

QPAD-XIAO Embedded Systems Programming

Chat transcript and development record

Generated September 30, 2026

This PDF preserves the user/assistant conversation used to set up, program, test, debug, and extend a QPAD-XIAO embedded system based on the Sseed Studio XIAO RP2040. System/developer/tool messages are omitted; attached screenshots and photos are included at the points where they appeared in the conversation.

User

I need to complete this assignment:

write and test a program for an embedded system using a microcontroller to interact (with input &/or output devices) and communicate (with wired or wireless connections)

I am using the QPAD Xiao. How do I go about doing the programming bit? I have already soldered/assembled my circuit

Assistant

Since you've already assembled the QPAD-Xiao, the programming part is basically:

computer → USB-C → XIAO RP2040 → read buttons → drive LEDs / send data

The QPAD-Xiao is specifically based on the RP2040, and the project was designed to work with both Arduino and MicroPython. For your assignment, I'd use Arduino IDE unless your class explicitly prefers Python, because it makes GPIO, serial communication, and testing very straightforward.

A good first project that satisfies the assignment would be: press a QPAD button → microcontroller detects it → lights an LED → sends the button state over USB Serial to your computer. That covers input, output, and wired communication.

1. Install Arduino IDE and the RP2040 board support

Install Arduino IDE, then add support for the XIAO RP2040. Once installed, connect the QPAD to your laptop with a USB data cable. In Arduino IDE, select the appropriate XIAO RP2040 board and the serial port that appears.

2. Start with the simplest possible program

Before worrying about the whole QPAD, make sure you can upload code at all.

```
void setup() {  
  Serial.begin(115200);  
}  
  
void loop() {  
  Serial.println("QPAD is alive");  
  delay(1000);  
}
```

Upload it, then open Tools → Serial Monitor and set it to 115200 baud. You should see “QPAD is alive” once per second.

3. Then test one button

Once you know the GPIO used by a button, the code is simple. For an ordinary mechanical button you might use INPUT_PULLUP and digitalRead().

4. Add an output

You can also control an LED and send a message over Serial. That gives you input, processing, output, and communication.

5. For the actual QPAD, read all the buttons

Once you know the six pad pins, you can make it act like a small controller and send a command for each input.

What I would document for the assignment

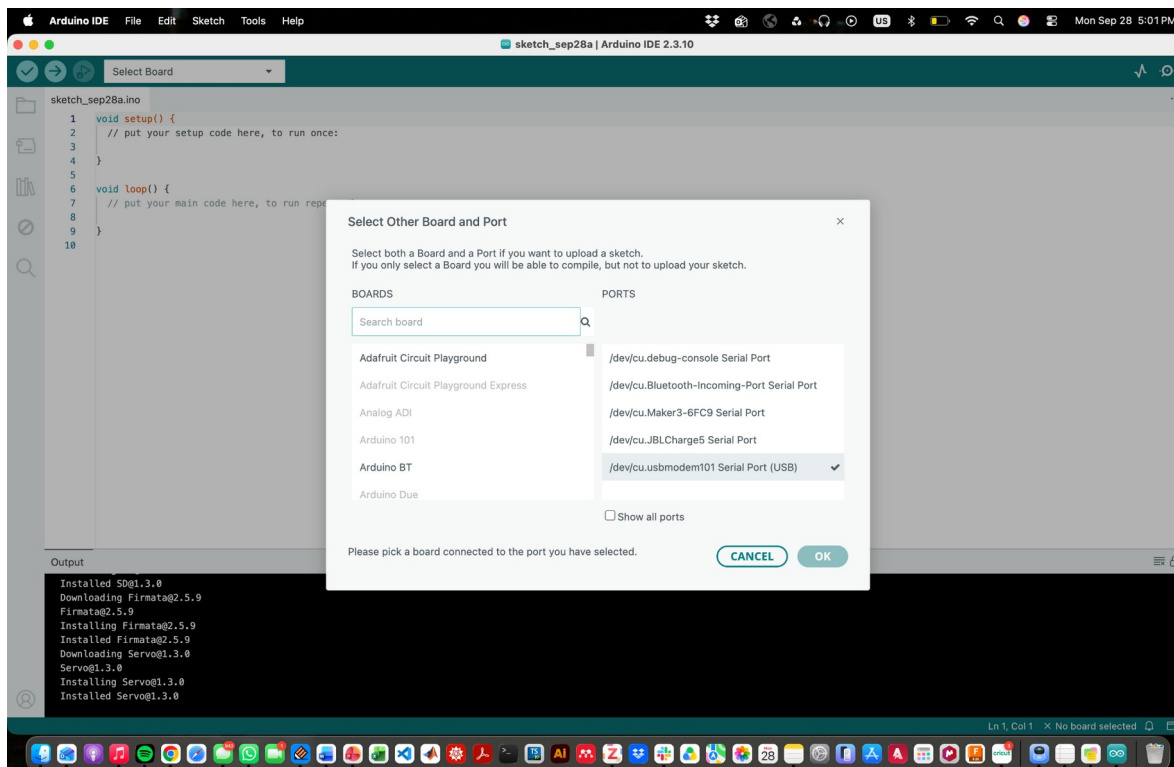
You can describe the system roughly like this:

“I programmed a QPAD-Xiao embedded system based on the RP2040 microcontroller. The microcontroller reads physical inputs using GPIO, updates an output, and transmits state to a computer over USB serial communication. The program was written in C++ using the Arduino framework and tested using the Arduino Serial Monitor.”

The main thing I wouldn't guess is the QPAD's actual input/display pin mapping. If you send me a photo of your finished QPAD, your schematic, or the pinout from the QPAD-Xiao files, I can give you the exact Arduino sketch for your board rather than placeholder pins.

User

this is what I see after plugging in the qpad to my laptop, what do I need to do now



Attached image from the conversation

Assistant

You're in the right place. The important thing in your screenshot is that your Mac sees the QPAD as a USB serial device — `/dev/cu.usbmodem101` — but Arduino IDE does not yet have the XIAO RP2040 board definition installed, so you can't select the board yet.

Do this next:

1. Click Cancel on that “Select Other Board and Port” window.
2. In Arduino IDE, go to Arduino IDE → Settings (or File → Preferences).
3. Find Additional Boards Manager URLs.
4. Add this exact URL:

```
https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json
```

5. Click OK.
6. Open Tools → Board → Boards Manager...

7. Search for arduino-pico.

8. Install Raspberry Pi Pico/RP2040/RP2350.

After it finishes installing, go back to the board selector at the top left. Search for “Seeed Studio XIAO RP2040” and select it. Then select /dev/cu.usbmodem101.

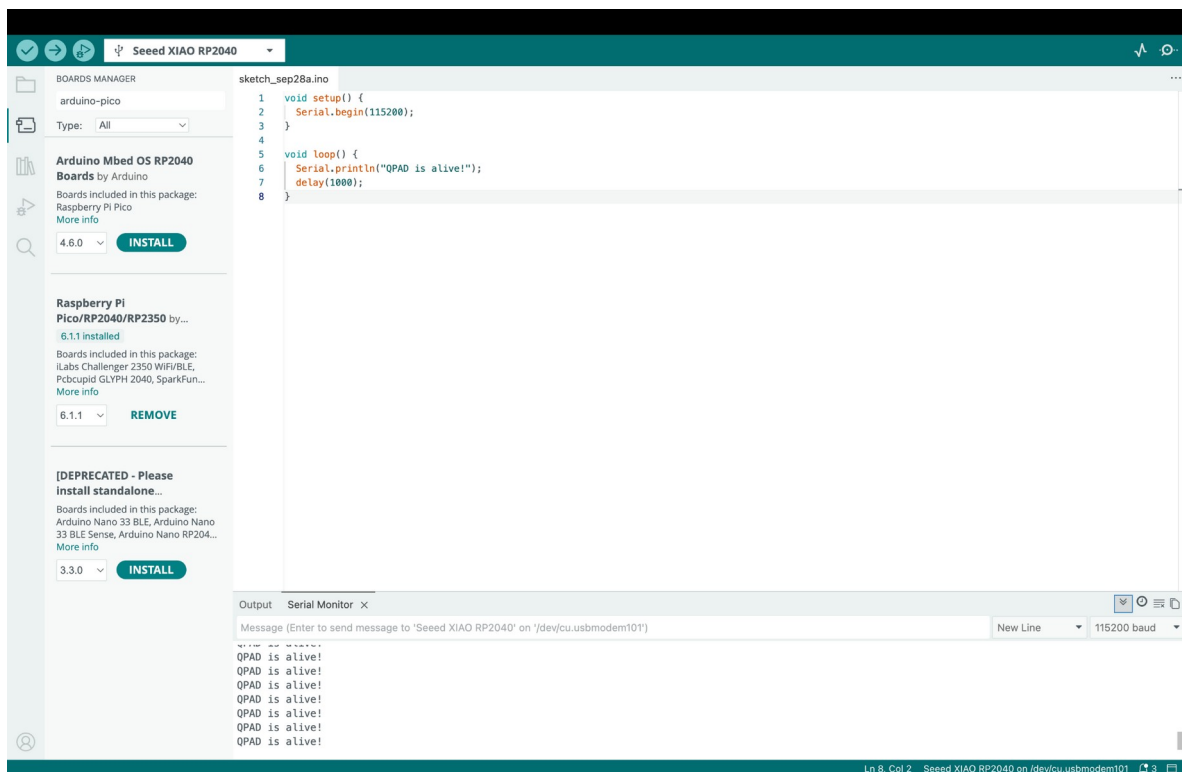
Then do a first upload test with:

```
void setup() {  
  Serial.begin(115200);  
}  
  
void loop() {  
  Serial.println("QPAD is alive!");  
  delay(1000);  
}
```

