INDIANA STATE UNIVERSITY

BAILEY COLLEGE OF ENGINEERING & TECHNOLOGY

Semester Project #2: Gate Entry Security System

Lea Tice

Department of Electronic and Computer Engineering Technology

ECT 308: IoT Microcontrollers

Dr. Alister McLeod

May 05, 2025

Author Note

Correspondence concerning this article should be addressed to Lea Tice, Indiana State University, 200 N 7th St. Terre Haute, IN 47809, United States.
Email: lea.tice@indstate.edu

Table of Contents

Abstract

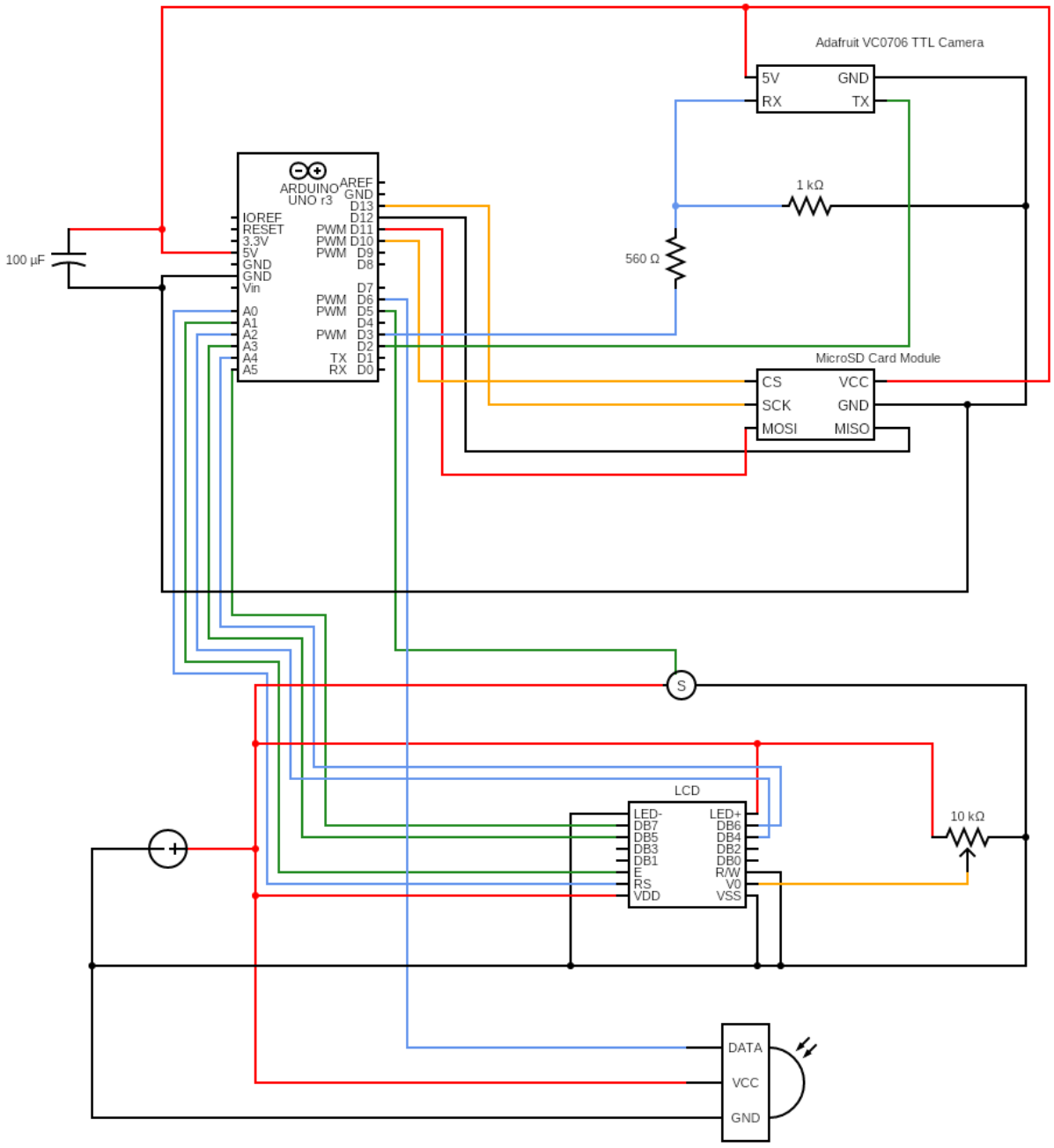
This project involved the development of a microcontroller-based gate entry security system using an Arduino platform. The goal was to detect and validate individuals requesting access using a combination of motion detection, image capture, and infrared (IR) remote ID verification. The system was designed to trigger a TTL serial camera to capture a photo when motion was detected, store the image to a microSD card, and prompt the user via an LCD screen to present an access code. An IR receiver was used to validate input from a remote control against authorized codes. Upon successful verification, a servo motor opened a simulated gate for a predefined duration. The LCD display updated at each critical stage, including motion detection, ID request, access granted or denied, and status reset. Although initial project instructions included web-based moderator verification and DC motor relay control via ThingSpeak, these were not implemented in the final version. The completed system demonstrates a functional prototype for low-cost, embedded access control using multiple hardware interfaces and real-time input processing.

Keywords: Arduino, security system, IR remote, microSD, TTL camera, servo motor, motion detection, LCD display

Gate Entry Security System

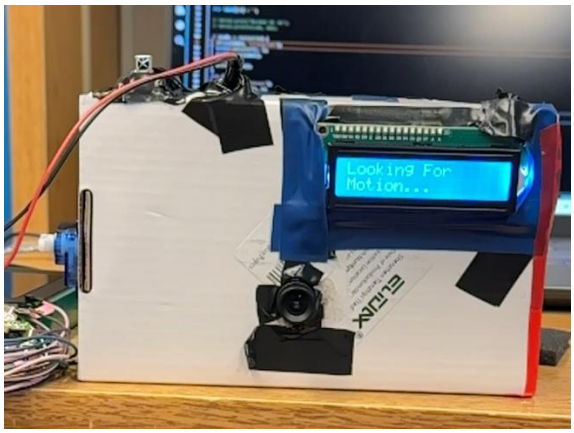
This project presents the development of a microcontroller-based gate entry security system designed to demonstrate practical applications of embedded systems in access control environments. The system is implemented using the Arduino platform and incorporates a range of components including a TTL serial camera, infrared remote input, servo motor, microSD card storage, and a character-based LCD interface. The purpose of the system is to autonomously monitor for motion at a controlled entry point, capture a photographic record of activity, and authenticate individuals through coded input via an infrared remote. Upon successful identification, the system simulates gate access by activating a servo-driven mechanism. Throughout the development process, attention was given to hardware-software integration, signal reliability, and power stability. This paper outlines the project objectives, system design, implementation methodology, and testing procedures that support the creation of a functional and efficient embedded access control prototype suitable for educational and applied research setting.

Schematics

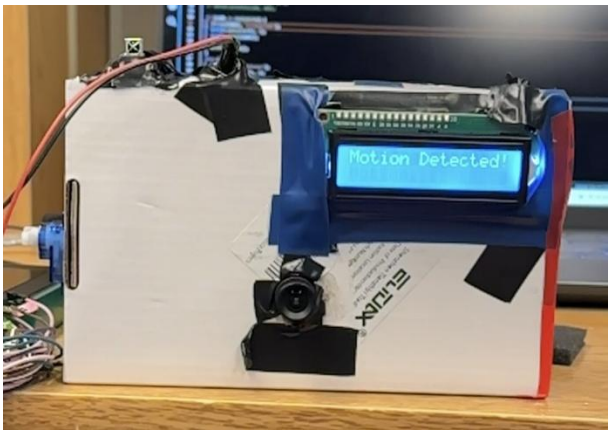

Adafruit VC0706 TTL Camera
5V
GND
RX
TX
1 kΩ
560 Ω
100 µF
ARDUINO
UNO r3
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
PWM
TX
RX
MicroSD Card Module
CS
SCK
MOSI
VCC
GND
MISO
S
LCD
LED-
DB7
DB5
DB3
DB1
E
RS
VDD
LED+
DB6
DB4
DB2
DB0
R/W
V0
VSS
10 kΩ
DATA
VCC
GND

Project Images

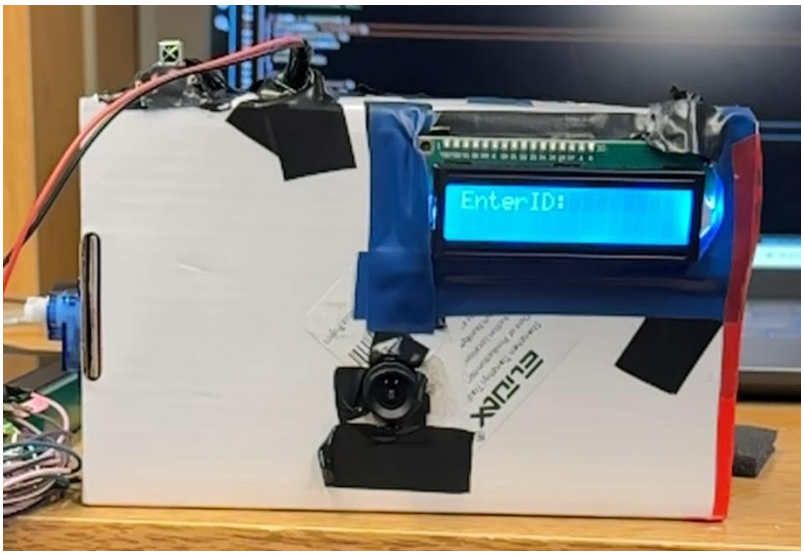
Looking For Motion



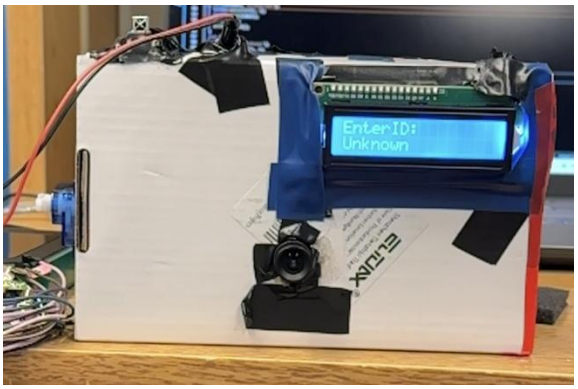
Motion Detected



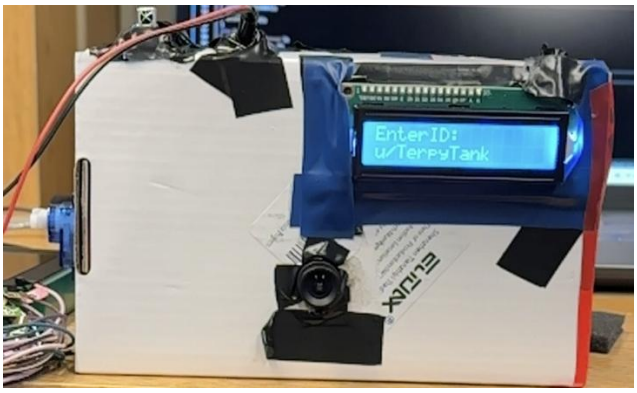
EnterID:



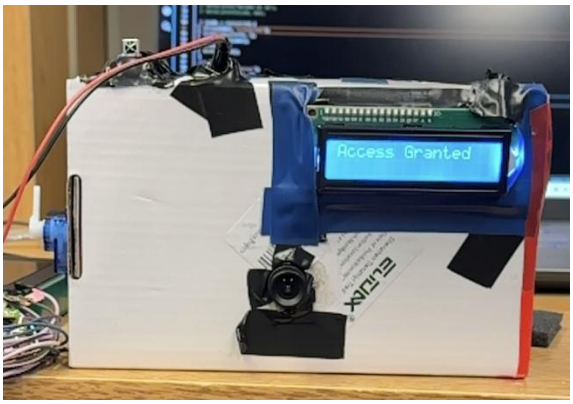
Unknown User



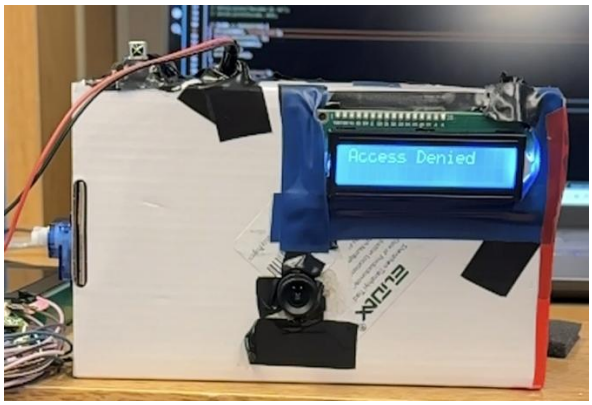
Known User



Access Granted



Access Denied



Objectives for the Security Camera and System

The objective of this project was to design and implement an Arduino-based gate entry security system capable of detecting motion, capturing images, and verifying identity using an infrared (IR) remote. The system aimed to automate the process of monitoring and granting access to a secure area using a combination of sensors and peripheral devices. Specifically, the project required the setup of a TTL serial camera to detect motion and capture images, a microSD card module to store those images, and a 1602 LCD screen to notify the user of system status and access prompts. Upon motion detection, the system would prompt the user to provide identification via an IR remote. If the transmitted IR code matched a predefined authorized value, a servo motor would rotate to simulate gate opening, then return to its original position after a brief delay. The LCD would update at each stage, displaying messages such as “Motion Detected,” “Enter ID,” and either “Access Granted” or “Access Denied.” Although the initial project requirements included integration with a web platform (ThingSpeak) for moderator verification and online status monitoring, this feature was not implemented in the final version. Similarly, plans to use a DC motor and relay for physical gate control were replaced with a servo motor simulation. The project successfully demonstrated a functional and compact security access system using real-time sensor input and output control via a microcontroller.

Algorithm for the Security Camera and System

The algorithm for the gate entry security system was designed to coordinate multiple modules in a sequential process triggered by motion detection. The program begins by initializing all connected peripherals, including the TTL camera, LCD display, servo motor, SD card module, and IR receiver. Once initialized, the system enters an idle loop where the LCD displays a message indicating that it is actively scanning for motion. When the camera detects motion, the LCD is updated to notify the user, and a snapshot is immediately taken and saved to the microSD card with a unique filename. Following image capture, the system prompts the user to enter an ID via an IR remote. The IR signal is decoded, and the received code is compared to pre-approved hexadecimal values stored in the program. If the input matches a recognized code, the LCD displays the associated username and confirms access permission. The system then activates the servo motor to rotate to an open position, simulating the gate opening, maintains the position for five seconds, and returns the servo to its original state. If the IR code is not recognized, the LCD displays an “Access Denied” message, and no further action is taken. Each operation is accompanied by timed LCD updates to ensure users are informed throughout the sequence.

1) **System Initialization**
    a) Begin serial communication for debug output.
    b) Initialize the LCD screen and set the cursor position.
    c) Attach the servo motor to its control pin and move it to the closed gate position.
    d) Initialize the microSD card and prepare it for file operations.
    e) Configure the TTL camera settings, including image resolution and motion detection.
    f) Enable the IR receiver for capturing input from an IR remote.

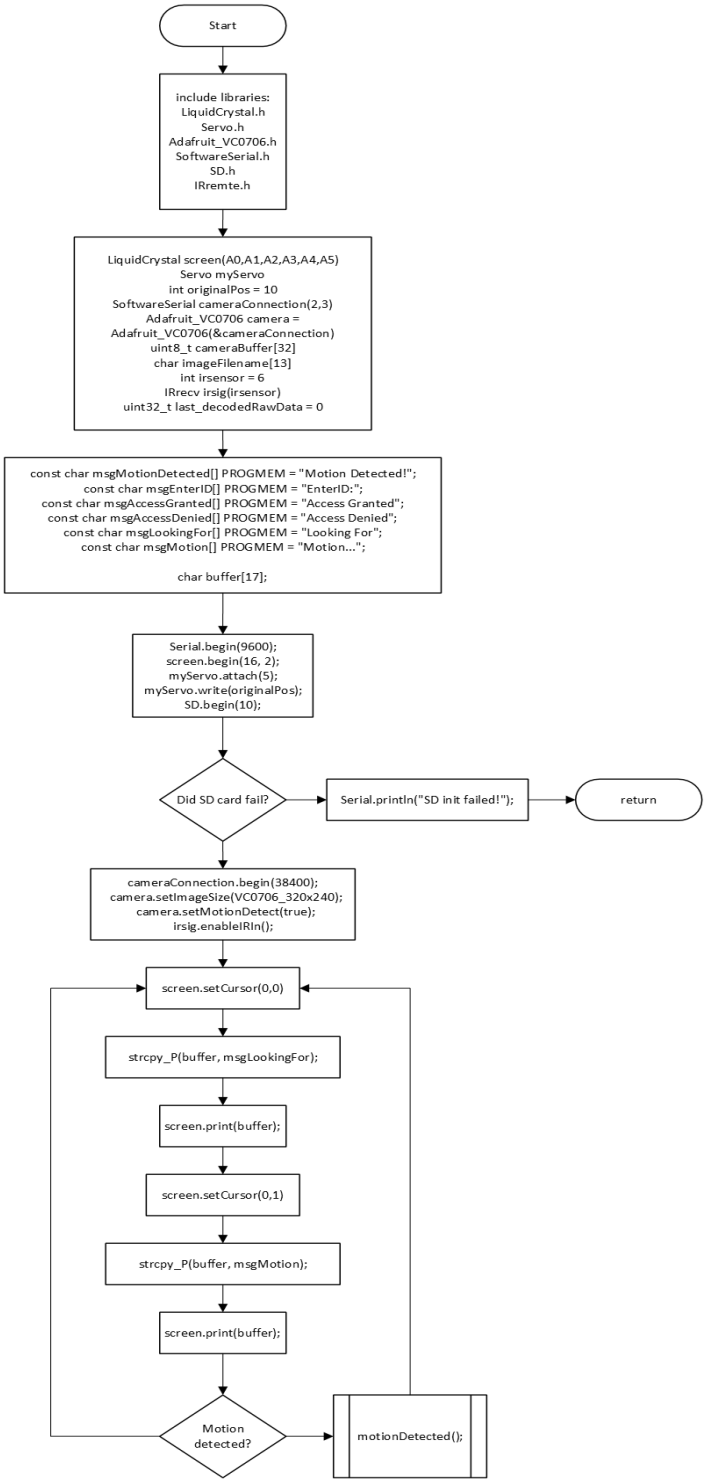
2) **Main Monitoring Loop**
    a) Display a message on the LCD indicating that the system is actively scanning for motion.
    b) Continuously check the camera for a motion detection signal.
    c) If motion is detected, proceed to the next steps.
3) **Motion Response Routine**
    a) Disable motion detection temporarily to avoid interference during image capture.
    b) Display a "Motion Detected" message on the LCD.
    c) Capture a still image using the TTL camera and save the file to the SD card with an automatically incremented filename.
    d) Prompt the user to input an access ID using the IR remote.
    e) Wait for and decode the IR signal input.
4) **Access Verification**
    a) Compare the decoded IR code against authorized values.
    b) If the IR code matches a predefined valid ID (e.g., 0xE916FF00), display the user label, show an "Access Granted" message, and trigger the servo motor to simulate gate opening for a predefined time interval.
    c) If the IR code does not match, display an "Access Denied" message and do not activate the gate mechanism.
5) **System Reset**
    a) Clear the LCD screen after each operation.
    b) Re-enable camera motion detection.
    c) Return to the main monitoring loop to await the next motion event.

Hardware Overview

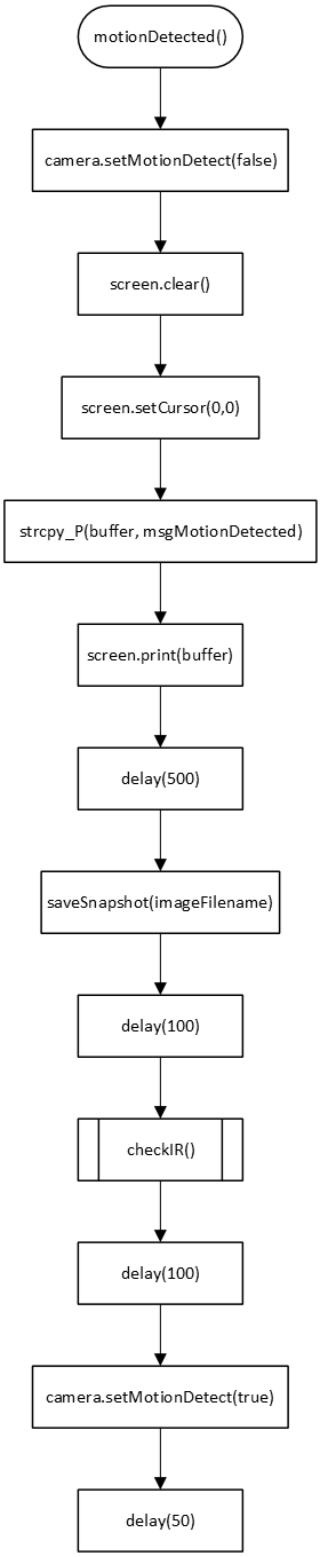
- Elegoo UNO R3
- Jumper Wires
- SD Card Module
- 8 GB SD Card
- 10 KΩ Potentiometer
- Servo Motor
- IR Sensor
- IR remote
- Adafruit VC0706 TTL Camera
- 100 µF Capacitor
- 1 KΩ Resistor
- 5 V DC Power Supply
- LCD 1602 Module
- Breadboards
- Arduino IDE

