Adafruit OV7670 Camera Library For SAMD51 Processors

Created by Phillip Burgess

https://learn.adafruit.com/adafruit-ov7670-camera-library-samd51

Last updated on 2023-08-29 04:29:38 PM EDT
# Table of Contents

**Overview**
- Parts

**Hardware**
- Modifying the Camera
- Power and Connections
- Camera Orientation

**Arduino Examples**
- Install Arduino Library

**Arduino Library**
- Pixel Format

**Arduino Library: Effects**
- In-Camera Effects
- Postprocessing Effects

**Other Hardware Notes**
- Manual Focus
- Pinhole Version
Overview

A feature we were excited to see in the SAMD51 microcontroller — the heart of all our “M4” boards — was the inclusion of a Parallel Capture Controller (PCC), a camera interface that can quickly (30 frames/second) pipe images straight into RAM with only negligible work from the CPU. Whenever possible, we’ve designed our M4 boards to make sure all the necessary pins are routed and accessible for this.

One camera that can interface with the Parallel Capture Controller is the OmniVision OV7670. This model is super common and affordable for hobbyists.

While not up to standards we’d demand from a current smartphone or laptop, it’s nicely balanced to the capabilities of recent 32-bit microcontrollers. A SAMD51 has enough RAM for a 320x240 pixel image. Many projects, like basic machine vision, need only a fraction of that resolution.

Now we have an Arduino library to tie these two together, with examples for the Adafruit M4 Grand Central board. But first we’ll show how to modify one of these inexpensive OV7670 carrier boards to interface with Grand Central’s pin headers...
Parts

Items Needed

- Adafruit Grand Central M4
- Adafruit TFT Touch Shield (resistive or capacitive) or 1.8" Color TFT shield.
- OV7670 camera module — pinout must match images above
- Two 2.2K resistors (color bands: red+red+red+gold)
- Wire
- Soldering iron and related paraphernalia
- Optional: microSD card

OV7670 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...occasionally there are incompatible variants. The cameras are sometimes sold in sets of two...which is a good idea, as they’re easily damaged with the wrong voltage or rough handling as we make some modifications...

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

2.8" TFT Touch Shield for Arduino with Resistive Touch Screen
Spice up your Arduino project with a beautiful large touchscreen display shield with built in microSD card connection. This TFT display is big (2.8" diagonal) bright (4 white-LED... https://www.adafruit.com/product/1651
2.8" TFT Touch Shield for Arduino with Capacitive Touch
Add some sizzle to your Arduino project with a beautiful large touchscreen display shield with built in microSD card connection and a capacitive touchscreen. This TFT...
https://www.adafruit.com/product/1947

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
Hardware

There are two code examples for the library, one for the 2.8" Touch Shields, another for the 1.8" TFT Shield with buttons. The latter example lets you save images to a microSD card.

Modifying the Camera

Gather up...

Camera

2.2K resistors (2)

Two wires, about 4 inches (10 cm) long, color coded if you like

If your camera arrived with a lens cap, keep that on for now to prevent solder splatter or other mayhem on the lens.
Two pins on the OV7670 camera board — 3V and GND — get completely removed.

A good hot iron will both melt the solder and soften the plastic of the row header, allowing the pin to be pulled straight out with tweezers or pliers.

If your iron isn’t up to the task, get creative with flush cutters.

Don’t struggle or you’ll peel a trace off the PCB. Please, gently the kobolds.

With the pins removed, clip away that part of the row header plastic, then clear out the vias with a solder sucker or wick. You should be able to see right through those two holes.
The two resistors will connect to the SDA and SCL pins at one end, and both go to 3.3V at the other end.

We’ll also be putting a wire through the 3.3V via but there’s only so much room, so you might need to economize on space by joining the resistors so only one leg goes into the hole.

Melt the solder on SDA and SCL and tack each resistor into place. Might take a few tries. Then the other end, plus one wire, are soldered to the 3.3V spot.

