Home Security System

Hiram Y. Miranda-Pomales[1]
hmiranda22@email.uagm.edu

Universidad Ana G. Méndez, Recinto Gurabo- Electrical Engineering Department
ELEN 442- 447 students

Abstract- This project introduces an affordable home security system harnessing the capabilities of the ESP32 microcontroller and ESP32-CAM. Priced at approximately $62.11, the system integrates with the Blynk platform, employing strategically placed door and window reed switches, a servo for door control, and an alarm system. The ESP32-CAM enhances surveillance with a 2-megapixel camera, seamlessly storing images on a microSD card. Remote monitoring and control are facilitated through the Blynk web console and phone app. Despite challenges in pin allocation and data storage, the project demonstrates resilience, providing a cost-effective solution for comprehensive home protection.

I. INTRODUCTION

The ESP32, a game-changing System on a Chip (SoC) microcontroller from Espressif, has redefined the landscape of affordable and power-efficient embedded systems. Priced between $6 to $10, its remarkable features, including Wi-Fi and Bluetooth capabilities, a dual-core processor, and exceptional power efficiency, position it as an ideal choice for various applications, especially in IoT and home automation.[3]

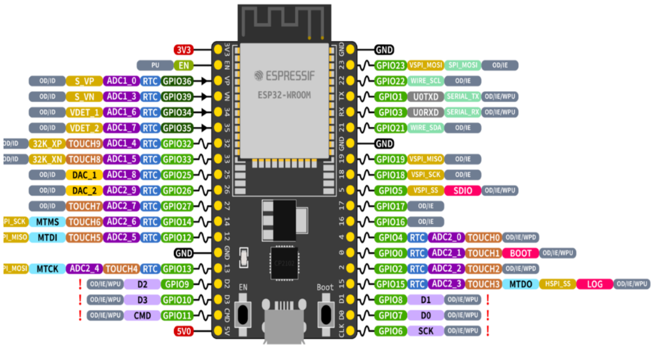

ESPRESSIF
ESP32-WROOM
3V3
PU EN
OD/ID S_VP ADC1_0 RTC GPIO36
OD/ID S_VN ADC1_3 RTC GPIO39
OD/ID VDET_1 ADC1_6 RTC GPIO34
OD/ID VDET_2 ADC1_7 RTC GPIO35
OD/ID 32K_XP TOUCH9 ADC1_4 RTC GPIO32
OD/ID 32K_XN TOUCH8 ADC1_5 RTC GPIO33
OD/ID DAC_1 ADC1_8 RTC GPIO25
OD/ID DAC_2 ADC2_9 RTC GPIO26
OD/ID TOUCH7 ADC2_7 RTC GPIO27
PI_SCK MTMS TOUCH6 ADC2_6 RTC GPIO14
PI_MISO MTDI TOUCH5 ADC2_5 RTC GPIO12
GND
PI_MOSI MTCK ADC2_4 TOUCH4 RTC GPIO13
! OD/IE/WPU D2 GPIO9
! OD/IE/WPU D3 GPIO10
! OD/IE/WPU CMD GPIO11
5V0
3V3 EN VP VN 34 35 32 33 25 26 27 14 12 GND 13 D2 D3 CMD 5V
GND 23 22 TX RX 21 GND 19 18 5 17 16 4 0 2 15 D1 D0 CLK
EN
Boot
GND
GPIO23 VSPI_MOSI SPI_MOSI OD/IE
GPIO22 WIRE_SCL OD/IE
GPIO1 U0TXD SERIAL_TX OD/IE/WPU
GPIO3 U0RXD SERIAL_RX OD/IE/WPU
GPIO21 WIRE_SDA OD/IE
GND
GPIO19 VSPI_MISO OD/IE
GPIO18 VSPI_SCK OD/IE
GPIO5 VSPI_SS SDIO OD/IE/WPU
GPIO17 OD/IE
GPIO16 OD/IE
GPIO4 RTC ADC2_0 TOUCH0 OD/IE/WPD
GPIO0 RTC ADC2_1 TOUCH1 BOOT OD/IE/WPU
GPIO2 RTC ADC2_2 TOUCH2 OD/IE/WPD
GPIO15 RTC ADC2_3 TOUCH3 MTDO HSPI_SS LOG OD/IE/WPU
GPIO8 D1 OD/IE/WPU !
GPIO7 D0 OD/IE/WPU !
GPIO6 SCK OD/IE/WPU !

Figure 1: ESP32 Pin Layout [2]

A notable variant, the ESP32-CAM, extends the ESP32's capabilities with a 2-megapixel OV2640 camera sensor. With a maximum resolution of 1600x1200 pixels, it becomes a valuable asset for surveillance and image capture. The ESP32-CAM also includes a microSD card slot for data logging and image storage.[1]

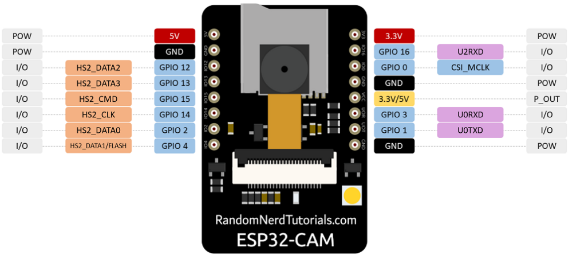


Figure 2: ESP32-CAM Pin Layout [1]

Building on this foundation, we introduce an advanced home security system leveraging the ESP32's capabilities. This system seamlessly integrates with IoT through the Blynk platform, incorporating strategically placed door and window reed switches. A servo for door control adds an extra layer of security, enabling swift intruder detection.

Upon detecting unauthorized entry, the system triggers immediate notifications, activates an alarm, and captures a snapshot stored on an SD card for reference. The system allows seamless control over the door lock and activation/deactivation of the entire security system.

II. Design Details

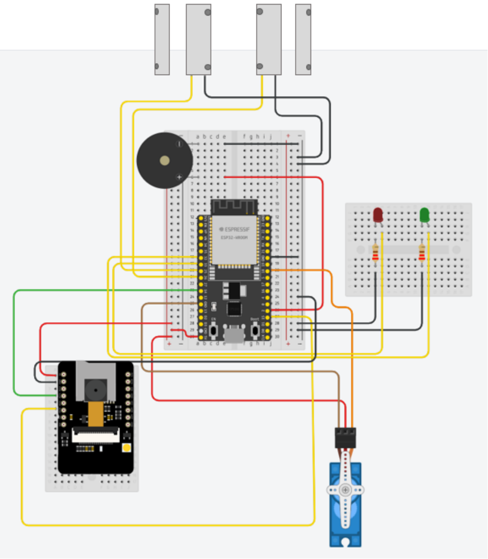
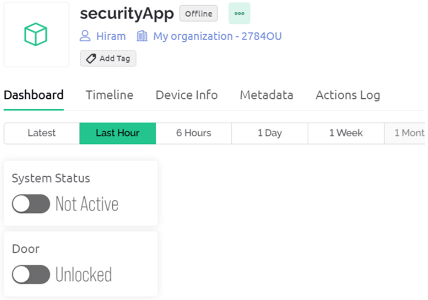


Figure 3: Circuit Design[1][2]

Our home security system is anchored by the versatile ESP32 microcontroller, functioning as the central intelligence hub for seamless IoT control and Wi-Fi integration. Strategically positioned magnetic reed switches on doors and windows, connected to designated pins, provide crucial input by detecting entry point states. Complementing this, a micro servo motor on pin 18 efficiently controls the door lock, while an active buzzer on pin 2 serves as the audible alarm, activating a distinctive sound upon sensor-triggered events.

