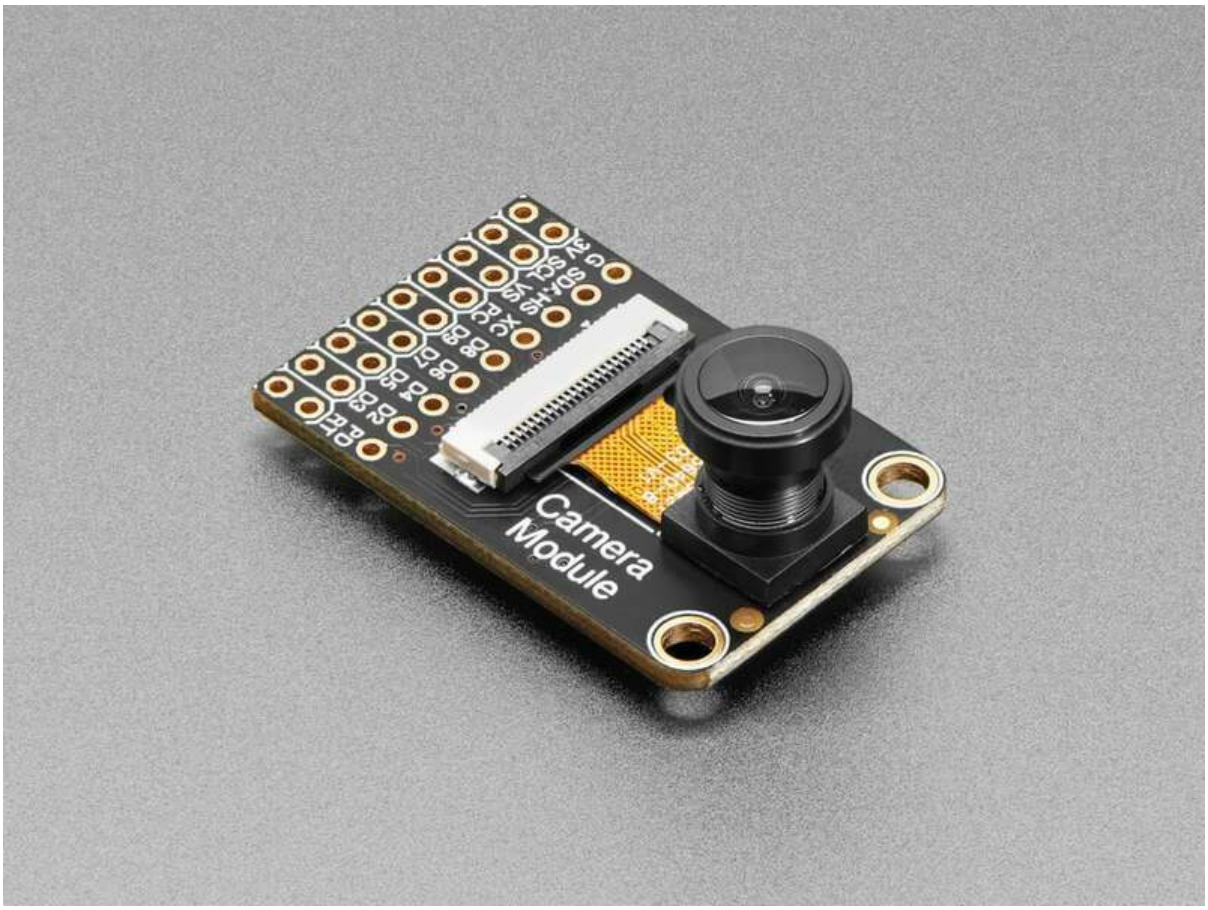
Adafruit OV5640 Camera Breakouts

Created by Jeff Epler



https://learn.adafruit.com/adafruit-ov5640-camera-breakout

Last updated on 2025-02-14 05:37:39 PM EST

Table of Contents

Overview 5

Pinouts 7

- A note on silkscreen labels
- Camera Lens
- Power Supply
- Power Down & Reset
- Power LED
- I2C Pins
- Clock and Synchronization Pins
- Data Pins
- Autofocus Motor Power
- Test Bar Mode

Assembly 13

CircuitPython Camera Libraries 14

- Camera Pin Naming

Espressif ESP32-family Overview 15

- Setting 'reserved PSRAM'
- Pin Choices

Install TinyUF2 on Espressif Kaluga 17

- 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

Install CircuitPython on Espressif Kaluga with TinyUF2 28

- CircuitPython Quickstart
- Kaluga USB Connection

Espressif Kaluga Pinout 30

- Setting 'reserved PSRAM'
- Camera Module Connections
- LCD variants

LCD Mirror Demo 32

- Install & Use the Demo
- Code Walkthrough

ASCII Mirror Demo 37

- Install & Use the Demo

JPEG Capture Demo 41

- Install & Use the Demo

• Code walkthrough

espcamera documentation 46

Raspberry Pi RP2040 Overview 46

• Image storage
• Pin choices

Raspberry Pi Pico Usage 47

• Camera connections
• LCD connections

ASCII Mirror Demo 49

• Install & Use the Demo

LCD Mirror Demo 53

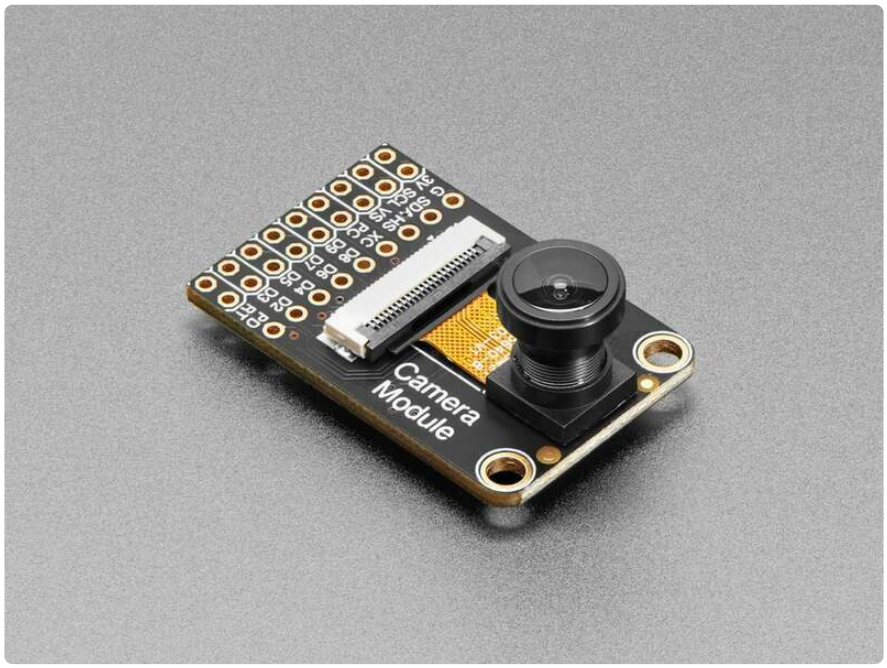
• Install & Use the Demo
• Code Walkthrough

adafruit_ov5640 documentation 56

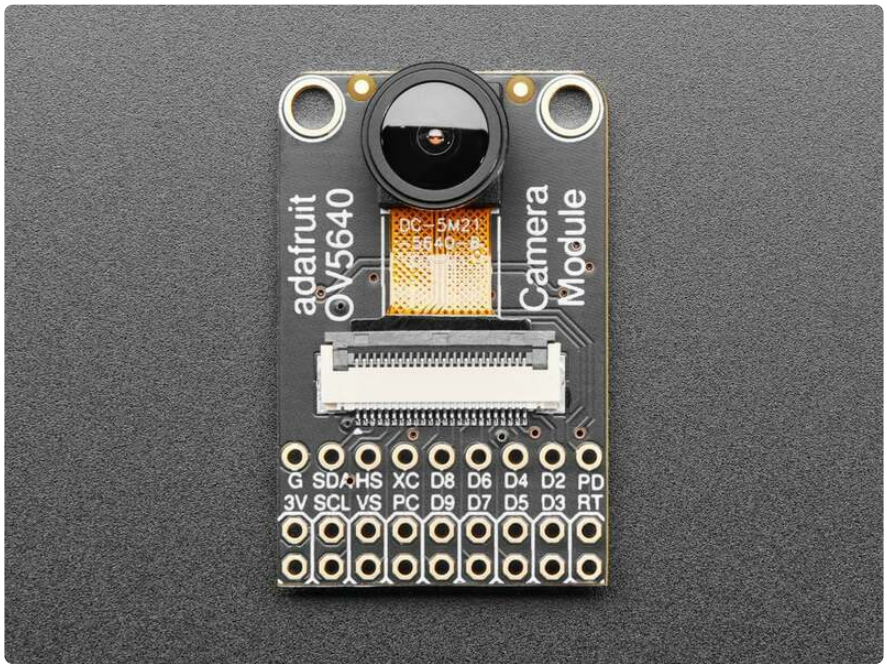
Downloads 56

• Files
• Schematic and Fab Print

Overview



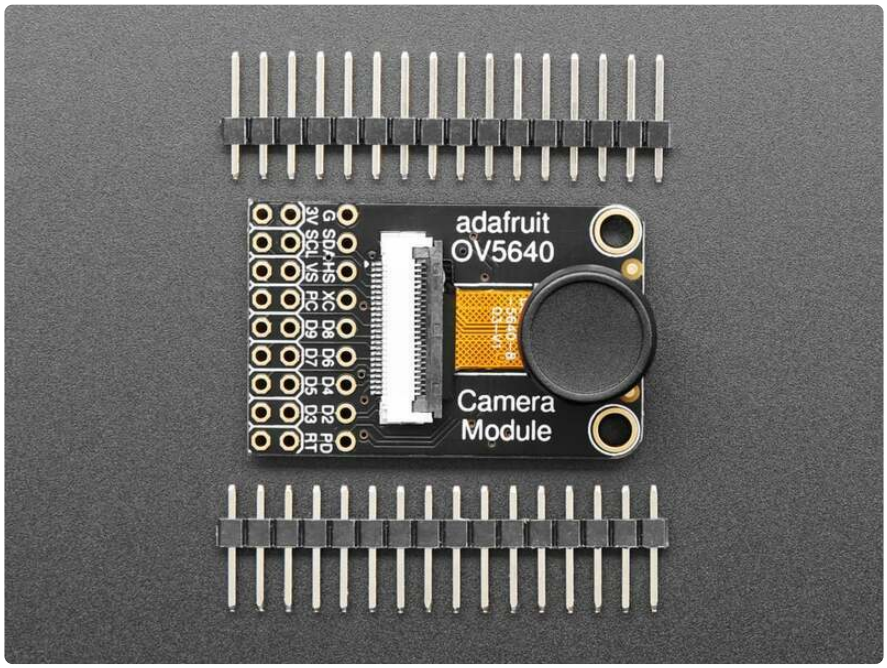
Hobby-level microcontrollers are finally getting powerful enough to start handling camera modules that historically would have required a full computer or FPGA to handle. The RP2040 and ESP32-Sx series of chips, for example, have enough pins to communicate with the 8-bit data output, DMA to quickly grab a frame, and the necessary RAM to buffer a raw snapshot. Now all that is needed is a nice camera module to make interfacing easy!



This Adafruit OV5640 Camera Breakout with 120 Degree Lens has a nice quality OV5640 camera with a 5 Megapixel sensor element, 120-degree wide angle lens, and

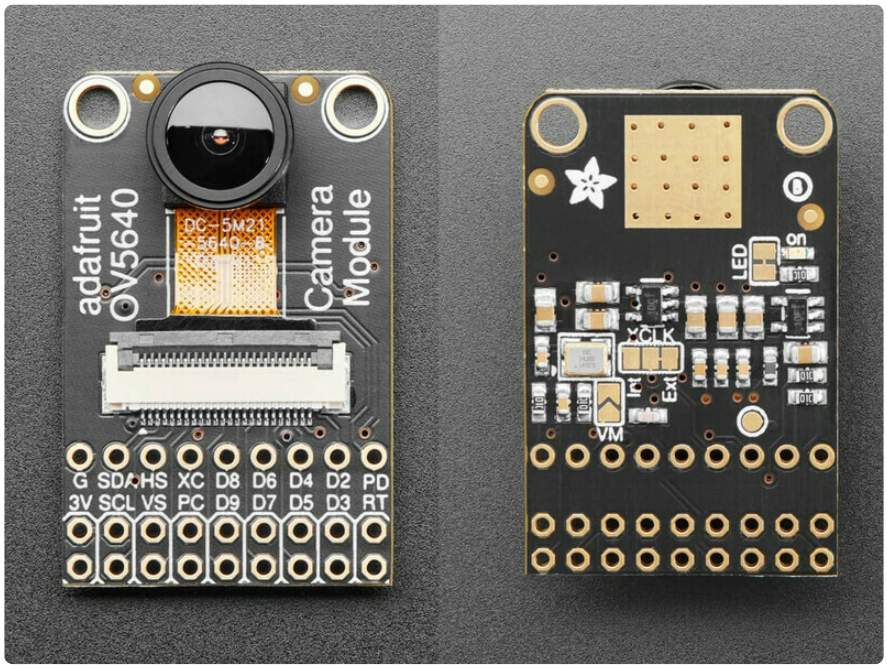
all the support circuitry you need. Adafruit looked at existing camera modules and while this breakout board is backwards compatible, they made some improvements:

- Standard 2x9 header if you want it, but also a duplicated header strip 0.3" apart so you can plug it into a breadboard or perfboard
- Selectable external or internal 24MHz "XCLK" clock generation - save one GPIO pin, or just have a nice stable 24 MHz signal even if your microcontroller can't generate it for you.
- Heat-sinking camera area with exposed ground pad, with lots of vias for good thermal transfer. Helpful for when doing continuous encoding and reducing thermal image drift.
- Optional VMotor 3.3V power jumper on DATA1, for auto-focusing camera modules
- 3.3V power-good LED on back that can be disabled

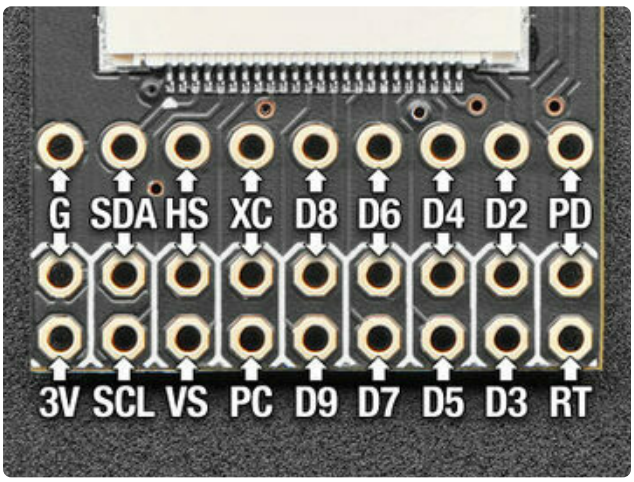


The module comes with header pins so you can solder it for use on a breadboard or on the standard 2x9 header—It's up to you.