Open Tools → Serial Monitor and set the baud rate to 115200. If you see the message repeatedly, you’ve completed the board/USB setup and we can move on to the QPAD hardware.

User

it's not popping up on my screen (the one on the qpad)



Attached image from the conversation

Assistant

That behavior is expected from the code you uploaded. Right now your program only contains:

```
Serial.println("QPAD is alive!");
```

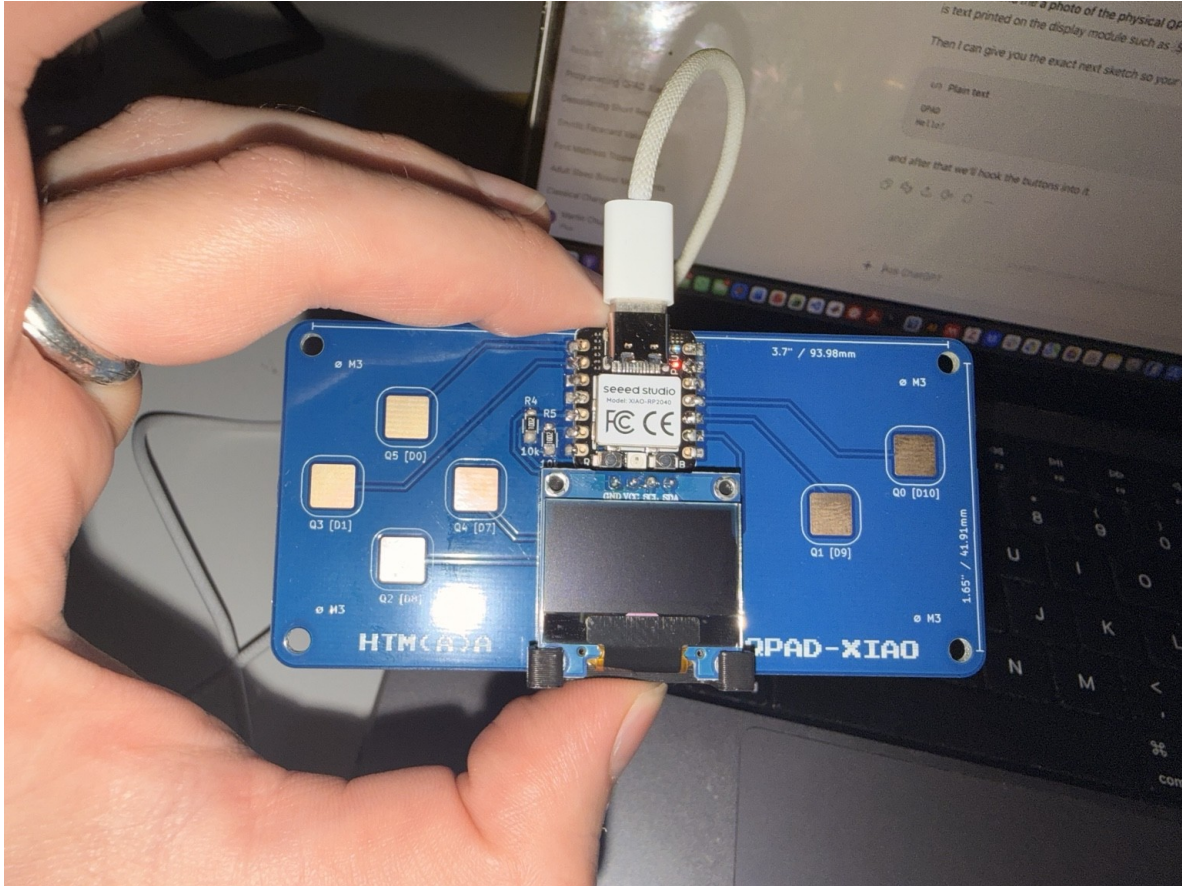
Serial.println() sends text over USB to your computer, which is why you correctly see “QPAD is alive!” in the Serial Monitor. It does not draw anything on the QPAD’s physical screen.

So the good news is: your XIAO RP2040 is programmed correctly and communication with your laptop is working.

To put text on the QPAD display, we need to initialize and write to the actual display hardware using its display controller/library. Can you send me a photo of the physical QPAD you assembled, especially the screen and PCB? If there is text printed on the display module such as SSD1306, SH1106, ST7789, etc., include that too.

User

[Photo of assembled QPAD-XIAO]



Attached image from the conversation

Assistant

Yep — this photo helps a lot.

Your display is a 4-pin I²C OLED. The labels on the module are:

```
GND VCC SCL SDA
```

That means the next step is to talk to the screen over I²C. The display is very likely an SSD1306-style OLED.

First, install two Arduino libraries:

```
Adafruit SSD1306
Adafruit GFX Library
```

Then replace your current sketch with this:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
```

```

Adafruit_SSD1306 display(
  SCREEN_WIDTH,
  SCREEN_HEIGHT,
  &Wire,
  OLED_RESET
);

void setup() {
  Serial.begin(115200);
  Wire.begin();

  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println("OLED failed to start");
    while (true);
  }

  Serial.println("OLED started!");

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 10);
  display.println("QPAD");

  display.setTextSize(1);
  display.setCursor(10, 40);
  display.println("Hello!");

  display.display();
}

void loop() {
}

```

If everything is wired/configured correctly, the physical QPAD OLED should show “QPAD” and “Hello!”.

User

okay that worked, I see it display on the screen

Assistant

Perfect — that means the display, I²C wiring, board selection, and upload process are all working.

Your next step should be to add the QPAD buttons as inputs. A good next program is: press a pad → show its name on the OLED → also print it to Serial. That gives you input, output, and wired communication in one demo.

An initial sketch might look like this:

```

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Example QPAD inputs
const int PAD_0 = D0;
const int PAD_1 = D1;
const int PAD_7 = D7;
const int PAD_9 = D9;
const int PAD_10 = D10;

```

```

void showMessage(const char* msg) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(10, 20);
  display.println(msg);
  display.display();
  Serial.println(msg);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);

  pinMode(PAD_0, INPUT_PULLUP);
  pinMode(PAD_1, INPUT_PULLUP);
  pinMode(PAD_7, INPUT_PULLUP);
  pinMode(PAD_9, INPUT_PULLUP);
  pinMode(PAD_10, INPUT_PULLUP);

  showMessage("READY");
}

void loop() {
  if (digitalRead(PAD_0) == LOW) { showMessage("D0"); delay(200); }
  if (digitalRead(PAD_1) == LOW) { showMessage("D1"); delay(200); }
  if (digitalRead(PAD_7) == LOW) { showMessage("D7"); delay(200); }
  if (digitalRead(PAD_9) == LOW) { showMessage("D9"); delay(200); }
  if (digitalRead(PAD_10) == LOW) { showMessage("D10"); delay(200); }
}

```

If some inputs don't respond, we can correct the sensing method once we confirm the QPAD touch-pad hardware.

User

wait, before I keep trying random shit, walk me through the code. use the one that I ran to get the screen to display "QPAD Hello!"

what is each part of the code doing? what is the associated syntax?

Assistant

Absolutely. Here's the exact sketch you ran, broken down piece by piece.