Enhancing the system's capabilities is the ESP32-Cam, seamlessly integrated through UART pins, offering camera functionality that captures images every 10 seconds upon sensor activation. These images are securely stored on a Micro SD card, providing a reliable local solution for visual records of security events. Additionally, two LEDs connected to specific pins serve as visual indicators for the door lock status, with carefully selected resistors ensuring optimal performance while safeguarding the system's integrity. The entire system is monitored and controlled through the Blynk web console and Blynk phone app, allowing users to manage and oversee security features remotely.



securityApp Offline

Hiram My organization - 2784OU

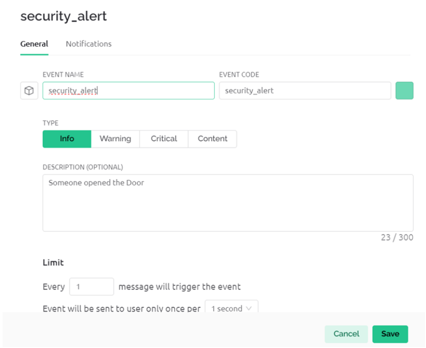
System Status

Not Active

Door

Unlocked

Figure 4: Web Console



security_alert

General | Notifications

EVENT NAME: security_alert

EVENT CODE: security_alert

TYPE: [✓ Info] | Warning | Critical | Content

DESCRIPTION (OPTIONAL): Someone opened the Door

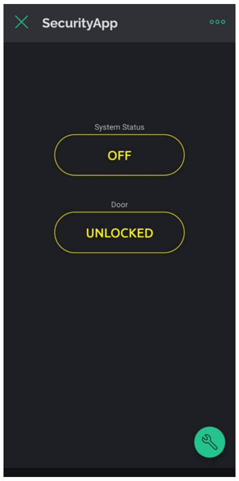
23 / 300

Limit

Every 1 message will trigger the event

Event will be sent to user only once per 1 second

Figure 5: Notification Event



SecurityApp

System Status: OFF

Door: UNLOCKED

Figure 6: Phone Interface

The ESP32, configured through the Arduino IDE, seamlessly integrates into our home security system. Key preferences and libraries, including "BlynkSimpleEsp32" for Blynk integration and "ESP32Servo" for servo motor control, enhance communication with other system components.

In the ESP32 code, Blynk integration enables IoT control for remote system management. The code orchestrates servo motor functions for door control, monitors magnetic reed switches for door and window conditions, and employs LED indicators for status feedback.

On the other hand, the ESP32-CAM code focuses on visual components, initializing the camera module for image capture. It sets up the MicroSD card for secure local storage, responding intelligently to commands to capture and store images. This interaction enhances the security system's capabilities, allowing image capture on user command or detection of security events.

III. Design Verification

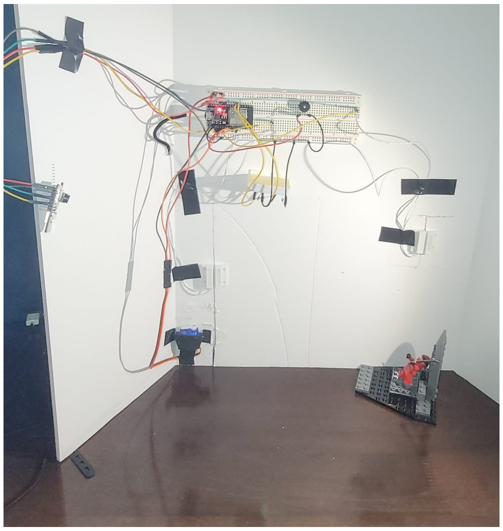
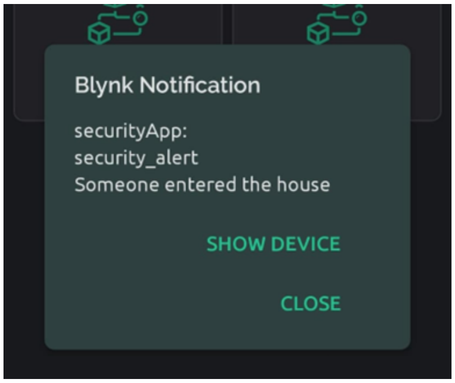


Figure 7: Circuit Implementation



Blynk Notification

securityApp:
security_alert
Someone entered the house

Figure 8: Phone Notification

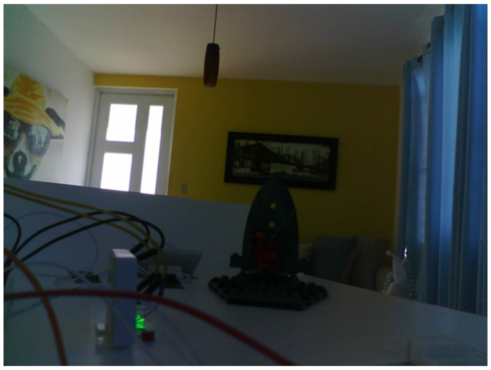


Figure 9: Micro SD pictures

Upon powering on the system, users gain control over the door lock state. The red LED indicates a closed door, while the green LED signifies an open door. Users can seamlessly activate or deactivate the security system, prompting the ESP32 to vigilantly monitor sensor states. If a door or window is detected as open, the ESP32 springs into action, activating the buzzer, dispatching a Blynk notification to the user's phone, and commanding the ESP32-CAM to capture images at 10-second intervals. As soon as the door/window closes, the alarm subsides, and image capture gracefully halts. For a more immersive experience, users can effortlessly retrieve the SD card from the ESP32-CAM to view captured photos on their computer. All of this security and user interaction comes at an estimated cost of approximately $62.11, factoring in the quantity and unit price for each component—a small investment for comprehensive home protection.

IV. CONCLUSION

In conclusion, this project represents a pioneering step towards the development of more advanced and secure IoT-based home security systems. Leveraging the capabilities of the ESP32 microcontroller and ESP32-CAM, the system seamlessly integrates with Blynk, providing remote control and monitoring through a user-friendly interface. The strategic placement of reed switches, servo motor for door control, and the incorporation of an alarm and camera enhance the overall security features. Despite encountering challenges in pin allocation and data storage solutions, the project demonstrates resilience by addressing these constraints and opting for a reliable SD card-based photo storage system. With a cost-effective approach and a comprehensive set of features, this project lays the foundation for future innovations in IoT security, marking the beginning of a promising journey towards more robust and sophisticated home automation solutions.

V. REFERENCES

[1] RandomNerdTutorials, “ESP32-cam ai-thinker Pinout Guide: Gpios usage explained,” Random Nerd Tutorials, https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/ (accessed Dec. 9, 2023).

[2] ESPRESSIF, “ESP32-DEVKITC V4 getting started guide,” ESPRESSIF, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html (accessed Dec. 9, 2023).

[3] RandomNerdTutorials, “Getting started with the ESP32 Development Board,” Random Nerd Tutorials, https://randomnerdtutorials.com/getting-started-with-esp32/ (accessed Dec. 4, 2023).

