Camera Smart Band Project Report

Submitted for: Electronic Design Workshop (ECECC404)

Submitted by :
ADVAY MAKHIJA : 2023UEC2567
MAHIB : 2023UEC2569
BHUWAN CHANDRA PANDEY : 2023UEC2576

17th May, 2025

CERTIFICATE

This is to certify that Mr. Advay Makhija (2023UEC2567), Mr. MAHIB (2023UEC2569) and Mr. Bhuwan Chandra Pandey (2023UEC2576) of B. Tech 4th semester, 2nd Year of branch Electronics And Communication Engineering have satisfactorily completed Electronic Design Workshop (ECECC404) Project during the academic year 2024-2025.

Signature of HOD

Signature of Teacher

Signature of Professor

Acknowledgement

We would like to express our sincere gratitude to everyone who played a pivotal role in the successful completion of this project. This endeavor has been a valuable learning experience, made possible through consistent guidance, support, and collaboration.

First and foremost, we are profoundly grateful to our mentor, Prof. Dhananjay V. Gadre, for his exceptional mentorship and insightful feedback. His expertise, encouragement, and constant support were instrumental in shaping this project and guiding us through every stage of development.

We are equally thankful to our faculty and peers, whose constructive inputs and collaborative spirit greatly enriched our journey.

We would also like to extend our heartfelt appreciation to our parents for their unwavering encouragement, patience, and belief in us. Their constant motivation and emotional support formed the backbone of our efforts.

Lastly, we thank everyone—directly or indirectly—who contributed their time, knowledge, or support in helping us bring this idea to fruition.

Contents

Chapter 1

The Idea

The original concept for this project was to design and implement a QR code scanner. However, as the project progressed, additional features were envisioned and integrated, including the ability to capture photos and toggle a flashlight on and off.

At the heart of our project lies the ESP32-CAM module, a compact yet powerful microcontroller with an integrated camera. This module is responsible for capturing frames, processing them to detect QR codes, and relaying data to a server if a QR code is detected.

Functional Overview

- **QR Detection:** The ESP32-CAM runs a detection algorithm which continuously analyzes incoming frames. If a QR code is found, it forwards the relevant frame to a server for decoding.
- **Server-Side Decoding:** The server, running custom code, takes the frame and attempts to decode the QR code using standard libraries. Once decoded, the extracted data is sent to the user.
- **Telegram Bot Integration:** The server communicates with a Telegram bot that sends the decoded data to the user's Telegram account. This makes the setup highly portable and easy to access.
- **Modes of Operation:** The system can be toggled between different modes — photo capture mode and QR scan mode. In the photo mode, users can capture and receive photos directly via Telegram.

This project showcases how low-cost, off-the-shelf microcontrollers can be leveraged to implement real-time computer vision tasks with additional cloud-based functionalities.

Chapter 2

Hardware Implementation

To power and support the ESP32-CAM, we designed a complete power delivery system comprising three main modules: an adapter module, a charging module, and a booster module. Each module is crucial for ensuring stability, portability, and efficient performance.

2.1 Adapter Module

The adapter is a basic AC to DC power supply circuit that converts 220V AC to a regulated 5V DC.

- A step-down transformer converts 220V AC to 12V AC.
- A Full-Wave Bridge Rectifier (FWBR) converts this to DC.
- An LM7805 voltage regulator regulates it to 5V at 1A.
- Initial filtering used a 0.33µF capacitor, but was later upgraded to 2200µF due to ripple issues and voltage dips under load.

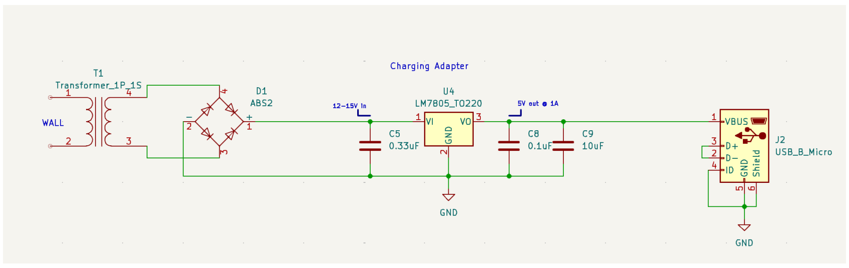

Figure 2.1: Schematic of the Adapter Module

2.2 Charging Module

The charging module allows the system to operate wirelessly with a LiPo battery.