```

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(
  SCREEN_WIDTH,
  SCREEN_HEIGHT,
  &Wire,
  OLED_RESET
);

void setup() {
  Serial.begin(115200);
  Wire.begin();
}

```

```

if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
  Serial.println("OLED failed to start");
  while (true);
}

Serial.println("OLED started!");
display.clearDisplay();
display.setTextSize(2);
display.setTextColor(SSD1306_WHITE);
display.setCursor(10, 10);
display.println("QPAD");
display.setTextSize(1);
display.setCursor(10, 40);
display.println("Hello!");
display.display();
}

void loop() {
}

```

The easiest way to understand Arduino code is to separate it into four ideas: libraries, constants/variables, setup(), and loop().

The `#include` lines import libraries. `Wire` handles I²C communication. `Adafruit_GFX` provides general drawing and text commands. `Adafruit_SSD1306` knows how to control the SSD1306 OLED.

The `#define` lines create named constants: the screen is 128 × 64 pixels, -1 indicates no separate reset GPIO, and 0x3C is the display's I²C address.

`Adafruit_SSD1306 display(...)` creates an object called `display`. Its type is `Adafruit_SSD1306`. The arguments tell it the width, height, I²C interface (`&Wire`), and reset configuration.

`void setup()` defines a function Arduino runs once at startup. `void` means the function doesn't return a value. `Serial.begin(115200)` starts serial communication. `Wire.begin()` starts I²C.

The `if (!display.begin(...))` block checks whether display initialization failed. `!` means NOT. If initialization fails, the program prints an error and stops in `while (true);`.

`display.clearDisplay()` clears the display's internal framebuffer. `setTextSize`, `setTextColor`, and `setCursor` configure the text. `display.println()` draws text into the framebuffer. `display.display()` sends that framebuffer to the physical OLED.

`void loop()` is repeatedly executed forever. In this sketch it is empty, so the screen stays as it was after `setup()`.

User

I uploaded this on arduino and my screen on qpad now says READY, but it does nothing when I hit the buttons

```

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Example QPAD inputs
const int PAD_0 = D0;
const int PAD_1 = D1;
const int PAD_7 = D7;

```

```

const int PAD_9 = D9;
const int PAD_10 = D10;

void showMessage(const char* msg) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(10, 20);
  display.println(msg);
  display.display();
  Serial.println(msg);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  pinMode(PAD_0, INPUT_PULLUP);
  pinMode(PAD_1, INPUT_PULLUP);
  pinMode(PAD_7, INPUT_PULLUP);
  pinMode(PAD_9, INPUT_PULLUP);
  pinMode(PAD_10, INPUT_PULLUP);
  showMessage("READY");
}

void loop() {
  if (digitalRead(PAD_0) == LOW) { showMessage("D0"); delay(200); }
  if (digitalRead(PAD_1) == LOW) { showMessage("D1"); delay(200); }
  if (digitalRead(PAD_7) == LOW) { showMessage("D7"); delay(200); }
  if (digitalRead(PAD_9) == LOW) { showMessage("D9"); delay(200); }
  if (digitalRead(PAD_10) == LOW) { showMessage("D10"); delay(200); }
}

```

Assistant

The issue is that I treated those gold pads like ordinary mechanical buttons. They are capacitive touch pads, so `digitalRead(...)=LOW` with INPUT_PULLUP is the wrong way to read them.

The RP2040 can sense these pads by timing how long each pad takes to charge. A finger increases capacitance and therefore increases the charge time.

First run a stripped-down diagnostic:

```

#define N_TOUCH 6
#define THRESHOLD 6

int touch_pins[N_TOUCH] = {3, 4, 2, 27, 1, 26};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};

void updateTouch() {
  int t;
  int t_max = 200;

  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];

    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);

    t = 0;
    while (!digitalReadFast(p) && t < t_max) {
      t++;
    }
  }
}

```

```

        touch_values[i] = t;
    }
}

void setup() {
    Serial.begin(115200);
}

void loop() {
    updateTouch();
    for (int i = 0; i < N_TOUCH; i++) {
        Serial.print(touch_values[i]);
        Serial.print(" ");
    }
    Serial.println();
    delay(50);
}

```

The code discharges each pad, enables a weak pull-up, then counts how long the pin remains LOW. A finger adds capacitance, so that count rises.

User

all the numbers read either 0 or 1, and when I touch the pad corresponding to the column (i.e., Q0 is the first column, Q1 is the second, etc), the numbers jump to about 100

Assistant

Perfect — that means the touch sensing is working exactly the way we want.

Your baseline is roughly 0–1 untouched and about 100 touched, so a threshold like 20 gives a comfortable margin.

Here is a combined touch + OLED version:

```

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
int touch_pins[N_TOUCH] = {3, 4, 2, 27, 1, 26};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};

void updateTouch() {
    int t;
    int t_max = 200;
    for (int i = 0; i < N_TOUCH; i++) {
        int p = touch_pins[i];
        pinMode(p, OUTPUT);
        digitalWriteFast(p, LOW);
        delayMicroseconds(25);
        pinMode(p, INPUT_PULLUP);
        t = 0;
        while (!digitalReadFast(p) && t < t_max) { t++; }
        touch_values[i] = t;
    }
}

void showMessage(const char* msg) {

```

```

display.clearDisplay();
display.setTextColor(SSD1306_WHITE);
display.setTextSize(2);
display.setCursor(10, 20);
display.println(msg);
display.display();
Serial.println(msg);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  showMessage("READY");
}

void loop() {
  updateTouch();
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    Serial.print(" ");
    if (touch_values[i] > THRESHOLD) {
      if (i == 0) showMessage("Q0");
      if (i == 1) showMessage("Q1");
      if (i == 2) showMessage("Q2");
      if (i == 3) showMessage("Q3");
      if (i == 4) showMessage("Q4");
      if (i == 5) showMessage("Q5");
      delay(200);
    }
  }
  Serial.println();
}

```

Now touching each pad should display the matching Q0–Q5 label.

User

Yes, everything is displayed correctly (i.e. the text matches the button pressed)

Assistant

Great — that means the core system is now working end-to-end: capacitive touch input → RP2040 logic → OLED output, with USB Serial available as the wired communication channel.

A useful cleanup is to detect the moment a pad becomes touched rather than repeatedly triggering while it remains held. You can track the previous state:

```

bool wasTouched[N_TOUCH] = {
  false, false, false, false, false, false
};

```

Then use:

```

bool isTouched = touch_values[i] > THRESHOLD;
if (isTouched && !wasTouched[i]) {
  // trigger once when touch begins
}
wasTouched[i] = isTouched;

```

For your assignment, you already have enough to explain the system clearly: the RP2040 measures the charging time of each capacitive pad, compares that measurement against a threshold, identifies which input was touched, updates the OLED via I²C, and transmits data to the laptop over USB Serial.

User

okay, how can I pause this (i.e. disconnect from my laptop) and resume at a later time

Assistant

You can just unplug it.

The program you uploaded is stored in the XIAO RP2040's flash memory, so disconnecting power does not erase it. When you reconnect the QPAD later, the microcontroller will power up and automatically run the same sketch again from the beginning.

Save your Arduino sketch, unplug the USB cable, and later plug it back in. The QPAD does not need Arduino IDE open to run; once programmed, it operates independently as long as it has power. USB is currently doing two jobs: supplying power and providing Serial communication to your laptop.

User

explain this code to me

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Q0, Q1, Q2, Q3, Q4, Q5
int touch_pins[N_TOUCH] = {3, 4, 2, 27, 1, 26};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};

void updateTouch() {
  int t;
  int t_max = 200;
  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];
    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);
    t = 0;
    while (!digitalReadFast(p) && t < t_max) { t++; }
    touch_values[i] = t;
  }
}

void showMessage(const char* msg) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(10, 20);
  display.println(msg);
  display.display();
  Serial.println(msg);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
```

```

    showMessage("READY");
}

void loop() {
    updateTouch();
    for (int i = 0; i < N_TOUCH; i++) {
        Serial.print(touch_values[i]);
        Serial.print(" ");
        if (touch_values[i] > THRESHOLD) {
            if (i == 0) showMessage("Q0");
            if (i == 1) showMessage("Q1");
            if (i == 2) showMessage("Q2");
            if (i == 3) showMessage("Q3");
            if (i == 4) showMessage("Q4");
            if (i == 5) showMessage("Q5");
            delay(200);
        }
    }
    Serial.println();
}

```

Assistant

Think of this program as having five jobs: load libraries, describe the hardware, measure the touch pads, display messages, and repeat forever.