[4] RandomNerdTutorials, “ESP32-cam save picture in Firebase Storage,” Random Nerd Tutorials, https://randomnerdtutorials.com/esp32-cam-save-picture-firebase-storage/ (accessed Dec. 4, 2023).

[5] DroneBot Workshop, “Using servo motors with the ESP32,” DroneBot Workshop, https://dronebotworkshop.com/esp32-servo/ (accessed Dec. 4, 2023).

[6] DroneBot Workshop, “Using the ESP32-cam microsd card,” DroneBot Workshop, https://dronebotworkshop.com/esp32-cam-microsd/ (accessed Dec. 4, 2023).

[7] esp32io, “ESP32 - door sensor,” ESP32 Tutorial, https://esp32io.com/tutorials/esp32-door-sensor (accessed Dec. 4, 2023).

VI. APPENDIX A - Text programs

a. ESP32 Code:

```
    // Blynk template information
#define BLYNK_TEMPLATE_ID "TMPL2dY2iBFzI"
#define BLYNK_TEMPLATE_NAME "HomeSecurity"
#define BLYNK_AUTH_TOKEN "zGGEwmcqyS7lS69zQgcrAdLC3TbhVuM6"

// Include necessary libraries
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32Servo.h>

// Pin definitions
#define RXD1 14
#define TXD1 15
#define DOOR_SENSOR  25
#define buzzer 2
#define WINDOW_SENSOR 26

// Servo setup
Servo myservo;
int servoPin = 18;
int ledPinClose = 32;
int ledPinOpen = 33;
int ADC_Max = 4096;

int val;
BlynkTimer timer;

// Blynk authentication and WiFi credentials
char auth[] = "zGGEwmcqyS7lS69zQgcrAdLC3TbhVuM6"; // Enter the authentication code sent by Blynk to your Email
//char ssid[] = "LIB-9225380"; // Enter your WIFI SSID
//char pass[] = "wnZwyg8n4ynd"; // Enter your WIFI Password
//char ssid[] = "MyASUS2024"; // Enter your WIFI SSID
//char pass[] = "asusisgod"; // Enter your WIFI Password
char ssid[] = "Hiram"; // Enter your WIFI SSID
char pass[] = "12345678"; // Enter your WIFI Password
int flag = 0;
int systemState = 0;  // Variable to store system state

// Blynk app write event for V1
BLYNK_WRITE(V1)
{
  int state = param.asInt();
  Serial.println(state);

  // Assuming you want to turn the system on when the state is 1 and off when it's 0
  if (state == 1) {
    systemState = 1;
    Serial.println("System is ON");
  } else {
    systemState = 0;
    Serial.println("System is OFF");
  }
}

// Blynk app write event for V2
BLYNK_WRITE(V2) {
  int val = param.asInt();

  if (val == 1023) {
     // Increase the output range to make the servo move even more
    int mappedValue = map(val, 0, ADC_Max, 0, 540);
    myservo.write(mappedValue);
    digitalWrite(ledPinOpen, HIGH);
    digitalWrite(ledPinClose, LOW);
    delay(200);
  } else {
     // Increase the output range to make the servo move even more
    digitalWrite(ledPinOpen, LOW);
    digitalWrite(ledPinClose, HIGH);
    int value = map(0, 0, ADC_Max, 0, 540);
    myservo.write(value);
    delay(200);
  }
}

// Function to notify when the door or
window is opened
void notifyOnButtonPress()
{
            int isButtonPressed =
digitalRead(DOOR_SENSOR);
            int isButtonpressed1 =
digitalRead(WINDOW_SENSOR);

       if ((isButtonPressed == 1 ||
isButtonpressed1 == 1) && flag == 0 &&
systemState == 1) {
       Serial.println("Someone entered the
house");
     Serial1.write('C'); // You can use any
character as a command
    delay(1000);
            Blynk.logEvent("security_alert",
"Someone entered the house");
    digitalWrite(buzzer, HIGH);
    flag = 1;
  } else {
    flag = 0;
          Serial.println("Door and Window
Closed");
    Serial1.print(0);
    digitalWrite(buzzer, LOW);
  }
}

// Setup function
void setup()
{
  Serial.begin(115200);
  Serial1.begin(9600, SERIAL_8N1, RXD1,
TXD1);
  Blynk.begin(auth, ssid, pass);
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  myservo.setPeriodHertz(50);
  myservo.attach(servoPin, 500, 2400);
                pinMode(DOOR_SENSOR,
INPUT_PULLUP);
            pinMode(WINDOW_SENSOR,
INPUT_PULLUP);
  pinMode(ledPinClose, OUTPUT);
  pinMode(ledPinOpen, OUTPUT);
  pinMode(buzzer, OUTPUT);
                  timer.setInterval(5000L,
notifyOnButtonPress);
}

// Loop function
void loop()
{
  Blynk.run();
  timer.run();
}
```