- A USB-C connector feeds power into the module.
- The MCP738311/2 LiPo charging IC handles battery charging at a max of 500mA.

- PMOS switching circuitry automatically disables battery discharge during external power input.
- Once unplugged, battery resumes supplying power.

2.3 Booster Module

To drive the ESP32-CAM at stable voltage levels, we use the MT3608 booster module.

- It boosts the 3.7V LiPo battery to 5.1V with max output current of 1.5A.
- The battery used is rated at 1000mAh with a discharge capacity of 2C.
- This makes the module portable and supports moderate current draw without brownouts.

The system is designed to switch seamlessly between charging and discharging modes, ensuring stable operation throughout usage cycles.

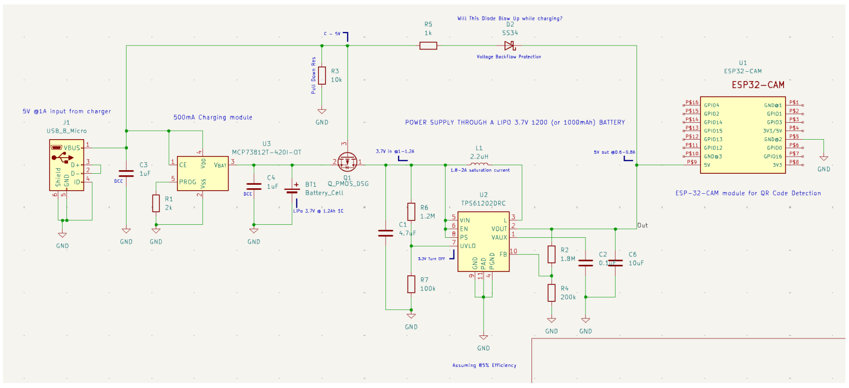

Figure 2.2: Schematic of the Multiplexed Charger and Booster Module

2.4 ESP32 Cam Scanner Modue

This is the Last module which uses the ESP32-CAM to Detect QR codes and take photos and send it to the user!

- Camera Module - **OV2640**
- Supply Pin Used was 5V one, as evident from the above module.
- The ESP Send the image and QR data to the user via a telegram bot for ease of access and minimal setup!

Chapter 3

Challenges and Issues Faced

3.1 Adapter Module Power Constraints

We aimed to extract 5W of power at 5V (1A) using the LM7805 regulator. However, the regulator's inefficiency meant we needed 10W input power. At 7V input, this translated to a current draw of 1.42A, which practically shorted the Input of the 7805. The input resistance appeared too low, leading to significant ripple and voltage drops.

Solution: We increased the input capacitor from 0.33µF to a much larger 1100µF, which effectively dampened voltage ripple and ensured smoother regulation.

3.2 Component Unavailability

Despite extensive searching across local markets in Delhi, we encountered repeated unavailability of key components:

- **TPS61202 IC** (intended for boosting): Not found. Replaced with an MT3608 breakout.
- **MCP73831/2 Charger IC**: Eventually ordered online after many failed attempts to find it in stores.

3.3 ESP32 Flashing Issues

The ESP32 initially refused to flash firmware. After investigation, we realized the user needed permissions to access USB devices. Adding the user to the correct system group ('dialout' or similar on Linux) resolved the problem.

3.4 QR Code Decoding Errors

While the ESP32 detected QR codes reliably, the actual decoding sometimes failed due to ECC (Error Correction Code) issues. Despite tweaking buffer sizes and memory allocations (using PSRAM), results were inconsistent.

Workaround: We moved decoding to the server. This improved performance and also allowed decoding of stylized QR codes — a feature that wouldn't be feasible on the ESP itself.

3.5 Booster modules not working as intended

All the booster modules we bought were appearently not working as intended, they would output a 5V open circuit voltage, but as soon as a load was connected the voltage would sag to as low as 2V and the maximum current under load was limited far below the statistics we saw in the datasheets!