Flowchart and Code Description

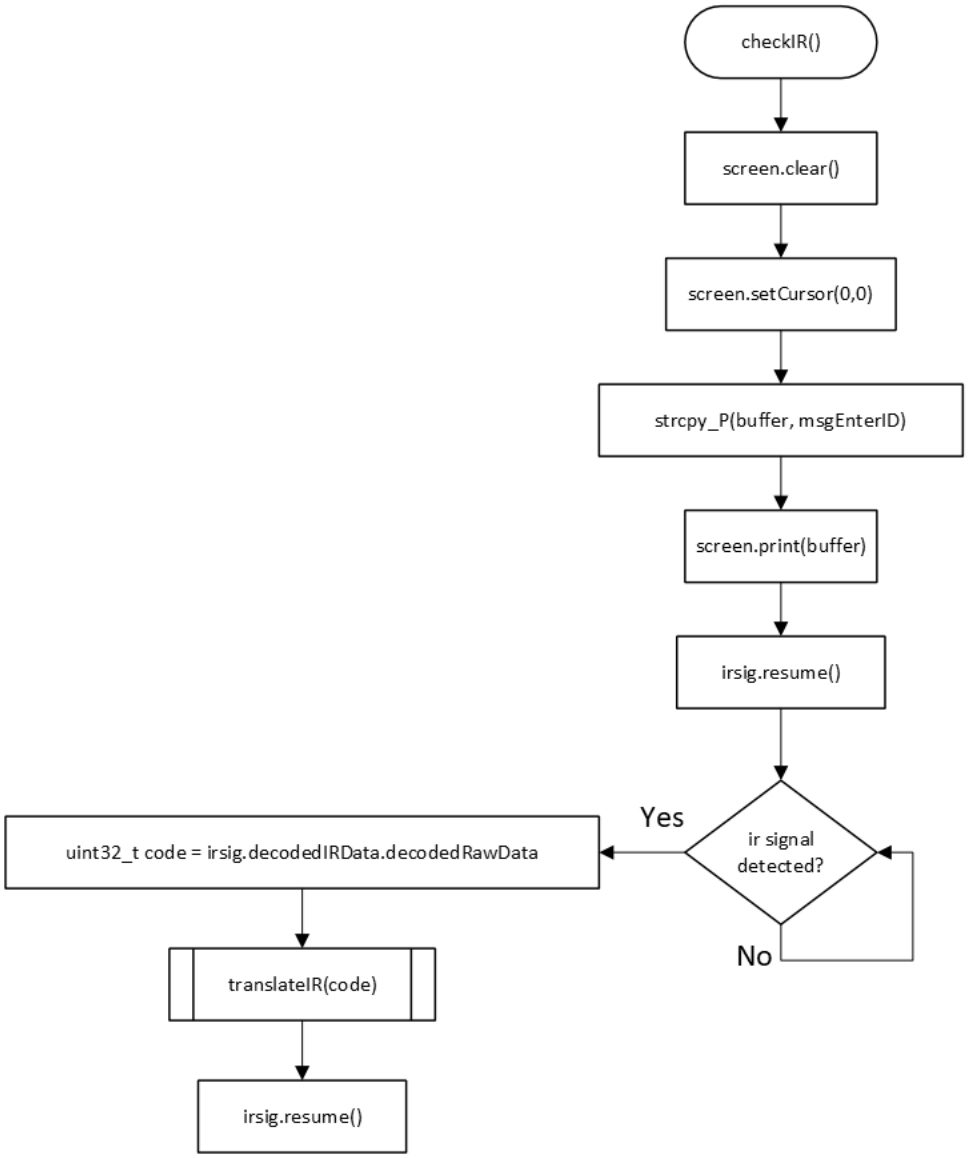
Main Setup & Loop



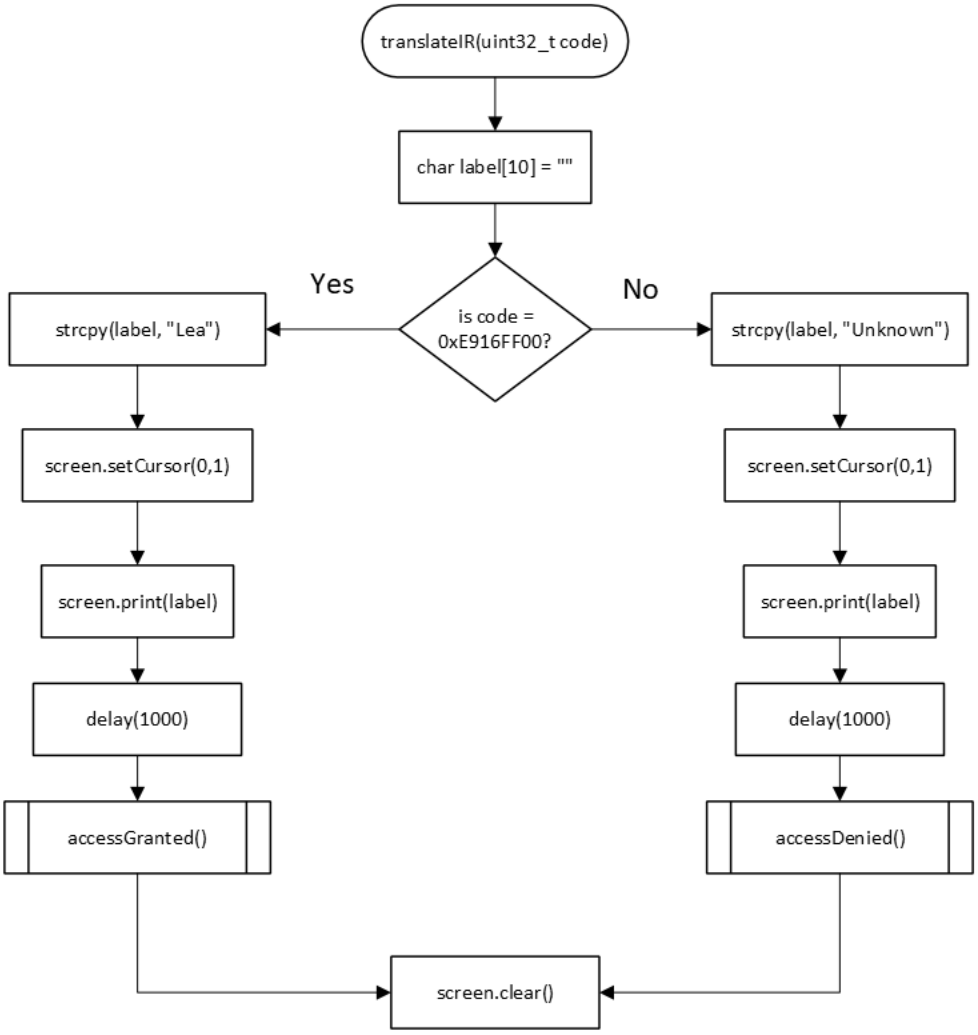
motionDetected() Function



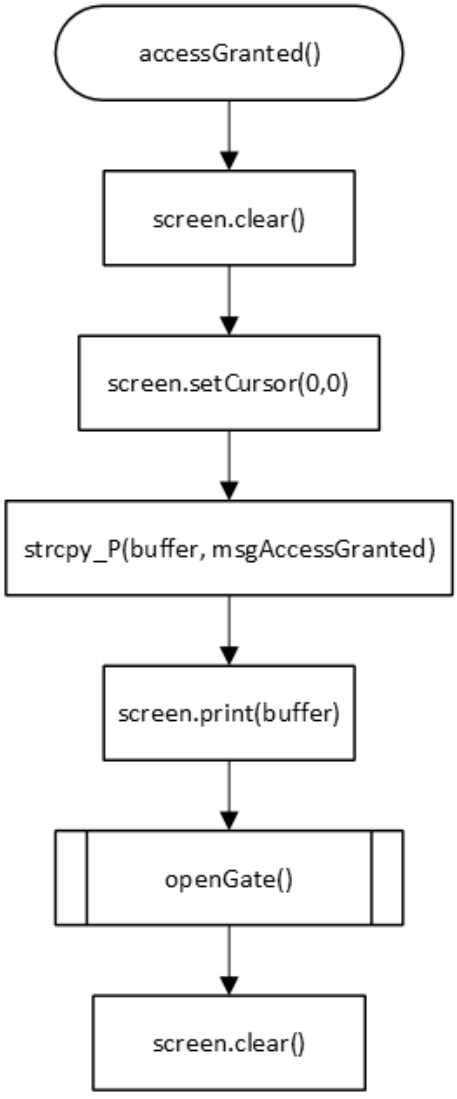
checkIR() Function



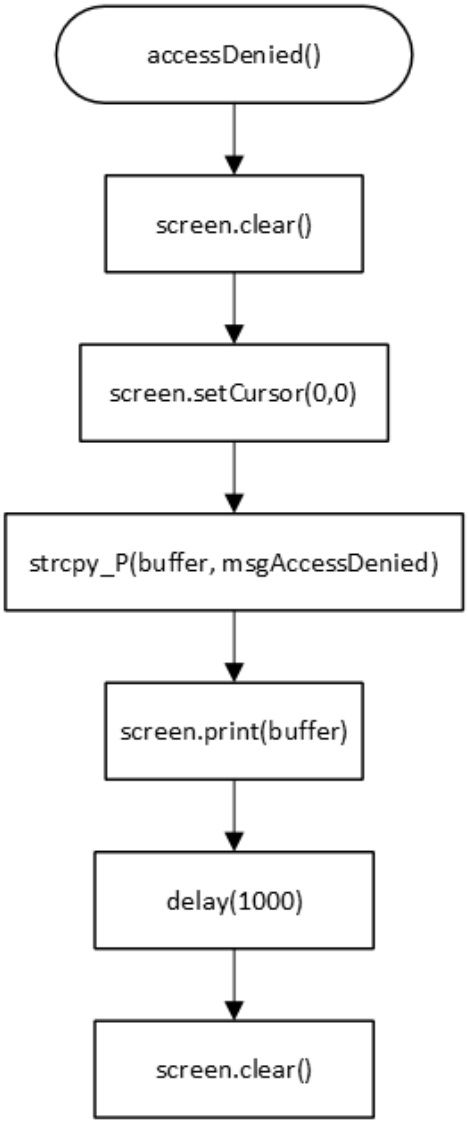
translateIR() Function



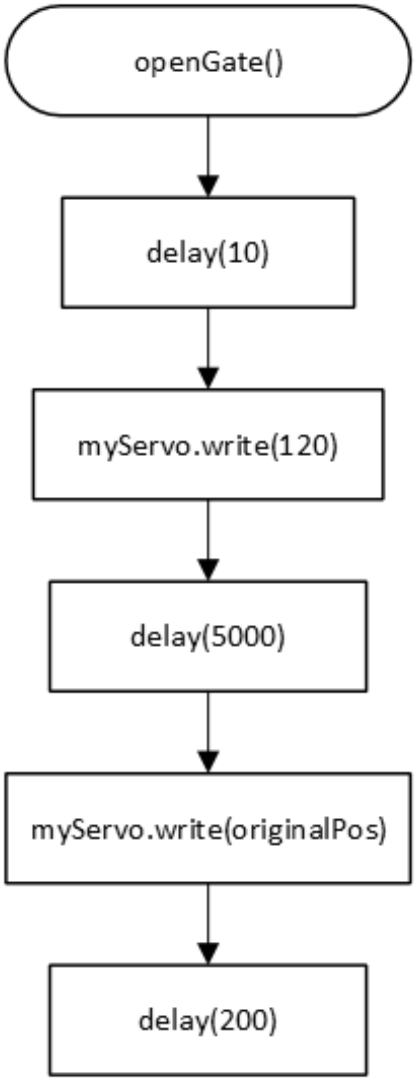
accessGranted() Function



accessDenied() Function



openGate() Function



Test and Debug

Extensive testing and debugging were required throughout the development of the Gate Entry Security System to ensure proper functionality of all integrated components. Challenges emerged during the interfacing of the TTL serial camera, IR receiver, SD card module, LCD display, and servo motor. Each issue was resolved through step-by-step troubleshooting, code refinement, and hardware verification.

One of the initial challenges encountered was establishing reliable serial communication between the Arduino and the VC0706 TTL camera module. Serial communication requires careful matching of transmit (TX) and receive (RX) lines, which are unidirectional. Early wiring attempts incorrectly connected TX-to-TX and RX-to-RX between the Arduino and camera, which resulted in failed camera initialization. To resolve this, the TX pin of the Arduino was connected to the RX pin of the camera, and the RX pin of the Arduino was connected to the TX pin of the camera, following standard UART protocol. Because the Arduino UNO has only one hardware serial port, which was occupied by the USB connection for debugging, a SoftwareSerial interface was used to create a second serial connection dedicated to the camera. Pins D2 and D3 were assigned for RX and TX respectively, allowing the camera to operate on its own serial channel and eliminating communication interference with the serial monitor.

SD card integration presented a different set of issues related to memory management and initialization timing. In an effort to reduce SRAM usage, the standard SD library (SD.h) and its dependent File class were modified to strip out unused functions. However, this caused compile-time errors because critical virtual functions such as read() and peek() were removed, breaking compatibility with the Arduino core libraries. After restoring these methods, the SD card still failed to initialize during runtime. Diagnostic serial output revealed that SD.begin(cs) returned false, indicating a hardware or communication issue. This was ultimately traced to the order of peripheral initialization in the setup() function. The SPI bus is shared by both the SD card module and the camera library, and initializing the

camera before the SD card caused bus contention. Reordering the initialization by calling SD.begin(10) before starting the camera resolved the conflict and allowed the SD card to be accessed reliably.