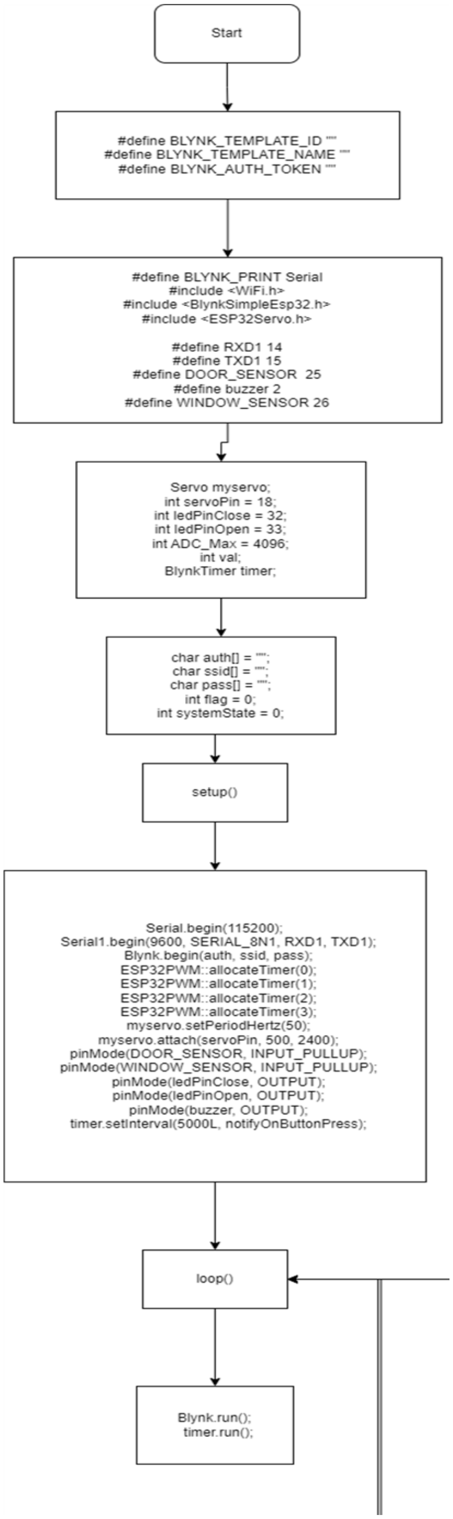
b. ESP32-CAM Code:

```
// Camera libraries
#include "esp_camera.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"

// MicroSD Libraries
#include "FS.h"
#include "SD_MMC.h"
#define RXD1 14
#define TXD1 13

// Pin definitions 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

// Counter for picture number
unsigned int pictureCount = 0;

// Delay time in milliseconds
unsigned int delayTime = 10000;

void configESPCamera() {
  // Configure Camera parameters

  // Object to store the camera configuration
parameters
  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; // Choices are
YUV422, GRAYSCALE, RGB565, JPEG

  // Select lower framesize if the camera
doesn't support PSRAM
  if (psramFound()) {
    config.frame_size =
FRAMESIZE_UXGA; // FRAMESIZE_ +
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXG
A
    config.jpeg_quality = 10; // 10-63 lower
number means higher quality
    config.fb_count = 2;
  } else {
    config.frame_size =
FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // Initialize the Camera
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error
0x%x", err);
    return;
  }

  // Camera quality adjustments
  sensor_t *s = esp_camera_sensor_get();

  // BRIGHTNESS (-2 to 2)
  s->set_brightness(s, 0);
  // CONTRAST (-2 to 2)
  s->set_contrast(s, 0);
  // SATURATION (-2 to 2)
  s->set_saturation(s, 0);
  // SPECIAL EFFECTS (0 - No Effect, 1 -
Negative, 2 - Grayscale, 3 - Red Tint, 4 -
Green Tint, 5 - Blue Tint, 6 - Sepia)
  s->set_special_effect(s, 0);
  // WHITE BALANCE (0 = Disable, 1 =
Enable)
  s->set_whitebal(s, 1);
  // AWB GAIN (0 = Disable, 1 = Enable)
  s->set_awb_gain(s, 1);
  // WB MODES (0 - Auto, 1 - Sunny, 2 - Cloudy, 3 - Office, 4 - Home)
  s->set_wb_mode(s, 0);
  // EXPOSURE CONTROLS (0 = Disable, 1 = Enable)
  s->set_exposure_ctrl(s, 1);
  // AEC2 (0 = Disable, 1 = Enable)
  s->set_aec2(s, 0);
  // AE LEVELS (-2 to 2)
  s->set_ae_level(s, 0);
  // AEC VALUES (0 to 1200)
  s->set_aec_value(s, 300);
  // GAIN CONTROLS (0 = Disable, 1 = Enable)
  s->set_gain_ctrl(s, 1);
  // AGC GAIN (0 to 30)
  s->set_agc_gain(s, 0);
  // GAIN CEILING (0 to 6)
  s->set_gainceiling(s, (gainceiling_t)0);
  // BPC (0 = Disable, 1 = Enable)
  s->set_bpc(s, 0);
  // WPC (0 = Disable, 1 = Enable)
  s->set_wpc(s, 1);
  // RAW GMA (0 = Disable, 1 = Enable)
  s->set_raw_gma(s, 1);
  // LENC (0 = Disable, 1 = Enable)
  s->set_lenc(s, 1);
  // HORIZ MIRROR (0 = Disable, 1 = Enable)
  s->set_hmirror(s, 0);
  // VERT FLIP (0 = Disable, 1 = Enable)
  s->set_vflip(s, 0);
  // DCW (0 = Disable, 1 = Enable)
  s->set_dcw(s, 1);
  // COLOR BAR PATTERN (0 = Disable, 1 = Enable)
  s->set_colorbar(s, 0);
}

void initMicroSDCard() {
  // Start the MicroSD card

  Serial.println("Mounting MicroSD Card");
  if (!SD_MMC.begin()) {
    Serial.println("MicroSD Card Mount Failed");
    return;
  }
  uint8_t cardType = SD_MMC.cardType();
  if (cardType == CARD_NONE) {
    Serial.println("No MicroSD Card found");
    return;
  }
}

void takeNewPhoto(String path) {
  // Take Picture with Camera

  // Setup frame buffer
  camera_fb_t *fb = esp_camera_fb_get();

  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }

  // Save picture to MicroSD card
  fs::FS &fs = SD_MMC;
  File file = fs.open(path.c_str(), FILE_WRITE);
  if (!file) {
    Serial.println("Failed to open file in write mode");
  } else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
  }
  // Close the file
  file.close();

  // Return the frame buffer back to the driver for reuse
  esp_camera_fb_return(fb);
}

void setup() {
  // Disable brownout detector

WRITE_PERI_REG(RTC_CNTL_BROWN
_OUT_REG, 0);

  // Start Serial Monitor
  Serial.begin(115200);
   Serial1.begin(9600, SERIAL_8N1, RXD1,
TXD1);

  // Initialize the camera
      Serial.print("Initializing the camera
module...");
  configESPCamera();
  Serial.println("Camera OK!");

  // Initialize the MicroSD
   Serial.print("Initializing the MicroSD card
module... ");
  initMicroSDCard();

  Serial.print("Delay Time = ");
  Serial.print(delayTime);
  Serial.println(" ms");
}

void loop() {
  // Path where a new image will be saved on
the MicroSD card
  if (Serial1.available() > 0) {
    char command = Serial1.read();

    // Check the received command
    if (command == 'C') {
      // Trigger the camera to take a picture
                  String path = "/image" +
String(pictureCount) + ".jpg";
      takeNewPhoto(path);
      pictureCount++;
    }
  }
}
```

VII. APPENDIX B - Flowcharts



```
#define BLYNK_TEMPLATE_ID ""
#define BLYNK_TEMPLATE_NAME ""
#define BLYNK_AUTH_TOKEN ""
```

```
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32Servo.h>

#define RXD1 14
#define TXD1 15
#define DOOR_SENSOR  25
#define buzzer 2
#define WINDOW_SENSOR 26
```

```
Servo myservo;
int servoPin = 18;
int ledPinClose = 32;
int ledPinOpen = 33;
int ADC_Max = 4096;
int val;
BlynkTimer timer;
```

```
char auth[] = "";
char ssid[] = "";
char pass[] = "";
int flag = 0;
int systemState = 0;
```

```
setup()
```

```
Serial.begin(115200);
Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);
Blynk.begin(auth, ssid, pass);
ESP32PWM::allocateTimer(0);
ESP32PWM::allocateTimer(1);
ESP32PWM::allocateTimer(2);
ESP32PWM::allocateTimer(3);
myservo.setPeriodHertz(50);
myservo.attach(servoPin, 500, 2400);
pinMode(DOOR_SENSOR, INPUT_PULLUP);
pinMode(WINDOW_SENSOR, INPUT_PULLUP);
pinMode(ledPinClose, OUTPUT);
pinMode(ledPinOpen, OUTPUT);
pinMode(buzzer, OUTPUT);
timer.setInterval(5000L, notifyOnButtonPress);
```

```
loop()
```

```
Blynk.run();
 timer.run();
```