The Real Problem: The battery we bought was not the right one for our use case. It was labelled 3.7V at 1000mAhm, but the shopkeeper told us it was a 2C battery. Maybe we damaged it while testing, or the shopkeeper said something he shouldn't have, because the battery was certainly not 2C and the voltage of the battery would sag to under 3V under load when trying ot draw anywhere close to 1A current. (We needed even higher current than 1A from the battery to power the ESP effectively!)

Workaround: We bought a new battery, and tested the circuit again, BOOM! this time it worked! Absolutely fabulously, without a hitch! (new battery was 3.7V rated 2000mAh @ 1C).

3.6 Mains Voltage isolation

As shown in the schematic, the original intended solution to switch between the charging and the ESP circuit was using a PMOS, some resistors and a diode, but then we quickly realised that we are amateurs at the ART OF ELECTRONICS and could short any connection accidentally, so to just separate the wall power from the ESP circuit, we had to get something which could physically disconnect the battery.

Workaround: We got rid of the PMOS contraption and just used a simple Toggle Switch instead! Made things easier and effective!

3.7 The Enclosure Design

While the PCB zero prototype was well and working, the actual size of it turned out to be far too large and heavy to be mounted on a spectacle. This is why the actual project appears to be mounted on a wrist band.

For the material of the enclosure, we decided to go with canvas board, as it was hard enough for our purpose and the laboratory 3D printer was not available by the time we had our prototype!

