3 * 4)  # CIEXYZ infrmation
    put_dword(144179)  # 2.2 gamma red
    put_dword(144179)  # 2.2 gamma green
    put_dword(144179)  # 2.2 gamma blue


def capture_image_bmp(the_bitmap):
    with open_next_image("bmp") as f:
        swapped = np.frombuffer(the_bitmap, dtype=np.uint16)
        swapped.byteswap(inplace=True)
        write_header(f, the_bitmap.width, the_bitmap.height, _bitmask_rgb565)
        f.write(swapped)


display.auto_refresh = False
old_record_pressed = True

while True:
    a_voltage = a.value * a.reference_voltage / 65535  # pylint: disable=no-member
    cam.capture(bitmap)
    bitmap.dirty()

    record_pressed = abs(a_voltage - V_RECORD) < 0.05
    display.refresh(minimum_frames_per_second=0)
    if record_pressed and not old_record_pressed:
        capture_image_bmp(bitmap)
    old_record_pressed = record_pressed
```

Coding Image Filters in C

CircuitPython 9's new **bitmaptools** (https://adafru.it/19e8) module includes a number of image filters: false color, lookup, mix, morph, and solarize.

If you have a different image processing task, and CircuitPython + ulab code is not fast enough, you can code a new algorithm in C. Note that this will require you to build your own custom CircuitPython firmware, so familiarize yourself with that process first. (https://adafru.it/Bfu) Then, get an overview of how to add a CircuitPython module coded in C (https://adafru.it/19e9).

There are 4 main parts you'll need:

- A declaration of the C function for processing the image
- The actual implementation of the image processing algorithm
- The function binding, which converts from CircuitPython arguments to C arguments
- The function's entry in the bitmaptools "globals table"

To illustrate each of these parts, the implementation of solarize will be used as an example.

In the file **shared-bindings/bitmapfilter/__init__.h** is a declaration of the image processing function. Solarize takes a bitmap (which it modifies in-place), an optional mask bitmap, and a threshold value from 0 to 1 as a float:

```
void shared_module_bitmapfilter_solarize(
    displayio_bitmap_t *bitmap,
    displayio_bitmap_t *mask,
    const mp_float_t threshold);
```

The C implementation of solarize in **shared-module/bitmapfilter/__init__.c** is shown below. Here are some key items to note:

- The threshold value is converted to a scaled integer just once. This is because on most microcontrollers, computations on integers are faster than computations on floating-point numbers.
- The bitmap depth is checked to determine if it's the right kind. In this example, only processing of 16-bit images is implemented. Furthermore, it is assumed (by the `IMAGE_GET_RGB565_PIXEL_FAST` and `IMAGE_PUT_RGB565_PIXEL_FAST` functions) that 16-bit images are always in **RGB565_SWAPPED** format.
- The image is processed by rows. `IMAGE_COMPUTE_RGB565_PIXEL_ROW_PTR` gets a pointer to the first pixel of a particular row (Y) value, while the GET/PUT macros take just a column (X) value.
- If the optional mask bitmap is not NULL, it is checked before deciding whether to alter a given pixel.
- Other functions can take apart or put together pixel values. There are macros for YUV & RGB conversion, etc. They are near the top of the file.
- If your algorithm needs temporary space it can use `scratchpad_alloc` or `scratch_bitmap16`. The **morph** algorithm does this, using small scratch bitmap so that it can process the image a row at a time.

```
void shared_module_bitmapfilter_solarize(
    displayio_bitmap_t *bitmap,
    displayio_bitmap_t *mask,
    const mp_float_t threshold) {

    int threshold_i = (int32_t)MICROPY_FLOAT_C_FUN(round)(256 * threshold);
    switch (bitmap->bits_per_value) {
        default:
            mp_raise_ValueError(MP_ERROR_TEXT("unsupported bitmap depth"));
        case 16: {
            for (int y = 0, yy = bitmap->height; y < yy; y++) {
                uint16_t *row_ptr = IMAGE_COMPUTE_RGB565_PIXEL_ROW_PTR(bitmap, y);
                for (int x = 0, xx = bitmap->width; x < xx; x++) {
                    if (mask && common_hal_displayio_bitmap_get_pixel(mask, x, y)) {
                        continue; // Short circuit.
                    }
```