Other wire goes into the GND via. No resistors there, this one’s straightforward.
Power and Connections

The camera requires 3.3V power. There is NO voltage regulator on the carrier board and 5V will DAMAGE it!

This is easy with the 1.8” TFT shield: there are pins labeled 3V and GND a few spots to the right of the A/B/C buttons.

The larger TFT Touch Shield obscures a lot and requires some creativity to access a 3.3V pin. In this case it was soldered to the corresponding header pad on the back of the shield.
With the camera header now reduced to 16 pins (2 rows x 8 pins), here’s where it plugs into the Grand Central board — pins 24 through 29.

You can quickly align this visually by just skipping the top 2x2 pins.

If the header is misaligned...up or down by one pin...nothing will happen, the camera just won’t respond. But be super extra careful of those top two pins, which carry 5 Volts and would damage the camera.

The 2-row header has a pair of GND pins at the bottom...that’s what we’ll use for ground when using the TFT Touch Shield.

Here’s how everything looks installed. The wiring is unattractive and delicate, but this is wild west stuff.

Camera Orientation

Please note that with the camera installed this way — with the silkscreen on both the camera board and Grand Central in a readable orientation — the camera’s actually rotated 180°. Its idea of “up” is the other way ’round.

The examples compensate for this by also rotating the display output 180°. These are mostly just copying data straight from the camera to the screen and don’t care. But if
you’re doing any actual image processing, anything requiring a specific orientation, keep that in mind, that you’ll want to hold the board the other way, with the silk upside-down.

---

**Arduino Examples**

If this is your first time using the Grand Central board, please begin with our *Introducing the Adafruit Grand Central M4 Express* guide. This will walk you through setting up the Arduino IDE for use with this board.

Do not continue until you have at least the basic “blink” sketch working on Grand Central.

**Install Arduino Library**

Our OV7670 library can be installed from the Arduino Library Manager:

Sketch→Include Library→Manage Libraries...

Enter “OV7670” in the search field. You’ll see more than one library returned...others are working on the idea too...but specifically install the Adafruit_OV7670 library because this one works with the SAMD51 Parallel Capture Controller.

Select Grand Central as the board type from the Tools menu:

Tools→Board→Adafruit Grand Central M4

Our example sketches can then be accessed from the File menu:

File→Examples→Adafruit OV7670
There are two examples to begin with:

- cameratest works with the 2.8" TFT Touch Shield (resistive or capacitive doesn’t matter — the example doesn’t use touch, just the display). This displays live video from the camera at 320x240 pixel resolution.
- selfie works with the 1.8" TFT Shield (V2). A live camera feed is displayed at 160x120 pixel resolution. With a FAT-formatted microSD card in Grand Central’s card slot (not the slot on the shield!), tap the “A” button to capture a still image at 320x240 pixels in BMP format.

Important notes about the “selfie” example

- Use the microSD card slot on Grand Central. Do not use the card slot on the TFT shield.
- Each time you run it and take stills, the code will sequentially overwrite any existing BMP images in the “selfies” folder on the card! Rude. This is a quick demo and not a Real Photography Tool™.
- The BMPs it writes are a lesser-seen variant that not everything can read. Photoshop, Preview on Mac and ImageMagick should all handle it.

Can this work with other boards besides Grand Central?

The examples, for now, are written for Grand Central, but perhaps others will come later.

In theory, any “M4” board should work with the library, provided the hardware exposes all of the PCC pins plus a Timer/Counter pin. But few if any boards besides Grand Central will have enough extra pins left to also connect much else.

Note the Arduino Due is not a SAMD51 / M4 compatible board and this process will NOT work for that board.

Arduino Library

The Arduino library is pretty minimal right now but handles the most important low-level ugly stuff.
The examples show all the vital steps, but it’s mixed in with a lot of weird TFT display-specific code. Let’s look at just the camera bits...

Sketches should begin by including the Wire and Adafruit_OV7670 libraries:

```cpp
#include <Wire.h>            // I2C comm to camera
#include "Adafruit_OV7670.h" // Camera library
```

The Wire library is used for camera control over I2C, while image data comes through the PCC data pins.