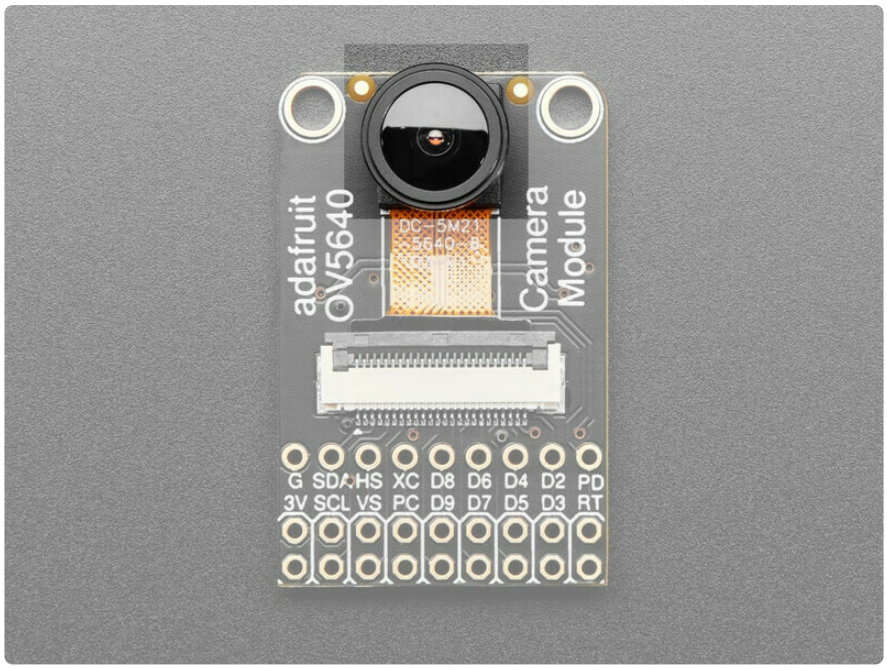
Pinouts



A note on silkscreen labels



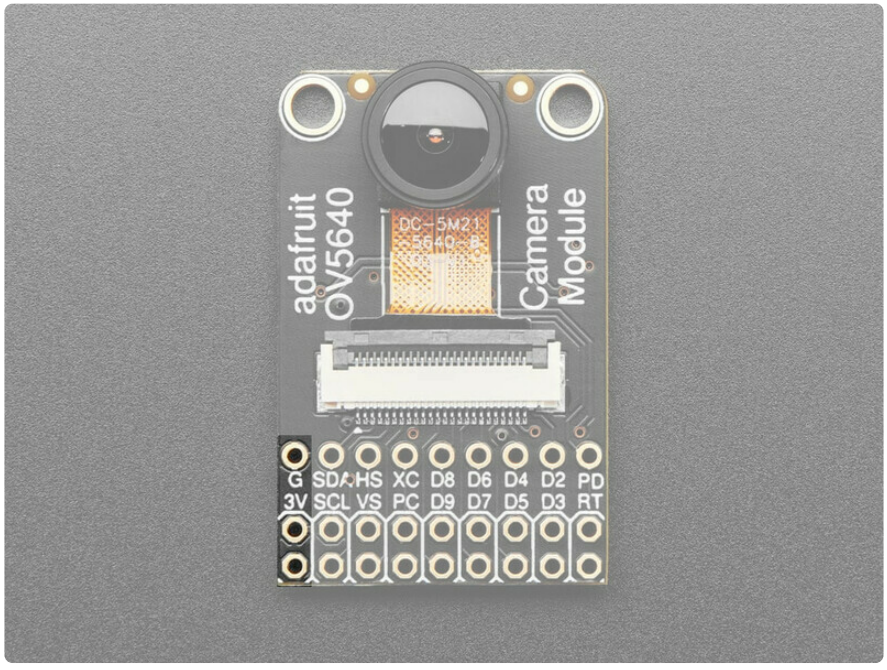
In each column, the top label (e.g., G, SDA, etc) applies to both the top and center rows of pins. The bottom label (e.g., 3V, SCL, etc) is for the bottom row of pins. As noted on the Assembly page, depending how you will install the module you will either be using the two outermost rows of pins, or the two bottom rows of pins.



Camera Lens

This OV5640 camera has a 120° wide-angle lens. Its focusing distance can't be adjusted. When shipped, it has an opaque lens cap on it. Remove the cap to capture images, unless you **really** like the color black.

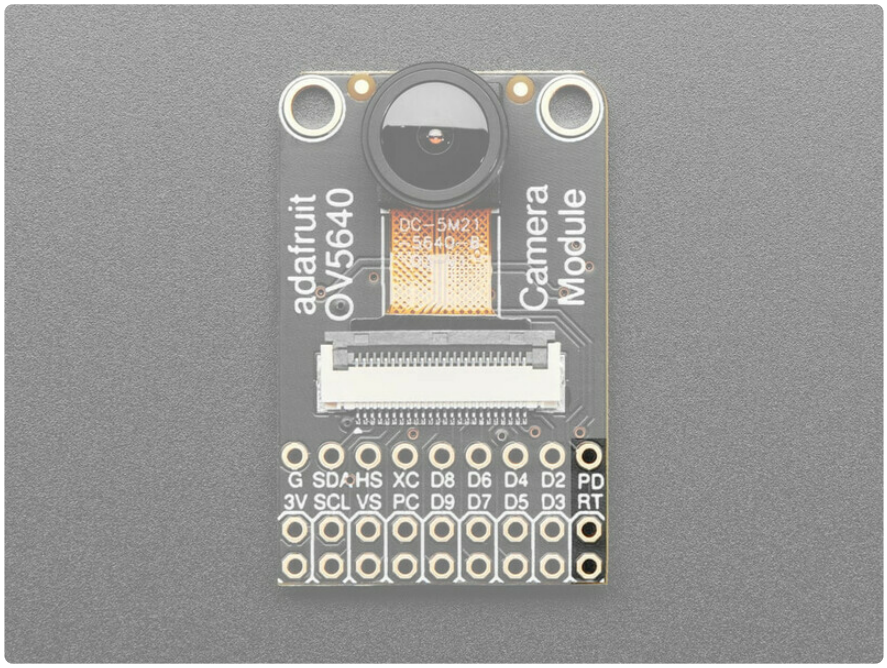
Power Supply



The camera module requires a well-regulated 3.3v supply in order to operate.

- **G** (GND): Connect to microcontroller GND
- **3V** (3.3V): Connect to microcontroller +3.3V supply

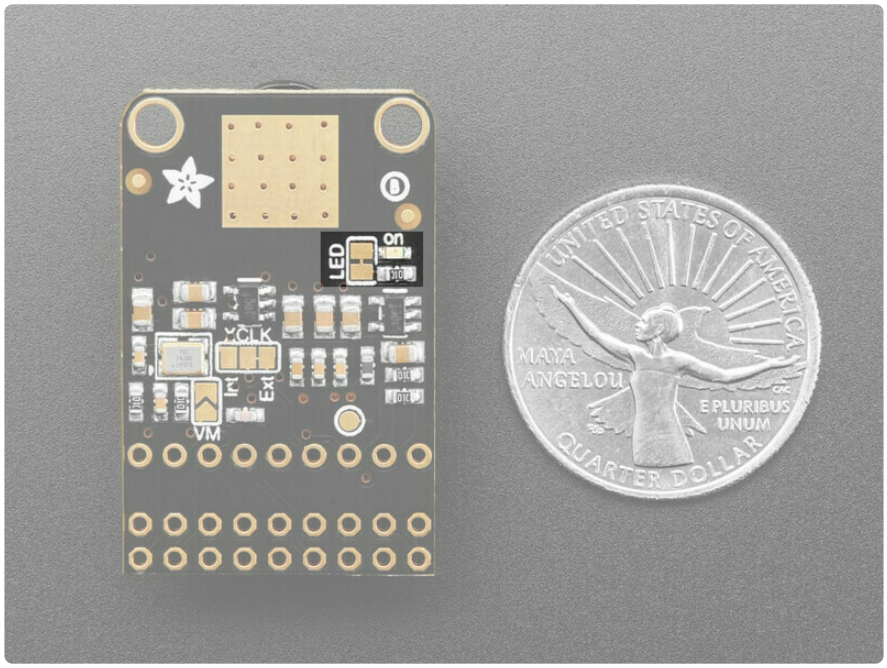
Power Down & Reset



- **PD** (powerdown): Optional connection to microcontroller GPIO. When pulled **HIGH** the camera module is put into power-down mode. When released or pulled **LOW** the camera is powered on. This pin can also be pulsed **HIGH** as a way to reset the camera.
- **RT** (reset): Optional connect to microcontroller GPIO. Pull the pin **LOW** to reset the camera module and release it or pull it **HIGH** to enter operating mode.

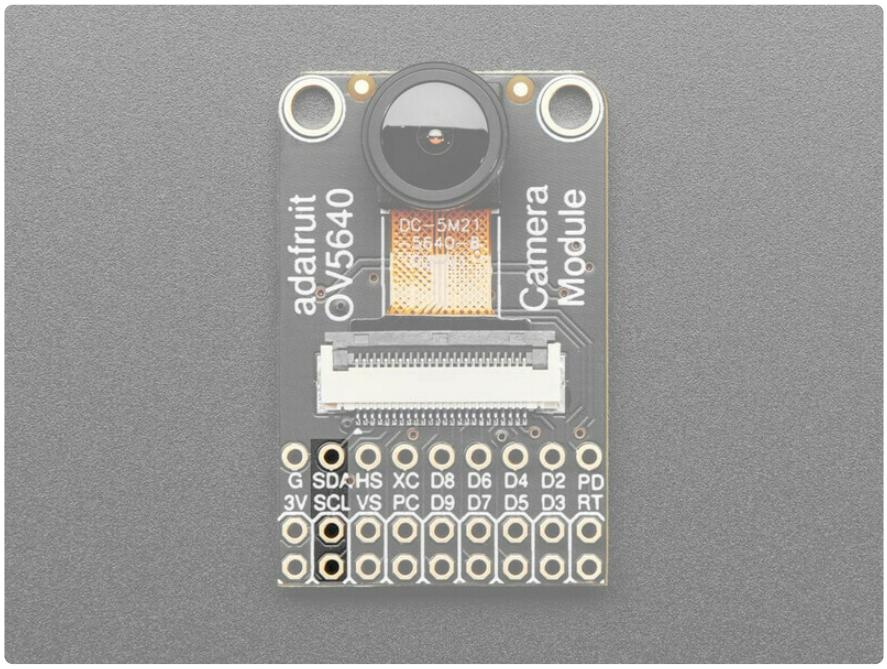
Both pins have built-in pull resistors, so by default the camera is powered on and allowed to exit reset mode.

Power LED



When powered on, this LED will light. To disable the LED, cut the small trace between the two pads of the solder jumper. To re-enable it, bridge the two pads with a blob of solder.