`#include` imports the I²C and OLED libraries. `#define` creates constants such as the screen size, touch-pad count, and threshold.

`int touch_pins[N_TOUCH] = {3, 4, 2, 27, 1, 26};` is an array containing the six GPIO numbers. `int touch_values[N_TOUCH] = {0,0,0,0,0,0};` is a parallel array holding the most recent measurement for each pad.

`updateTouch()` loops through each pad. It discharges the pad by making the pin an output and driving it LOW, waits 25 microseconds, switches the pin to `INPUT_PULLUP`, then counts how many loop iterations elapse before the pin reads HIGH. The count is stored in `touch_values[i]`.

`showMessage(const char* msg)` is a helper function that displays a passed-in string on the OLED and also prints it to Serial.

`setup()` starts Serial, starts I²C, initializes the display, and shows READY once.

`loop()` repeatedly calls `updateTouch()`, prints all six measurements, and checks each value against the threshold. If the value is above 20, it displays the corresponding Q label.

User

/ Q0, Q1, Q2, Q3, Q4, Q5

```
int touch_pins[N_TOUCH] = {3, 4, 2, 27, 1, 26};
```

```
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};
```

walk me thru this part of the code. how does each thing assign to a button on the qpad? also, why did you pick those numbers for the initial touch_pins?

Assistant

These two arrays do different jobs. `touch_pins` says which RP2040 GPIO belongs to each physical touch pad. `touch_values` stores the most recent measurement for each pad.

Because arrays are zero-indexed, the six positions are 0 through 5. With:

```
int touch_pins[6] = {3, 4, 2, 27, 1, 26};
```

the program interprets:

```
touch_pins[0] = 3
touch_pins[1] = 4
touch_pins[2] = 2
touch_pins[3] = 27
touch_pins[4] = 1
touch_pins[5] = 26
```

The second array uses the same indexes. If Q2 is being touched, for example, `touch\_values[2]` might become 103 while the others remain near zero.

The line `touch\_values[i] = t;` stores each measurement in the matching position. The mapping itself can be derived from the QPAD PCB traces/schematic plus the XIAO RP2040 pin mapping.

User

what published code are you talking about? even then, could this stuff be found in a datasheet instead?

Assistant

By “published code,” I meant public QPAD-XIAO example code in Fab Academy documentation, not an official Arduino library example.

Yes, you can derive the mapping from hardware documentation instead. There are three layers:

```
Physical QPAD pad
    ↓
QPAD PCB trace / schematic
    ↓
XIAO pin label
    ↓
RP2040 GPIO number
```

The XIAO RP2040 pinout tells you how the board’s labeled pins map to RP2040 GPIOs. For example, D10 maps to GPIO3, D9 to GPIO4, D8 to GPIO2, D1 to GPIO27, D7 to GPIO1, and D0 to GPIO26.

The QPAD PCB tells you which labeled XIAO pin each touch pad is physically connected to. Combining those two pieces lets you write the mapping yourself.

For readability, you can write:

```
int touch_pins[N_TOUCH] = {
    D10,
    D9,
    D8,
    D1,
    D7,
    D0
};
```

instead of raw GPIO numbers.

User

I'm very confused. Each button maps to a pin on the XIAO, and I can see what pin is connected to what button. From that, could I not find this info on a datasheet and write it into my code without relying on peaking at someone else's?

Assistant

Yes — exactly. You do not need someone else’s example code if you can trace each QPAD touch pad to a labeled XIAO pin.

The clean way to think about it is:

QPAD touch pad → XIAO pin label → RP2040 GPIO number

If you can physically see that Q0 is connected to D10, then you can use the XIAO pinout to find that D10 corresponds to the appropriate RP2040 GPIO. In Arduino code you can simply write `D10`, because the board definition already knows that mapping.

So if the pads connect as:

```
Q0 → D10
Q1 → D9
Q2 → D8
Q3 → D1
Q4 → D7
Q5 → D0
```

then this is a clear, defensible array:

```
int touch_pins[N_TOUCH] = {
  D10,
  D9,
  D8,
  D1,
  D7,
  D0
};
```

That is actually better for documentation because the pin names match what you physically see on the board.

User

do they have to be in this order?

```
int touch_pins[N_TOUCH] = {
  D10,
  D9,
  D8,
  D1,
  D7,
  D0
};
```

Assistant

Not inherently. The order only matters because your code uses the array index to mean a specific QPAD button.

If index 0 means Q0, index 1 means Q1, and so on, then the array order should match the actual wiring for Q0 through Q5.

You can choose a different order, but then the rest of the program must use that same ordering consistently. A more explicit alternative while learning is to use named variables such as `const int Q0\_PIN = D10;`, `const int Q1\_PIN = D9;`, etc.

User

okay, using this touch_pins, explain how touch_values works. be concise

Assistant

`touch\_values` is a parallel array: each position stores the latest measured touch value for the pin in the same position of `touch\_pins`.

```
int touch_pins[N_TOUCH] = {D10, D9, D8, D1, D7, D0};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};
```

So `touch\_pins[0] = D10` pairs with `touch\_values[0]`, `touch\_pins[1] = D9` pairs with `touch\_values[1]`, etc.

If Q0 is touched, the code might store `touch_values[0] = 105`. `touch_pins[i]` tells the program which pin to measure, and `touch_values[i]` stores what that measurement was.

User

```
for (int i = 0; i < N_TOUCH; i++) {
```

here, why is it `< N_TOUCH`, what is the size of that array according to arduino? how does the indexing work

Assistant

Because `N_TOUCH` is defined as 6, Arduino sees:

```
for (int i = 0; i < 6; i++)
```

The array has six elements, but C++ arrays are indexed starting at 0. Therefore the valid indexes are 0, 1, 2, 3, 4, 5.

There is no valid element 6. That is why the condition is `i < N_TOUCH` rather than `i <= N_TOUCH`. When `i` becomes 6, `6 < 6` is false and the loop stops.

User

does changing `touch_pins` to the array with the D-number affect the rest of the code? does calling `p = touch_pins[i]` remain unaffected?

Assistant

Yes. `p = touch_pins[i]` still works exactly the same.

If the array uses `D10`, `D9`, etc., those are still integer pin constants defined by the XIAO board package. So `int p = touch_pins[i];` still gives `p` the correct pin identifier, and `pinMode(p, ...)`, `digitalWriteFast(p, ...)`, and `digitalReadFast(p)` continue to work normally.

User

what are these functions?

```
pinMode(p, OUTPUT);  
digitalWriteFast(p, LOW);  
pinMode(p, INPUT_PULLUP);  
digitalReadFast(p);
```

Assistant

They're Arduino GPIO functions operating on the pin stored in `p`.

`pinMode(p, OUTPUT)` makes the pin an output. `digitalWriteFast(p, LOW)` drives that output to approximately 0 V, which discharges the touch pad. `pinMode(p, INPUT_PULLUP)` changes the pin into an input with an internal weak pull-up resistor. `digitalReadFast(p)` reads whether the pin is currently HIGH or LOW.

The sequence is: make the pin an output → drive it LOW → switch it to input with pull-up → watch how long it takes to become HIGH. The Fast versions reduce software overhead, which helps when measuring a short timing difference.

User

explain to me how reading the time acts as a button here

Assistant

The touch pad is not a mechanical switch. Your finger changes the pad's capacitance, and the code detects that change by measuring how long the pad takes to charge.

The code first forces the pad to 0 V, then switches it to an input with a weak pull-up. Without a finger, the capacitance is small and the voltage rises to HIGH quickly. With a finger touching the pad, your body adds capacitance, so the voltage rises more slowly.