Further improvements were made to ensure compatibility with FAT32 filesystems used on microSD cards. FAT32 requires file names to be in uppercase and follow the 8.3 format. To accommodate this, image filenames were programmatically converted to uppercase using a custom strupr() function before being passed to SD.open(). This step ensured that filenames like “IMG001.JPG” were correctly recognized by the filesystem, avoiding cases where lowercase names silently failed to create files. With these changes in place, the system successfully captured images upon motion detection and saved them to the SD card under incremented filenames.

The camera save routine was enhanced to reliably store images. This involved looping through the picture data using 32-byte buffer chunks and ensuring the camera’s motion detection feature was temporarily disabled during the save operation. For the IR remote, decoding issues were identified when repeated button presses generated identical values. These were resolved by properly resuming the receiver after each signal and ensuring each code was correctly parsed. To minimize memory usage, LCD display messages were stored in program memory using PROGMEM, and functions like strcpy_P were used to retrieve them efficiently.

Servo motor jitter was identified during testing, often causing unpredictable motion after the gate access routine. The issue was traced to voltage dips caused by current draw from the servo. Solutions included recommendations to use decoupling capacitors across the power and ground lines, or to power the servo from a separate source with a shared ground. Additional testing confirmed that improper grounding or shared power lines contributed to erratic behavior.

LCD initialization required precise timing to avoid display corruption. The custom ShiftedLCD library used SPI to communicate with a 74HC595 shift register driving the 1602 LCD in 4-bit mode. This approach initially offered the benefit of conserving digital I/O pins by shifting data through serial-to-parallel conversion. However, it required detailed analysis of the shift register's pin behavior and a firm

understanding of how to perform bitwise operations to control specific LCD lines such as RS, Enable, and data bits D4 through D7. Extensive research was conducted to break down how the LCD's command structure maps to 4-bit operations, and how those bits must be clocked through the shift register using SPI.transfer along with proper latch and pulse timing. Despite significant effort in reverse-engineering the instruction set and timing sequence, persistent display instability and difficulty troubleshooting SPI timing led to the shift register approach being abandoned. Instead, the LCD was connected directly to the Arduino's analog pins A0 through A5, allowing for more straightforward parallel communication. This change improved reliability and reduced debugging complexity, especially during the critical initialization phase which follows the HD44780 standard sequence of function set, display control, and entry mode commands.

Later in the development process, memory constraints caused SRAM overflow errors due to the use of dynamic String objects and large buffers for camera and SD card operations. These were corrected by switching to fixed-length char arrays, reducing buffer sizes, and streamlining code. After implementing all necessary optimizations and corrections, the system successfully completed its intended functions: detecting motion, displaying messages, capturing and storing images, and validating user access using the IR remote. Each fix was confirmed through LCD feedback and serial monitor output during test cycles.

Conclusion

The Gate Entry Security System project achieved its primary objective of creating a reliable Arduino based access control prototype that integrates motion detection, image capture, identity verification, and mechanical actuation. The final design successfully combined multiple peripherals including a TTL serial camera, LCD screen, IR receiver, microSD card module, and both a servo motor and a DC motor via relay control to simulate real world security entry operations. Careful attention was given to hardware timing, memory optimization, and electrical stability. Through extensive debugging and design revisions, the system was refined to handle motion triggered events, validate authorized users via IR codes, capture and store photographic records, and activate a physical gate mechanism using both servo and DC motors. The experience emphasized the importance of modular code structure, hardware software synchronization, and thoughtful peripheral interfacing. Future improvements may include wireless data upload, encrypted access logs, and mobile app control for enhanced functionality and scalability.

References

Monk, S. (2016). *Programming Arduino: Getting Started with Sketches* (2nd ed.). McGraw Hill.

Seneviratne, P. (2015). *Internet of Things with Arduino Blueprints: Develop Interactice Arduino-Based Internet Projects with Ethernet and Wi-Fi* . Packt Publishing.

Footnotes

Figure 1.

Complete wiring schematic of the project

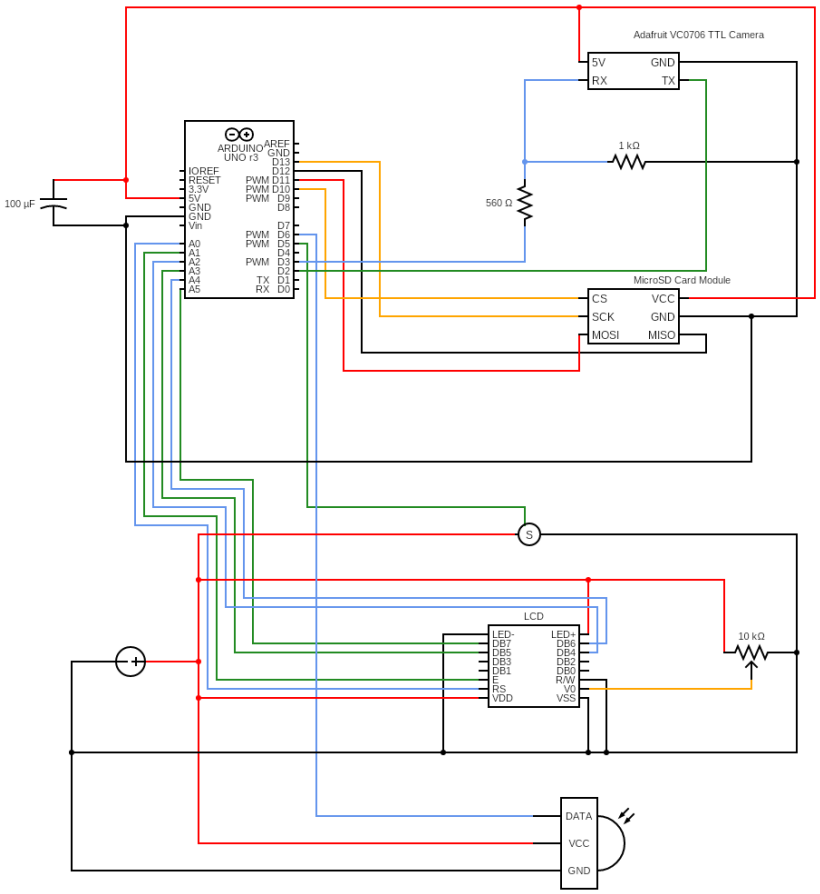
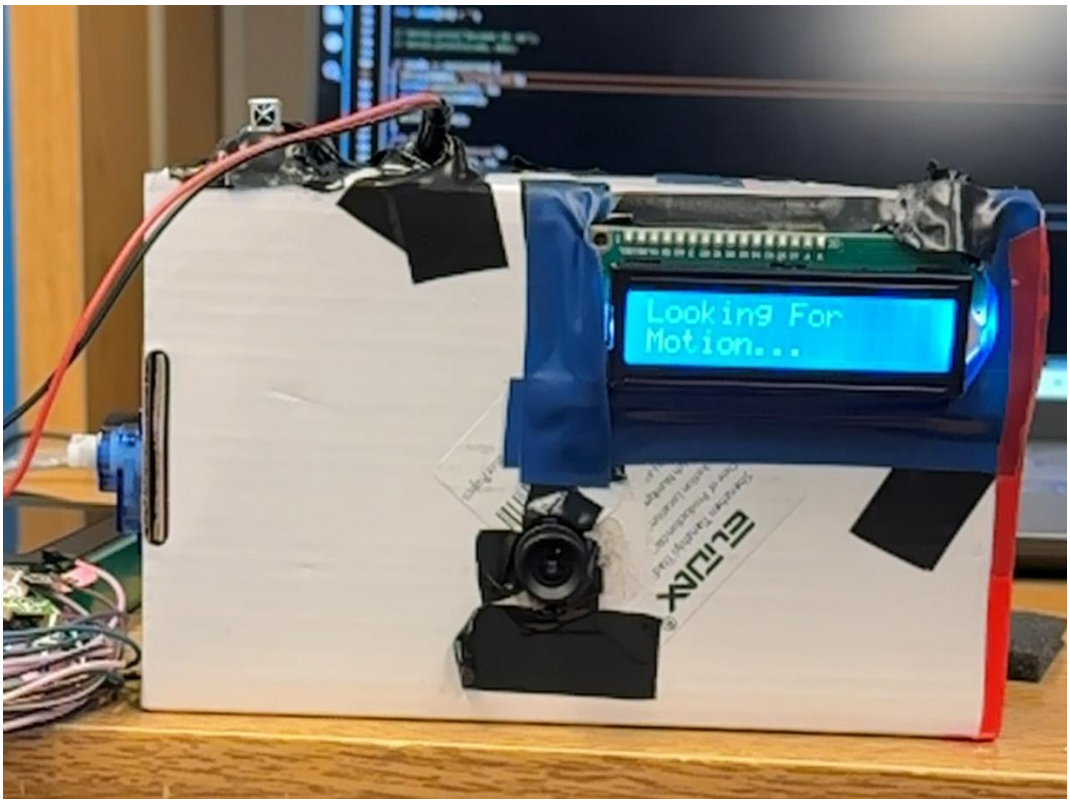


Figure 2.

Image of completed project demonstrating LCD prompt is informing user that the device is actively looking for motion. This step is only taken if the SD card has successfully initialized.



INDIANA STATE UNIVERSITY

BAILEY COLLEGE OF ENGINEERING & TECHNOLOGY

ECT 405 Final Project Lab Report

May 2025

Prepared by:

Lea Tice

Table of Contents

Objectives

The primary objective of this project was to investigate and understand the electrical signal behavior of a microcontroller-based, motion-activated security camera system. This system integrates an ELEGOO Uno R3 microcontroller with two peripheral components: a TTL serial camera (Adafruit VC0706) for image capture and a microSD card module for data storage. The focus of the analysis was on how these components interact, particularly during critical moments such as image capture and file writing.

To achieve this, the Analog Discovery 3 logic analyzer and the WaveForms software were utilized to probe and visualize real-time signals within the circuit. Specific attention was given to monitoring the SPI communication lines, Chip Select (CS), Serial Clock (SCK), Master Out Slave In (MOSI), and Master In Slave Out (MISO), to confirm that the data transfer processes were being executed correctly and in the appropriate sequence. These signals were chosen due to their critical roles in orchestrating communication between the microcontroller and the SD card during write operations.

Additionally, power stability was analyzed by measuring the analog voltage supply during both idle and active states. This helped identify any dips, surges, or irregularities in voltage that could interfere with normal operation, particularly during moments of high current demand, such as when saving images to the SD card.