Chapter 4

Final Design and Implementation details

4.1 List of Components

PCB-Zero Board

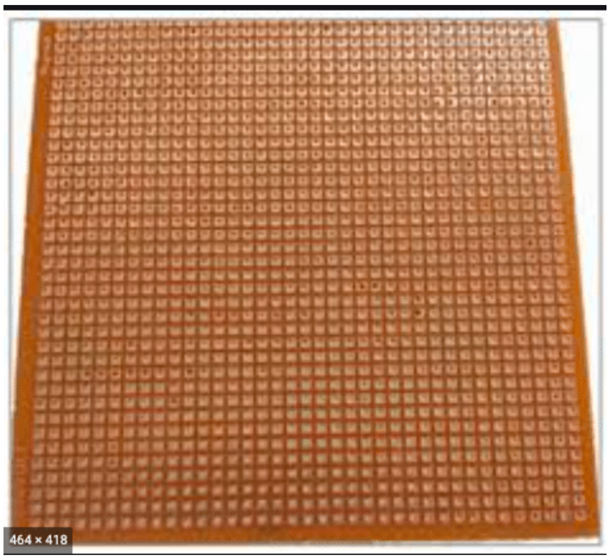

Figure 4.1: Zero PCB

Connecting Wires

MT3608 Breakout Board

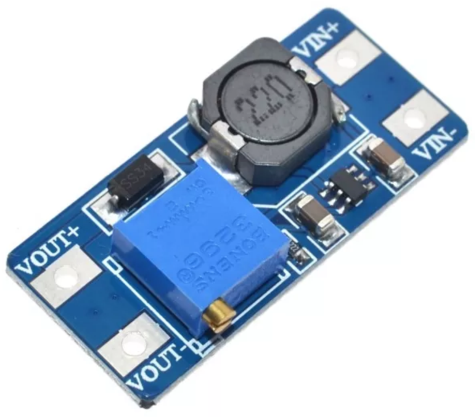

Figure 4.2: MT3608 Breakout Board

TP4056 Charging Module

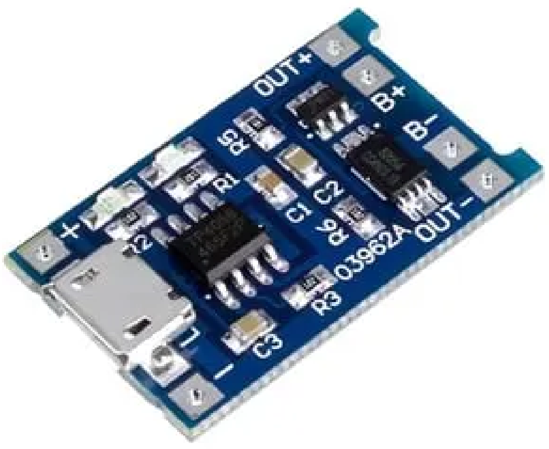

Figure 4.3: TP4056 Charging Module

ESP32-CAM Module

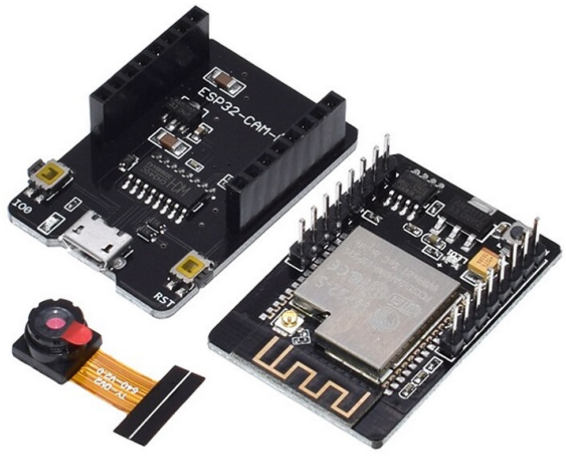

Figure 4.4: ESP32-CAM Module

LM7805 Voltage regulator

Figure 4.5: LM7805 Voltage Regulator

220-12V RMS Transformer @ 1.5A RMS

Diodes (Full-Wave Bridge Rectifier)

Capacitors (470uF, 1100uF, 10uF, 0.1uF)

4.2 The Adapter

Did not deviate much from this circuit, just had to add an extra 2200uF beefy capacitor at the input of the LM7805. This successfully started providing us with up to 400mA on the output of the battery's charging module which was directly hooked up to the Adapter via a USB C cable (Yes, we did switch out the USB B Micro for the USB C).

4.3 The Main Circuit

4.3.1 Charging module

Used a standard TP4056 charger module instead of the MCP738312/1 due to ease of availability.

4.3.2 Switching Section

Used a Simple toggle switch to turn off/turn on the QR Scanner Circuit. This is connected with the TP4056 and MT3608 module, the former for charging the battery when power is supplied AND the ESP32 is OFF, and protecting the battery from over discharging when ESP32 is ON. The latter is simply a boost converter to boost the battery's 3.7V to 5V to be supplied to the ESP32.

4.3.3 The Actual QR Scanner Section

Used a TPS60123 Boster module to boost battery from 3.7V @ 1.6A to 5V @ up to 1A and fed the output into the ESP. The ESP uses a camera and buttons to switch modes between qr scanning and/or photo capture only modes. There is also the feature to turn on the flashlight (The inbuilt LED of the ESP). The Scanned QR/Photo are sent to the user via a telegram bot.

4.4 Enclosure

Used a canvas which was cut to match the dimensions of the soldered PCB0 approximately.