In the globals section of the sketch...outside the `setup()` and `loop()` functions...we set up some structures and call the OV7670 constructor.

The arch structure contains values specific to the SAMD51 hardware. In principle, in the future, there might be different arch structures for different hardware. If using Grand Central, you can just copy this line to your own code. For other M4 boards, you need to specify what timer/counter peripheral (PWM out) connects to the camera’s XCLK input, and, if the timer is a TCC peripheral, if special pin multiplexing is required for it (super esoteric, probably won’t need to change).

The pins structure specifies Arduino pin numbers where the camera’s enable, reset and XCLK pins are connected. On the SAMD51, the PCC pins are set in stone and can’t be assigned to other locations, but these few pins are OK being routed to other locations. The examples are set up for the Grand Central header.

Then the constructor is invoked...we’ll call our camera object “cam,” and it expects, in this order:

1. An I2C address (pass OV7670_ADDR, the standard address for an OV7670).
2. A pointer (&) to a pins structure (previously declared).
3. A pointer (&) to a Wire (I2C) instance. Grand Central has several...one of those, Wire1, is conveniently on pins 24 (SCL) and 25 (SDA), which aligns with the camera board, almost like it was planned this way.
4. A pointer (&) to an arch structure (previously declared).

```cpp
OV7670_arch arch = {.timer = TCC1, .xclk_pdec = false};
OV7670_pins pins = {.enable = PIN_PCC_D8, .reset = PIN_PCC_D9, .xclk = PIN_PCC_XCLK};
Adafruit_OV7670 cam(OV7670_ADDR, &pins, &Wire1, &arch);
```
Later, inside the `setup()` function, we initialize the camera by calling its `begin()` function. This expects at least three arguments:

1. A color mode, either `OV7670_COLOR_RGB` or `OV7670_COLOR_YUV`. RGB is best for showing color images on TFTs, but some applications such as object tracking may want grayscale data, which is more easily extracted from YUV.
2. An initial image size, from one of the values #defined in `Adafruit_OV7670.h`. In most situations the library will attempt to allocate a buffer large enough for an image this size. Possible values include:
   - `OV7670_SIZE_DIV1` — 640x480 pixels, don’t bother using this right now because there’s not enough RAM to buffer a full image this size on current SAMD51 chips.
   - `OV7670_SIZE_DIV2` — 320x240 pixels (a division-by-two of 640x480). This is the largest size that most M4s can handle.
   - `OV7670_SIZE_DIV4` — 160x120 pixels (division by 4 of 640x480).
   - `OV7670_SIZE_DIV8` — 80x60 pixels (ditto, 8).
   - `OV7670_SIZE_DIV16` — 40x30 pixels (16).
3. A desired frame rate, as a floating-point value. The actual frame rate may be different from this, but it will do its best to match. The maximum supported by this camera is 30.0 frames/second.

It is IMPORTANT to check the return value from the `begin()` function — this tells you whether it initialized successfully and the camera is working. Possible return values include:

- `OV7670_STATUS_OK` on success.
- `OV7670_STATUS_ERR_MALLOC` if image buffer couldn’t be allocated (insufficient or fragmented RAM).
- `OV7670_STATUS_ERR_PERIPHERAL` if an invalid timer peripheral was passed to the constructor earlier.

```cpp
OV7670_status status = cam.begin(OV7670_COLOR_RGB, OV7670_SIZE_DIV2, 30.0);
if (status != OV7670_STATUS_OK) {
  Serial.println("Camera begin() fail");
  for(;;);
}
```

An optional fourth argument to `begin()` takes an image buffer size, in bytes. There are some situations (as in the “selfie” example) where the camera might change between small and large image resolutions. It’s best in these cases to pre-allocate the image buffer to the largest anticipated image size, because it might not be possible to change later. Here we’re initially using a DIV4 (160x120 pixel) image...but allocating enough for a DIV2 (320x240) image (each pixel is 2 bytes, whether RGB or YUV):
0V7670_status status = cam.begin(OV7670_COLOR_RGB, OV7670_SIZE_DIV4, 30.0, 320 * 240 * 2);

After `begin()` returns an **OK** status, the camera is now continuously dumping frames into a section of RAM. We can query the address of the image buffer, and the image dimensions in pixels, using:

```c
uint16_t *pixel_data = cam.getBuffer();
uint16_t width = cam.width();
uint16_t height = cam.height();
```

However...because the camera is continually dumping data, it's possible you or the camera might overtake one another when accessing this, resulting in a visible “tear” across the image.

