Documenting Senior Project:

The project is broken down into 4 major steps. First, being able to set up the microcontroller to take images, store them, and transfer them to the Color filter device. All while being able to reconstruct the image and display the results on a separate display. Second, the color filter device will be made to be able to filter out at least 1 color and would simultaneously influence how the data is communicated between the MCU and filter. Third, testing will be done to make sure everything functions properly, and prototyping will be made here as well. Finally, optimization will be made. Some improvements could be better filtering, image capturing, better components, and so on.

Date Document:

1/24/2025-2/14/2025:  
Project has been worked on since first week of Spring semester. This is the first logging of the work, so it can be used later on as reference. Successfully able to create a .bmp file using reference code and understanding how BMP formatting works.  
Websites used:

* 1) <https://engineering.purdue.edu/ece264/19sp/hw/HW11>
* 2) <https://www.synalysis.blog/blog/what-is-the-file-format-bmp>
* 3) <https://en.wikipedia.org/wiki/BMP_file_format>

All 3 used to understand BMP formatting.

* 4) <https://www.color-hex.com/color/da211a>

Used to understand what the hex representation means.

* <https://docs.sunfounder.com/projects/esp32-starter-kit/en/latest/arduino/basic_projects/ar_take_photo_sd.html>

This site is used as a reference code to understand how to setup the ESP32 with camera capabilities. This program only takes JPEG images. Sections of the code are used to begin but will be modified to increase efficiency and simplicity. A section dedicated to explaining the code and the way we come to a solution will be explained later on.



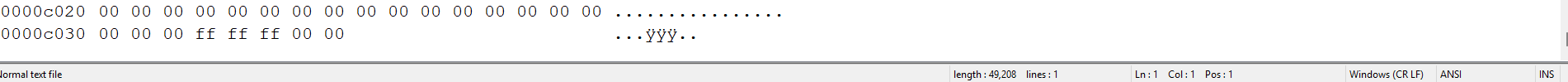
This is the original BMP image taken. To fully test the file formatting, website 1 was used. From there the basic header was able to be taken out and allow us to edit only the raw pixel information. To test how the file changes to modifications it was opened in Notepad++ with the HEX editor plugin. The image below shows the HEX code of the original image.

A screen shot of a computer

AI-generated content may be incorrect.

Afterwards, a copy of the BMP file was opened in Photoshop and changed to pure black. This results in the rest of the file containing 0 throughout and in the end.  
A screenshot of a computer

AI-generated content may be incorrect.



In Photoshop the top left and bottom right corner were changed to white to see whether the HEX file changes. After editing the file and opening it in Notepad++ we can see ffffff is in address 0x00000036 and is 6 HEX symbols large. This is significant because this indicates the color per pixel is 6 HEX symbols long as indicated in the aforementioned resources. A similar collection of HEX symbols can be found at the end of the file. However, there are 4 HEX symbols which are presumably used for padding. We still need to test how each pixel is separated and the best way to access them without losing the order. Real quickly we changed the final HEX section to #f000ff which is a pink color to see if the end of the file is truly there. Image without modifications except for white is seen below.A screenshot of a computer screen

AI-generated content may be incorrect.

After saving the changes, we see that the change does work and the pink is at the end. This indicated the file is saved sequentially. This goes against the information in website 1 stating the pixels are saved backwards. Regardless, this is looking promising. Image is below.

A black background with a white line

AI-generated content may be incorrect.

2/17/2025

While developing a method to send out the bit information, the best way would be to send out 6 bytes at a time. All the files taken are divisible by 6 bytes. This is important because the header is 54 bytes in length. 6 bytes is the same as 24 bits.

A screenshot of a computer

AI-generated content may be incorrect.

3/19/2025

Created Excel sheet to document parts and inventory.

Coding Documentation:

References:

* <https://docs.sunfounder.com/projects/esp32-starter-kit/en/latest/arduino/basic_projects/ar_take_photo_sd.html>

Part 1 of the project and my main role is to implement MCU and data communication. Logan’s is to develop and find the specifications of the Color filter components.

When begging to write the code for the MCU, the code provided by the website was used to understand the basic functionality of the MCU with the camera attachment. The code is below and highlighted in Yellow.