Figure 10: ESP32 Flowchart part 1

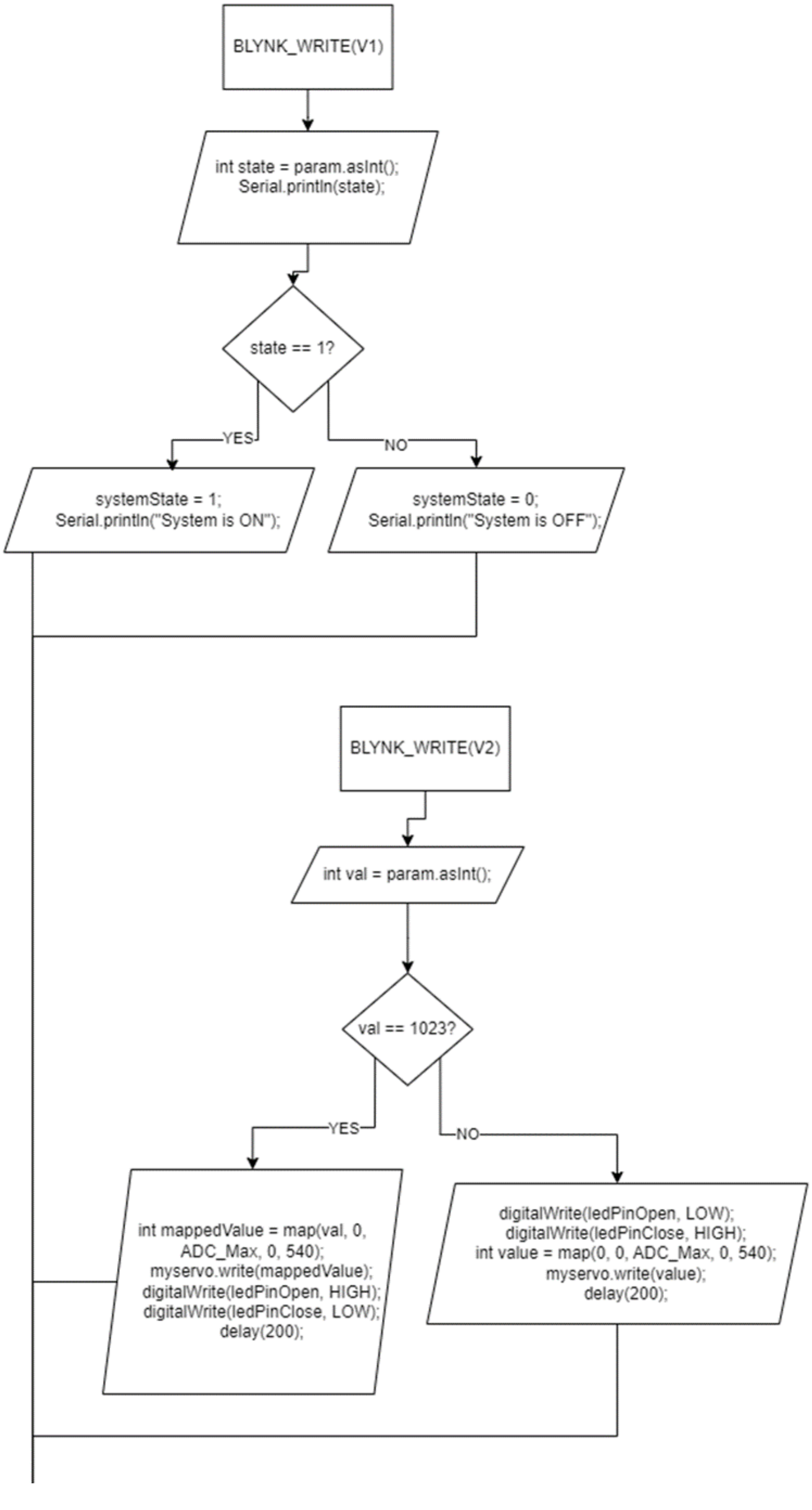
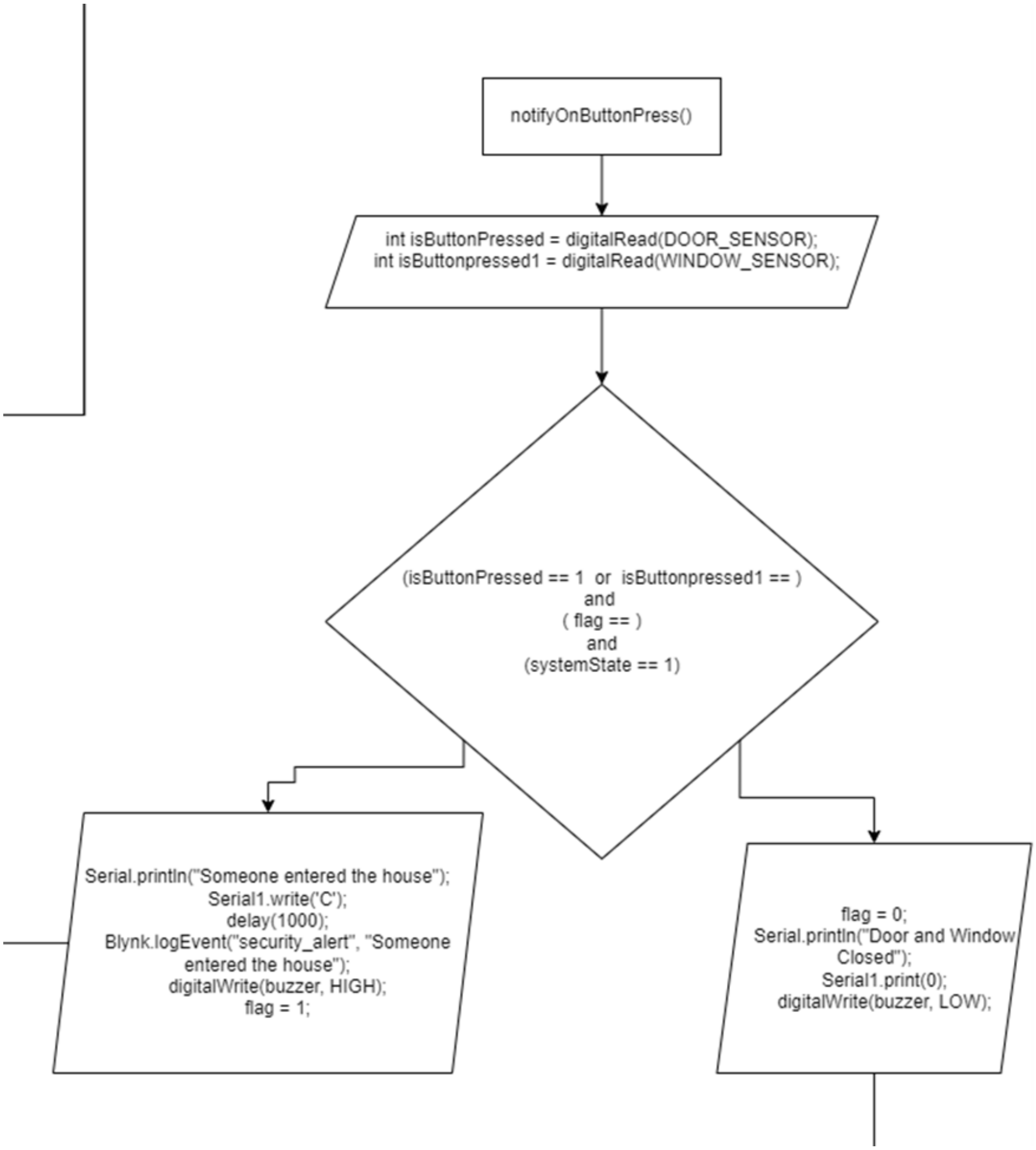
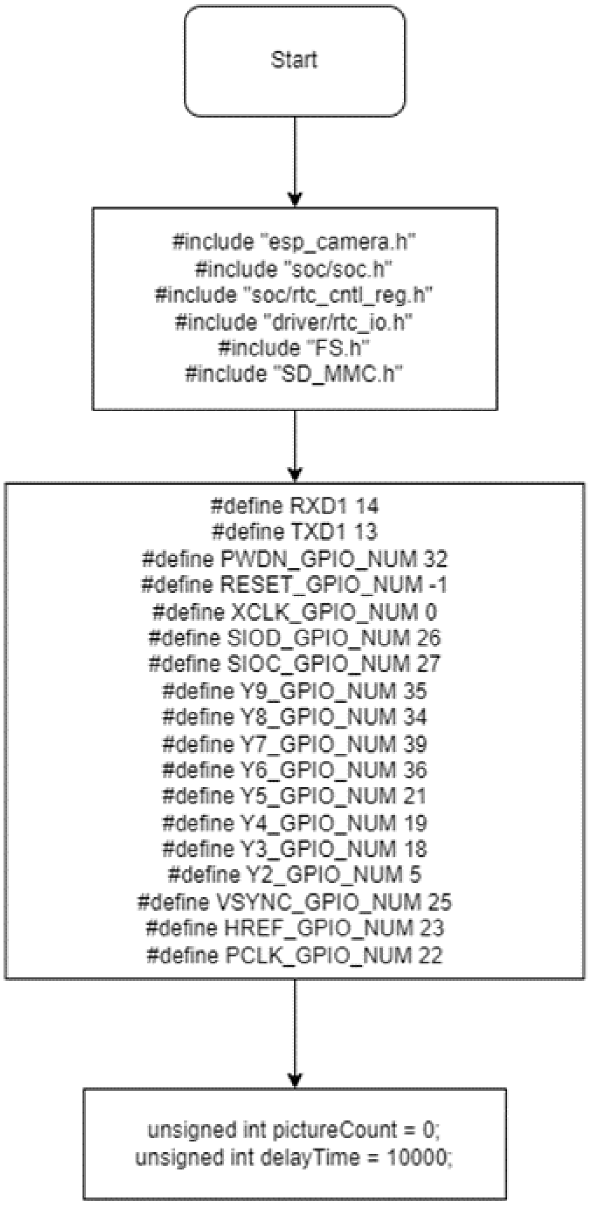