I2C Pins

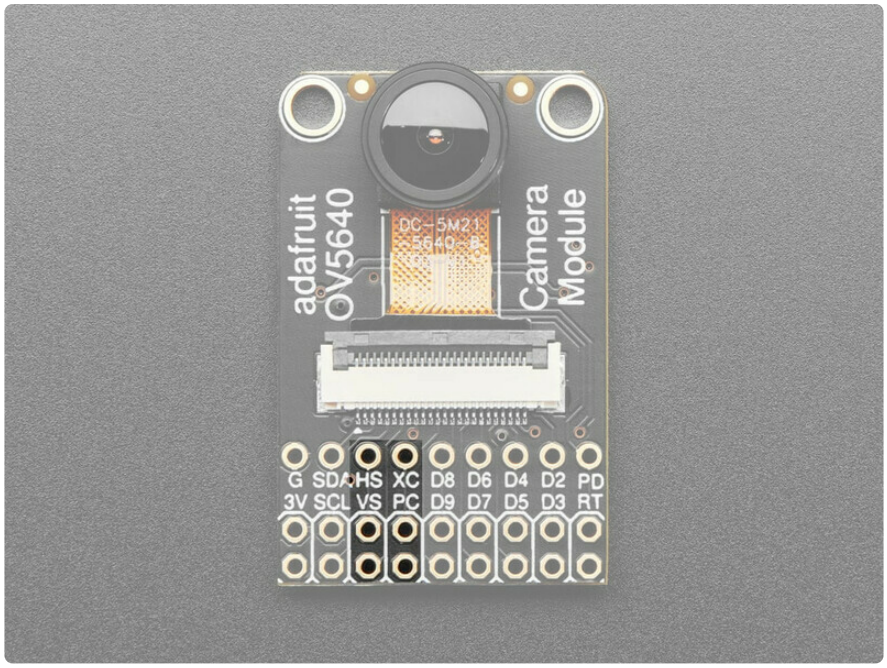


The camera module must be configured using I2C.

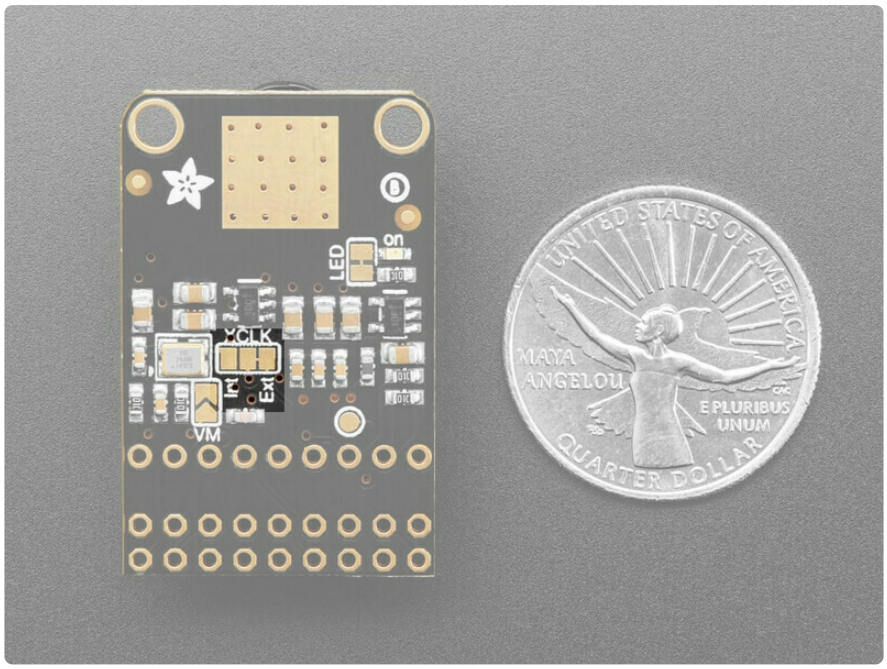
- **SDA**: Connect to microcontroller SDA
- **SCL**: Connect to microcontroller SDL

Adafruit's OV5640 camera module has built-in pull ups, so you don't need to add external ones. But note that many modules from other vendors do not have these pull-ups.

Clock and Synchronization Pins

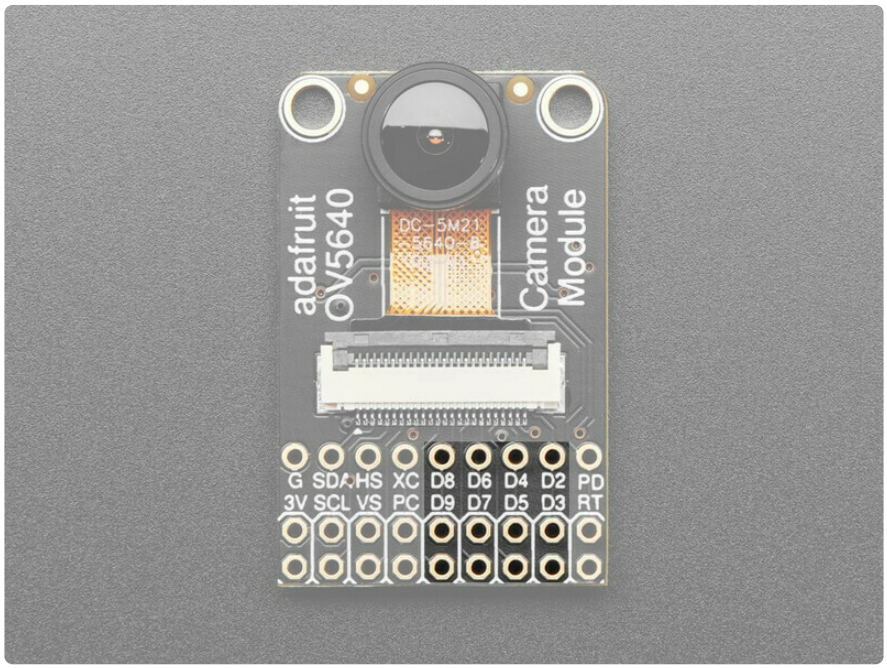


- **XC** (external clock): When the **XCLK** jumper is set to "**EXT**" (the default), this pin must be driven with a 24MHz square wave from the microcontroller or other source. When the jumper is changed to "**INT**", then an on-board clock generator is used instead. In this case, the **XC** pin should not be driven by the microcontroller and may be left unconnected.
- **PC** (pixel clock) tells the microcontroller when image data is available. This may need to be connected to a specific microcontroller pin.
- **VS** and **HS** are synchronization signals, which tell the microcontroller when a new frame (**VS**) or row (**HS**) of data begins. These may need to be connected to specific microcontroller pins.



The OV5640 Camera Breakout includes an on-board 24MHz crystal oscillator. To use this as the OV5640 clock, cut the jumper from the center pad to **EXT** then solder between the center pad and **INT**. In this mode, any input on the **XC** pin is ignored and **XC** may be left unconnected.

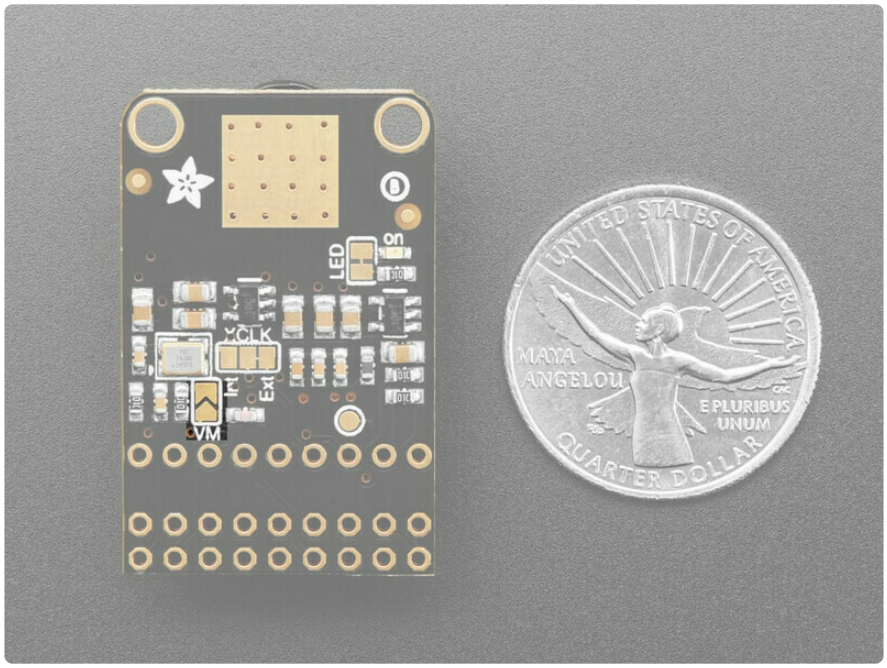
Data Pins



The 8 data pins (numbered from **D2** to **D9** because reasons) carry data out of the camera into the microcontroller.

Depending on mode, these 8 data bits can be half of a 16-bit pixel value, or one byte of JPEG data.

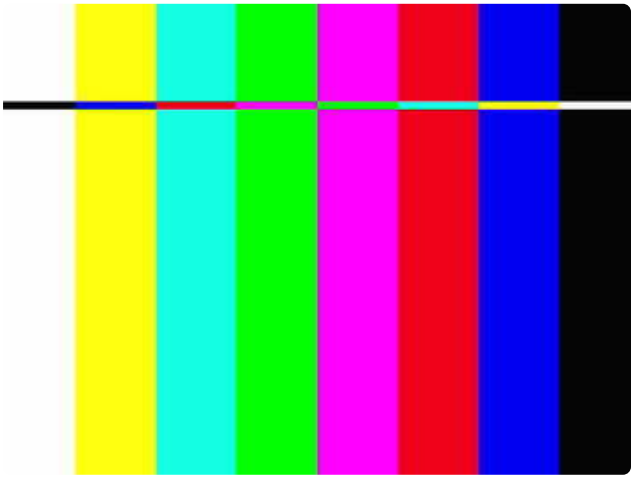
Autofocus Motor Power



While the camera supplied with the OV5640 Camera Breakout does not have an auto-focus motor, some compatible camera modules include a "voice coil" motor for auto-focus.

If you install such a module, close the **VM** jumper with a blob of solder. Internally, this connects the 3.3V supply to the camera module on its **D1** connection.

Test Bar Mode



In test bar mode, the camera shows color bars in the order **white** - **yellow** - **cyan** - **green** - **purple** - **red** - **blue** - **black**. A small vertical bar of inverted colors moves from top to bottom. Here's a typical test bar, captured in JPEG mode at VGA resolution.

Assembly

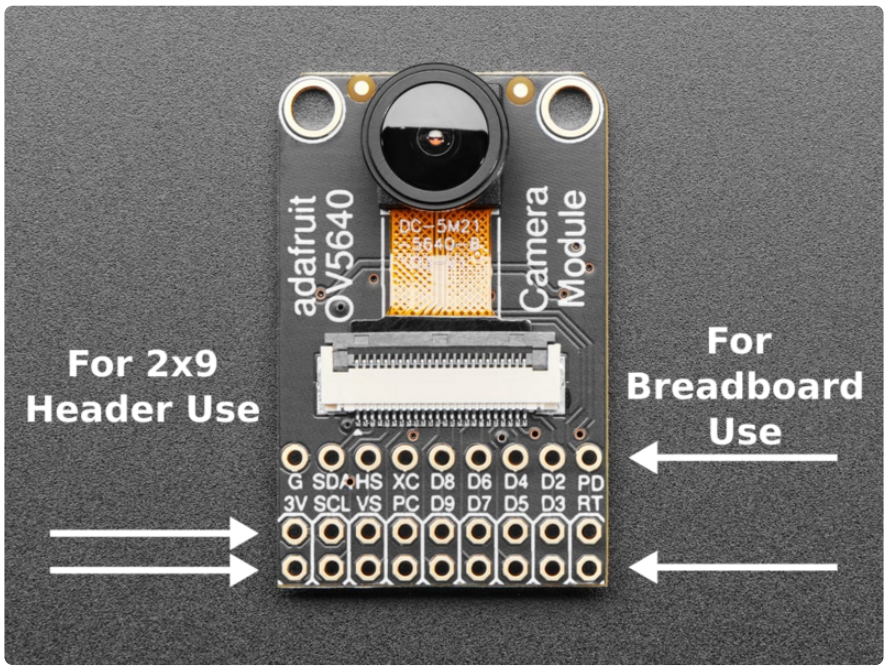
First, decide how you are going to use your camera module: on a board which has a standard 2x9 header or on a standard breadboard.

If your microcontroller has a 2x9 header available, that method is strongly preferred! The high speed signals of the camera module can be scrambled even with short lengths of jumper wire on a solderless breadboard.

Using side cutters, snap off two 9-pin sections of headers.

If you are soldering for the 2x9 header then place the pins in the two rows at the edge of the board

If you are soldering for breadboard use then place the pins in the outermost rows, leaving the middle row empty.



Carefully align the headers. You can use tacky clay to hold them in place. Solder just one pin in each row, then check again for alignment. If the pins are poorly aligned, re-heat the solder joint while adjusting it until it is straight. Add flux if needed so that you don't end up with a cold solder joint.

Then, solder the rest of the pins.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)).

CircuitPython Camera Libraries

There are multiple libraries for camera support on CircuitPython.

For Espressif boards, the built in **espcamera** (https://adafru.it/18tC) module (added in CircuitPython 8) interfaces with multiple types of cameras including the OV5640.

For other boards, such as the Raspberry Pi Pico with RP2040 microcontrollers, an installable library called **adafruit_ov5640** (https://adafru.it/18tD) is used instead. Internally, this uses a module called **imagecapture** (https://adafru.it/18tE) for low-level camera interfacing.

Remember to use the right library for your board!