/\*\*\*\*\*\*\*\*\*

Rui Santos

Complete project details at https://RandomNerdTutorials.com/esp32-cam-take-photo-save-microsd-card

IMPORTANT!!!

- Select Board "AI Thinker ESP32-CAM"

- GPIO 0 must be connected to GND to upload a sketch

- After connecting GPIO 0 to GND, press the ESP32-CAM on-board RESET button to put your board in flashing mode

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

\*\*\*\*\*\*\*\*\*/

#include "esp\_camera.h"

#include "Arduino.h"

#include "FS.h" // SD Card ESP32

#include "SD\_MMC.h" // SD Card ESP32

#include "soc/soc.h" // Disable brownour problems

#include "soc/rtc\_cntl\_reg.h" // Disable brownour problems

#include "driver/rtc\_io.h"

#include <EEPROM.h> // read and write from flash memory

// define the number of bytes you want to access

#define EEPROM\_SIZE 1

// Pin definition for CAMERA\_MODEL\_AI\_THINKER

#define PWDN\_GPIO\_NUM 32

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 0

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 21

#define Y4\_GPIO\_NUM 19

#define Y3\_GPIO\_NUM 18

#define Y2\_GPIO\_NUM 5

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

int pictureNumber = 0;

void setup() {

WRITE\_PERI\_REG(RTC\_CNTL\_BROWN\_OUT\_REG, 0); //disable brownout detector

Serial.begin(115200);

//Serial.setDebugOutput(true);

//Serial.println();

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

config.pin\_d1 = Y3\_GPIO\_NUM;

config.pin\_d2 = Y4\_GPIO\_NUM;

config.pin\_d3 = Y5\_GPIO\_NUM;

config.pin\_d4 = Y6\_GPIO\_NUM;

config.pin\_d5 = Y7\_GPIO\_NUM;

config.pin\_d6 = Y8\_GPIO\_NUM;

config.pin\_d7 = Y9\_GPIO\_NUM;

config.pin\_xclk = XCLK\_GPIO\_NUM;

config.pin\_pclk = PCLK\_GPIO\_NUM;

config.pin\_vsync = VSYNC\_GPIO\_NUM;

config.pin\_href = HREF\_GPIO\_NUM;

config.pin\_sscb\_sda = SIOD\_GPIO\_NUM;

config.pin\_sscb\_scl = SIOC\_GPIO\_NUM;

config.pin\_pwdn = PWDN\_GPIO\_NUM;

config.pin\_reset = RESET\_GPIO\_NUM;

config.xclk\_freq\_hz = 20000000;

config.pixel\_format = PIXFORMAT\_JPEG;

if (psramFound()) {

config.frame\_size = FRAMESIZE\_UXGA; // FRAMESIZE\_ + QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA

config.jpeg\_quality = 10;

config.fb\_count = 2;

} else {

config.frame\_size = FRAMESIZE\_SVGA;

config.jpeg\_quality = 12;

config.fb\_count = 1;

}

// Init Camera

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK) {

Serial.printf("Camera init failed with error 0x%x", err);

return;

}

//Serial.println("Starting SD Card");

if (!SD\_MMC.begin()) {

Serial.println("SD Card Mount Failed");

return;

}

uint8\_t cardType = SD\_MMC.cardType();

if (cardType == CARD\_NONE) {

Serial.println("No SD Card attached");

return;

}

// initialize EEPROM with predefined size

EEPROM.begin(EEPROM\_SIZE);

pictureNumber = EEPROM.read(0) + 1;

for (int shoot = 0; shoot < 5; shoot++) {

camera\_fb\_t \*fb = NULL;

// Take Picture with Camera

fb = esp\_camera\_fb\_get();

if (!fb) {

Serial.println("Camera capture failed");

return;

}

// Path where new picture will be saved in SD Card

String path = "/picture" + String(pictureNumber) + ".jpg";

fs::FS &fs = SD\_MMC;

// Serial.printf("Picture file name: %s\n", path.c\_str());

File file = fs.open(path.c\_str(), FILE\_WRITE);

if (!file) {

Serial.println("Failed to open file in writing mode");

} else {

file.write(fb->buf, fb->len); // Write image data to file

if (shoot == 4) {

Serial.printf("Saved file to path: %s\n", path.c\_str());