The program counts how long the pin remains LOW:

```
t = 0;
while (!digitalReadFast(p) && t < t_max) {
  t++;
}
```

Then it converts that measurement to a yes/no decision:

```
if (touch_values[i] > THRESHOLD) {
  // treat as pressed
}
```

So the chain is: finger touches copper → capacitance increases → pad charges more slowly → measured count rises → count exceeds threshold → software treats it as a button press.

User

how does it measure time?

Assistant

In your current code, it does not measure time in microseconds directly. It measures time by counting how many times a very fast loop executes before the pin becomes HIGH.

```
t = 0;
while (!digitalReadFast(p) && t < t_max) {
  t++;
}
```

So `t` is a loop count rather than a physical unit of time. Each iteration takes a tiny amount of processor time, so a larger `t` means a longer charge time.

If you wanted a true elapsed-time measurement in microseconds, you could use Arduino's `micros()` function and subtract a start timestamp from an end timestamp.

User

okay, now explain the physics again. describe the capacitor, and give a broad overview of capacitors, and what they accomplish in circuits. be concise.

Assistant

A capacitor is a component or structure that stores electric charge temporarily. In its simplest form it consists of two conductive regions separated by an insulator (dielectric). Applying a voltage causes opposite charges to accumulate, storing energy in an electric field.

In circuits, capacitors are commonly used to store small amounts of energy, smooth noisy power supplies, filter signals, create timing effects, block DC while passing changing signals, and sense changes in capacitance.

For your QPAD, the copper touch pad behaves like part of a small capacitor. Your body and the surrounding environment provide the other effective conductor. Touching the pad increases the capacitance.

When a capacitor charges through a resistance, the characteristic time is proportional to $R \times C$. The pull-up resistance stays roughly fixed, so increasing C with your finger makes the charging slower. Your code detects that longer charge time.

User

```
void showMessage(const char* msg) {  
    display.clearDisplay();  
  
    display.setTextColor(SSD1306_WHITE);  
    display.setTextSize(2);  
    display.setCursor(10, 20);  
  
    display.println(msg);  
    display.display();  
    Serial.println(msg);  
}
```

explain this part of the code to me. what is `char*` ?

Assistant

This defines a function named `showMessage` that takes some text, displays it on the OLED, and also prints it to Serial.

`void` means the function returns no value. `showMessage` is the function name. `const char* msg` is the input parameter.

A `char` is one character such as `'Q'`. A C-style string is a sequence of `char` values ending with a special null character `\0`. `char*` is a pointer to a `char`; in the context of a string it points to the first character of that character sequence. `const` means the function promises not to modify the characters.

So `showMessage("Q3")` makes `msg` point to the characters in the string `"Q3"`.

`display.clearDisplay()` clears the drawing buffer. The text settings configure size, color, and position.

`display.println(msg)` draws the passed text into the buffer. `display.display()` sends the buffer to the physical OLED. `Serial.println(msg)` sends the same text to the computer.

User

I don't understand the pointer thing. elaborate on that please

Assistant

A pointer is a variable that stores a memory address instead of storing the actual value itself.

If you write:

```
char letter = 'Q';
```

then `letter` stores the character `Q` somewhere in memory. If you write:

```
char* ptr = &letter;
```

then `ptr` stores the address of `letter`. The `&` operator means "address of." The expression `*ptr` means "go to the address stored in `ptr` and read the value there."

For a string like `""READY"`, the characters are stored next to each other in memory as ``R E A D Y \0``. A ``const char* msg`` stores the address of the first character. A function such as ``println(msg)`` starts at that address and keeps reading consecutive characters until it reaches ``\0``.

So ``msg`` does not contain the whole string itself; it remembers where the string starts.

User

so "char" is string?

Assistant

Not quite. ``char`` means one character:

```
char letter = 'Q';
```

A string is a sequence of chars. A traditional C/C++ string is commonly represented by a pointer to the first character:

```
const char* msg = "Hello";
```

So ``char`` is one character, while ``const char*`` is commonly used to represent a string of characters.

User

okay, i still don't understand how the input of `showMessage` is stored. I understand it stores a pointer, and that passing that pointer to `println` shows the sequence of characters (or the string). what confuses me is: if you are defining it as `char` (which is a single character), how can it "remember" a string?

Assistant

Because ``const char* msg`` is not defining ``msg`` as a `char`. It is defining ``msg`` as a pointer to a `char`.

``char x;`` means `x` stores one character. ``char* x;`` means `x` stores an address, and at that address there is a `char`.

For `""HELLO""`, the characters are stored consecutively in memory: H, E, L, L, O, then ``\0``. The pointer stores the address of H. ``println()`` starts at that address and keeps reading one `char` after another until it reaches ``\0``.

So the type says "pointer to the first character of a sequence," not "one character only."

User

what if two strings start with the same letter?

Assistant

That is fine, because the pointer identifies a string by its memory address, not by its first letter.

Two strings can both start with H but be stored at different addresses. One pointer can point to the H in "HELLO" and another pointer can point to the H in "HI." ``println()`` follows the address it was given and reads until that particular string's null terminator.

User

what does `display.display()` do?

Assistant

``display.display()`` sends the image you built in the `display` buffer to the actual OLED.

Commands such as ``clearDisplay()``, ``setCursor()``, and ``println()`` modify a framebuffer in the RP2040's RAM. The physical screen is updated when you call ``display.display()``.

Typical sequence:

```
display.clearDisplay();
display.println("Hello");
display.display();
```

Without the final call, the new drawing may exist in memory but not appear on the OLED.

User

okay, I changed the code a bit and have the following:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
int touch_pins[N_TOUCH] = {D10, D9, D8, D1, D7, D0};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};

void updateTouch() {
  int t;
  int t_max = 200;
  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];
    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);
    t = 0;
    while (!digitalReadFast(p) && t < t_max) {
      t++;
    }
    touch_values[i] = t;
  }
}

void showMessage(const char* msg) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(10, 20);
  display.println(msg);
  display.display();
  Serial.println(msg);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  showMessage("READY");
}

void loop() {
  updateTouch();
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    Serial.print(" ");
    if (touch_values[i] > THRESHOLD) {
```

```

    if (i == 0) showMessage("YOU ARE TOUCHING Q0");
    if (i == 1) showMessage("YOU ARE TOUCHING Q1");
    if (i == 2) showMessage("YOU ARE TOUCHING Q2");
    if (i == 3) showMessage("YOU ARE TOUCHING Q3");
    if (i == 4) showMessage("YOU ARE TOUCHING Q4");
    if (i == 5) showMessage("YOU ARE TOUCHING Q5");
    delay(200);
  }
}
Serial.println();
}

```

nothing is coming up on my computer except the live readings of the touch values.

Assistant

Your logic is basically fine. The Serial Monitor is being flooded by the continuously printed touch values, while `showMessage()` only prints a message when a touch is detected. That message can get buried among the readings. Also, `showMessage("READY")` runs during setup, often before the Serial Monitor is opened.

For a clearer test, temporarily remove the continuous `Serial.print(touch\_values[i])` lines and leave only the messages. The raw-value prints were debugging code; they are not required for the touch sensing itself.

Also note that "YOU ARE TOUCHING Q0" is long for a 128×64 OLED at text size 2, so you may want a smaller text size or shorter message.

User

okay, now I want to add another part to the code which:

1) stops the message from permanently reading "Q0" or "Q3" etc in the display, but that it displays it and goes back.

Assistant

The simplest version is to show the message briefly and then restore READY:

```

if (i == 0) {
  showMessage("YOU ARE TOUCHING Q0");
  delay(1000);
  showMessage("READY");
}

```

A cleaner approach is to create:

```

void showTemporaryMessage(const char* msg) {
  showMessage(msg);
  delay(1000);
  showMessage("READY");
}

```

Then call that helper for each pad. The tradeoff is that `delay(1000)` pauses all other processing for one second.

User

that did not work

```

if (i == 0) {
  showMessage("YOU ARE TOUCHING Q0");
  delay(1000);
  showMessage("READY");
}

```

this did not work, it is still frozen

Assistant

The reason is that after the 1-second delay, the loop starts again and the pad may still be detected as touched. The program immediately shows Q0 again, so READY may only appear for a tiny fraction of a second.

What you want is to trigger only when the pad changes from not touched to touched. Add:

```
bool wasTouched[N_TOUCH] = {  
  false, false, false, false, false, false  
};
```

Then use:

```
bool isTouched = touch_values[i] > THRESHOLD;  
  
if (isTouched && !wasTouched[i]) {  
  // show the matching message  
  delay(1000);  
  showMessage("READY");  
}  
  
wasTouched[i] = isTouched;
```

Now holding your finger down does not retrigger continuously. Releasing and touching again creates a new event.

User

when did i define wastouched?

Assistant

You hadn't defined it yet. You need to add it near your other arrays:

```
bool wasTouched[N_TOUCH] = {  
  false, false, false, false, false, false  
};
```

`bool` is a Boolean type, so each element is either true or false. It remembers whether each pad was being touched during the previous loop.

User

what if now I want to have it read the capacitance time back to my computer during the 1 ms delay?

Assistant

While Arduino is inside `delay(1000)`, it does no useful sensing work. Also, `delay(1000)` is 1000 ms = 1 second, not 1 ms.

To keep measuring during that second, replace the delay with a timed loop using `millis()`:

```
unsigned long startTime = millis();  
  
while (millis() - startTime < 1000) {  
  updateTouch();  
  
  for (int j = 0; j < N_TOUCH; j++) {  
    Serial.print(touch_values[j]);  
    Serial.print(" ");  
  }  
  
  Serial.println();  
}
```

This keeps the processor active: it measures all six pads and sends the values repeatedly until one second has elapsed.

User

No, I want it to give me all six columns. how do i incorporate it into my current code

Assistant

Wrap the “print all six columns” logic into its own helper function:

```
void printTouchValues() {
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    Serial.print(" ");
  }
  Serial.println();
}
```

Then create a helper that shows the touch message while continuing to measure and print for one second:

```
void showTouchMessage(const char* msg) {
  showMessage(msg);

  unsigned long startTime = millis();

  while (millis() - startTime < 1000) {
    updateTouch();
    printTouchValues();
  }

  showMessage("READY");
}
```

Then each pad case can simply call `showTouchMessage(...)`.

User

can you fix this

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
int touch_pins[N_TOUCH] = {D10, D9, D8, D1, D7, D0};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};
bool wasTouched[N_TOUCH] = {false, false, false, false, false, false};

void updateTouch() {
  int t;
  int t_max = 200;
  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];
    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);
    t = 0;
    while (!digitalReadFast(p) && t < t_max) { t++; }
    touch_values[i] = t;
  }
}
```

```

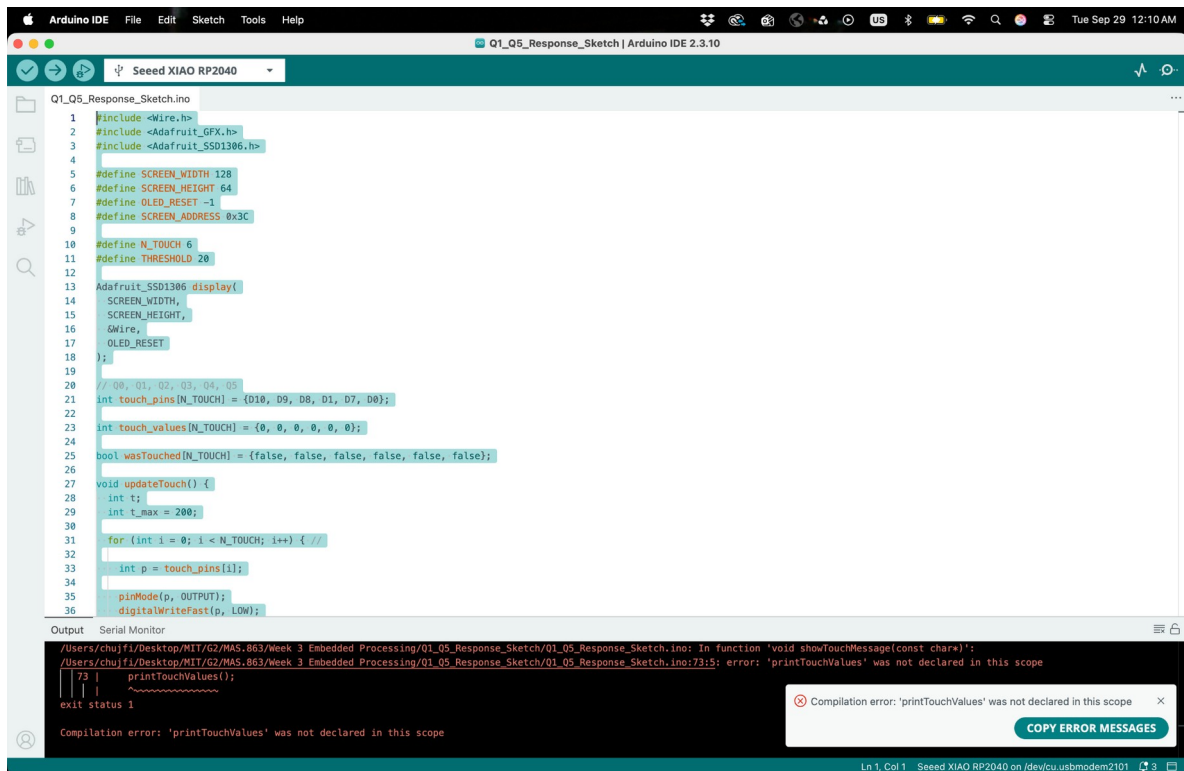
void showMessage(const char* msg) {
    display.clearDisplay();
    display.setTextColor(SSD1306_WHITE);
    display.setTextSize(1);
    display.setCursor(10, 20);
    display.println(msg);
    display.display();
    Serial.println(msg);
}

void showTouchMessage(const char* msg) {
    showMessage(msg);
    unsigned long startTime = millis();
    while (millis()-startTime < 1000) {
        updateTouch();
        printTouchValues();
    }
    showMessage("READY");
}

void setup() {
    Serial.begin(115200);
    Wire.begin();
    display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
    showMessage("READY");
}

void loop() {
    updateTouch();
    for (int i = 0; i < N_TOUCH; i++) {
        bool isTouched = touch_values[i] > THRESHOLD;
        if (isTouched && !wasTouched[i]) {
            if (i == 0) showTouchMessage("YOU ARE TOUCHING Q0");
            if (i == 1) showTouchMessage("YOU ARE TOUCHING Q1");
            if (i == 2) showTouchMessage("YOU ARE TOUCHING Q2");
            if (i == 3) showTouchMessage("YOU ARE TOUCHING Q3");
            if (i == 4) showTouchMessage("YOU ARE TOUCHING Q4");
            if (i == 5) showTouchMessage("YOU ARE TOUCHING Q5");
            delay(1000);
            showMessage("READY");
        }
        wasTouched[i] = isTouched;
    }
}

```



Attached image from the conversation

Assistant

The compile error is because you call `printTouchValues()` but never defined it. You also have the one-second wait/READY logic twice: once in `showTouchMessage()` and again in `loop()`.

Add:

```
void printTouchValues() {
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    Serial.print(" ");
  }
  Serial.println();
}
```

Keep `showTouchMessage()` as:

```
void showTouchMessage(const char* msg) {
  showMessage(msg);
  unsigned long startTime = millis();

  while (millis() - startTime < 1000) {
    updateTouch();
    printTouchValues();
  }

  showMessage("READY");
}
```

Then remove the extra `delay(1000); showMessage("READY");` from `loop()`. The resulting behavior is: touch a pad → show its message → for one second continuously measure and print all six values → return to READY.

User

Okay, the assignment was to write and test a program for an embedded system using a microcontroller to interact (with input and/or output devices) and communicate (via wired or wireless connection).

I believe my QPAD meets this, as it flashes different things based on what button I hit and communicates the live t readings to my computer.

I want to try something a bit more creative or cooler. Give me some ideas

Assistant

Yes, your current project already satisfies the assignment: capacitive touch is the input, the RP2040 is the processing element, the OLED is the output, and USB Serial is the wired communication channel.

Possible extensions include a reaction-time game, Simon Says memory game, mini menu/controller, capacitance visualizer, secret-code lock, musical controller, or two-way computer interaction.

For where you are now, a capacitance visualizer is especially good because it builds directly on `touch\_values[]` and makes the underlying sensor physics visible.