So, before accessing the image, it's recommended to first pause the camera:

```c
cam.suspend();
```

(This is not a sleep mode, it just pauses the data spigot.)

Read what you need from the image buffer, then resume camera streaming with:

```c
cam.resume();
```

You can also keep the camera paused and read individual frames with:

```c
cam.capture();
```

The image data will then be in the same location as returned by `getBuffer()`.

`suspend()` has slightly less latency than `capture()`, since it returns as soon as the current in-flight frame is received, rather than starting on the next frame.

**Pixel Format**

The OV7670 is “big endian” — for each 16-bit pixel, the most significant byte is at the lower address in memory.

This is the opposite of the SAMD51 and most other 32-bit microcontrollers, which are “little-endian.” If you need to dismantle and process individual pixels, a byte swap is often necessary:
uint16_t le_pixel = __builtin_bswap16(be_pixel);

Most TFT displays are also big-endian, so the examples don’t need to do this byte swapping…they can just move data directly from the camera to the display, it’s super smooth and buttery.

In OV7670_COLOR_RGB mode, each 16-bit pixel has 5 bits of red, 6 bits green and 5 bits blue. In OV7670_COLOR_YUV mode, 8 bits are brightness and 8 for color… you can work with just the brightness byte for higher-quality grayscale than in RGB mode.

Arduino Library: Effects

The Adafruit_OV7670 library provides a number of special image effects. Some of these are “in-camera” effects — every frame from the camera continuously comes out this way in real time, with no processing required on the host microcontroller. Others are “postprocessing” effects, requiring that the camera be paused while the host microcontroller does a number on the last-received image in memory.

For reference, here’s a normal unmodified scene captured from the OV7670 and shown on a TFT display.
In-Camera Effects

The camera can mirror (flip) the image on the horizontal and/or vertical axes, set with the `flip()` function. This expects two boolean values (true or false, or 1 or 0) to select flips for the horizontal and vertical axes, respectively:

```javascript
cam.flip(false, false);  // Disables flips, captures images normally.

// Enables horizontal flip only.
// This can be helpful for selfie previews...like a mirror, what happens on your left or right shows on the screen's corresponding left or right.

// Enables vertical flip only.

// Flips both axes. This is equivalent to 180 degree rotation, and can be helpful if your camera and display must be physically mounted in opposite orientations (or, many screens also have their own function providing the same).
```
Night mode can sometimes take better images in low-light situations. The tradeoff is a reduced frame rate, as the camera is adding up several images over time. The maximum number of frames to accumulate can be specified...though, if lighting is sufficient, the camera might ignore this and use a lesser setting. It’s enabled or disabled with:

```c
cam.night(setting);
```

where `setting` is one of:

- `OV7670_NIGHT_MODE_8` — Accumulate up to 8 frames maximum.
- `OV7670_NIGHT_MODE_4` — Up to 4 frames max.
- `OV7670_NIGHT_MODE_2` — Up to 2 frames.
- `OV7670_NIGHT_MODE_OFF` — Disable night mode and return to normal full frame rate.
The camera can output test patterns, if that's useful to anybody:

```c
cam.test_pattern(setting);
```

where `setting` is one of:

- `OV7670_TEST_PATTERN_SHIFTING_1` — single-pixel-wide vertical RGB stripes.
- `OV7670_TEST_PATTERN_COLOR_BAR` — 8 color bars (second bar is yellow, but is “blown out” by the exposure in this photo, sorry).
- `OV7670_TEST_PATTERN_COLOR_BAR_FADE` — 8 color bars with a fade to white.
- `OV7670_TEST_PATTERN_NONE` — Turn off test pattern and return to normal image capture.