BLYNK_WRITE(V1)
int state = param.asInt();
Serial.println(state);
state == 1?
YES
NO
systemState = 1;
Serial.println("System is ON");
systemState = 0;
Serial.println("System is OFF");
BLYNK_WRITE(V2)
int val = param.asInt();
val == 1023?
YES
NO
int mappedValue = map(val, 0, ADC_Max, 0, 540);
myservo.write(mappedValue);
digitalWrite(ledPinOpen, HIGH);
digitalWrite(ledPinClose, LOW);
delay(200);
digitalWrite(ledPinOpen, LOW);
digitalWrite(ledPinClose, HIGH);
int value = map(0, 0, ADC_Max, 0, 540);
myservo.write(value);
delay(200);

Figure 11: ESP32 Flowchart part 2



```
notifyOnButtonPress()

int isButtonPressed = digitalRead(DOOR_SENSOR);
int isButtonpressed1 = digitalRead(WINDOW_SENSOR);

(isButtonPressed == 1  or  isButtonpressed1 == )
and
( flag == )
and
(systemState == 1)

Serial.println("Someone entered the house");
Serial1.write('C');
delay(1000);
Blynk.logEvent("security_alert", "Someone
entered the house");
digitalWrite(buzzer, HIGH);
flag = 1;

flag = 0;
Serial.println("Door and Window
Closed");
Serial1.print(0);
digitalWrite(buzzer, LOW);
```

Figure 12: ESP32 Flowchart part 3

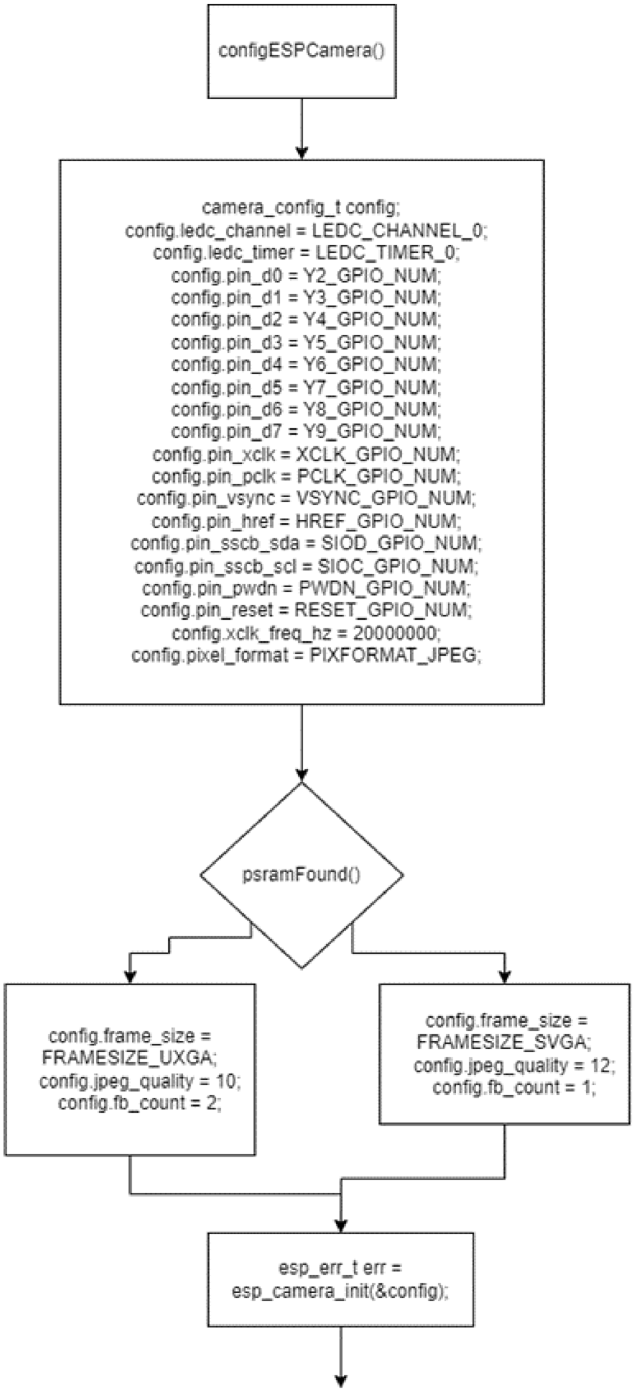


```
#include "esp_camera.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"
#include "FS.h"
#include "SD_MMC.h"
```

```
#define RXD1 14
#define TXD1 13
#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
```

```
unsigned int pictureCount = 0;
unsigned int delayTime = 10000;
```

Figure 13: ESP32-CAM Flowchart part 1



```
configESPCamera()
```

```
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;
```