```
                    int pixel = IMAGE_GET_RGB565_PIXEL_FAST(row_ptr, x);
                    int y = COLOR_RGB565_TO_Y(pixel);
                    if (y > threshold_i) {
                        y = MIN(255, MAX(0, 2 * threshold_i - y));
                        int u = COLOR_RGB565_TO_U(pixel);
                        int v = COLOR_RGB565_TO_V(pixel);
                        pixel = COLOR_YUV_TO_RGB565(y, u, v);
                        IMAGE_PUT_RGB565_PIXEL_FAST(row_ptr, x, pixel);
                    }
                }
            }
            break;
        }
    }
}
```

In the file **shared-bindings/bitmapfilter/__init__.c**, is the adapter function from CircuitPython to C. It opens with the C function definition, which will always have the following form. Next, the possible Python function arguments are declared, first as an `enum{}` and second as an `allowed_args[]`. The `args[]` array is created to have the same number of elements as `allowed_args[]`. Each element in the `enum` must match the element of `allowed_args[]` and can be used to index into the `args[]` array. The function `mp_arg_parse_all` takes care of converting the input arguments into the `args[]` array.

```
STATIC mp_obj_t bitmapfilter_solarize(size_t n_args, const mp_obj_t *pos_args,
mp_map_t *kw_args) {
    enum { ARG_bitmap, ARG_threshold, ARG_mask };
    static const mp_arg_t allowed_args[] = {
        { MP_QSTR_bitmap, MP_ARG_REQUIRED | MP_ARG_OBJ, { .u_obj = MP_OBJ_NULL } },
        { MP_QSTR_threshold, MP_ARG_OBJ, { .u_obj = MP_OBJ_NULL } },
        { MP_QSTR_mask, MP_ARG_OBJ, { .u_obj = MP_ROM_NONE } },
    };
    mp_arg_val_t args[MP_ARRAY_SIZE(allowed_args)];
    mp_arg_parse_all(n_args, pos_args, kw_args, MP_ARRAY_SIZE(allowed_args),
allowed_args, args);
```

Next, the required processing of each object into the correct C type, such as `mp_float_t` or `displayio_bitmap_t *`, and dealing with the case where an optional argument was not specified:

```
// (continued from above)
    mp_float_t threshold = (args[ARG_threshold].u_obj == NULL) ?
MICROPY_FLOAT_CONST(0.5) : mp_obj_get_float(args[ARG_threshold].u_obj);
    mp_arg_validate_type(args[ARG_bitmap].u_obj, &amp;displayio_bitmap_type,
MP_QSTR_bitmap);
    displayio_bitmap_t *bitmap = MP_OBJ_TO_PTR(args[ARG_bitmap].u_obj);


    displayio_bitmap_t *mask = NULL;
    if (args[ARG_mask].u_obj != mp_const_none) {
        mp_arg_validate_type(args[ARG_mask].u_obj, &amp;displayio_bitmap_type,
MP_QSTR_mask);
        mask = MP_OBJ_TO_PTR(args[ARG_mask].u_obj);
    }
```

Finally, the converted arguments are passed to the function that implements the filter, and the modified bitmap is used as the CircuitPython return value. Immediately following the function definition is a line to create the CircuitPython function object.

If your algorithm needs to return something other than the modified bitmap, then you would need to add calls to build the CircuitPython object corresponding to the output or result of the C image processing function, and return this instead of the bitmap argument.