### Postprocessing Effects

The following effects are generated in code, not by the camera. It's therefore necessary to pause the camera output before performing any of these operations, else the next incoming frame will overwrite the interim results in RAM.

```c
// Pause camera before processing image
cam.suspend();

// Call processing function(s)
cam.image_edges();
```
All of the postprocessing function names begin with `image_`

You can chain multiple processing functions, each will modify the output of the prior function. For example, `image_median()` then `image_edges()`. The order of operations will affect the outcome; they are not interchangeable.

Remember that these only process the last-captured image in memory. They are not continuously applied while the camera is “live.” You must suspend, process, do something with the image data, then resume.

Some of these functions only work in RGB mode, YUV is not always supported.

```java
// Do something with image data here -- TFT display, SD card, etc.
// Finished with image, return image buffer to camera
cam.resume();
```

`cam.image_negative()` inverts the image — light becomes dark, dark becomes light, a red dragon becomes cyan.
`cam.image_threshold()` reduces an image to 1 bit each for red, green and blue. It accepts an optional argument (0 to 255, default is 128) to set the brightness level below/above which the 1-bit determination is made.

`cam.image_posterize(n)` reduces the number of brightness “steps” or levels in an image. Whereas `image_threshold()` always results in 2 levels, `image_posterize()` can generate some in-between brightness levels — 3, 4, 5 and so forth — specified by the single argument.

The following work with RGB images only, YUV is not handled.

`cam.image_mosaic(tile_width, tile_height)` applies a low-res “shower door effect” to the image, averaging all the pixels within each block.

The two arguments are the width and height (in pixels) of the mosaic tiles. These do not need to be powers of two, anything >= 1 will suffice. If the image size does not divide evenly into the tile size, the fractional tiles will always be along the right and/or bottom edge(s).
cam.image_median() reduces the amount of pixel “noise” in an image while generally preserving higher contrast details.

This is a fairly math-intensive operation and might only manage 1-2 frames per second, that’s normal.

cam.image_edges() looks for high-contrast changes in an image and generally highlights object edges.

This requires objects be in-focus and adequately lit. In some cases the result might be nearly empty or totally full of pixel “snow,” so image_edges() accepts an optional value to configure the edge sensitivity — pass a value from 0 to 31 (default is 4), where smaller values make it more sensitive to edges, larger values less so.

Other Hardware Notes

Manual Focus

The OV7670 is a fixed-focus camera — it can’t automatically compensate for near or far subjects. From the factory, it seems to work best about 1 meter (3 feet) from a subject. Anything farther or closer (especially closer) will be progressively more blurry. But we can manually tweak that.
There's a tiny set screw on the side of the camera. Use a correspondingly tiny jeweler's screwdriver to loosen this screw a couple turns.

The lens is threaded and can be turned now.

Turning clockwise (looking at the lens) will tune this to focus on more distant subjects.

Turning counterclockwise will help focus on closer subjects.

A couple of full counterclockwise turns and the camera can do incredible close-ups, just a few millimeters in front of it! This could even be used for a simple "video microscope," if the subject is adequately lit.
Pinhole Version

There’s also an incredibly tiny (7mm square) “pinhole” version of the OV7670. Costs a bit more, focus isn’t so adjustable, but it’s so tiny. This version often comes installed on a similar carrier board.

If you make the same power and resistor changes on the carrier board, this version is interchangeable with the larger camera. There are two extra pins…it still fits in the Grand Central 2-row header, those extra pins just aren’t used right now.

Although the extra pins are labeled D0 and D1 (with the usual other 8 being D2 through D9), that’s not really true...D2-D9 are really the PCC D0-D7 pins. Not certain, but this might be a “FIFO” variant of the camera, which could allow capturing images larger than will fit in RAM...though the library doesn’t support this yet.