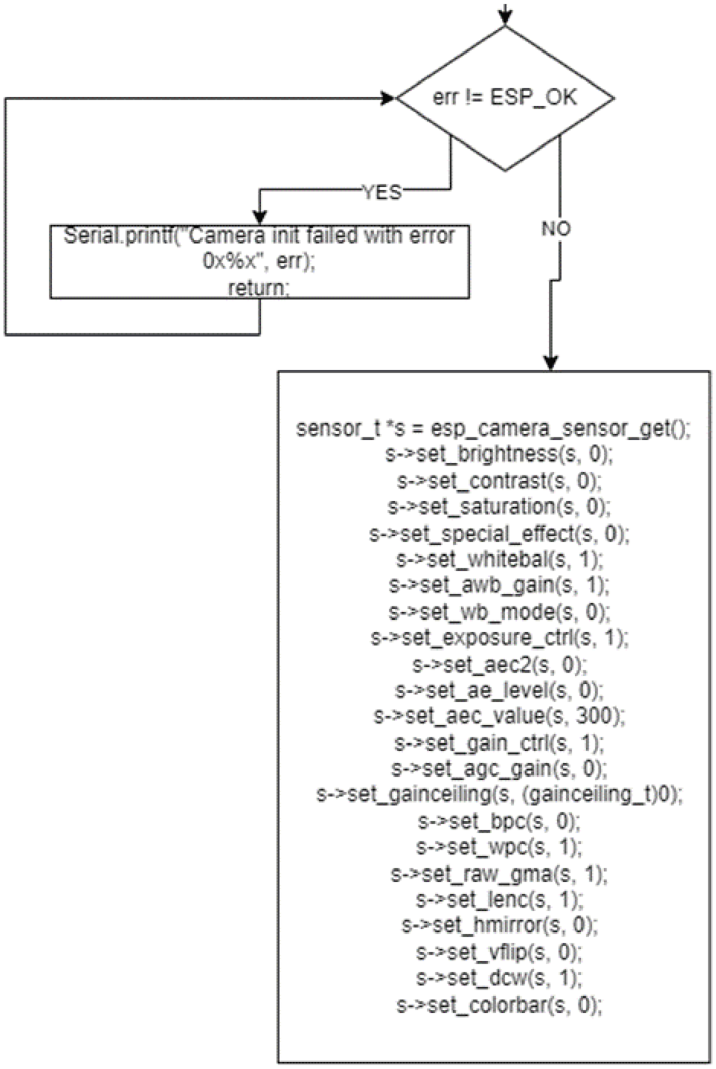
```
psramFound()
```

```
config.frame_size =
FRAMESIZE_UXGA;
config.jpeg_quality = 10;
config.fb_count = 2;
```

```
config.frame_size =
FRAMESIZE_SVGA;
config.jpeg_quality = 12;
config.fb_count = 1;
```

```
esp_err_t err =
esp_camera_init(&config);
```

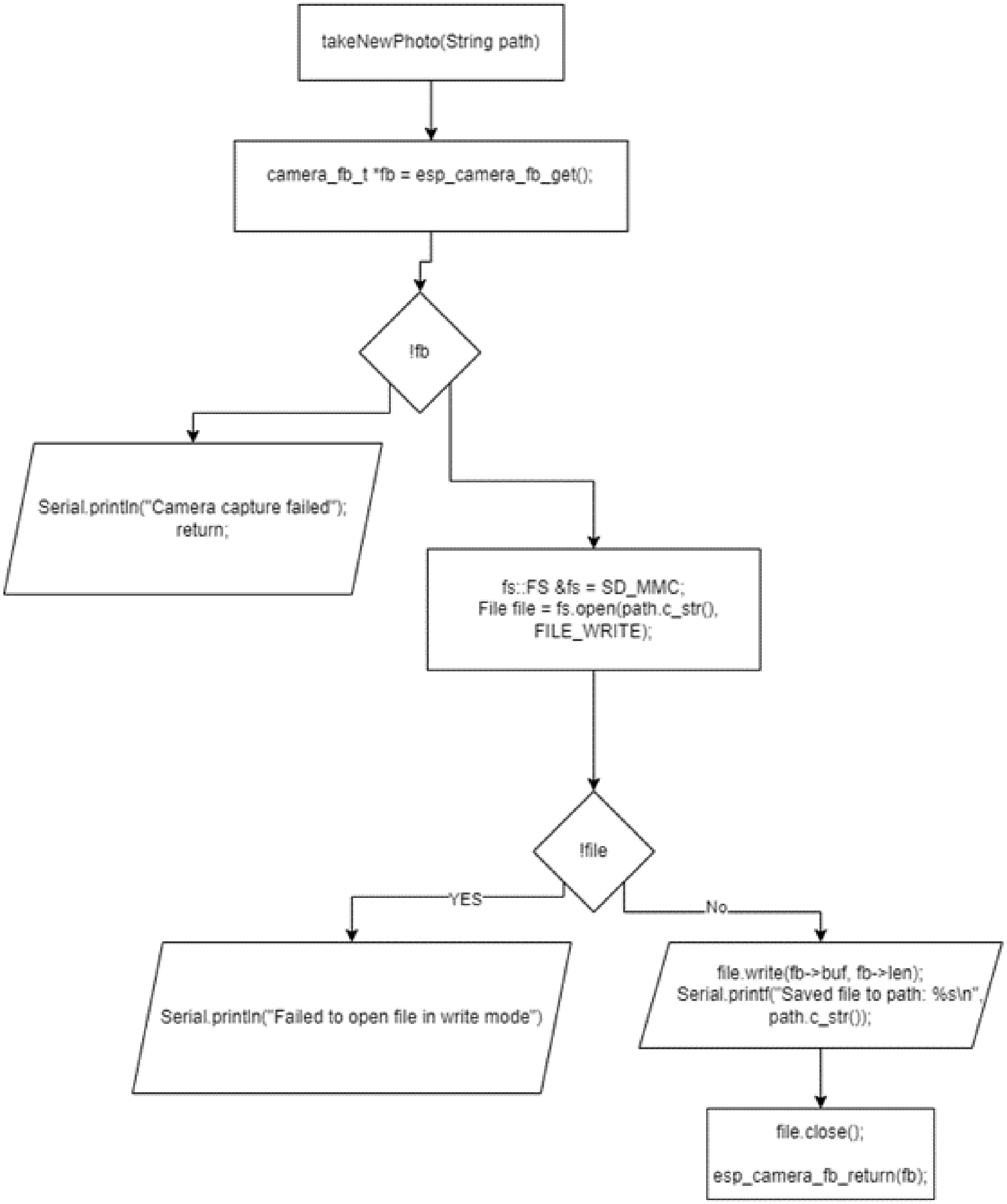
Figure 14: ESP32-CAM Flowchart part 2



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

```
sensor_t *s = esp_camera_sensor_get();
s->set_brightness(s, 0);
s->set_contrast(s, 0);
s->set_saturation(s, 0);
s->set_special_effect(s, 0);
s->set_whitebal(s, 1);
s->set_awb_gain(s, 1);
s->set_wb_mode(s, 0);
s->set_exposure_ctrl(s, 1);
s->set_aec2(s, 0);
s->set_ae_level(s, 0);
s->set_aec_value(s, 300);
s->set_gain_ctrl(s, 1);
s->set_agc_gain(s, 0);
s->set_gainceiling(s, (gainceiling_t)0);
s->set_bpc(s, 0);
s->set_wpc(s, 1);
s->set_raw_gma(s, 1);
s->set_lenc(s, 1);
s->set_hmirror(s, 0);
s->set_vflip(s, 0);
s->set_dcw(s, 1);
s->set_colorbar(s, 0);
```