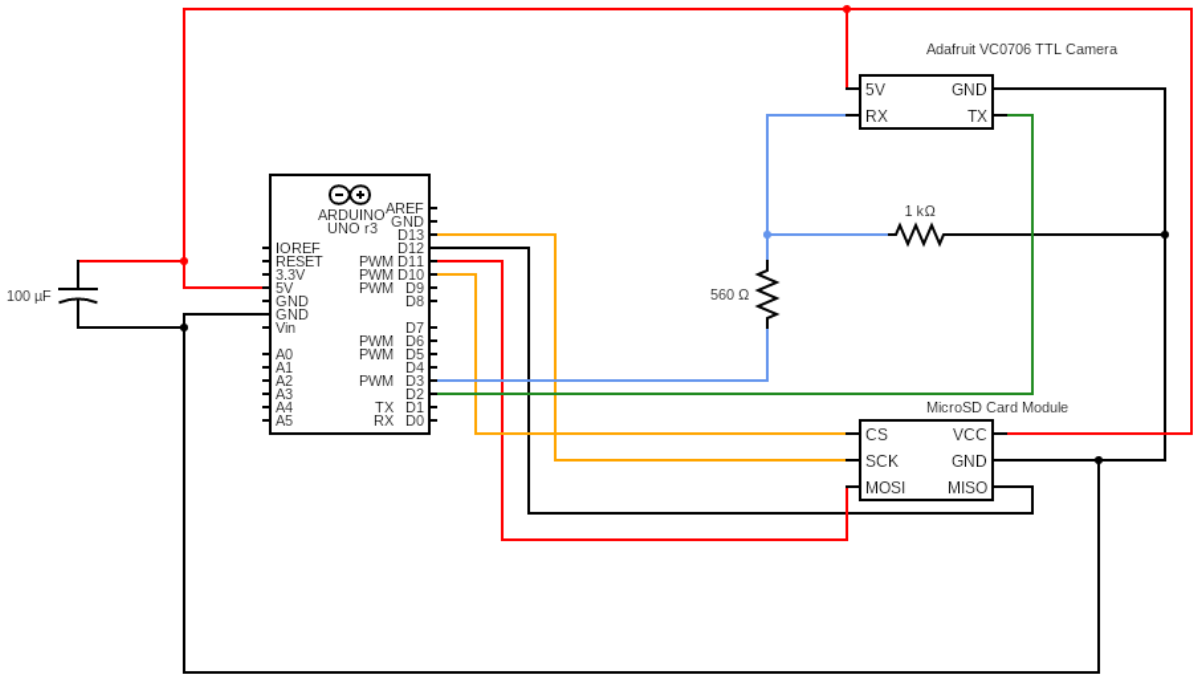
Another objective of this project was to troubleshoot and mitigate issues commonly associated with power delivery in embedded systems, such as current spikes and potential brownouts. The project aimed to assess the effectiveness of using decoupling components, like capacitors, and other methods to stabilize voltage and protect against resets or communication failures. By examining both the data and power aspects of the system, this project intended to provide a well-rounded understanding of embedded signal behavior under real-world operating conditions.

Equipment and Materials

- Analog Discovery 3
- WaveForms Software
- USB-C cable
- AD3 Wire Bundle w/Harness
- Jumper Wires
- ELEGOO Uno R3 Microcontroller
- MicroSD Card Module
- Adafruit VC0706 TTL Serial Camera Module
- 1 100 µF Capacitor
- 1 1KΩ Resistor
- 1 560 Ω Resistor

Methods

Circuit Schematic:



This circuit is powered by an ELEGOO Uno R3 microcontroller, which controls two key modules: the Adafruit VC0706 TTL Serial Camera and a MicroSD card module. The circuit is part of a larger system that utilizes the motion detection feature of the TTL camera. Once motion is detected, the camera captures a snapshot, which is then saved onto the SD card.

The VC0706 camera module has six pins, though only the first four are used in this project. The final two pins are intended for video output and are not relevant here. The active connections are:

- **5V** – connected directly to the 5V output of the microcontroller.
- **GND** – connected directly to ground.
- **TX (transmit)** – connected to digital pin 2 on the Uno (used as the microcontroller's RX pin).
- **RX (receive)** – connected to digital pin 3 on the Uno (used as the microcontroller's TX pin).

Note: The camera's TX must connect to the Uno's RX and vice versa; TX-to-TX or RX-to-RX connections will not work. Pins 0 (RX) and 1 (TX) on the Uno are avoided because they are used for USB communication with the computer.

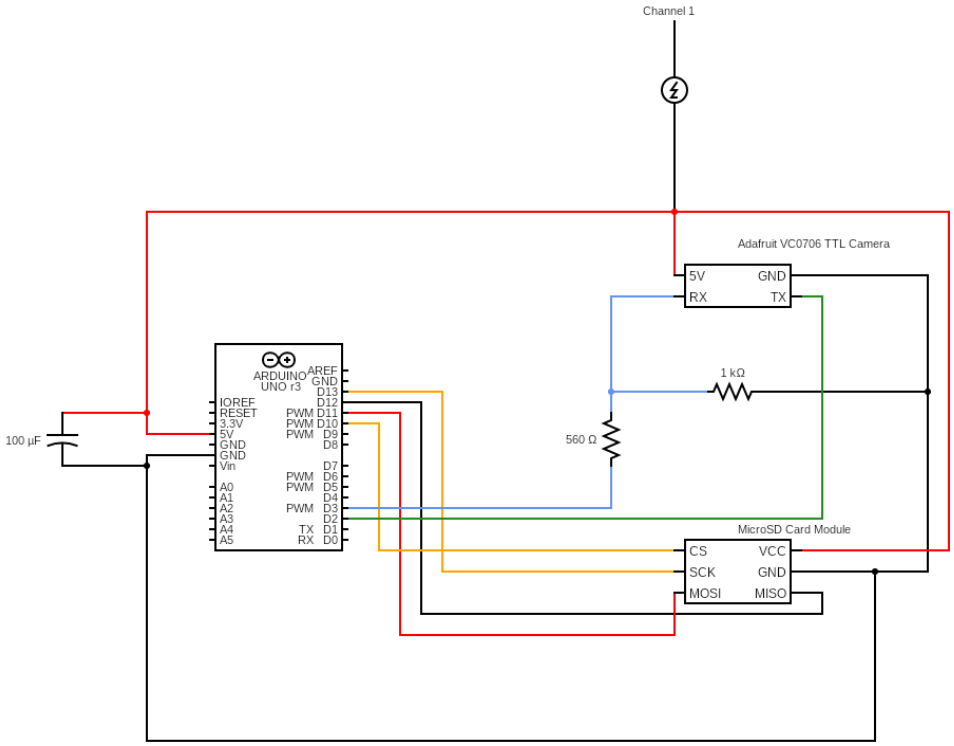
The MicroSD card module communicates using the SPI protocol and has the following six pins:

- **VCC** – connected to 5V power.
- **GND** – connected to ground.
- **CS (Chip Select)** – connected to digital pin 10 on the Uno. This pin is set LOW to activate communication with the SD card and HIGH to deactivate it.
- **MOSI (Master Out, Slave In)** – connected to digital pin 11. This line sends data from the microcontroller to the SD card (e.g., image data).
- **MISO (Master In, Slave Out)** – connected to digital pin 12. This line sends data from the SD card back to the microcontroller.

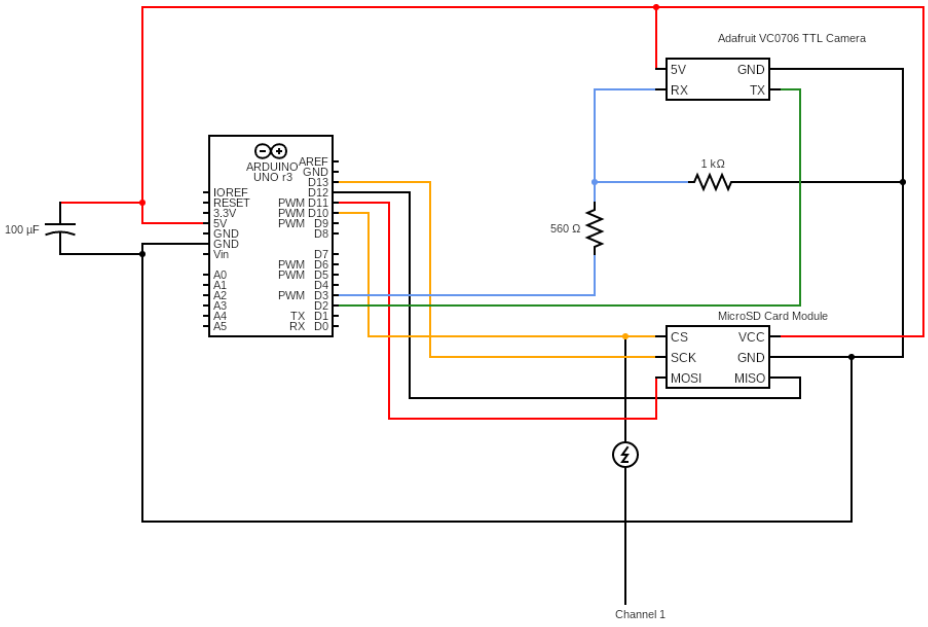
- **SCK (Serial Clock)** – connected to digital pin 13. This pin provides the clock signal to synchronize data transmission over SPI.

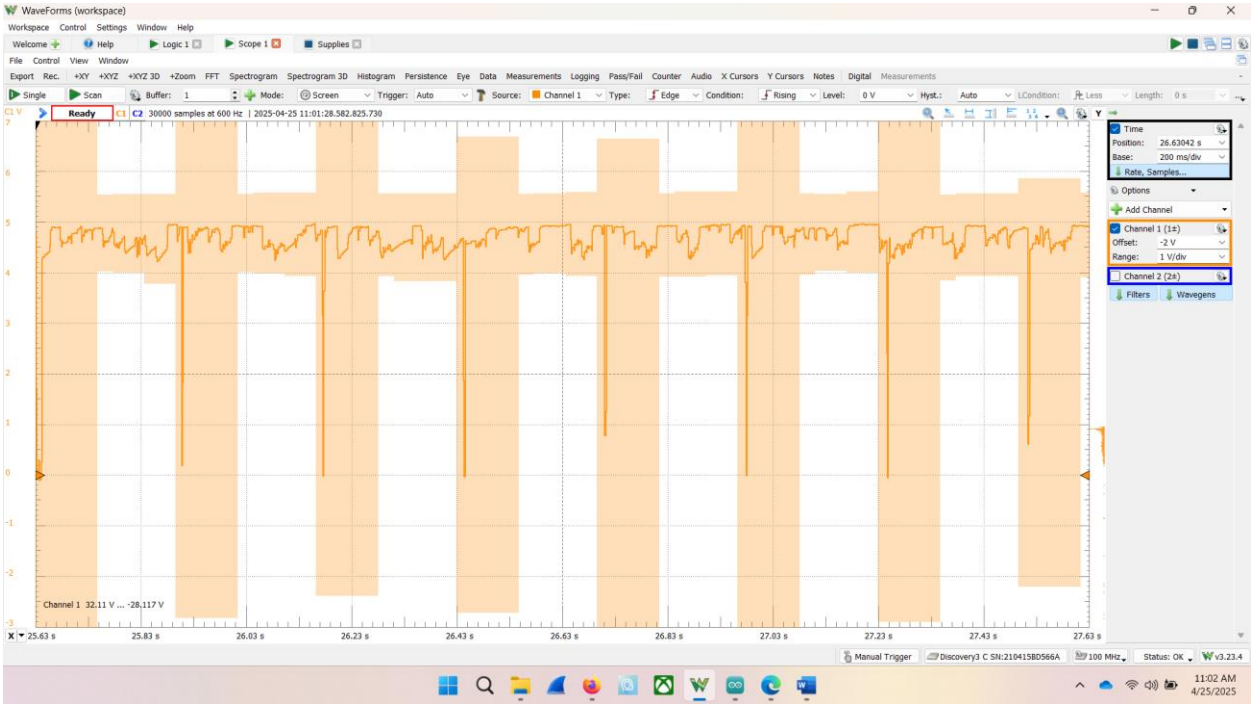
Using the Analog Discovery 3 and WaveForms software, I probed the following five points in the circuit to analyze signal behavior:

1. **Analog Voltage Supply** – Measured voltage stability during idle (passive) state and during active image capture/storage to observe current draw and supply dips.