}else{

Serial.printf("Shooting... \n");

}

EEPROM.write(0, pictureNumber); // Update the picture number in EEPROM

EEPROM.commit();

}

file.close(); // Close the file

esp\_camera\_fb\_return(fb); // Return the frame buffer back to the camera driver

delay(200); // Short delay between shots

}

// Turns off the ESP32-CAM white on-board LED (flash) connected to GPIO 4

pinMode(4, OUTPUT);

digitalWrite(4, LOW);

rtc\_gpio\_hold\_en(GPIO\_NUM\_4);

// Put the ESP32-CAM to deep sleep

delay(2000);

Serial.println("Going to sleep now");

delay(2000);

esp\_deep\_sleep\_start();

Serial.println("This will never be printed");

}

void loop() {

// Empty loop as we are putting the ESP32-CAM to deep sleep

}

After uploading the code, we were able to take screenshots using the instructions on the website. The image taken was saved as a “.jpg” format and saved in the root of the microSD card connected to the extension board. Every time an image is taken, the counter also increases. The image is roughly the same size every time and the resolution is 1600x1200. Below is an example image.A screenshot of a computer

AI-generated content may be incorrect.

A white tile floor with a black object on it

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

After testing, the code seems to function correctly. However, for our application we need a Bitmap image type (BMP). We need this file since we want to directly manipulate the binary information in the Color filter device. Originally, I attempted to simply change the variables such as the pixel format but the version of our ESP32 does not allow direct conversion to RGB888, RAW, and other useful formats. Only JPEG is functional. To see how a BMP file would look, we took an image and converted it to BMP using an online JPEG to BMP converter (<https://convertio.co/jpg-bmp/>).

A black and white image with numbers

AI-generated content may be incorrect.



After using an online converter, the resulting file is much larger than the original one. This is possibly because we are storing the raw unaltered JPEG data.

3/24/2025

The new Microcontroller arrived.  
  
sd\_read\_write.cpp  
Saves the same information similar to the original  
    // Save BMP file

    String path = "/picture" + String(pictureNumber) + ".bmp";

    fs::FS &fs = SD\_MMC;

    File file = fs.open(path.c\_str(), FILE\_WRITE);

    if (!file) {

      Serial.println("Failed to open file in writing mode");

    } else {

      file.write(bmp\_buf, bmp\_len);  // Write BMP data

      if (shoot == 4) {

        Serial.printf("Saved file to path: %s\n", path.c\_str());

      } else {

        Serial.printf("Shooting... \n");

      }

      EEPROM.write(0, pictureNumber);

      EEPROM.commit();

    }

    file.close();

In the new file structure, saved in the sd\_read\_write.h and sd\_read\_write.cpp

3/29/2025

Attempting to display image on LCD and create data transmission. To approach this, I decided to create two arrays. One to store the header information and another to store the pixel information To make sure the files are as expected we look at the file sizes.

A screen shot of a computer

AI-generated content may be incorrect.  
Each file is 49,206. The expected file size is 16,384 which is found from the resolution 128x128. With some more research I found that this is multiplied by the bit depth (24 in our case) and then divided by 8 (8 bits to 1 byte). This gave us the size of 49,152. This is still off, but once we add the size of the header of 54 bytes, we get matching size of 49,206. With this, the temporary arrays can be created. This is useful because the resolution can be found by taking the square of the Pixel amount and manipulated to find other values.

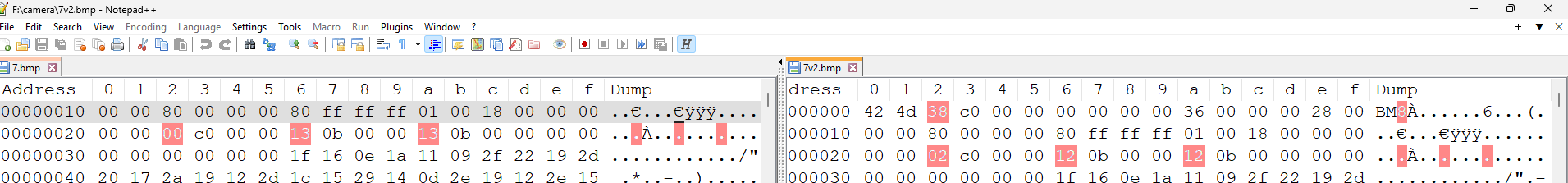
A screenshot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

From here, we can see the array stores the image properly. I have also observed that the image is reconstructed if opened in photoshop and saved. At the end a padding of 00 00 appears. However if the image is directly opened and edited there appears to be no changes.  
  