User

I like the capacitance visualizer

Assistant

The idea is to convert each `touch\_values[i]` reading into a vertical bar height and draw six live bars on the OLED.

A first version can use `display.fillRect()`:

```
void drawVisualizer() {
  display.clearDisplay();

  for (int i = 0; i < N_TOUCH; i++) {
    int barHeight = map(
      touch_values[i],
      0,
      120,
      0,
      50
    );

    barHeight = constrain(barHeight, 0, 50);
    int x = i * 20;

    display.fillRect(
      x,
      63 - barHeight,
      15,
      barHeight,
      SSD1306_WHITE
    );
  }

  display.display();
}
```

Then your loop becomes:

```
void loop() {
  updateTouch();
  drawVisualizer();
  printTouchValues();
}
```

`map()` converts the sensor's numeric range to a pixel-height range. `constrain()` prevents out-of-range values from drawing outside the intended area.

User

why did you choose 120? the readings usually go up to 200 when fully pressed

Assistant

Then 120 is too low for your actual sensor range. Use 200 as the input maximum:

```
int barHeight = map(
  touch_values[i],
  0,
  200,
  0,
  50
);
```

That gives roughly: $0 \rightarrow 0$ pixels, $100 \rightarrow 25$ pixels, $200 \rightarrow 50$ pixels. Keep `constrain()` as protection against unusual readings above 200.

User

where do I put this in the code? I still see my screen flash READY

Assistant

You're still seeing READY because your current program is still calling `showMessage("READY")` somewhere. For the visualizer, make the screen dedicated to the bars.

Add `drawVisualizer()` above `setup()`, remove `showMessage("READY")` from `setup()`, and replace your old `loop()` with:

```
void loop() {
  updateTouch();
  drawVisualizer();
  printTouchValues();
}
```

Now the program repeatedly measures the pads, draws six bars, prints six values to Serial, and repeats.

User

okay, what if I want there to be text at the bottom of each rectangle indicating which rectangle corresponds to what button

Assistant

Reserve a little space at the bottom for labels and make the bars slightly shorter:

```
void drawVisualizer() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  for (int i = 0; i < N_TOUCH; i++) {
    int barHeight = map(touch_values[i], 0, 200, 0, 42);
    barHeight = constrain(barHeight, 0, 42);

    int x = i * 20;

    display.fillRect(
      x,
      50 - barHeight,
      15,
      barHeight,
      SSD1306_WHITE
    );
  }
}
```

```

    display.setCursor(x, 55);
    display.print("Q");
    display.print(i);
}

display.display();
}

```

This produces six live bars with Q0 through Q5 labels at the bottom.

User

okay, now what if I want to make it have a starting screen? say it opens to a logo of MIT for a second and then it goes to the rectangles. is that possible?

Assistant

Yes. `setup()` runs once when the QPAD powers on, so it is the natural place for a splash screen.

A simple text splash would be:

```

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);

  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(45, 20);
  display.println("MIT");
  display.display();

  delay(1000);
}

```

Then `loop()` takes over and draws the capacitance bars. You can also use an actual 1-bit bitmap graphic with `display.drawBitmap()`.

User

yes, how do I do the graphic thing?

Assistant

The SSD1306 cannot display a PNG/JPEG directly. Convert the image into a 1-bit bitmap: each pixel is either off or on. The bitmap becomes a byte array stored in your Arduino program.

The process is: MIT logo image → resize to fit 128×64 → convert to black/white bitmap → store as byte array → draw with `display.drawBitmap()`.

For example, a generated array looks like:

```

const unsigned char mit_logo[] PROGMEM = {
  0x00, 0x3F, 0xC0, 0x00,
  // ...more bytes...
};

```

Then draw it with:

```

void showSplashScreen() {
  display.clearDisplay();
  display.drawBitmap(
    32,
    16,
    mit_logo,

```

```

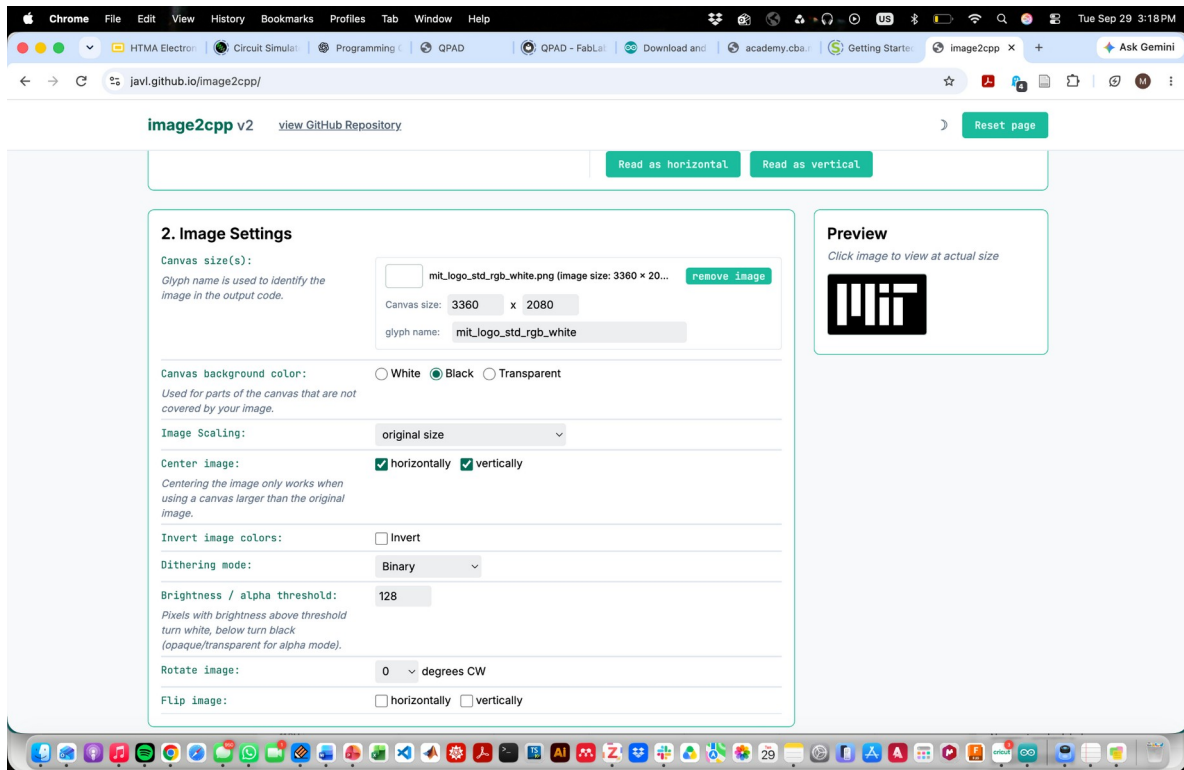
64,
32,
SSD1306_WHITE
);
display.display();
delay(1000);
}

```

If the bitmap is the full 128×64 screen, draw it at x=0, y=0 instead.