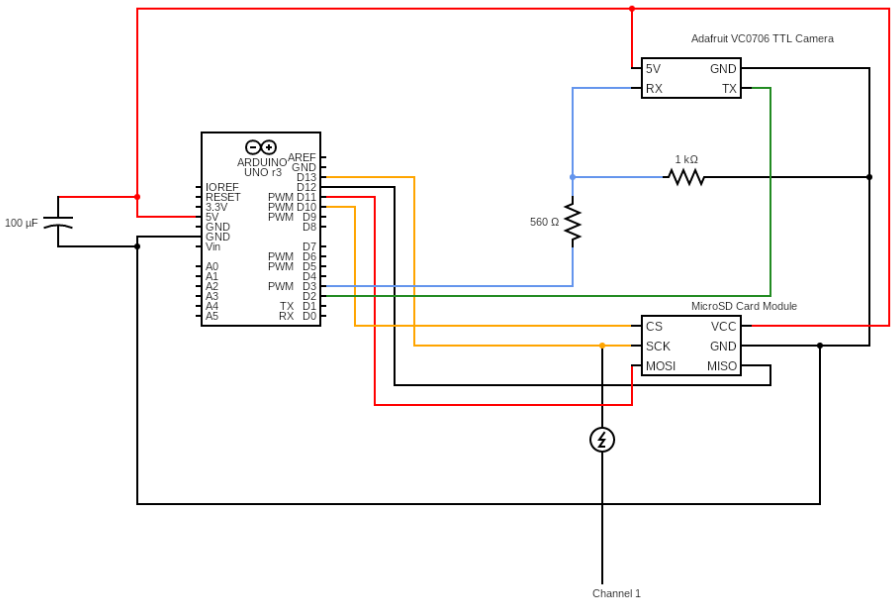


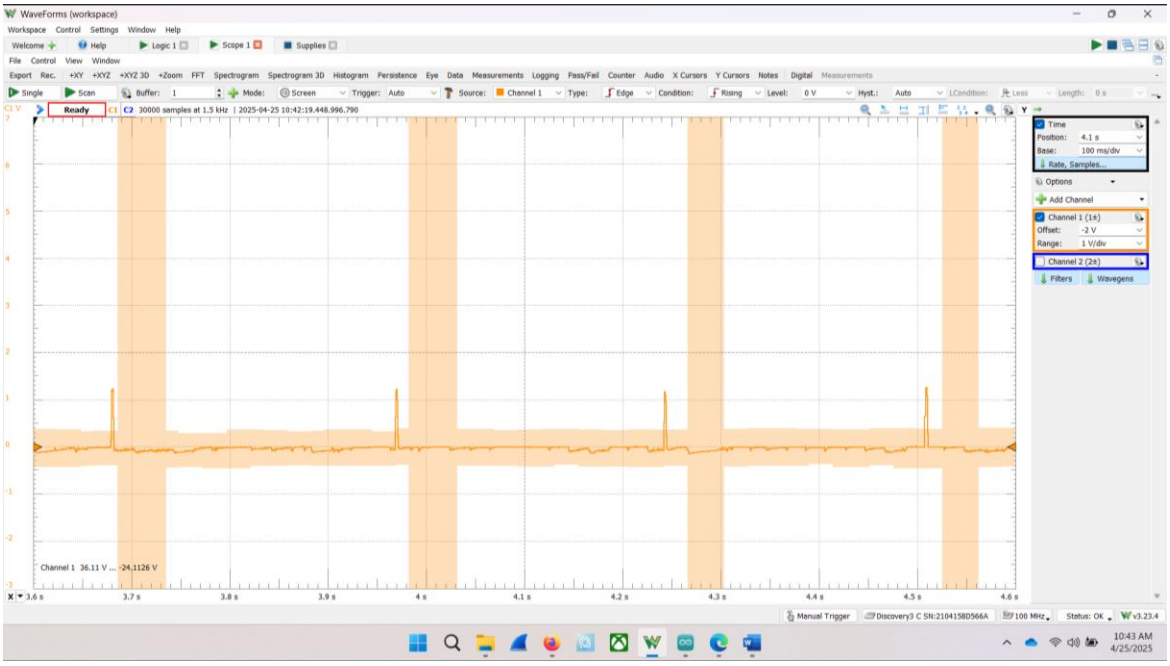
2. **CS Pin (Chip Select)** – Monitored to observe when the SD card is being actively accessed (LOW state indicates active communication).



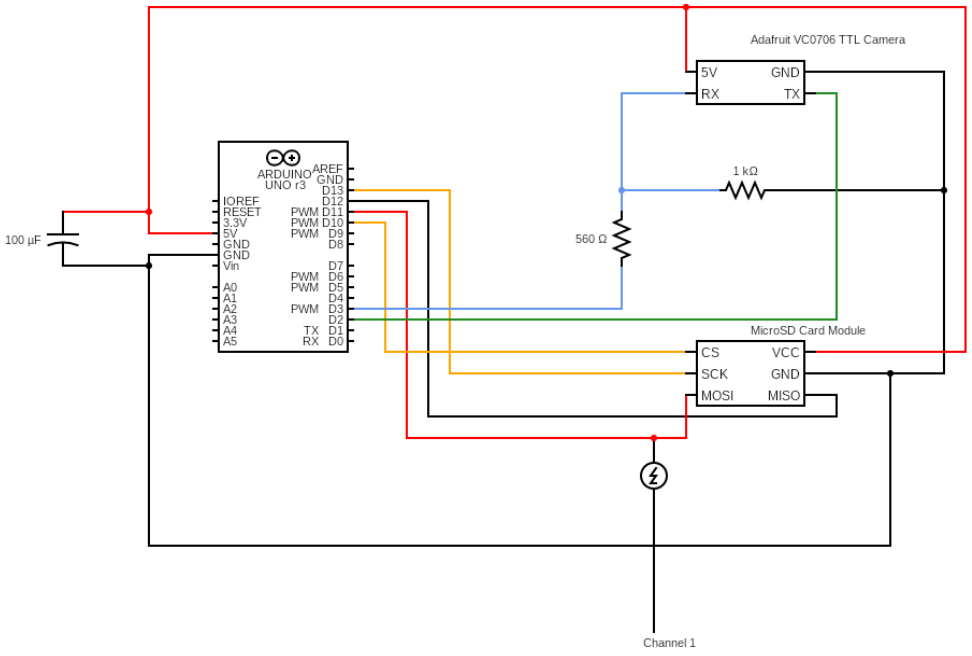


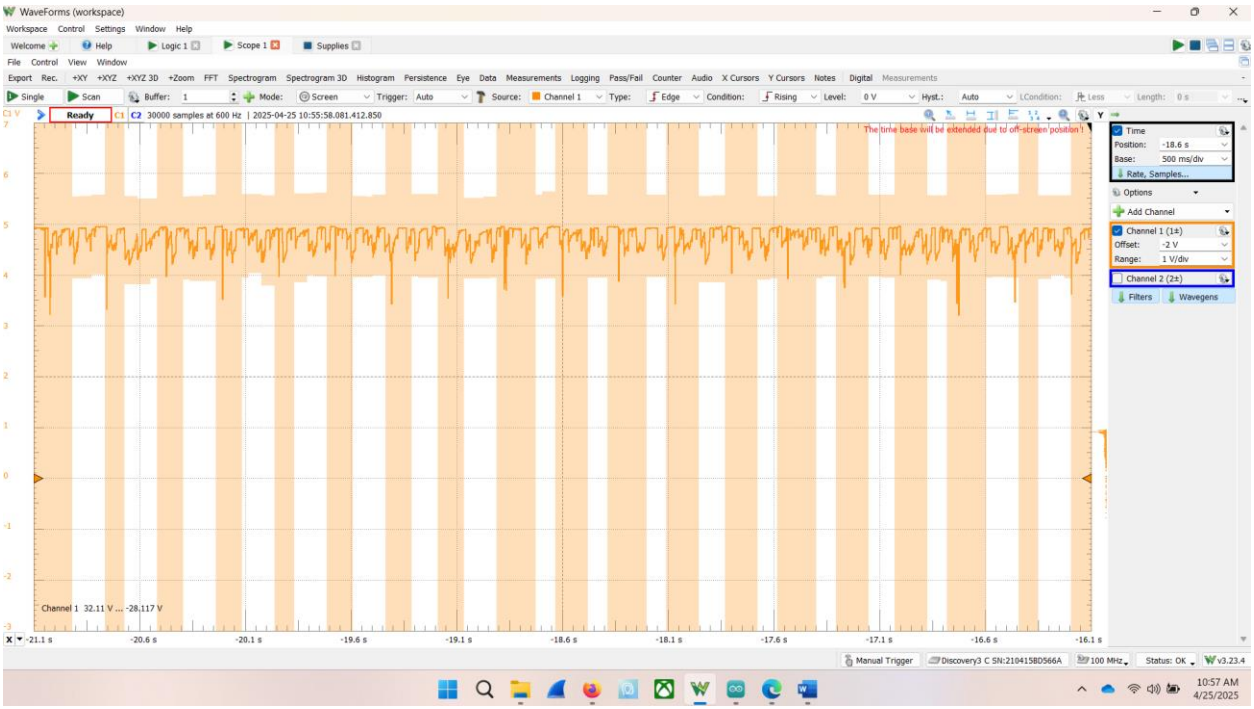
3. **SCK Pin (Serial Clock)** – Checked for timing pulses that indicate active SPI communication.