Here is an example file “7.bmp”. This is the section of the header.  
A screenshot of a computer

AI-generated content may be incorrect.  
Here is the same image, but with the end section edited to ee77ee. When opened in photoshop the change is visible.

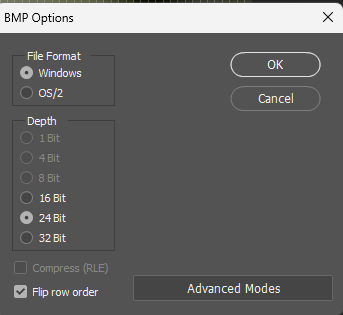
A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a computer

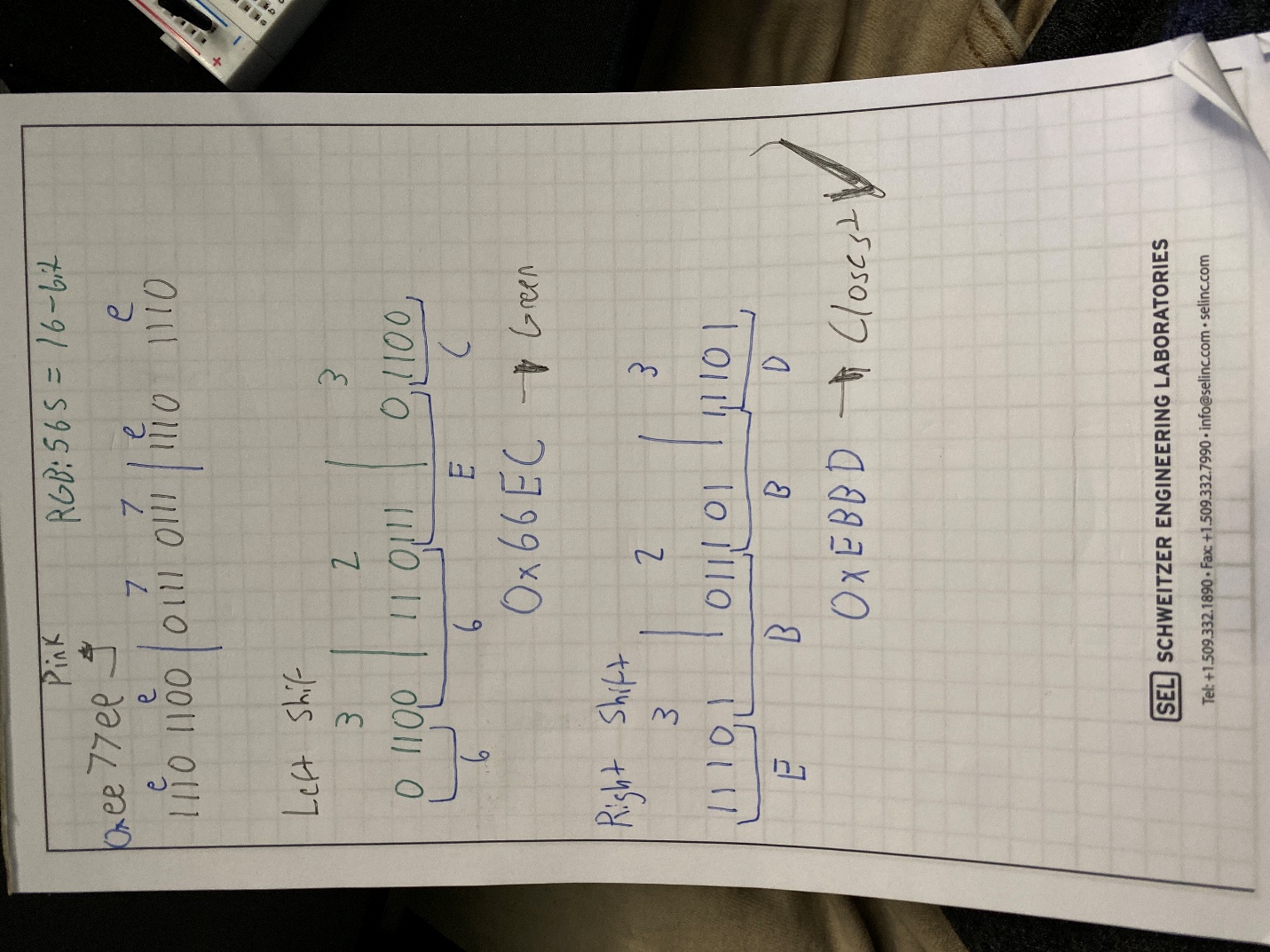
AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