Camera Pin Naming

By convention, if a board has an integrated camera or a dedicated camera connector, the following will exist in the **board** module:

- **CAMERA_SIOC** - the **SCL** pin of the camera
- **CAMERA_SIOD** - the **SDA** pin of the camera
- **CAMERA_PCLK** - the pixel clock of the camera
- **CAMERA_VSYNC** - the vertical sync of the camera
- **CAMERA_HREF** - the horizontal sync of the camera
- **CAMERA_XCLK** - the input clock pin of the camera
- **CAMERA_DATA** - the 8 data pins of the camera

Espressif ESP32-family Overview

CircuitPython's **espcamera** module is available on most supported ESP32-S2 and ESP32-S3 boards with PSRAM. It incorporates both the camera configuration code and the image capturing code in a single library that works across multiple types of camera modules, instead of being available for regular CircuitPython objects.

> CircuitPython 9 does not require reserving PSRAM: the CIRCUITPYRESERVEDPSRAM= directive is ignored. Read the below only if you are using CircuitPython 8.

Setting 'reserved PSRAM'

Because of how CircuitPython and ESP-IDF (Espressif IoT Development Framework) manage memory together, a portion of memory has to be set aside for the camera framebuffers.

Usually, 1MB (1048576 bytes) is a reasonable amount of memory to reserve. This is plenty of space for two 320×240 bitmap images or a 5-megapixel JPEG image, along with other memory that the esp-idf allocates internally.

Boards with built-in cameras include a default reserved PSRAM setting. Boards with only a dedicated camera header do not.

To set the reserved memory amount, edit the **settings.toml** file within the **CIRCUITPY** drive. Add a line that says `CIRCUITPY_RESERVED_PSRAM=1048576`

The setting will become effective when the board is reset with the reset button. You can check it by opening the repl and running the following lines:

```
Adafruit CircuitPython 8.0.0-rc.1 on 2023-01-30; Kaluga 1 with ESP32S2
>>> import espidf
>>> espidf.get_reserved_psram()
1048576
```

Pin Choices

By selecting appropriate pins, you can use the espcamera CircuitPython module on other boards with supported ESP32, ESP32-S2 and ESP32-S3 microcontrollers:

- **xclk**, **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.
- **data_pins**: Free choice of any 8 pins

By convention, if a board has an integrated camera or a dedicated camera connector, the following will exist in the **board** module:

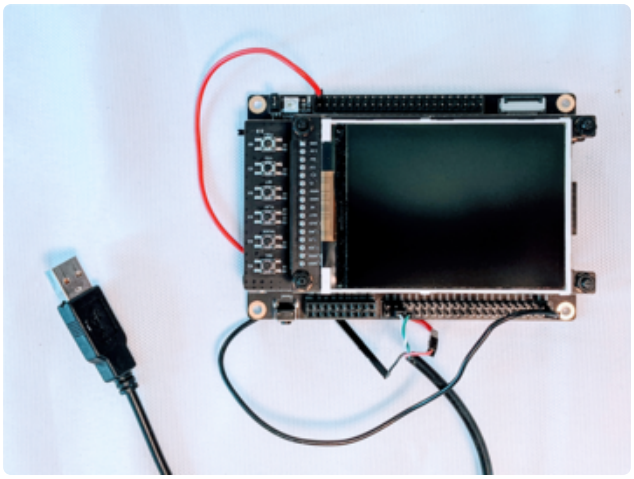
- **CAMERA_SIOC** - the **SCL** pin of the camera
- **CAMERA_SIOD** - the **SDA** pin of the camera
- **CAMERA_PCLK** - the pixel clock of the camera
- **CAMERA_VSYNC** - the vertical sync of the camera
- **CAMERA_HREF** - the horizontal sync of the camera
- **CAMERA_XCLK** - the input clock pin of the camera
- **CAMERA_DATA** - the 8 data pins of the camera

Continue to the next page to see how to use the camera module with the Espressif Kaluga, which has a compatible 18-pin camera connector built in.

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

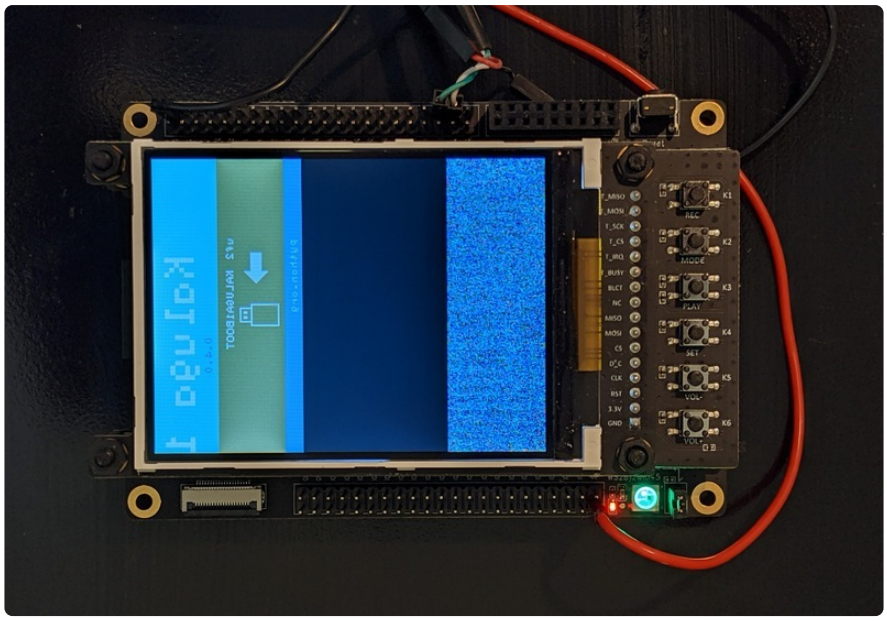
combined.bin

https://adafru.it/TAI

Step 2. Place your board in bootloader mode

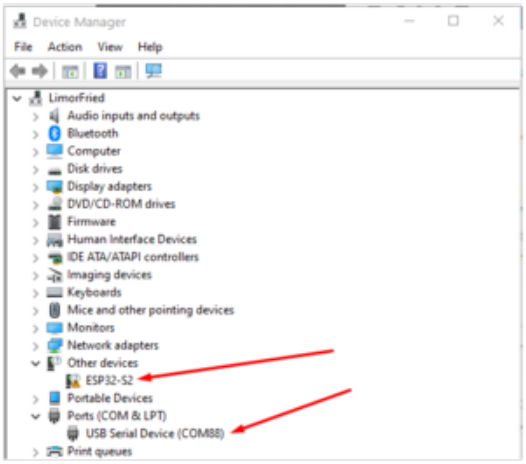
Entering the bootloader is easy. Complete the following steps.

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**



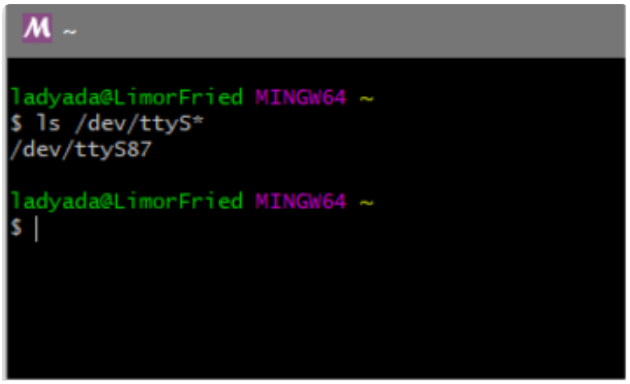
Because there are several incompatible versions of the Kaluga TFT display, the bootloader's screen may appear incorrectly or not at all. This does not affect its operation.

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

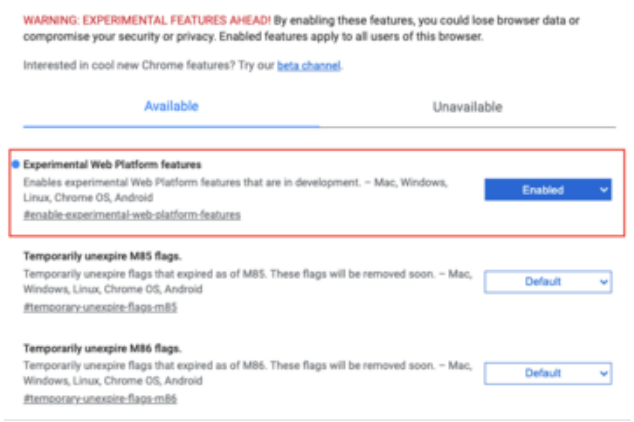
Step 3 Option A. Use the Web Serial ESPTool to upload

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)

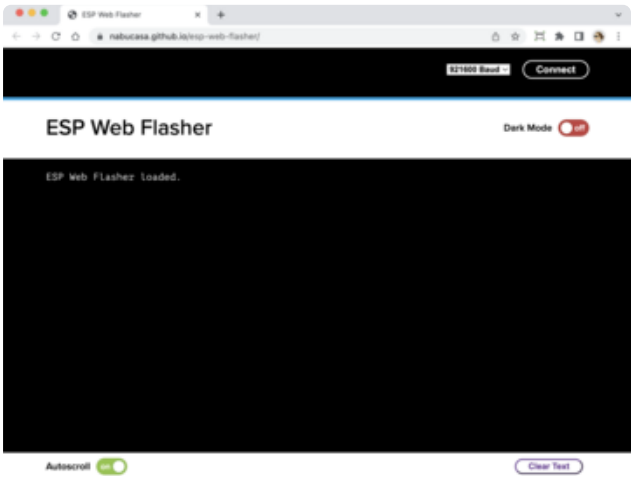
As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.



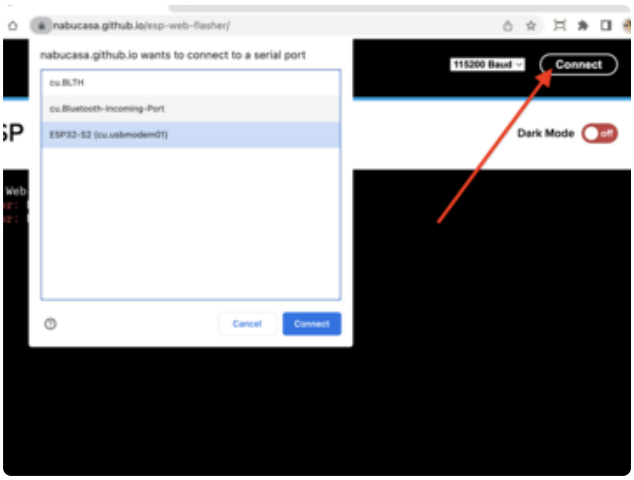
Visit **chrome://flags** from within Chrome. Find and enable the **Experimental Web Platform features**

Restart Chrome

Connecting



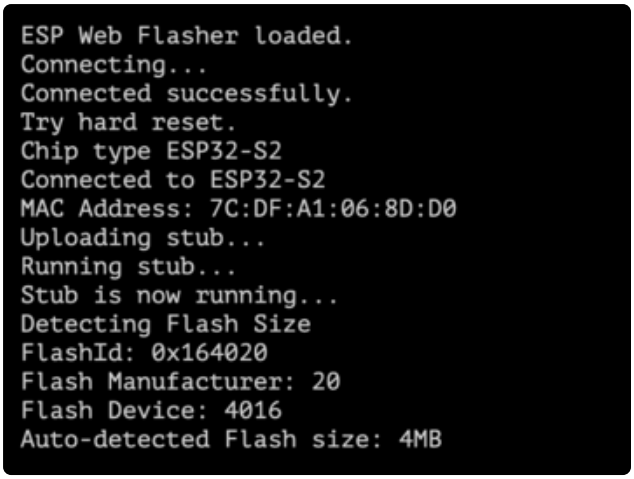
In the **Chrome browser** visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru.it/PMB). It should look like the image to the left.



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.

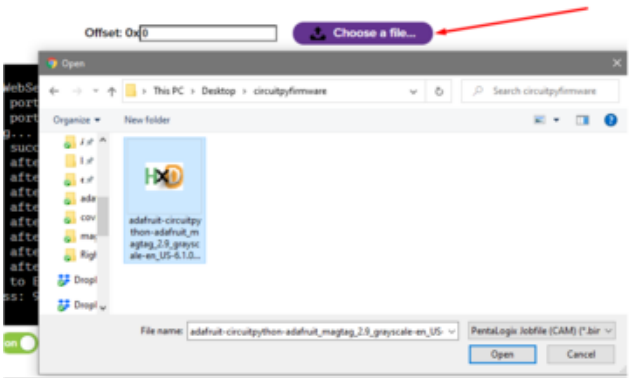
Erasing the Contents

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.