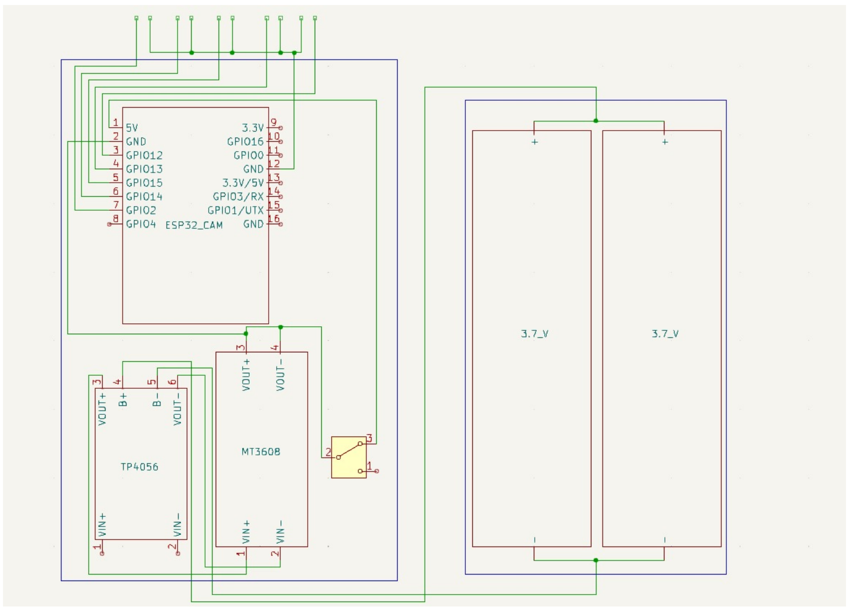

Figure 4.6: Schematic of the Final Design of the Project

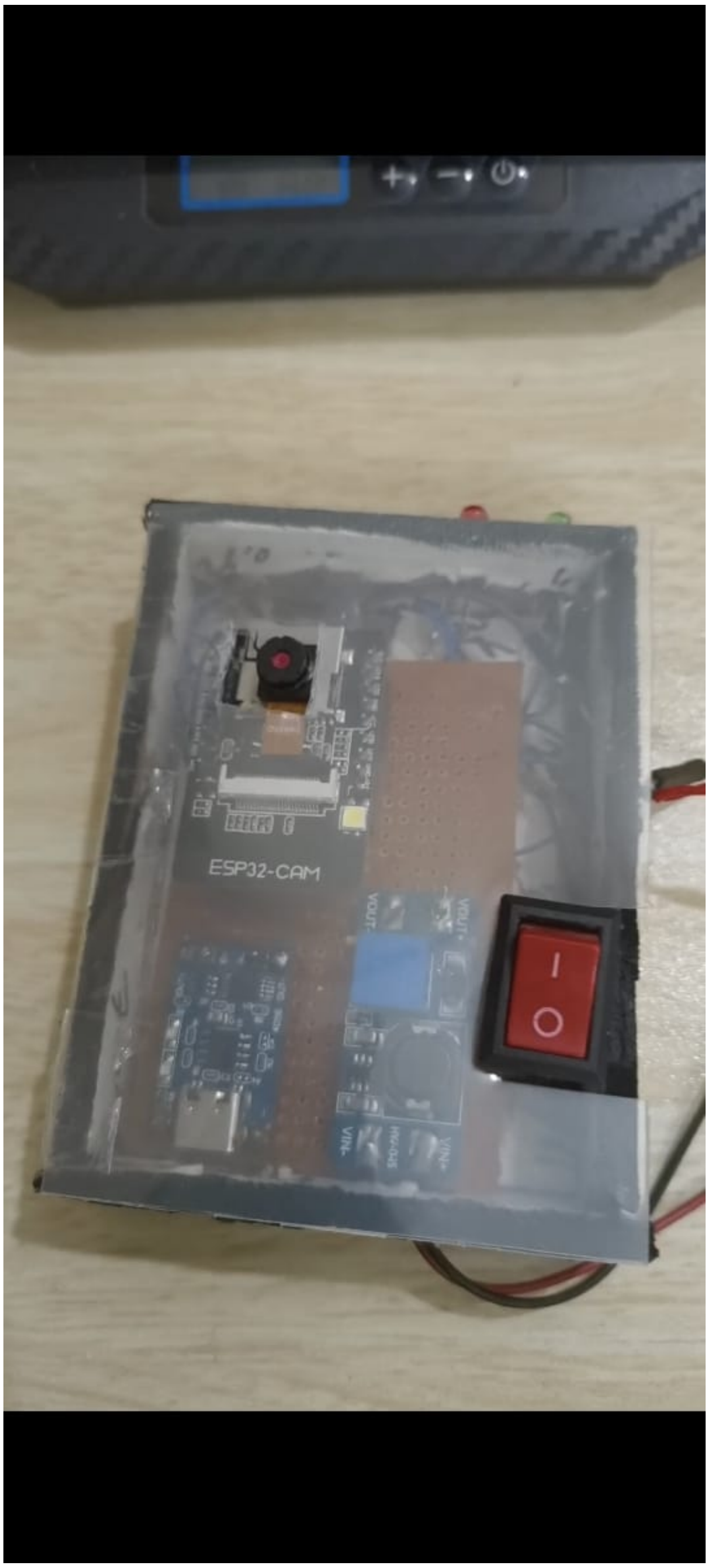

Figure 4.7: Final Project

4.5 The Code, Verbatim!

```
#include "esp_camera.h"
```

```
#include "esp_timer.h"
#include "Arduino.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>
#include "img_converters.h"

// Defining 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

#define BUTTON_FLASHLIGHT 2 //Button for flashlight
#define BUTTON_TAKE_PHOTO 14 //Button for clicking pictures
#define BUTTON_QR_MODE 15 //Button for scanning QR Codes

#define CAMERA_LED 4 //ESP 32 CAM in built flashlight
#define RED_LED 13 //Red LED
#define GREEN_LED 12 //Green LED

bool cameraOn = false;
bool inQRMode = false;

//Wifi Credentials
const char *ssid = "Maruti Niwas";
const char *password = "Bhuw@n@1301";

//server URL to send post request
const char *serverURL = "https://smartglasses.onrender.com/upload";

// Camera config
camera_config_t config;
```

```
void setupCamera() {
  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_RGB565;
  config.fb_location = CAMERA_FB_IN_PSRAM;

  if (psramFound()) {
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  } else {
    Serial.println("Not found");
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x\n", err);
    switch (err) {
      case ESP_ERR_INVALID_ARG:
        Serial.println("ESP_ERR_INVALID_ARG");
        break;
      case ESP_ERR_INVALID_STATE:
        Serial.println("ESP_ERR_INVALID_STATE");
        break;
      case ESP_ERR_NOT_FOUND:
        Serial.println("ESP_ERR_NOT_FOUND");
        break;
      default:
        Serial.println("Unknown error");
        break;
    }
  }

  Serial.println("Camera initialized.");
}

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  Serial.begin(115200);

  pinMode(BUTTON_FLASHLIGHT, INPUT_PULLUP);
  pinMode(BUTTON_TAKE_PHOTO, INPUT_PULLUP);
  pinMode(BUTTON_QR_MODE, INPUT_PULLUP);

  pinMode(CAMERA_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  digitalWrite(RED_LED, HIGH);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected.");

  digitalWrite(GREEN_LED, HIGH);
  delay(300);
  digitalWrite(GREEN_LED, LOW);

  setupCamera();
}

void loop() {

  if (digitalRead(BUTTON_FLASHLIGHT) == LOW) {
    digitalWrite(CAMERA_LED, digitalRead(CAMERA_LED) == LOW ? HIGH : LOW);
  }
  if (digitalRead(BUTTON_TAKE_PHOTO) == LOW) {
    takePhoto();
  }
  if (digitalRead(BUTTON_QR_MODE) == LOW) {
    inQRMode = true;
    takePhoto();
    inQRMode = false;
  }

  delay(300);
}


void takePhoto() {
  Serial.println("Taking photo...");
  digitalWrite(RED_LED, HIGH);

  bool success = false;

  for (int i = 0; i < 3; i++) {
    camera_fb_t *flush_fb = esp_camera_fb_get();
    if (flush_fb) {
      esp_camera_fb_return(flush_fb);
      delay(100);
    }
  }