```
// (continued from above)
    shared_module_bitmapfilter_solarize(bitmap, mask, threshold);
    return args[ARG_bitmap].u_obj;
}
MP_DEFINE_CONST_FUN_OBJ_KW(bitmapfilter_solarize_obj, 0, bitmapfilter_solarize);
```

The final element is an entry in the bitmapfilter's module globals table for the function, near the bottom of **shared-bindings/bitmapfilter/__init__.c**:

```
STATIC const mp_rom_map_elem_t bitmapfilter_module_globals_table[] = {
    // ...
    { MP_ROM_QSTR(MP_QSTR_solarize), MP_ROM_PTR(&bitmapfilter_solarize_obj) },
    // ...
};
```

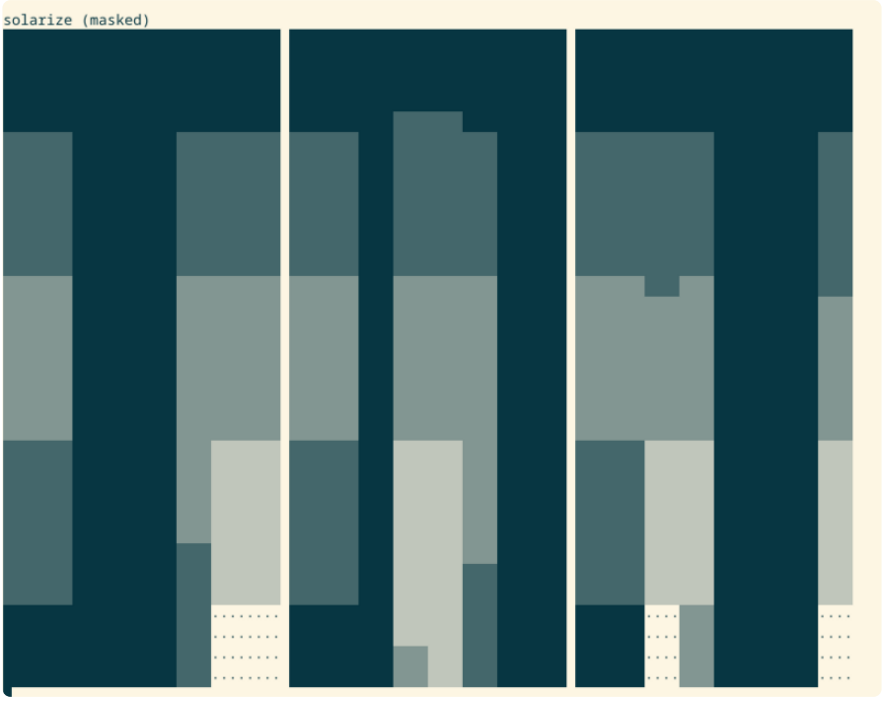
At this point, you can compile your code and address any build errors that occur.

If you're using a Linux or Mac based development environment, you can also test your filter on a host computer before uploading firmware to a board. You can do this by building in the **ports/unix** subdirectory with a commandline like **make -j8 VARIANT=coverage**. The created program **build-coverage/micropython** has importable **bitmapfilter** and **displayio** modules (among others). You can run this **micropython** (including under the gdb debugger) for testing your algorithm on a host computer. By setting **MICROPYPATH=/complete/path/to/circuitpython/tests/testlib** in your shell environment, you will be able to import some useful routines for getting bitmap test data (`import blinka_image`) and for printing out representations of bitmaps on screen (`import dump_bitmap`)

You can run the test suite, including image processing tests, with **make -j8 VARIANT=coverage test**. This will run the tests including those in **tests/circuitpython**. Have a look at an existing test such as **tests/circuitpython/bitmapfilter_solar.py** and its "expected output" file **tests/circuitpython/bitmapfilter_solar.py.exp**. After some preliminaries, it creates a test bitmap with several color ramps, and runs the solarize algorithm on it. It dumps the image using Unicode characters that represent 5 brightness levels, showing the R, G, and B image channels separately:

```
print("solarize (masked)")
bitmapfilter.solarize(b, mask=q)
dump_bitmap_rgb_swapped(b)
```

In a proper, wide terminal window the results can be seen, though they can be difficult to interpret.



Documentation: adafruit_ov2640

Documentation: adafruit_ov2640 (https://adafru.it/TnE)

Documentation: adafruit_ov7670

Documentation: adafruit_ov7670 (https://adafru.it/Toa)

Documentation: adafruit_ov5640

Documentation: adafruit_ov5640 (https://adafru.it/VfR)