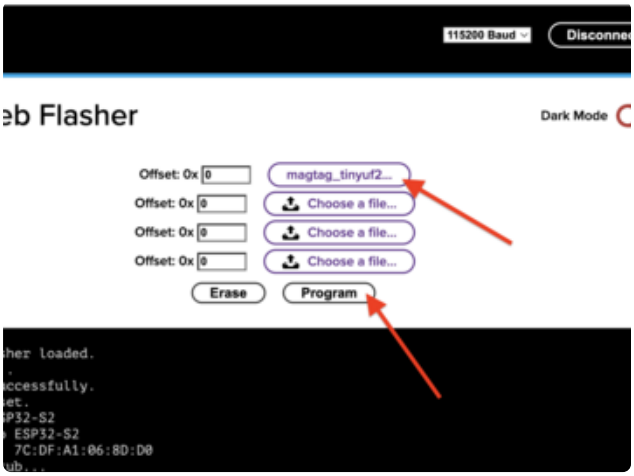
Programming the Microcontroller

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):

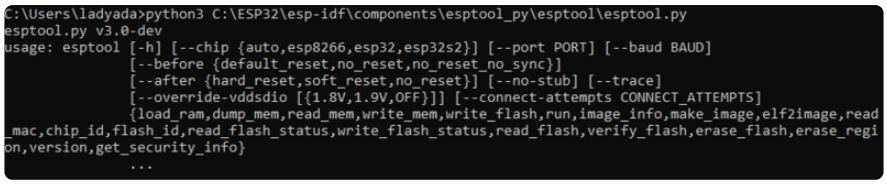
```
pip3 install --upgrade esptool
```

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

```
C:\Users\ladyada>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py --port COM88  chip_id
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:3f:3e
Uploading stub...
Running stub...
Stub running...
Warning: ESP32-S2 has no Chip ID. Reading MAC instead.
MAC: 7c:df:a1:00:3f:3e
Hard resetting via RTS pin...
ERROR: ESP32-S2 chip was placed into download mode using GPIO0.
esptool.py can not exit the download mode over USB. To run the app, reset the chip manually.
To suppress this error, set --after option to 'no_reset'.
```

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:

```
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:05:f8:9a
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 3081264 bytes to 98937...
Wrote 3081264 bytes (98937 compressed) at 0x00000000 in 22.8 seconds (effective 1080.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
ERROR: ESP32-S2 chip was placed into download mode using GPIO0.
esptool.py can not exit the download mode over USB. To run the app, reset the chip manually.
To suppress this error, set --after option to 'no_reset'.
```

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.

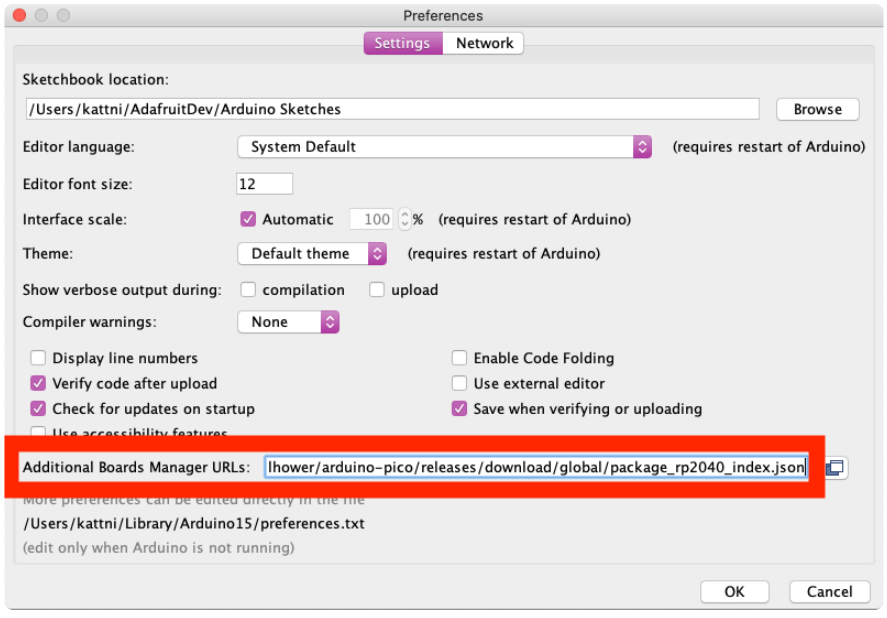
The **Preferences** window will open.

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**.

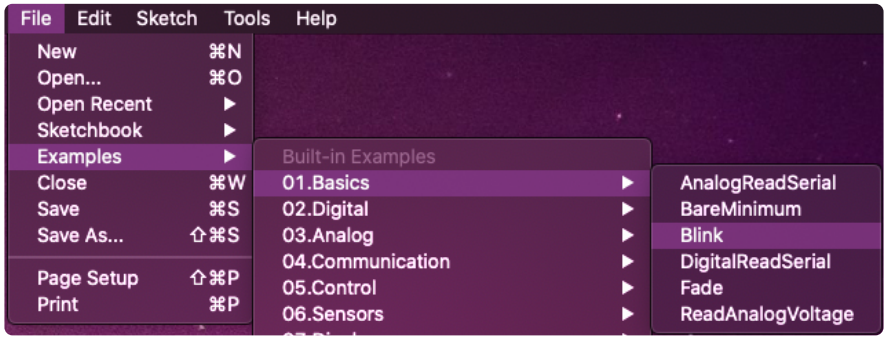
In the **Tools** > **Boards** menu you should see the **ESP32 Arduino** menu. In the expanded menu, it should contain the ESP32 boards along with all the latest ESP32-S2 boards.

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

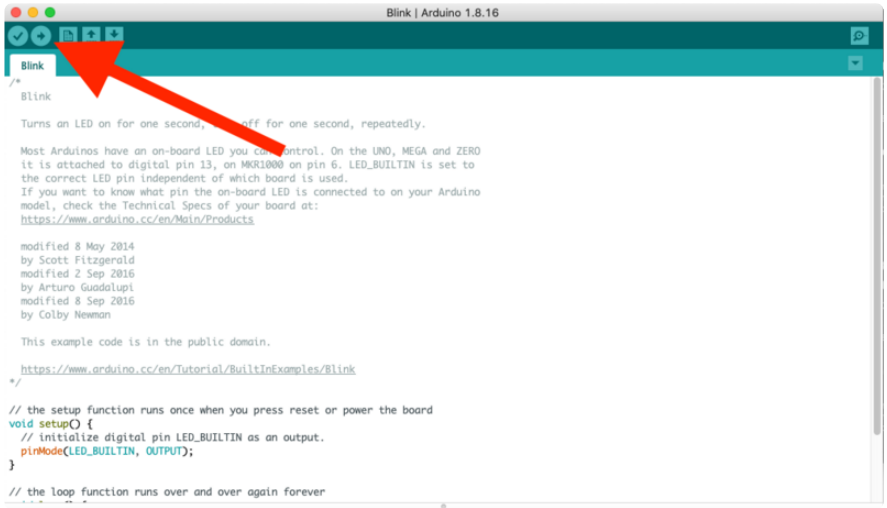
Load the Blink Sketch

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.

Install CircuitPython on Espressif Kaluga with TinyUF2

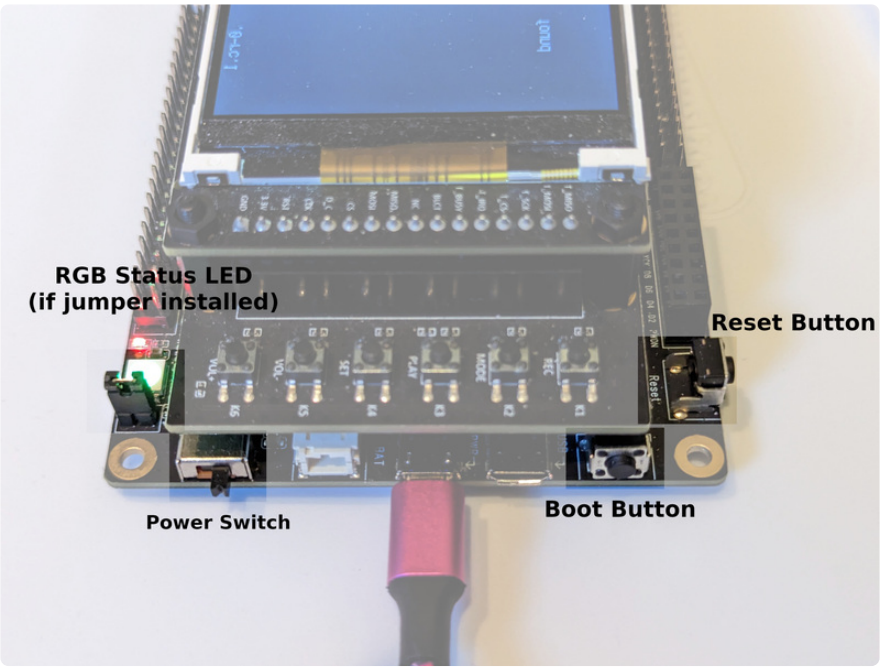
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your a Kaluga 1.3 board (a Kaluga 1.2 will NOT WORK).

Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/18tF



Kaluga USB Connection

Neither of the two Micro USB connectors on the Kaluga provide access to the native USB connection used by CircuitPython. Instead, a USB breakout cable must be connected as follows:

- USB +**5V** (red) to Kaluga +**5V**
- USB **GND** (black) to Kaluga **GND**
- USB **D+** (green) to Kaluga **GPIO20**

- USB **D-** (white) to Kaluga **GPIO19**

Advanced USB connection

Non-populated resistor positions R151 & R152 connect the "J10" micro-USB connector to native USB. By soldering in a 0-ohm link at these positions, you can use the J10 connector for native USB. Some photos in this guide show a board which has been modified in this way.

Click the **reset** button once, and then the **boot** button once when you see the **RGB status LED** turn purple (approximately half a second later). The second tap needs to happen while the LED is still purple.

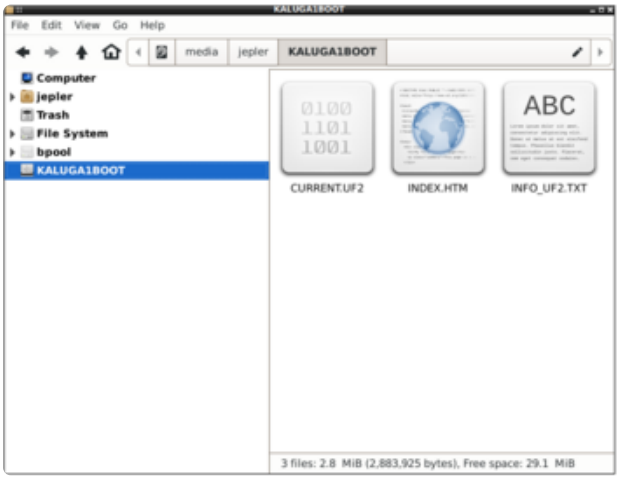
Once successful, you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

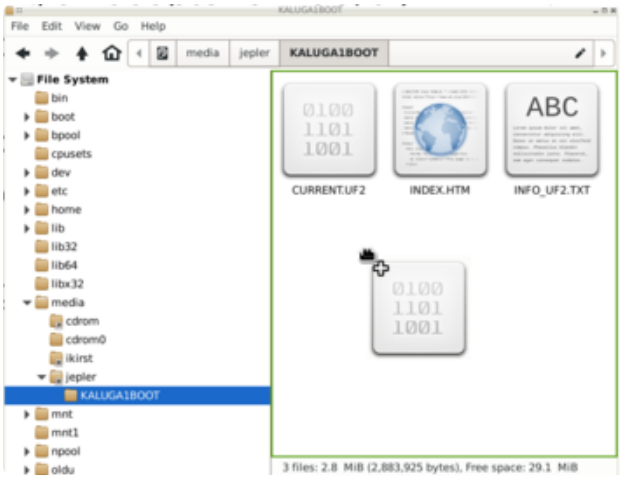
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

Remember that unless you've modified your Kaluga, you have to use a USB breakout cable, not the built-in USB Micro-B ports!

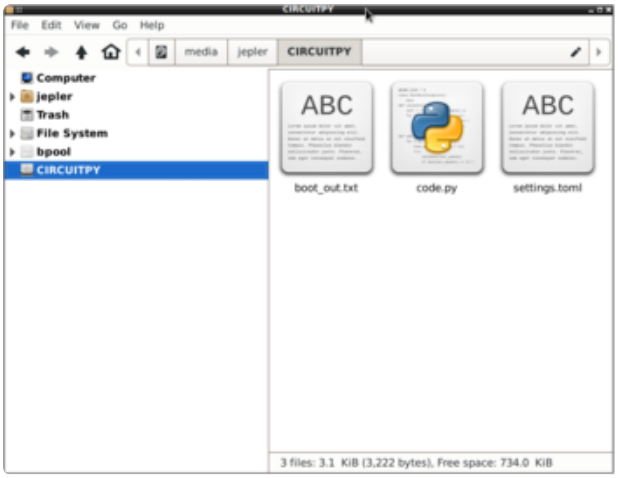
> Because there are multiple variants of the LCD bundled with the Espressif Kaluga development kit, TinyUF2 may show a distorted image on the LCD. This is harmless.



You will see a new disk drive appear called **KALUGA1BOOT**.



Drag the **adafruit-circuitpython-espressif_kaluga_1.3.uf2** file to **KALUGA1BOOT**



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

Espressif Kaluga Pinout

These demos only work with the Kaluga version 1.3. They are not compatible with version 1.2. Check your board's silkscreen to find the version.

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.