  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    digitalWrite(RED_LED, LOW);
    for (int i = 0; i < 2; i++) {
      digitalWrite(RED_LED, HIGH);
      delay(200);
      digitalWrite(RED_LED, LOW);
      delay(200);
    }
    return;
  }

  uint8_t *jpg_buf = NULL;
  size_t jpg_len = 0;
  bool converted = false;

  if (fb->format != PIXFORMAT_JPEG) {
    converted = frame2jpg(fb, 80, &jpg_buf, &jpg_len);
  } else {
    jpg_buf = fb->buf;
    jpg_len = fb->len;
    converted = true;
  }

  if (!converted) {
```

```
  Serial.println("JPEG compression failed");
  esp_camera_fb_return(fb);
  digitalWrite(RED_LED, LOW);
  for (int i = 0; i < 2; i++) {
    digitalWrite(RED_LED, HIGH);
    delay(200);
    digitalWrite(RED_LED, LOW);
    delay(200);
  }
  return;
}

digitalWrite(RED_LED, LOW);
digitalWrite(GREEN_LED, HIGH);

//POST REQUEST TO SERVER
WiFiClientSecure client;
HTTPClient http;


client.setInsecure();

String requestURL(serverURL);
if (inQRMode) requestURL += "?qr=true";

String boundary = "PerplexityBoundary" + String(millis());
String header = "--" + boundary + "\r\n";
header += "Content-Disposition: form-data; name=\"image\"; filename=\"image.jpg\"\r
header += "Content-Type: image/jpeg\r\n\r\n";
String footer = "\r\n--" + boundary + "--\r\n";

size_t headerLen = header.length();
size_t footerLen = footer.length();
size_t totalLength = headerLen + jpg_len + footerLen;

uint8_t *postData = (uint8_t *)malloc(totalLength);
if (!postData) {
  Serial.println("Memory allocation failed!");
  if (converted) free(jpg_buf);
  esp_camera_fb_return(fb);
  digitalWrite(GREEN_LED, LOW);
  for (int i = 0; i < 2; i++) {
    digitalWrite(RED_LED, HIGH);
    delay(200);
    digitalWrite(RED_LED, LOW);
    delay(200);
  }
  return;
```

```
    }

    memcpy(postData, header.c_str(), headerLen);
    memcpy(postData + headerLen, jpg_buf, jpg_len);
    memcpy(postData + headerLen + jpg_len, footer.c_str(), footerLen);

    http.begin(client, requestURL);
    http.addHeader("Content-Type", "multipart/form-data; boundary=" + boundary);

    int httpCode = http.POST(postData, totalLength);
    free(postData);

    if (httpCode == HTTP_CODE_OK) {
      String response = http.getString();
      Serial.println("Server response:");
      Serial.println(response);
      if (response.indexOf("true") != -1) {
        success = true;
      }
    } else {
      Serial.printf("HTTP error: %d\n", httpCode);
      Serial.println(http.errorToString(httpCode));
    }

    http.end();

    if (converted) free(jpg_buf);
    esp_camera_fb_return(fb);
    digitalWrite(GREEN_LED, LOW);
    delay(200);

    if (success) {
      int counter = inQRMode ? 3 : 2;

      for (int i = 0; i < counter; i++) {
        digitalWrite(GREEN_LED, HIGH);
        delay(200);
        digitalWrite(GREEN_LED, LOW);
        delay(200);
      }

    }
  }
```