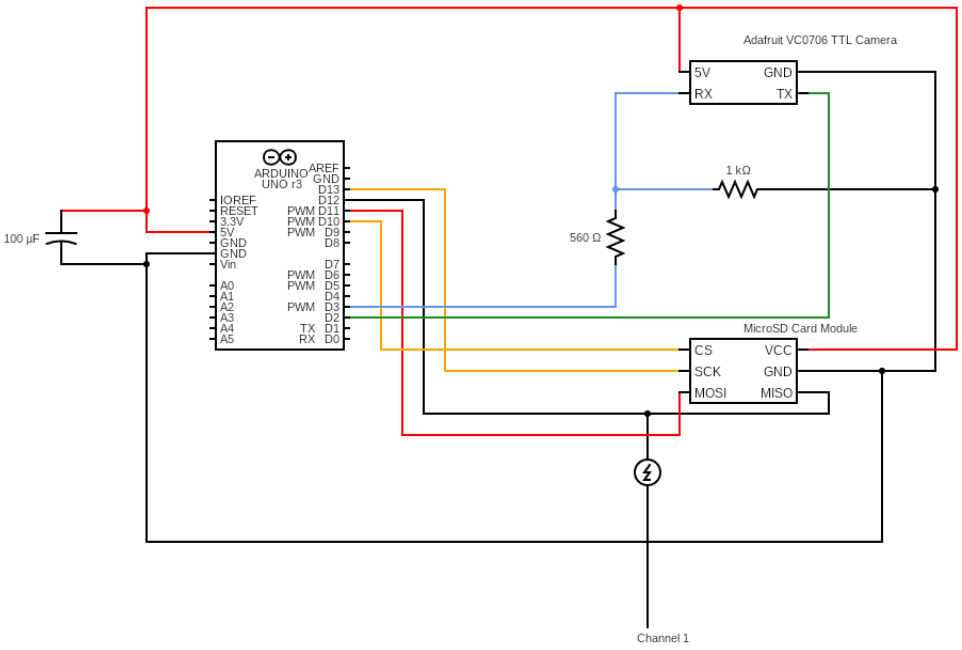


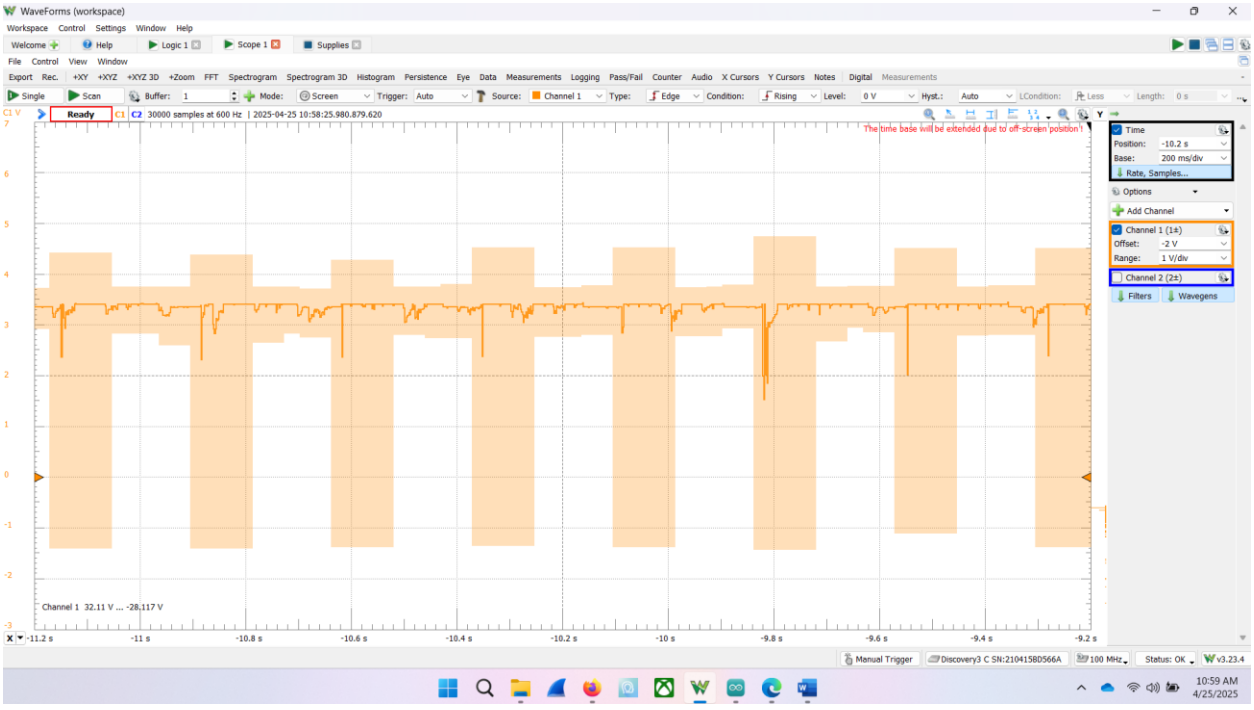
4. **MOSI Pin** – Observed to confirm data transmission from the Uno to the SD card during file writes.





5. **MISO Pin** – Captured to view data responses from the SD card back to the Uno.

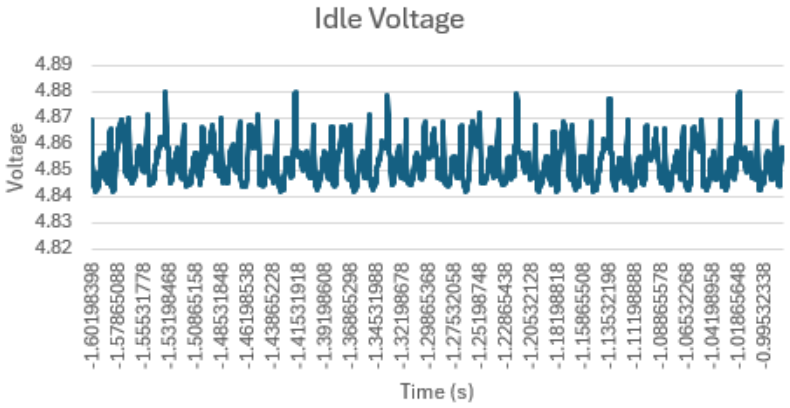


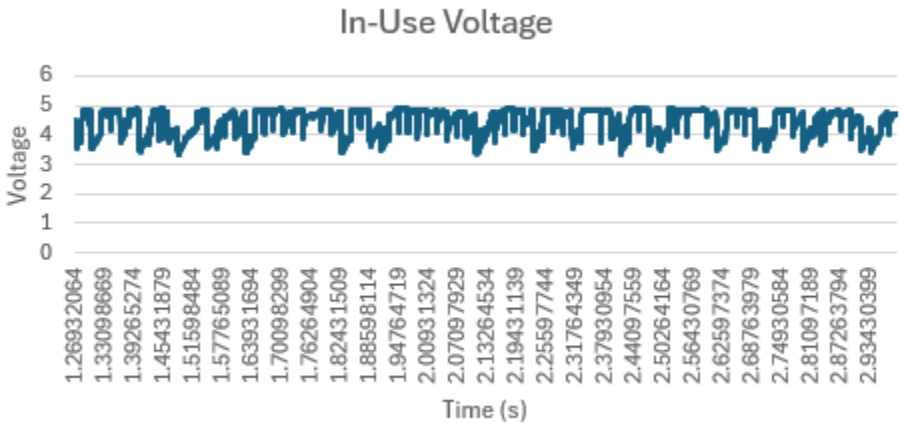


Results

Voltage behavior was measured during both the idle and active states of the circuit. While troubleshooting the full system, it became evident that current fluctuations occurred during image-saving operations to the SD card, likely due to the sudden current surges SD modules are known to cause. To mitigate potential brownouts or system resets, a 100 µF capacitor was added to stabilize the power supply. An attempt was made to filter power using diodes; however, this approach was unsuccessful, suggesting that my understanding of diode behavior in power filtering applications may need further refinement.

Min	3.325305
Max	4.921599
Avg	4.78406

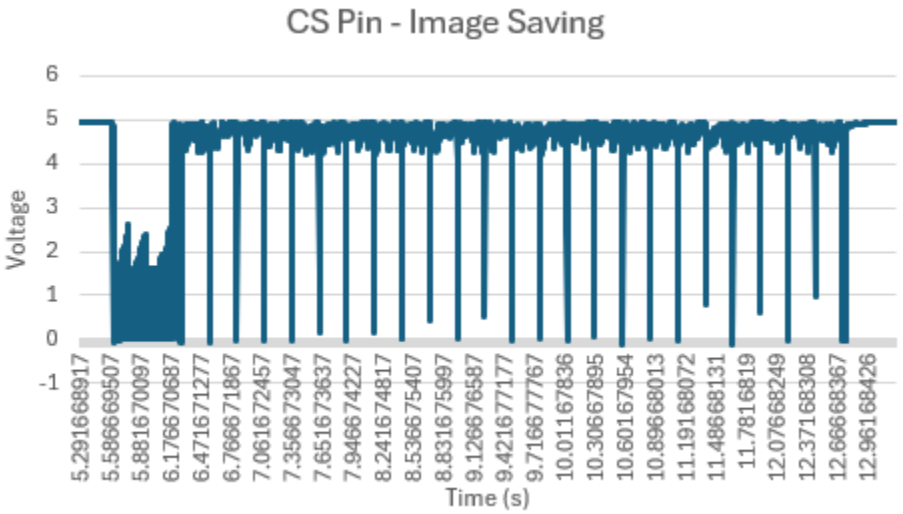


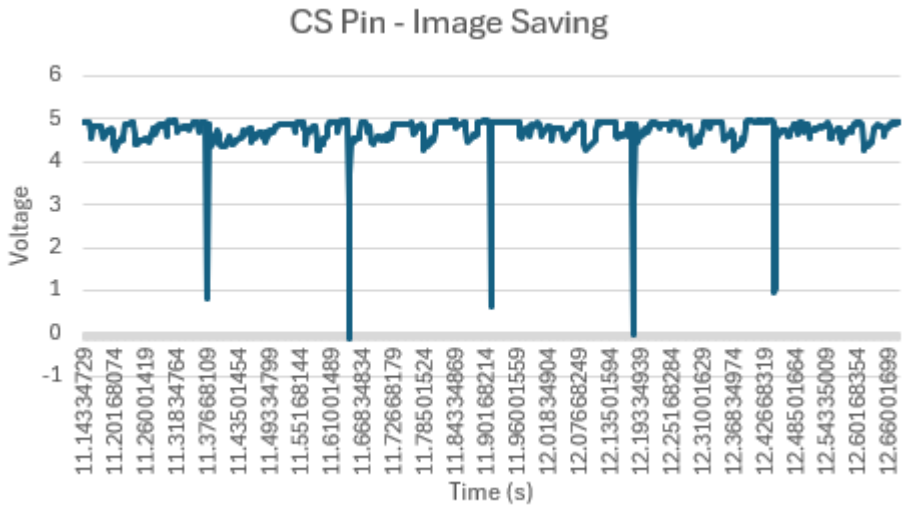


The CS (Chip Select) pin was monitored to observe its behavior during SD card access. While the system was idle, when no image was being captured or saved—the CS pin

remained HIGH, indicating that the SD card was not selected. Upon motion detection and image capture, the CS pin transitioned to LOW, marking the start of the write operation. This transition confirms that the microcontroller correctly selected the SD card for SPI communication. The shift from HIGH to LOW occurred precisely at the beginning of the file-saving process, validating proper SPI protocol implementation.

Min	-0.10602
Max	4.9558
Avg	4.704499

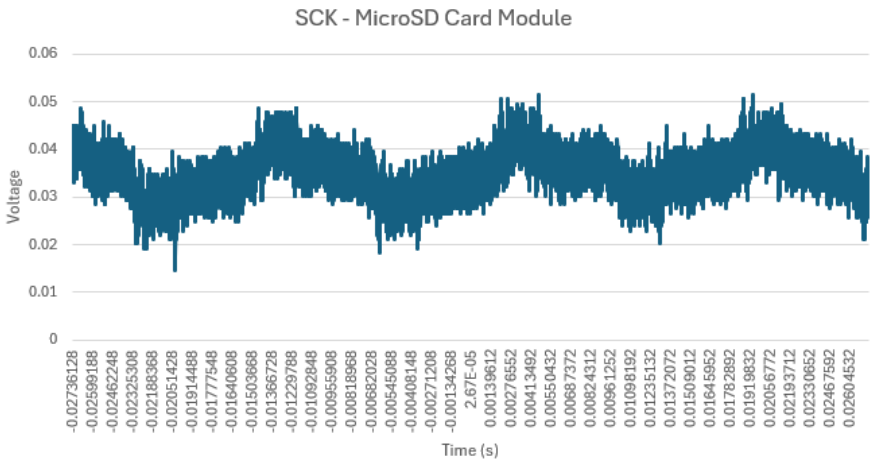




The SCK (Serial Clock) pin was measured to observe the timing pulses that synchronize SPI communication. Clock activity was expected only during actual data transfers, and the

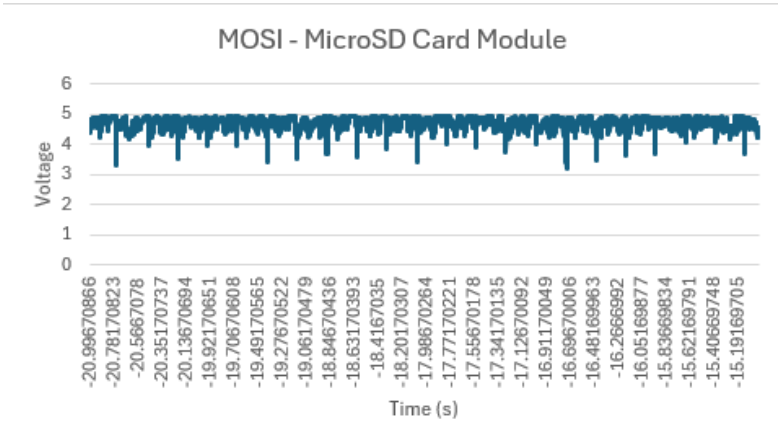
results confirmed this behavior. Clean, regular pulses were observed with no irregularities, indicating proper synchronization and reliable SPI timing.