Setting 'reserved PSRAM'

The 'reserved PSRAM' setting was discussed earlier, but here's a reminder now that you've got CircuitPython installed:

Edit the **settings.toml** file within the **CIRCUITPY** drive, creating it as an empty file if necessary. Add a line that says `CIRCUITPY_RESERVED_PSRAM=1048576`

The setting will become effective when the board is reset with the reset button. You can check it by opening the REPL and running the following lines:

```
Adafruit CircuitPython 8.0.0-rc.1 on 2023-01-30; Kaluga 1 with ESP32S2
>>> import espidf
>>> espidf.get_reserved_psram()
1048576
```

Camera Module Connections

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

The camera pins are as follows (though in CircuitPython they also have meaningful names that should be used when available):

- **GPIO8** is **CAMERA_SIOC** - the **SCL** pin of the camera
- **GPIO9** is **CAMERA_SIOD** - the **SDA** pin of the camera
- **GPIO33** is **CAMERA_PCLK** - the pixel clock of the camera
- **GPIO2** is **CAMERA_VSYNC** - the vertical sync of the camera
- **GPIO3** is **CAMERA_HREF** - the horizontal sync of the camera
- **GPIO1** is **CAMERA_XCLK** - the input clock pin of the camera
- **GPIO36, 37, 41, 42, 39, 40, 21, 38** are **CAMERA_DATA** - the 8 data pins of the camera

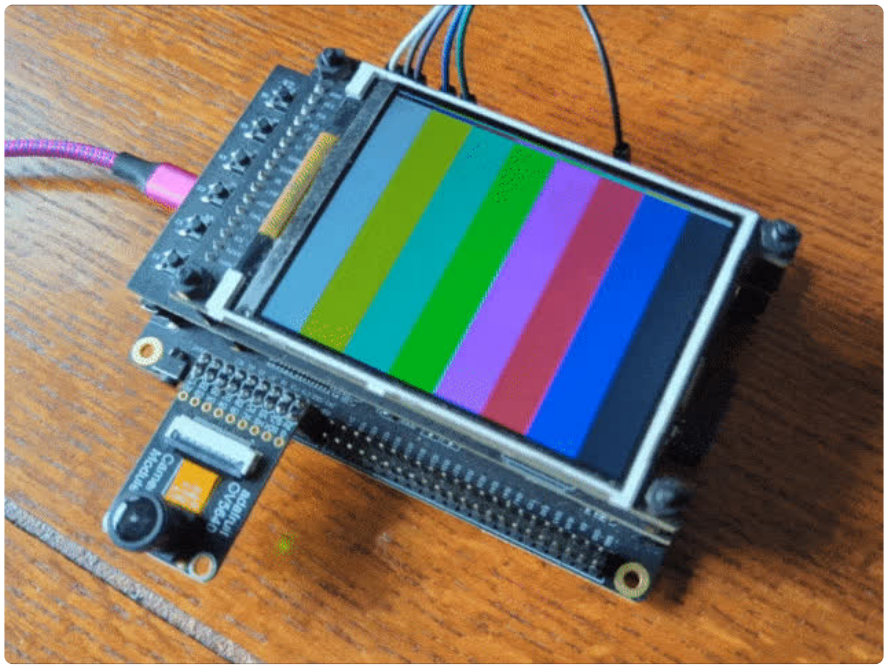
LCD variants

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 for demos that use the LCD you will need to try each variant until you find the one that works.

LCD Mirror Demo



Install & Use the Demo

First, make sure you can see the Kaluga **CIRCUITPY** drive and connect to the REPL. Open the REPL and double check that `import espcamera` works without showing an error. Then, copy one of the Adafruit software bundles to your device (the first appears below, more on subsequent pages). It will automatically reload and start displaying the image from the camera on the built-in LCD.

By clicking the BOOT button you can swap the camera between live mode & test pattern mode.

If the live mode image is black, remove the lens cap from the 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.

```
# SPDX-FileCopyrightText: Copyright (c) 2021 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. It requires CircuitPython 8.

To fix the MemoryError when creating a Camera object, Place the line
```toml
CIRCUITPY_RESERVED_PSRAM=1048576
```
in the file **CIRCUITPY/settings.toml** and restart.
"""
```

```
import struct

import board
import busio
import keypad
import displayio
import busdisplay
import fourwire
import espcamera
import espidf

print("Initializing display")
displayio.release_displays()
spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = fourwire.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\x40"  # 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 = busdisplay.BusDisplay(display_bus, _INIT_SEQUENCE, width=320, height=240)

if espidf.get_reserved_psram() < 1047586:
    print("""Place the following line in CIRCUITPY/settings.toml, then hard-reset
the board:
CIRCUITPY_RESERVED_PSRAM=1048576
""")
    raise SystemExit


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.QVGA,
```

```
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2)
print(cam.width, cam.height)
display.auto_refresh = False

k = keypad.Keys([board.IO0], value_when_pressed=False)

ow = (display.width - cam.width) // 2
oh = (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:
    if (e := k.events.get()) is not None and e.pressed:
        cam.colorbar = not cam.colorbar

    frame = cam.take(1)
    display_bus.send(44, frame)
```

Code Walkthrough

First, the code performs necessary imports and sets up the display. It will stop if it detects the necessary reserved PSRAM setting is not active:

```
# SPDX-FileCopyrightText: Copyright (c) 2021 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. It requires CircuitPython 8.

To fix the MemoryError when creating a Camera object, Place the line
```toml
CIRCUITPY_RESERVED_PSRAM=1048576
```
in the file **CIRCUITPY/settings.toml** and restart.
"""

import struct

import board
import busio
import keypad
import displayio
import fourwire
import espcamera
import espidf

print("Initializing display")
displayio.release_displays()
spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = fourwire.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\x40"  # 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)

if espidf.get_reserved_psram() < 1047586:
    print("""Place the following line in CIRCUITPY/settings.toml, then hard-reset
the board:

CIRCUITPY_RESERVED_PSRAM=1048576
""")
    raise SystemExit
```

Next, the camera object is created. Since this board has a dedicated camera header, there are shorthand names in the **board** module to name the pins; see the Espressif Overview page for how to adapt the example to other boards.

```
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.QVGA,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2)
```

The **BOOT** button (also known as IO0) can be used when the demo is running to turn the test pattern on and off:

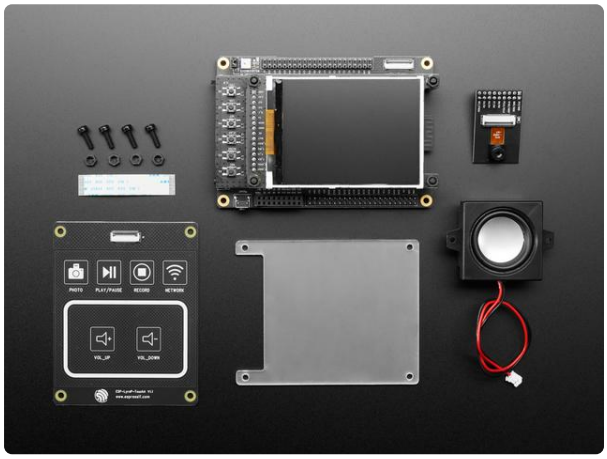
```
k = keypad.Keys([board.IO0], value_when_pressed=False)
```

To improve the refresh rate of the display, the demo directly sends data to the LCD rather than going through displayio. The program's forever-loop repeatedly fetches a frame and then sends its data to the LCD over the fourwire bus:

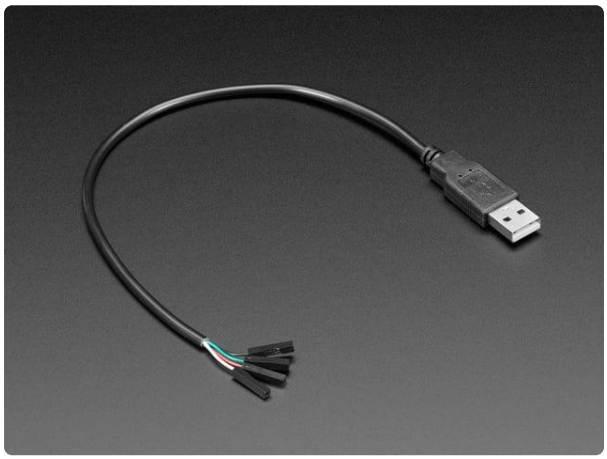
```
ow = (display.width - cam.width) // 2
oh = (display.height - cam.height) // 2
display_bus.send(42, struct.pack(">hh", ow, cam.width + ow - 1))
display_bus.send(43, struct.pack(">hh", oh, cam.height + ow - 1))

while True:
    if (e := k.events.get()) is not None and e.pressed:
        cam.colorbar = not cam.colorbar

    frame = cam.take(1)
    display_bus.send(44, frame)
```



[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

ASCII Mirror Demo

Install & Use the Demo

First, make sure you can see the Kaluga **CIRCUITPY** drive and connect to the REPL. Open the REPL and double check that `import espcamera` 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 serial REPL as lo-fi ASCII art.

By clicking the BOOT button you can swap the camera between live mode & test pattern mode.

You will need to use a terminal program that understands ANSI escape codes such as **screen** or **tio**.

If the live mode image is black, remove the lens cap from the 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.

```
# SPDX-FileCopyrightText: Copyright (c) 2021 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

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

To fix the MemoryError when creating a Camera object, Place the line
```toml
CIRCUITPY_RESERVED_PSRAM=1048576
```
in the file **CIRCUITPY/settings.toml** and restart.
"""

import sys

import board
import keypad
import displayio
import espcamera
import espidf

# The demo runs very slowly if the LCD display is enabled!
# It's intended to be viewed on the REPL on a host computer
displayio.release_displays()

if espidf.get_reserved_psram() < 1047586:
    print("""Place the following line in CIRCUITPY/settings.toml, then hard-reset
the board:
CIRCUITPY_RESERVED_PSRAM=1048576
""")
    raise SystemExit

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.GRAYSCALE,
    frame_size=espcamera.FrameSize.QQVGA,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=2)
print("initialized")

k = keypad.Keys([board.IO0], value_when_pressed=False)

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

sys.stdout.write("\033[2J")

while True:
    if (e := k.events.get()) is not None and e.pressed:
        cam.colorbar = not cam.colorbar

    frame = cam.take(1)

    for j in range(0, cam.height, 5):
        sys.stdout.write(f"\033[{j//5}H")
        for i in range(cam.width // 2):
            row[i] = remap[frame[width * j + 2 * i]]
        sys.stdout.write(row)
```

```
        sys.stdout.write("\033[K")
    sys.stdout.write("\033[J")
```

After code that is familiar from the LCD demo is the start of the code specific to the ASCII art part of the program:

- "chars" holds the ASCII characters to use, arranged from darkest to lightest (the demo is intended to be run on a terminal with a dark background color).
- "remap" is a 256-element look-up table from the raw brightness value to a character
- "width" is just a shorthand way to refer to the camera's width property
- "row" contains one byte for every 2 characters across the image, which gives a width of 80 characters, a standard terminal width.

Finally, the whole screen is cleared.

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

sys.stdout.write("\033[2J")
```

The forever loop grabs a fresh frame and then converts it to ASCII.

Every 5th row of the input image is used, giving 24 lines of height; every 2nd column is taken, given 80 characters of width.

First, an escape code is printed to move the cursor to the start of the correct line.

Then, the ASCII characters for the row are calculated by using the **remap** array

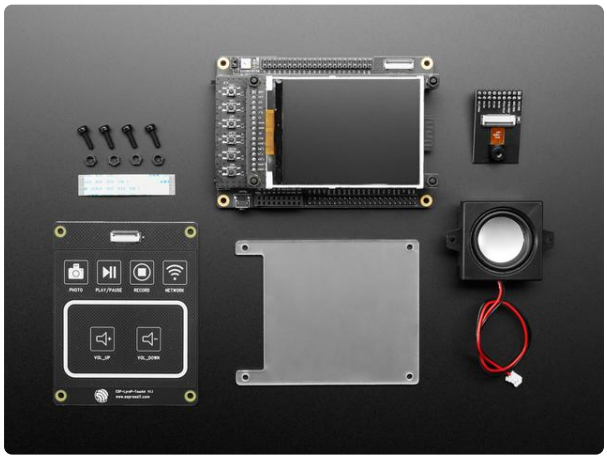
Finally, the row is written, followed by an escape code indicating "clear to end of line".

When the whole thing is written, the remainder of the screen (if any) is cleared.

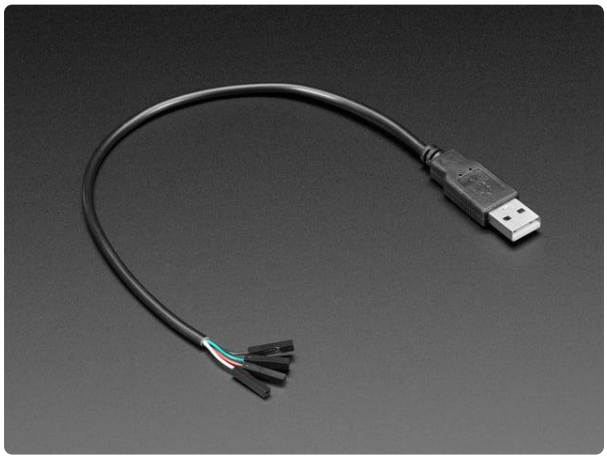
```
while True:
    if (e := k.events.get()) is not None and e.pressed:
        cam.colorbar = not cam.colorbar
```