However as soon as we save the file structure changes. 7v2 is the one changed in Photoshop. As seen above, Photoshop adds some padding at the end making the file 2 bytes larger. Besides this, the header also changes. The changes are highlighted in pink. Moving forward, photoshop should not be used. The changes were possible done because of this hub seen below that pops up when Save as is selected.



In order to display the RGB colors correctly on the LCD they must be converted from 24 bit to 16 bit. An attempt to reach this is by shifting the bits.



4/5/2025

In an attempt to make sure the circuit is setup correctly, some test code is written to send out a test bit information. The code below is the code used on the MCU to be used as the camera.   
#define RED\_MASK        0xFF0000   // Red channel

#define GREEN\_MASK      0x00FF00   // Green channel

#define BLUE\_MASK       0x0000FF   // Blue channel

#define RED\_BLUE\_MASK   0xFFFF00   // Red and Blue channels (no Green)

#define RED\_GREEN\_MASK  0xFF00FF   // Red and Green channels (no Blue)

#define BLUE\_GREEN\_MASK 0x00FFFF   // Blue and Green channels (no Red)

#define ALL\_CHANNELS\_MASK 0xFFFFFF // All Red, Green, and Blue channels

#define BUTTON\_PIN  0

#define PIXEL

// Pins

const uint8\_t DATA = 21;

const uint8\_t MASK = 47;

const uint8\_t S0 = 48; // LSB (A)

const uint8\_t S1 = 45; // (B)

const uint8\_t S2 = 0;  // MSB (C)

const uint8\_t MONITOR = 35;

void setup() {

  Serial.begin(115200);

  Serial.setDebugOutput(false);

  Serial.println();

  pinMode(BUTTON\_PIN, INPUT\_PULLUP);

  // Set selector pins as output

  pinMode(DATA, OUTPUT);

  pinMode(S0, OUTPUT);

  pinMode(S1, OUTPUT);

  pinMode(S2, OUTPUT);

  pinMode(MASK, OUTPUT);

  pinMode(MONITOR, INPUT);

}

int Aa = 1;

int Bb = 1;

int Cc = 0;

void loop() {

  uint8\_t pixel = 0xFF; // 10100111

  uint8\_t colorMask = 0xFF;  // 11111111

   if(digitalRead(BUTTON\_PIN)==LOW){

    //Serial.printf("0x%0X\n", pixel);

    //Serial.printf("0x%0X\n", colorMask);

    digitalWrite(S0, Aa);

    digitalWrite(S1, Bb);

    digitalWrite(S2, Cc);

    Serial.printf("Selector: %d%d%d\n", Cc,Bb,Aa);

    digitalWrite(DATA, 1);

    digitalWrite(MASK, 1);

    Serial.printf("00: %d\n", digitalRead(MONITOR));

    // digitalWrite(DATA, 1);

    // digitalWrite(MASK, 0);

    // Serial.printf("10: %d\n", digitalRead(MONITOR));

    // digitalWrite(DATA, 1);

    // digitalWrite(MASK, 1);

    // Serial.printf("11: %d\n", digitalRead(MONITOR));

    //sendData(pixel, colorMask);  // Send first 8 bits to channel 0

    Serial.printf("Pixel Sent\n");

    //delay(2000);