Chapter 5

Conclusion

The QR Code Scanner project was not only a learning opportunity but also a creative engineering endeavor. It encompassed embedded systems, electronics hardware design, real-time computer vision, and web-server integration — making it a well-rounded interdisciplinary project.

Key takeaways include:

- Understanding the real-world limitations of voltage regulators and power electronics.
- Working around hardware availability constraints by creatively adapting modules.
- Integrating a full-stack system: from camera sensing on an ESP32, to server-side decoding, to real-time messaging via Telegram.
- Dealing with bugs, flashing issues, and decoding limitations — and learning to design around those constraints.

5.1 Current Issues in the Project

The flashlight, its the flashlight. It flickers when we turn it on using the booster module and we can NOT understand why.
It works just fine when we turn it on using a wall power supply, but on battery it starts its tantrums back again!

5.2 Final Statements

While the project is not perfect, it has shown us the value of adaptability and the power of embedded computing when paired with cloud integration.
We hope that with a few final tweaks — especially to the charger and the enclosure — our QR scanner will be a working, demo-ready product. Regardless, the process has already taught us far more than a simple classroom assignment ever could.

Bibliography

[1] W3Schools.
ESP Programming Tutorial.
Available at: `https://www.w3schools.com`

[2] GitHub Repository.
ESP32 QR Code Reader Library.
Available at: `https://github.com/ESP32QRCodeReader/ESP32QRCodeReader.git`

[3] Espressif Systems.
Official ESP32-CAM Documentation and Models.
Available at: `https://www.espressif.com`

[4] The Art Of Electronics (Book)
Horowitz and Hill
For Miscellaneous references and circuit designing guide