Min	0.0146622
Max	0.051422
Avg	0.0350011



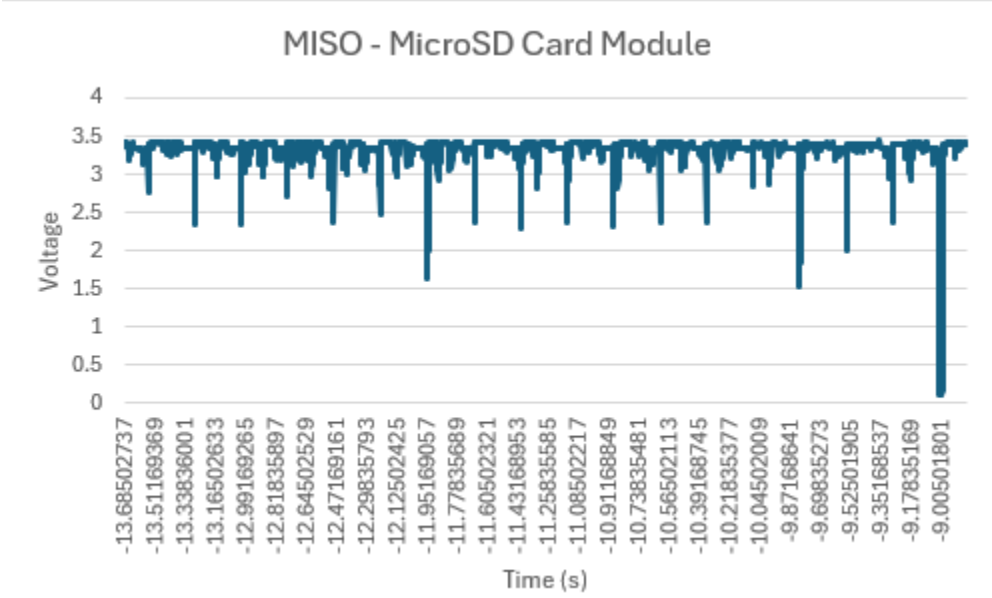
The MOSI (Master Out, Slave In) pin serves as the data transmission line from the microcontroller to the SD card. During image write operations, bursts of consistent signals were observed, indicating successful data transfer. This signal behavior also supports

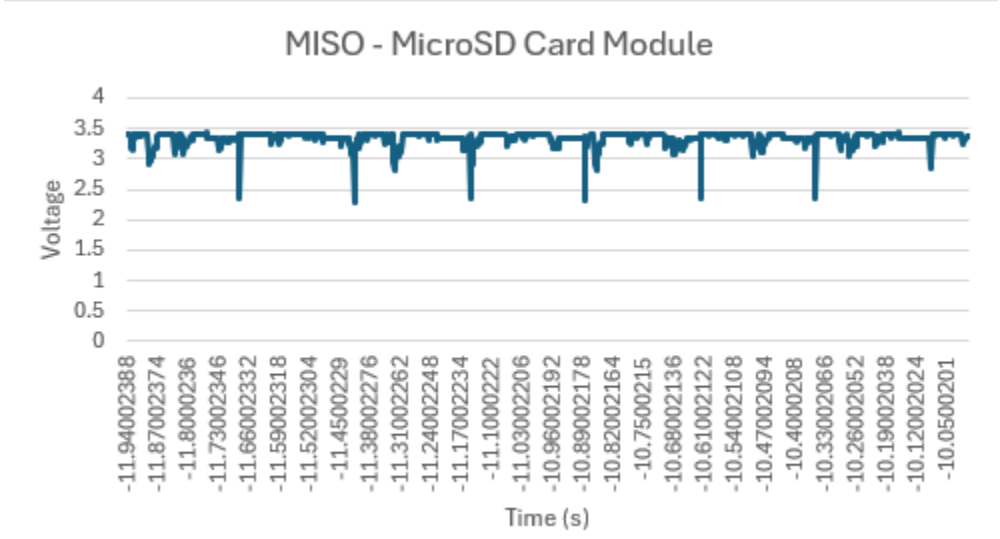
Min	3.215225
Max	4.956719
Avg	4.726027



proper timing coordination across all SPI lines, reinforcing that communication was functioning as intended.

The MISO (Master In, Slave Out) pin carries responses from the SD card back to the microcontroller. Signal activity was expected during SD card initialization, file opening, and writing processes. The presence of these signals confirms that bidirectional communication was functioning correctly and that the SD card was returning valid data as expected.





Conclusion

Through the use of the Analog Discovery 3 and WaveForms software, this project successfully captured and analyzed the behavior of critical signals in a microcontroller-based camera and storage system. The CS, SCK, MOSI, and MISO pins all exhibited expected activity aligned with SPI protocol operations, confirming proper communication between the Arduino and the microSD card. Voltage measurements revealed current fluctuations during data writing, which were mitigated by adding a 100 μF capacitor to stabilize the power supply. While diode-based power filtering was attempted, it proved ineffective, highlighting an area for further learning. Overall, the system demonstrated reliable functionality, and the signal measurements validated that the circuit behaved as designed during image capture and storage.

```
// include liquid crytal library
#include <LiquidCrystal.h>
// include library for servo motor
#include <Servo.h>
// include library for ttl camera and software serial connection for camera
#include <Adafruit_VC0706.h>
#include <SoftwareSerial.h>
// include library for SD card
#include <SD.h>
// include library that houses functions for the IR remote, remove whats unneeded to free memory
#define DECODE_NEC
#undef DECODE_SONY
#undef DECODE_RC5
#undef DECODE_PANASONIC
#undef DECODE_JVC
#include <IRremote.h>


// ****************** LIBRARY DEFINITIONS ****************

//////////////////////////////////////////////////////
// LCD SCREEN ////////////////////////////////////////
//////////////////////////////////////////////////////
LiquidCrystal screen(A0, A1, A2, A3, A4, A5);

//////////////////////////////////////////////////////
// SERVO MOTOR ///////////////////////////////////////
//////////////////////////////////////////////////////
Servo myServo;
int originalPos = 10;

//////////////////////////////////////////////////////
// ADAFRUIT TTL CAMERA ///////////////////////////////
//////////////////////////////////////////////////////
SoftwareSerial cameraConnection(2, 3); // RX, TX
Adafruit_VC0706 camera = Adafruit_VC0706(&cameraConnection);
uint8_t cameraBuffer[32];
char imageFilename[13];

//////////////////////////////////////////////////////
// IR SENSOR & REMOTE ////////////////////////////////
//////////////////////////////////////////////////////
// Define the signal pin for the IR reciever as PWM pin 3
int irsensor = 6;
// Create instance of IRrecv named irsig (IR Signal) using data from pin 3 (irsensor)
IRrecv irsig(irsensor);
// variable uses to store the last decodedRawData
// leading u says that type is unsigned
// int says it is a integer value (not necessarily an int)
// the value defines the number of bits used to store the value (8, 16, 32, etc.)
// _t says that sizes are standard across all platforms
uint32_t last_decodedRawData = 0;

//////////////////////////////////////////////////////
// PROGMEM VARIABLES /////////////////////////////////
//////////////////////////////////////////////////////
const char msgMotionDetected[] PROGMEM = "Motion Detected!";
const char msgEnterID[] PROGMEM = "EnterID:";
const char msgAccessGranted[] PROGMEM = "Access Granted";
const char msgAccessDenied[] PROGMEM = "Access Denied";
const char msgLookingFor[] PROGMEM = "Looking For";
const char msgMotion[] PROGMEM = "Motion...";

char buffer[17];


// ***************** PROGRAM FUNCTIONS ****************

```

```
// Depending on what button is pressed will output its value in HEX and user value
// If user holds button down, remote will detect a same value as old value and
// will read it as a repeat
void translateIR(uint32_t code) {

  char label[10] = "";

  // Serial.print("Decoded IR: 0x");
  // Serial.println(code, HEX);

  // only ID number that will work
  if (code == 0xE916FF00) {
    strcpy(label, "Lea");
    screen.setCursor(0, 1);
    screen.print(label);
    delay(1000);
    accessGranted();
  }
  else {
    strcpy(label, "Unknown");
    screen.setCursor(0, 1);
    screen.print(label);
    delay(1000);
    accessDenied();
  }

  screen.clear();
}


void checkIR(){
  // checks to see if ir signal is present
  screen.clear();
  screen.setCursor(0, 0);
  strcpy_P(buffer, msgEnterID);
  screen.print(buffer);
  irsig.resume();

  while (!irsig.decode()){
    ;
  }
  if (irsig.decode()){
    uint32_t code = irsig.decodedIRData.decodedRawData;
    // .resume readys the IR sensor to read the next signal
    // irsig.resume();
    translateIR(code);
  }
  irsig.resume();
}

void openGate(){
  delay(10);
  myServo.write(120);
  delay(5000);
  myServo.write(originalPos);
  delay(200);
}

void motionDetected(){
  camera.setMotionDetect(false);
  screen.clear();
  screen.setCursor(0, 0);
  strcpy_P(buffer, msgMotionDetected);
  screen.print(buffer);
  delay(500);
  saveSnapshot(imageFilename);
  delay(100);
  checkIR();
  delay(100);
```

```
138     camera.setMotionDetect(true);
139     delay(50);
140   }
141
142   void accessGranted(){
143     screen.clear();
144     screen.setCursor(0, 0);
145     strcpy_P(buffer, msgAccessGranted);
146     screen.print(buffer);
147     openGate();
148     screen.clear();
149   }
150
151   void accessDenied(){
152     screen.clear();
153     screen.setCursor(0, 0);
154     strcpy_P(buffer, msgAccessDenied);
155     screen.print(buffer);
156     delay(1000);
157     screen.clear();
158   }
159
160   // Helper function to convert string to uppercase
161   char *strupr(char *s) {
162     char *p = s;
163     while (*p) {
164       *p = toupper(*p);
165       p++;
166     }
167     return s;
168   }
169
170   bool saveSnapshot(char* outFilename) {
171     Serial.println("Taking picture...");
172     if (!camera.takePicture()) {
173       // Serial.println("Camera failed to take picture!");
174       return false;
175     }
176
177     uint16_t jpglen = camera.frameLength();
178     // Serial.print("Image size: ");
179     // Serial.println(jpglen);
180
181     for (uint8_t i = 0; i < 100; i++) {
182       sprintf(outFilename, "IMG%03d.JPG", i);
183       strupr(outFilename);
184       if (!SD.exists(outFilename)) break;
185     }
186
187     // Serial.print("Saving as: ");
188     // Serial.println(outFilename);
189
190     delay(50);
191     File imgFile = SD.open(outFilename, FILE_WRITE);
192     if (!imgFile) {
193       // Serial.println("Failed to open file for writing.");
194       return false;
195     }
196
197     while (jpglen > 0) {
198       uint8_t bytesToRead = min(32, jpglen);
199       uint8_t *buffer = camera.readPicture(bytesToRead);
200       if (!buffer) {
201         // Serial.println("Camera readPicture returned NULL.");
202         break;
203       }
204
205       imgFile.write(buffer, bytesToRead);
206       jpglen -= bytesToRead;
```

```
    }

    imgFile.close();
    Serial.println("Image saved successfully.");
    return true;
  }

  // ****************** SETUP *****************************

  void setup(){

    // begin serial monitor
    Serial.begin(9600);

    // start the LCD screen
    screen.begin(16, 2);

    // attach servo to pin 5 and move to start position 0
    myServo.attach(5);
    myServo.write(originalPos);

    SD.begin(10);

    if (!SD.begin(10)) {
      Serial.println("SD init failed!");
      return;
    }

    // File testFile = SD.open("test.txt", FILE_WRITE);
    // if (testFile) {
    //   testFile.println("SD card is working!");
    //   testFile.close();
    //   Serial.println("Test file written.");
    // }
    // else {
    //   Serial.println("Failed to write test file.");
    // }

    // start camera
    cameraConnection.begin(38400);
    camera.setImageSize(VC0706_320x240);
    camera.setMotionDetect(true);

    // start ir sensor receiver
    irsig.enableIRIn(); // Start the receiver
  }

  // ****************** LOOP *****************************

  void loop(){
    // let user know looking for motion...
    screen.setCursor(0, 0);
    strcpy_P(buffer, msgLookingFor);
    screen.print(buffer);
    screen.setCursor(0, 1);
    strcpy_P(buffer, msgMotion);
    screen.print(buffer);

    if (camera.motionDetected()) {
      motionDetected();
    }
  }
```