```
    frame = cam.take(1)

    for j in range(0, cam.height, 5):
        sys.stdout.write(f"\033[{j//5}H")
        for i in range(cam.width // 2):
            row[i] =  remap[frame[width * j + 2 * i]]
        sys.stdout.write(row)
        sys.stdout.write("\033[K")
    sys.stdout.write("\033[J")
```



[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

JPEG Capture Demo

As of CircuitPython 9, a mount point (folder) named /sd is required on the CIRCUITPY drive. Make sure to create that directory after upgrading CircuitPython.

Install & Use the Demo

For this demo, you'll need to attach an SD card breakout to your Kaluga using the following connections:

- **5V** to **5V**, or **3V** to **3V3** (depends on breakout)
- **GND** to **GND**
- **CLK** to **GPIO18**
- **MOSI** to **GPIO14**
- **MISO** to **GPIO10**
- **CS** to **GPIO12**

Now, insert a CircuitPython-supported, formatted SD card. Supported formats include FAT16 and FAT32, but not exFAT.

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

Click the boot button to capture an image in JPEG format to the SD card.

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.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 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. It requires CircuitPython 8.

This demo needs reserved psram properly configured in settings.toml:
CIRCUITPY_RESERVED_PSRAM=1048576

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 "BOOT" button to take a photo in BMP format.
"""

import os
import struct

import board
import busio
import displayio
import busdisplay
import fourwire
import espcamera
import espidf
import keypad
import sdcardio
import storage

print("Initializing display")
displayio.release_displays()
spi = busio.SPI(MOSI=board.LCD_MOSI, clock=board.LCD_CLK)
display_bus = fourwire.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\x40"  # 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 = busdisplay.BusDisplay(display_bus, _INIT_SEQUENCE, width=320, height=240)

if espidf.get_reserved_psram() < 1047586:
    print("""Place the following line in CIRCUITPY/settings.toml, then hard-reset
the board:
CIRCUITPY_RESERVED_PSRAM=1048576
""")
    raise SystemExit

print("Initializing SD card")
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")

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.QVGA,
    i2c=board.I2C(),
    external_clock_frequency=20_000_000,
    framebuffer_count=1)
print("initialized")
display.auto_refresh = False

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")

ow = (display.width - cam.width) // 2
oh = (display.height - cam.height) // 2

k = keypad.Keys([board.IO0], value_when_pressed=False)

while True:
```

```
    frame = cam.take(1)
    display_bus.send(42, struct.pack(">hh", ow, cam.width + ow - 1))
    display_bus.send(43, struct.pack(">hh", oh, cam.height + ow - 1))
    display_bus.send(44, frame)
    if (e := k.events.get()) is not None and e.pressed:
        cam.reconfigure(
            pixel_format=espcamera.PixelFormat.JPEG,
            frame_size=espcamera.FrameSize.SVGA,
        )
        frame = cam.take(1)
        if isinstance(frame, memoryview):
            jpeg = frame
            print(f"Captured {len(jpeg)} bytes of jpeg data")

            with open_next_image() as f:
                f.write(jpeg)
        cam.reconfigure(
            pixel_format=espcamera.PixelFormat.RGB565,
            frame_size=espcamera.FrameSize.QVGA,
        )
```

Code walkthrough

These functions help with opening the next numbered image in a sequence, so that files are named **img0000 img0001** etc.

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

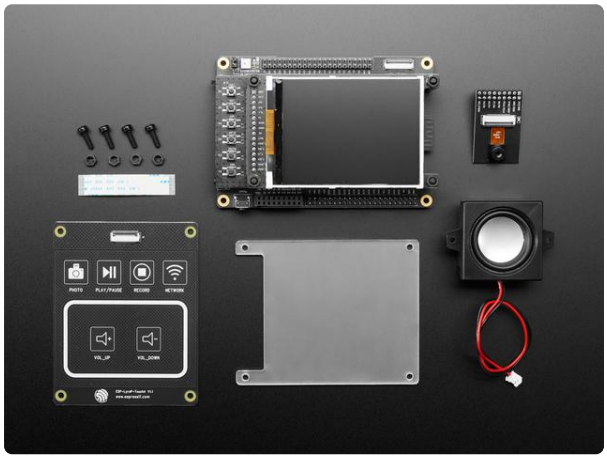

_image_counter = 0


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")
```

Inside the forever loop, when the shutter button is pressed, these lines change the camera to JPEG mode, capture an image to the SD card, and then set the camera back to RGB mode:

```
if (e := k.events.get()) is not None and e.pressed:
        cam.reconfigure(
            pixel_format=espcamera.PixelFormat.JPEG,
            frame_size=espcamera.FrameSize.SVGA,
        )
        frame = cam.take(1)
        if isinstance(frame, memoryview):
            jpeg = frame
            print(f"Captured {len(jpeg)} bytes of jpeg data")

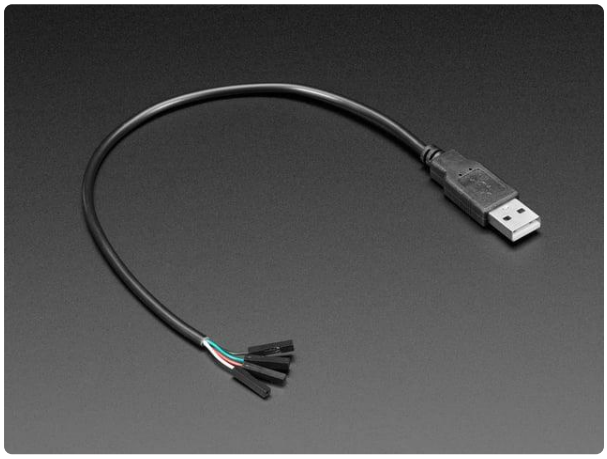
            with open_next_image() as f:
                f.write(jpeg)
        cam.reconfigure(
            pixel_format=espcamera.PixelFormat.RGB565,
            frame_size=espcamera.FrameSize.QVGA,
        )
```



[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

---

espcamera documentation

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

Raspberry Pi RP2040 Overview

The CircuitPython **imagecapture** module is available on most supported RP2040 boards including the Raspberry Pi Pico.

This low-level module works together with a camera-specific high-level module such as **adafruit_ov5640**, to set up the camera.

Image storage

A 320×240 image at 16bpp takes about 150kB of RAM to store. The RP2040 has this much storage, but as memory can become fragmented, it's generally a good idea to allocate storage for the image just once at the beginning of your program. As shown in the example, a `displayio.Bitmap` object can be used for this.

Pin choices

By selecting appropriate pins, you can use the **adafruit_ov5640** library on other boards with the RP2040 microcontroller:

- **xclk**, **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.
- **data_pins**: Any 8 sequential pins in GPIO ordering (e.g., GPIO2..GPIO9).

By convention, if a board has an integrated camera or a dedicated camera connector, the following will exist in the **board** module:

- **CAMERA_SIOC** - the **SCL** pin of the camera
- **CAMERA_SIOD** - the **SDA** pin of the camera
- **CAMERA_PCLK** - the pixel clock of the camera
- **CAMERA_VSYNC** - the vertical sync of the camera
- **CAMERA_HREF** - the horizontal sync of the camera
- **CAMERA_XCLK** - the input clock pin of the camera
- **CAMERA_DATA** - the 8 data pins of the camera

Continue to the next page to see how to use the camera module with the Raspberry Pi Pico module on a permaproto board. Be prepared to do quite a bit of wiring & soldering for the required connections.

Raspberry Pi Pico Usage

Camera connections

For the Raspberry Pi Pico setup shown in these examples, wire the following connections:

- OV5640 **GND** to Pi Pico **GND**
- OV5640 **3V** to Pi Pico **3V3**
- OV5640 **SDA** to Pi Pico **GP8**
- OV5640 **SCL** to Pi Pico **GP9**
- OV5640 **HS** to Pi Pico **GP21**
- OV5640 **VS** to Pi Pico **GP7**
- OV5640 **XC** to Pi Pico **GP20**
- OV5640 **PC** to Pi Pico **GP11**
- OV5640 **D2..D9** to Pi Pico **GP12..GP19**
- OV5640 **RT** to Pi Pico **GP10**

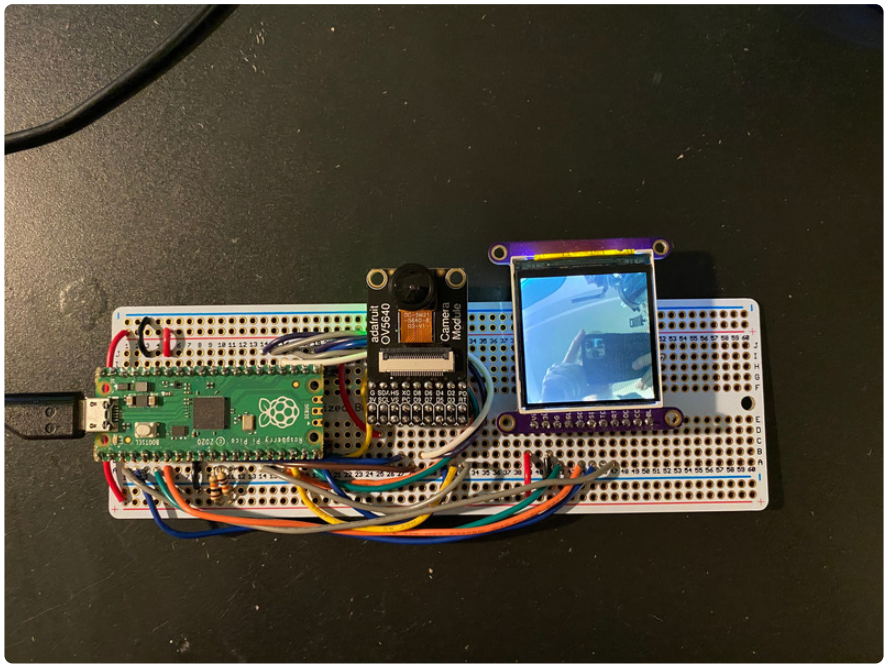
The pull-up resistors shown on GP8 and GP9 are not required for Adafruit's OV5640 camera break-out board but may be required for other camera breakout boards.

> Many of these connections carry high frequency signals. Keep wire lengths short and consider using a perma-proto board to improve signal integrity.

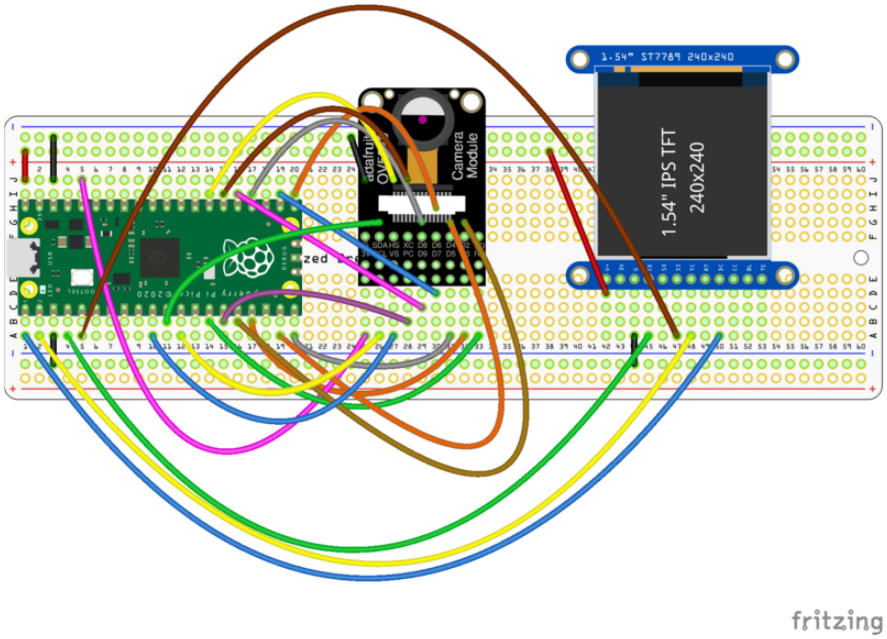
LCD connections

For the LCD display, make the following connections:

- Display **V+** to Pi Pico **VBUS**
- Display **GND** to Pi Pico **GND**
- Display **CK** to **Pi Pico GP2**
- Display **SI** to **Pi Pico GP3**
- Display **DC** to **Pi Pico GP0**
- Display **TC** to Pi Pico **GP1**



Here's a diagram of the many connections required; rather than trying to trace out the connections from this image, though, it's better to use the bullet list of connections above.



Depending whether you are using the optional LCD display, continue on to the LCD or ASCII mirror demo on the following pages.

ASCII Mirror Demo

Install & Use the Demo

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.

You will need to use a terminal program that understands ANSI escape codes such as `screen` or `tio`. Connect to your device using a compatible terminal program and you will see the image captured as lo-fi ASCII art.

If the live mode image is black, remove the lens cap from the camera.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
This demo is designed for the Raspberry Pi Pico.

It shows the camera image as ASCII art on the USB REPL.
"""


import sys
import time
import busio
import board
import digitalio
import adafruit_ov5640

print("construct bus")
bus = busio.I2C(board.GP9, board.GP8)
print("construct camera")
reset = digitalio.DigitalInOut(board.GP10)
cam = adafruit_ov5640.OV5640(
    bus,
    data_pins=(
        board.GP12,
        board.GP13,
        board.GP14,
        board.GP15,
        board.GP16,
        board.GP17,
        board.GP18,
        board.GP19,
    ),
    clock=board.GP11,
    vsync=board.GP7,
    href=board.GP21,
    mclk=board.GP20,
    shutdown=None,
    reset=reset,
    size=adafruit_ov5640.OV5640_SIZE_QQVGA,
)
print("print chip id")
print(cam.chip_id)


cam.colorspace = adafruit_ov5640.OV5640_COLOR_YUV
cam.flip_y = True
cam.flip_x = True
cam.test_pattern = False

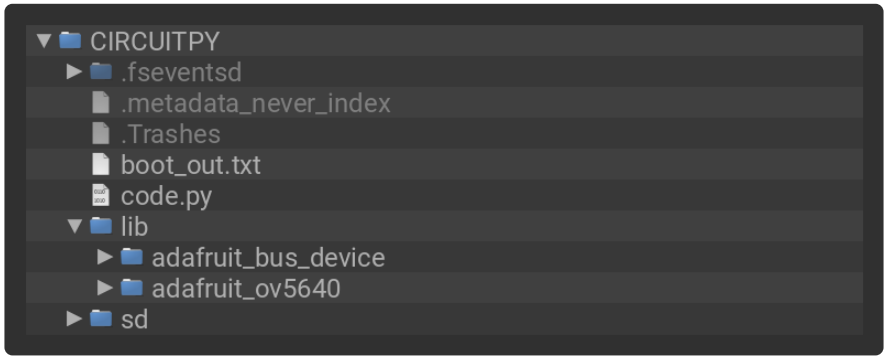
buf = bytearray(cam.capture_buffer_size)
chars = b" .':-+=*%$#"
remap = [chars[i * (len(chars) - 1) // 255] for i in range(256)]

width = cam.width
row = bytearray(width)

print("capturing")
cam.capture(buf)
print("capture complete")

sys.stdout.write("\033[2J")
while True:
    cam.capture(buf)
    for j in range(0, cam.height, 2):
        sys.stdout.write(f"\033[{j//2}H")
```

```
        for i in range(cam.width):
            row[i] = remap[buf[2 * (width * j + i)]]
        sys.stdout.write(row)
        sys.stdout.write("\033[K")
    sys.stdout.write("\033[J")
    time.sleep(0.1)
```