  }

}

void sendData(uint8\_t Ddata, uint8\_t Mmask) {

  for (int i = 7; i >= 0; i--) {

    switch (i) {

        case 0:

          digitalWrite(S0, 0);

          digitalWrite(S1, 0);;

          digitalWrite(S2, 0);

          break;

        case 1:

          digitalWrite(S0, 1);

          digitalWrite(S1, 0);

          digitalWrite(S2, 0);

          break;

        case 2:

          digitalWrite(S0, 0);

          digitalWrite(S1, 1);

          digitalWrite(S2, 0);

          break;

        case 3:

          digitalWrite(S0, 1);

          digitalWrite(S1, 1);

          digitalWrite(S2, 0);

          break;

        case 4:

          digitalWrite(S0, 0);

          digitalWrite(S1, 0);

          digitalWrite(S2, 1);

          break;

        case 5:

          digitalWrite(S0, 1);

          digitalWrite(S1, 0);

          digitalWrite(S2, 1);

          break;

        case 6:

          digitalWrite(S0, 0);

          digitalWrite(S1, 1);

          digitalWrite(S2, 1);

          break;

        case 7:

          digitalWrite(S0, 1);

          digitalWrite(S1, 1);

          digitalWrite(S2, 1);

          break;

        default:

          Serial.println("Invalid case");

          break;

    }

    uint8\_t temp = (Ddata >> i) & 1;

    uint8\_t temp1 = (Mmask >> i) & 1;

    Serial.printf("Data: 0x%02X\n", temp);

    Serial.printf("Mask: 0x%02X\n", temp1);

    digitalWrite(DATA, temp);

    digitalWrite(MASK, temp1);

    Serial.printf("Monitor: %d\n", digitalRead(MONITOR));

    //uint16\_t  tempValue = digitalRead(MONITOR);

    //Serial.printf("Captured: 0x%02X\n", tempValue);

  }

}

On a secondary MCU, this code was created as a makeshift logic analyzer.  
const uint8\_t MONITOR0 = 23;

const uint8\_t MONITOR1 = 22;

const uint8\_t MONITOR2 = 21;

const uint8\_t MONITOR3 = 19;

const uint8\_t MONITOR4 = 18;

const uint8\_t MONITOR5 = 5;

const uint8\_t MONITOR6 = 4;

const uint8\_t MONITOR7 = 0;

const uint8\_t MONITOR8 = 2;

const uint8\_t MONITOR9 = 15;

void setup() {

  Serial.begin(115200);

  pinMode(MONITOR0, INPUT);

  pinMode(MONITOR1, INPUT);

  pinMode(MONITOR2, INPUT);

  pinMode(MONITOR3, INPUT);

  pinMode(MONITOR4, INPUT);

  pinMode(MONITOR5, INPUT);

  pinMode(MONITOR6, INPUT);

  pinMode(MONITOR7, INPUT);

  pinMode(MONITOR8, INPUT);

  pinMode(MONITOR9, INPUT);

}

void loop() {

  Serial.printf("A: %d\n", digitalRead(MONITOR0));

  //Serial.printf("MONITOR1: %d\n", digitalRead(MONITOR1));

  Serial.printf("B: %d\n", digitalRead(MONITOR2));

  Serial.printf("ANDA: %d\n", digitalRead(MONITOR3));

  Serial.printf("ANDB: %d\n", digitalRead(MONITOR4));

  Serial.printf("OUT: %d\n", digitalRead(MONITOR5));

  //Serial.printf("MONITOR6: %d\n", digitalRead(MONITOR6));

  //Serial.printf("MONITOR7: %d\n", digitalRead(MONITOR7));

  //Serial.printf("MONITOR8: %d\n", digitalRead(MONITOR8));

  //Serial.printf("MONITOR9: %d\n", digitalRead(MONITOR9));

  Serial.println();

  delay(2000);

}

A screenshot of a computer

AI-generated content may be incorrect.

When writing the code, the main code loops indefinitely as seen here when the button is not pressed. However, the logic seems to work well. As proven by the other MCU

A black background with white text

AI-generated content may be incorrect.

4/16/2025

void setup() {

  Serial.begin(115200);

  Serial.println(F("\nI2C PINS"));

  Serial.print(F("\tSDA = ")); Serial.println(SDA);

  Serial.print(F("\tSCL = ")); Serial.println(SCL);

}

void loop() {}

Useful code to find SDA/SCL for I2C