Figure 15: ESP32-CAM Flowchart part 3



```
takeNewPhoto(String path)

camera_fb_t *fb = esp_camera_fb_get();

Serial.println("Camera capture failed");
return;

fs::FS &fs = SD_MMC;
File file = fs.open(path.c_str(),
FILE_WRITE);

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

file.write(fb->buf, fb->len);
Serial.printf("Saved file to path: %s\n",
path.c_str());

file.close();

esp_camera_fb_return(fb);
```

Figure 16: ESP32-CAM Flowchart part 4

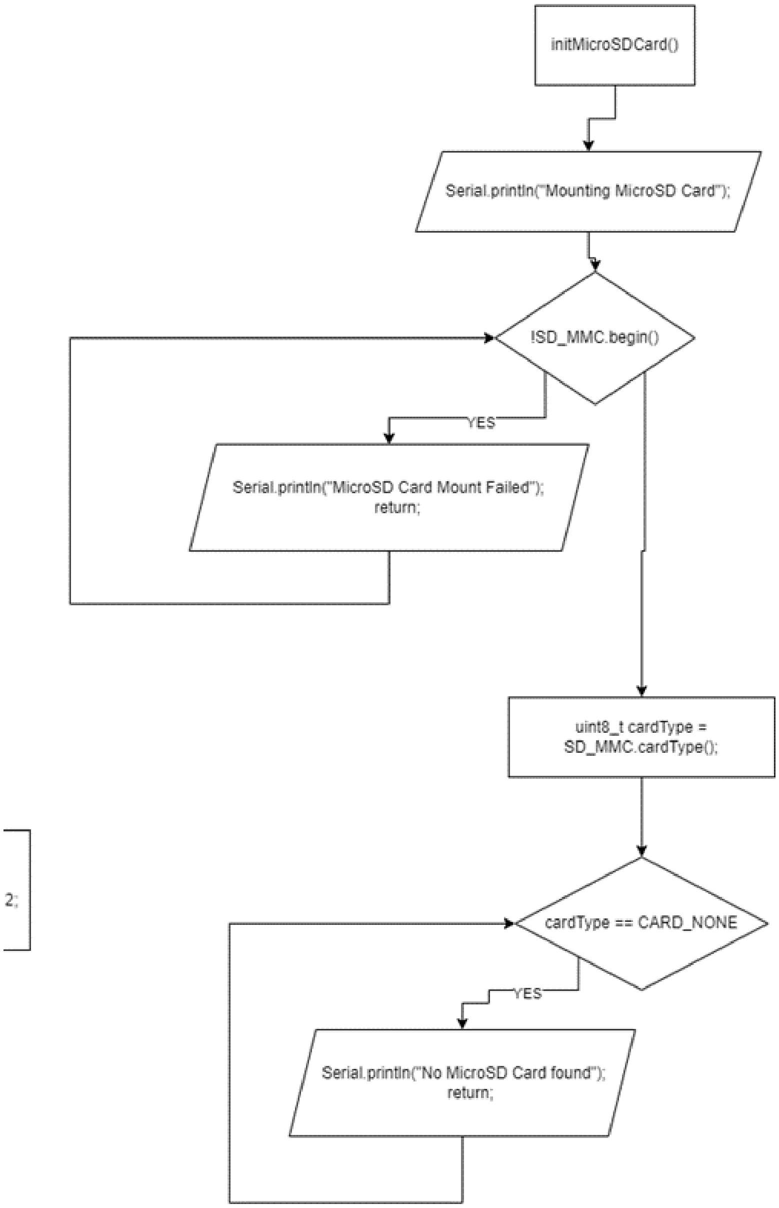
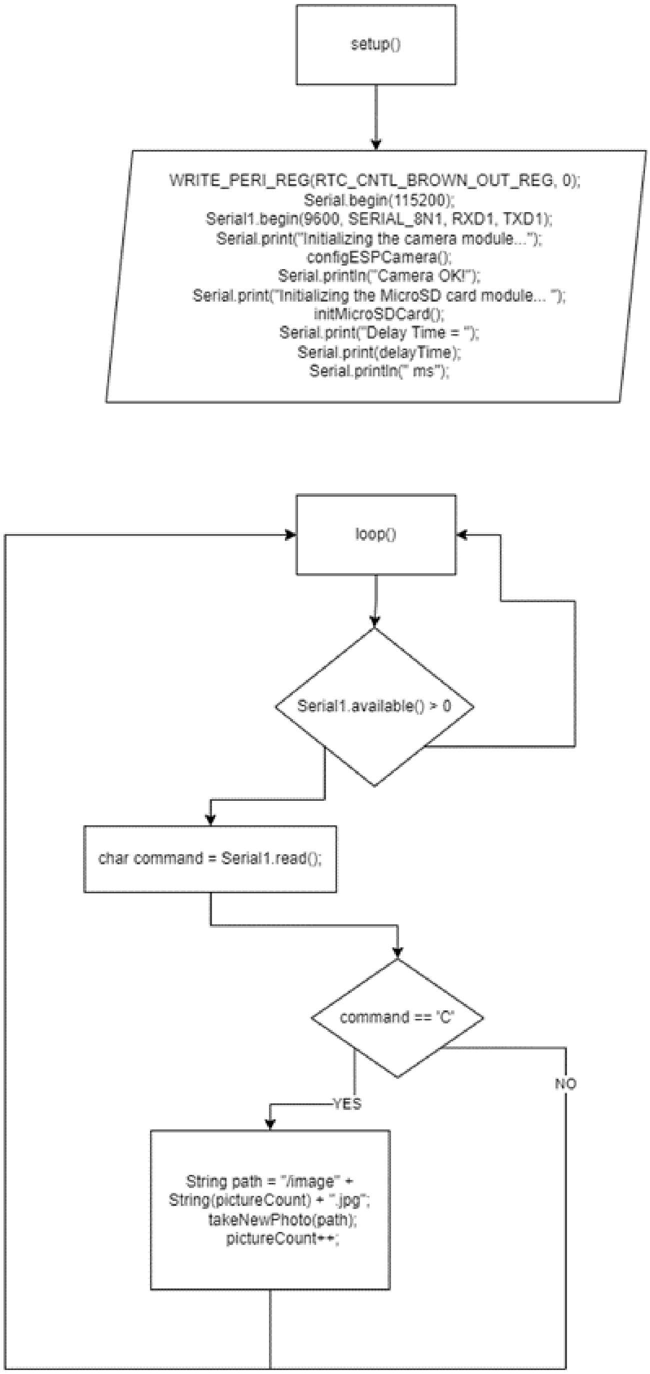

initMicroSDCard()
Serial.println("Mounting MicroSD Card");
!SD_MMC.begin()
YES
Serial.println("MicroSD Card Mount Failed");
return;
uint8_t cardType =
SD_MMC.cardType();
2;
cardType == CARD_NONE
YES
Serial.println("No MicroSD Card found");
return;

Figure 17: ESP32-CAM Flowchart part 5



```
WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
Serial.begin(115200);
Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);
Serial.print("Initializing the camera module...");
configESPCamera();
Serial.println("Camera OK!");
Serial.print("Initializing the MicroSD card module... ");
initMicroSDCard();
Serial.print("Delay Time = ");
Serial.print(delayTime);
Serial.println(" ms");
```

```
char command = Serial1.read();
```

```
String path = "/image" +
String(pictureCount) + ".jpg";
takeNewPhoto(path);
pictureCount++;
```

Figure 18: ESP32-CAM Flowchart part 6