After code that is familiar from the LCD demo is the start of the code specific to the ASCIi art part of the program:

- "chars" holds the ASCII characters to use, arranged from darkest to lightest (the demo is intended to be run on a terminal with a dark background color).
- "remap" is a 256-element look-up table from the raw brightness value to a character
- "width" is just a short-hand way to refer to the camera's width property
- "row" contains one byte for every 2 characters across the image, which gives a width of 80 characters, a standard terminal width.

Finally, the whole screen is cleared.

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

sys.stdout.write("\033[2J")
```

The forever loop grabs a fresh frame and then converts it to ASCII.

Every 5th row of the input image is used, giving 24 lines of height; every 2nd column is taken, given 80 characters of width.

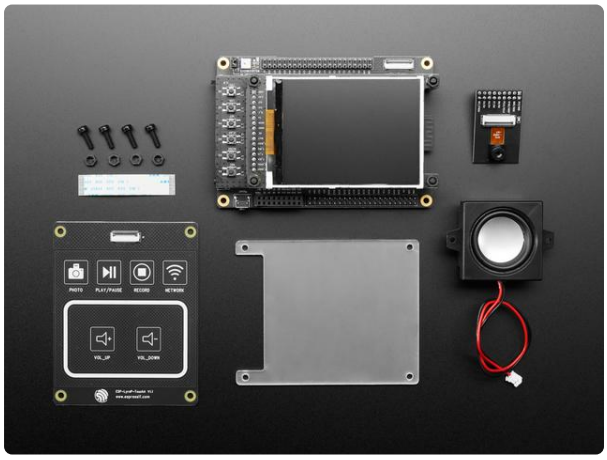
First, an escape code is printed to move the cursor to the start of the correct line.

Then, the ASCII characters for the row are calculated by using the `remap` array

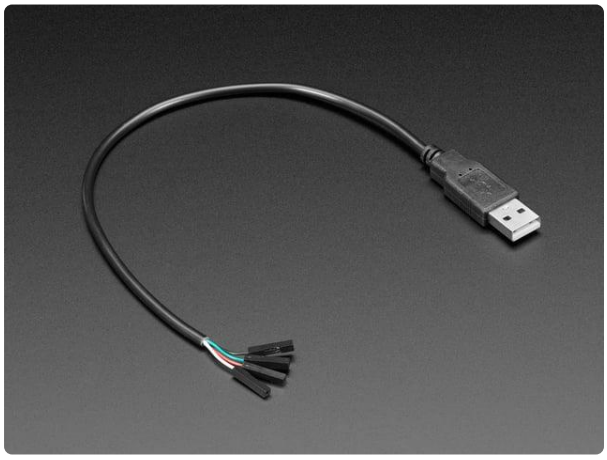
Finally, the row is written, followed by an escape code indicating "clear to end of line".

When the whole thing is written, the remainder of the screen (if any) is cleared.

```
sys.stdout.write("\033[2J")
while True:
    cam.capture(buf)
    for j in range(0, cam.height, 2):
        sys.stdout.write(f"\033[{j//2}H")
        for i in range(cam.width):
            row[i] = remap[buf[2 * (width * j + i)]]
        sys.stdout.write(row)
        sys.stdout.write("\033[K")
    sys.stdout.write("\033[J")
    time.sleep(0.1)
```



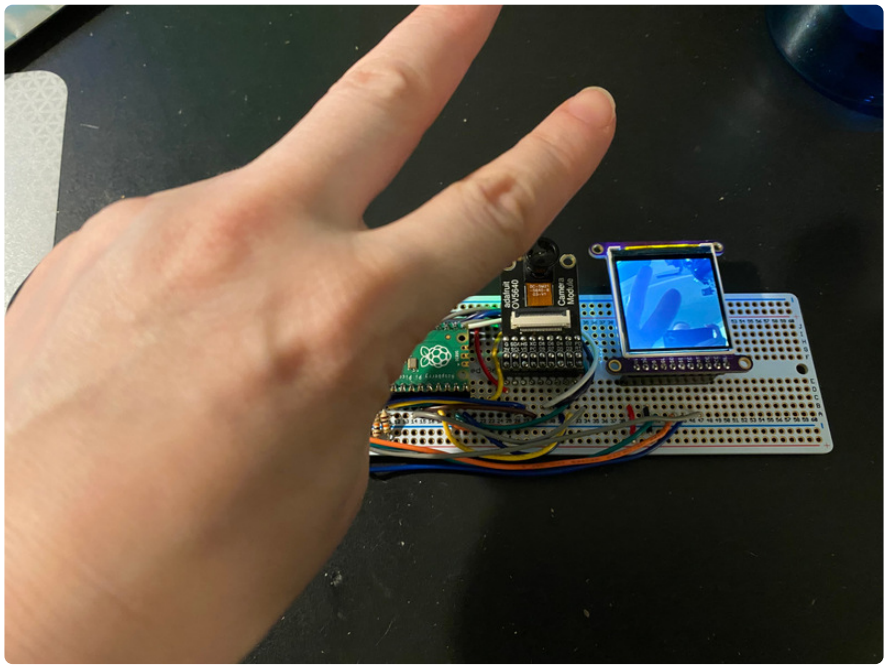
[ESP32-S2 Kaluga Dev Kit featuring ESP32-S2 WROVER]
The ESP32-S2-Kaluga-1 kit is a full featured development kit by Espressif for the ESP32-S2 that comes with everything but the kitchen sink! From TFTs to touch panels,...
https://www.adafruit.com/product/4729



[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

LCD Mirror Demo



Install & Use the Demo

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.

If the live mode image is black, remove the lens cap from the camera.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
This demo is designed for the Raspberry Pi Pico. with 240x240 SPI TFT display

It shows the camera image on the LCD
"""
import time
import busio
import board
import digitalio
import adafruit_ov5640
import adafruit_st7789
import displayio
import fourwire

# Set up the display (You must customize this block for your display!)
displayio.release_displays()
spi = busio.SPI(clock=board.GP2, MOSI=board.GP3)
display_bus = fourwire.FourWire(spi, command=board.GP0, chip_select=board.GP1,
reset=None)
display = adafruit_st7789.ST7789(display_bus, width=240, height=240, rowstart=80,
rotation=0)

print("construct bus")
bus = busio.I2C(board.GP9, board.GP8)
```

```
print("construct camera")
reset = digitalio.DigitalInOut(board.GP10)
cam = adafruit_ov5640.OV5640(
    bus,
    data_pins=(
        board.GP12,
        board.GP13,
        board.GP14,
        board.GP15,
        board.GP16,
        board.GP17,
        board.GP18,
        board.GP19,
    ),
    clock=board.GP11,
    vsync=board.GP7,
    href=board.GP21,
    mclk=board.GP20,
    shutdown=None,
    reset=reset,
    size=adafruit_ov5640.OV5640_SIZE_240X240,
)
print("print chip id")
print(cam.chip_id)

cam.colorspace = adafruit_ov5640.OV5640_COLOR_RGB
cam.flip_y = False
cam.flip_x = False
cam.test_pattern = False

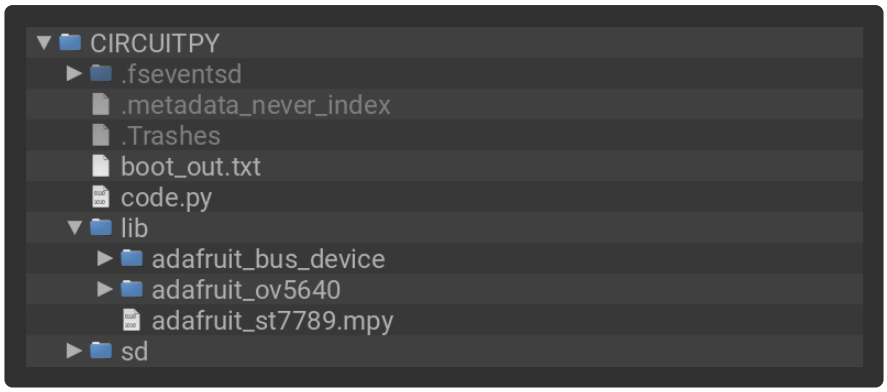
width = display.width
height = display.height

#cam.test_pattern = OV7670_TEST_PATTERN_COLOR_BAR_FADE
bitmap = displayio.Bitmap(cam.width, cam.height, 65535)
print(width, height, cam.width, cam.height)
if bitmap is None:
    raise SystemExit("Could not allocate a bitmap")

g = displayio.Group(scale=1, x=(width-cam.width)//2, y=(height-cam.height)//2)
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
```



Code Walkthrough

First, the code performs necessary imports and sets up the display.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
"""
This demo is designed for the Raspberry Pi Pico. with 240x240 SPI TFT display
It shows the camera image on the LCD
"""
import time
import busio
import board
import digitalio
import adafruit_ov5640
import adafruit_st7789
import displayio
import fourwire

# Set up the display (You must customize this block for your display!)
displayio.release_displays()
spi = busio.SPI(clock=board.GP2, MOSI=board.GP3)
display_bus = fourwire.FourWire(spi, command=board.GP0, chip_select=board.GP1,
reset=None)
display = adafruit_st7789.ST7789(display_bus, width=240, height=240, rowstart=80,
rotation=0)
```

Next, the camera object is created and configured.

```
print("construct bus")
bus = busio.I2C(board.GP9, board.GP8)
print("construct camera")
reset = digitalio.DigitalInOut(board.GP10)
cam = adafruit_ov5640.OV5640(
    bus,
    data_pins=(
        board.GP12,
        board.GP13,
        board.GP14,
        board.GP15,
        board.GP16,
        board.GP17,
        board.GP18,
        board.GP19,
    ),
    clock=board.GP11,
    vsync=board.GP7,
    href=board.GP21,
```

```
    mclk=board.GP20,
    shutdown=None,
    reset=reset,
    size=adafruit_ov5640.OV5640_SIZE_240X240,
)
print("print chip id")
print(cam.chip_id)

cam.colorspace = adafruit_ov5640.OV5640_COLOR_RGB
cam.flip_y = False
cam.flip_x = False
cam.test_pattern = False

width = display.width
height = display.height
```

This demo integrates with displayio for display, so a bitmap object is needed:

```
bitmap = displayio.Bitmap(cam.width, cam.height, 65535)
print(width, height, cam.width, cam.height)
if bitmap is None:
    raise SystemExit("Could not allocate a bitmap")

g = displayio.Group(scale=1, x=(width-cam.width)//2, y=(height-cam.height)//2)
tg = displayio.TileGrid(bitmap,

pixel_shader=displayio.ColorConverter(input_colorspace=displayio.Colorspace.RGB565_SWAPPED)
)
g.append(tg)
display.root_group = g
```

The forever loop grabs a frame to the bitmap and then refreshes the display. It tracks the approximate refresh rate (FPS) of the demo; this demo achieves about 2fps due mostly to the overhead of displayio & communication with the display.

```
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
```

---

adafruit_ov5640 documentation

adafruit_ov5640 documentation (https://adafru.it/18tD)

Downloads

Files

- OV5640 Datasheet (https://adafru.it/18em)
- OV5640 Register Datasheet (https://adafru.it/18en)

- OV5640 Firmware User Guide (https://adafru.it/18eo)
- EagleCAD PCB files on GitHub (https://adafru.it/18uc)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/18ud)

Schematic and Fab Print