User

here what do I input for the canvas size



Attached image from the conversation

Assistant

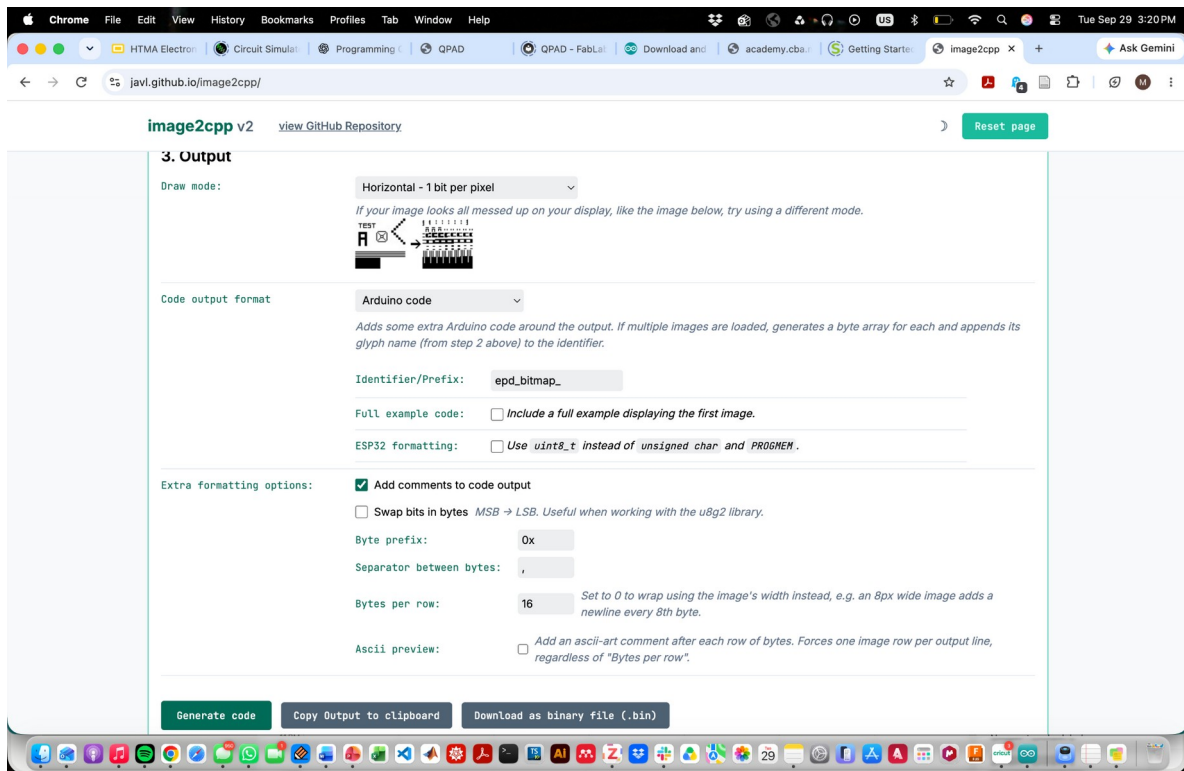
Set the canvas to the actual OLED size:

```
128 × 64
```

Your display is 128 pixels wide and 64 pixels tall. Use a black background and fit/contain the MIT logo within that canvas. You do not want to leave the canvas at the original multi-thousand-pixel PNG dimensions.

User

what do I do here?



Attached image from the conversation

Assistant

For your SSD1306 + Adafruit_GFX setup, use:

- Draw mode: Horizontal - 1 bit per pixel
- Code output format: Arduino code
- Identifier/Prefix: something like `mit\_logo\_`
- Full example code: unchecked
- ESP32 formatting: unchecked
- Swap bits in bytes: unchecked
- Byte prefix: `0x`
- Separator: ``
- Bytes per row: 16 is fine

Then click Generate code. You should get a `const unsigned char ... PROGMEM = { ... };` array. Copy that array into your Arduino sketch near the top.

If the bitmap is 128×64, draw it with:

```
display.drawBitmap(
  0,
  0,
  mit_logo,
  128,
  64,
  SSD1306_WHITE
);
```

User

This is what image2cpp returned:

```
// 'mit_logo_std_rgb_white', 128x64px
const unsigned char mit_logo_mit_logo_std_rgb_white [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, /* ... full generated bitmap bytes ... */
};

// Array of all bitmaps for convenience.
const int mit_logo_allArray_LEN = 1;
const unsigned char* mit_logo_allArray[1] = {
  mit_logo_mit_logo_std_rgb_white
};
```

Assistant

Perfect — that is exactly the kind of output you need.

You only really need the `const unsigned char mit\_logo\_mit\_logo\_std\_rgb\_white[] PROGMEM = { ... };` array. The extra allArray wrapper is optional if you have only one image.

Paste the bitmap array near the top of your sketch. Then add:

```
void showSplashScreen() {
  display.clearDisplay();

  display.drawBitmap(
    0,
    0,
    mit_logo_mit_logo_std_rgb_white,
    128,
    64,
    SSD1306_WHITE
  );

  display.display();
  delay(1000);
}
```

Call `showSplashScreen();` once from `setup()`. After the one-second splash, `loop()` will take over and run the visualizer.

User

yes please, do that

Assistant

Here is a clean combined sketch that shows the MIT logo for one second, reads all six capacitive pads, draws six live bars labeled Q0 through Q5, and prints all six readings to Serial.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int touch_pins[N_TOUCH] = {D10, D9, D8, D1, D7, D0};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};

const unsigned char mit_logo_mit_logo_std_rgb_white[] PROGMEM = {
  // paste full bitmap array here
```

```

};

void updateTouch() {
  int t;
  int t_max = 200;
  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];
    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);
    t = 0;
    while (!digitalReadFast(p) && t < t_max) { t++; }
    touch_values[i] = t;
  }
}

void printTouchValues() {
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    if (i < N_TOUCH - 1) Serial.print("  ");
  }
  Serial.println();
}

void showSplashScreen() {
  display.clearDisplay();
  display.drawBitmap(0, 0, mit_logo_mit_logo_std_rgb_white, 128, 64, SSD1306_WHITE);
  display.display();
  delay(1000);
}

void drawVisualizer() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  for (int i = 0; i < N_TOUCH; i++) {
    int barHeight = map(touch_values[i], 0, 200, 0, 42);
    barHeight = constrain(barHeight, 0, 42);
    int x = i * 20;
    display.fillRect(x, 50 - barHeight, 15, barHeight, SSD1306_WHITE);
    display.setCursor(x, 55);
    display.print("Q");
    display.print(i);
  }
  display.display();
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println("OLED failed to start");
    while (true) {}
  }
  showSplashScreen();
}

void loop() {
  updateTouch();
  drawVisualizer();
  printTouchValues();
}

```

The only remaining step is to paste the complete generated bitmap bytes into the `mit\_logo...[]` array.

User

actually, I just want you to add the start-up screen to my existing code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
#define N_TOUCH 6
#define THRESHOLD 20

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int touch_pins[N_TOUCH] = {D10, D9, D8, D1, D7, D0};
int touch_values[N_TOUCH] = {0, 0, 0, 0, 0, 0};
bool wasTouched[N_TOUCH] = {false, false, false, false, false, false};

void updateTouch() {
  int t;
  int t_max = 200;
  for (int i = 0; i < N_TOUCH; i++) {
    int p = touch_pins[i];
    pinMode(p, OUTPUT);
    digitalWriteFast(p, LOW);
    delayMicroseconds(25);
    pinMode(p, INPUT_PULLUP);
    t = 0;
    while (!digitalReadFast(p) && t < t_max) { t++; }
    touch_values[i] = t;
  }
}

void printTouchValues() {
  for (int i = 0; i < N_TOUCH; i++) {
    Serial.print(touch_values[i]);
    Serial.print(" ");
  }
  Serial.println();
}

void showMessage(const char* msg) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(10, 20);
  display.println(msg);
  display.display();
  Serial.println(msg);
}

void showTouchMessage(const char* msg) {
  showMessage(msg);
  unsigned long startTime = millis();
  while (millis() - startTime < 1000) {
    updateTouch();
    printTouchValues();
  }
  showMessage("READY");
}

void drawVisualizer() {
  display.clearDisplay();
  display.setTextSize(1);
```

```

display.setTextColor(SSD1306_WHITE);
for (int i = 0; i < N_TOUCH; i++) {
  int barHeight = map(touch_values[i], 0, 200, 0, 42);
  barHeight = constrain(barHeight, 0, 42);
  int x = i * 20;
  display.fillRect(x, 50 - barHeight, 15, barHeight, SSD1306_WHITE);
  display.setCursor(x, 55);
  display.print("Q");
  display.print(i);
}
display.display();
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
}

void loop() {
  updateTouch();
  drawVisualizer();
  printTouchValues();
}

```

Assistant

Here is your existing code with only the startup screen added. The changes are: paste the generated MIT bitmap array near the top, add a `showSplashScreen()` function, and call it once from `setup()`.

```

const unsigned char mit_logo_mit_logo_std_rgb_white[] PROGMEM = {
  // paste the full bitmap array here
};

void showSplashScreen() {
  display.clearDisplay();

  display.drawBitmap(
    0,
    0,
    mit_logo_mit_logo_std_rgb_white,
    128,
    64,
    SSD1306_WHITE
  );

  display.display();
  delay(1000);
}

```

Then in setup:

```

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  showSplashScreen();
}

```

The result is: power on → MIT logo for one second → live capacitance visualizer.

User

I want this whole chat in a pdf so that I can document my work