Project Report

Image Capturing Device

Table of Contents

Introduction

This image capturing device includes a remote optical module configured to capture an image of a particular object or place, that will be sent to the user's email ID at regular intervals selected by the user.

It is a 12-volt-operated device that is connected to a 12-volt external adapter with a built-in battery backup feature which is used as an alternate power source. It uses a push button as a power cut-off switch to manually turn on the device and an additional LED is integrated for indication of Wifi connectivity, whether the device is connected to a network or not, searching for network and capturing images sent to particular email status.

We are using ESP-32 as the main controller for the device which has a built-in Camera and Wifi module.

Working Principle:

The LED will blink when the device is on by switch and the LED will turn Off when the device is connected to the WIFI. If the device is not connected to the WIFI, after 1 minute of network searching, the device automatically goes to deep sleep mode, after 30 minutes of Deep Sleep time interval, the device will restart again and search for the WIFI connection and the whole process will repeat itself.

- ➔ Switching On: When the button is pushed, the device starts searching for the WIFI, until the WIFI is not connected the LED continues to blink for a time interval of 1 minute after which the device goes into deep sleep mode if the WIFI network is not found.

- ➔ Memory Storage Check: After successful connection with the WIFI the device then checks for the availability of the storage space (which in our device is a SPIFFS Filesystem). If the device does not find any storage or storage device in it, then the device will restart again and again until it does not find any storage.

- Camera Checking: After getting a storage space, the device will then check for its camera. If the device does not find any camera, it will be restarted. This will go on until the camera is not found

- Image Capturing: After the confirmation of the availability of all of the above three things, the device will start capturing the image from wherever the face of the camera will be. Until a fully clear image is not captured, the device will keep clicking the images and delete the unclear image from the device.

- Storing Image: After a clear image is captured, the image is stored in the memory card of the device.

- Transferring Image: From the device, an automatic email is generated to the user with the help of an online SMTP server (which transfers the image from the controller to the user). Firstly, the server checks the credentials and generates an email with the subject ESP32-CAM-D1 (here D1 is for device-1) the connection to the server link is found in reference.

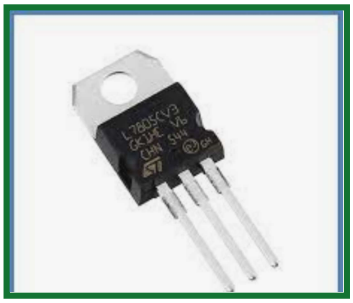
- Indication: When the image is sent to the user with a mail, the LED is blinked twice to indicate that the image is sent successfully and when the image is not sent due to any kind of error so there will be no indication from the LED. After all this, the image is deleted from the device memory so that it can store further images.

- Deep Sleep Mode: When all these things are done the device goes to deep sleep mode for 30 minutes and again performs all the above tasks.

Circuit Components:

1. **Regulator IC:** Here for regulation, we use LM7805 for 5-volt constant power to the controller
2. **Heat Sink:** A heat sink is used in this device to prevent regulating IC from any heating issue.
3. **Lithium Cells**: 2 Lithium cells of 2000mAh are used for backup power supply.
4. **2S-BMS:** A battery management system is used for 2 series lithium cells for short circuit protection and overcharges protection.
5. **DC-DC Buck converter**: Here LM2596 is used to convert the 12V power supply to 8V.
6. **ESP32-CAM:** ESP-32 is used as the main controller with the integrated camera feature.
7. **Diode:** It is used to verify the one-way flow of current.

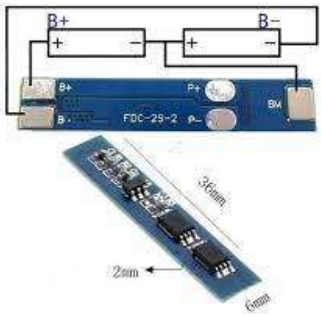
(1) LM7805 IC



(2) Heat Sink

(3) Lithium Cells

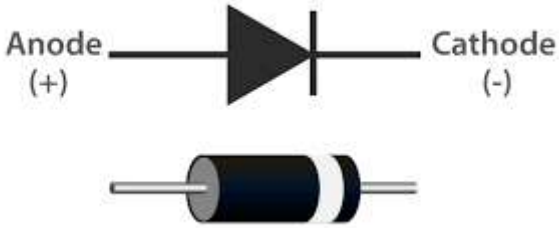
(4) 2S BMS



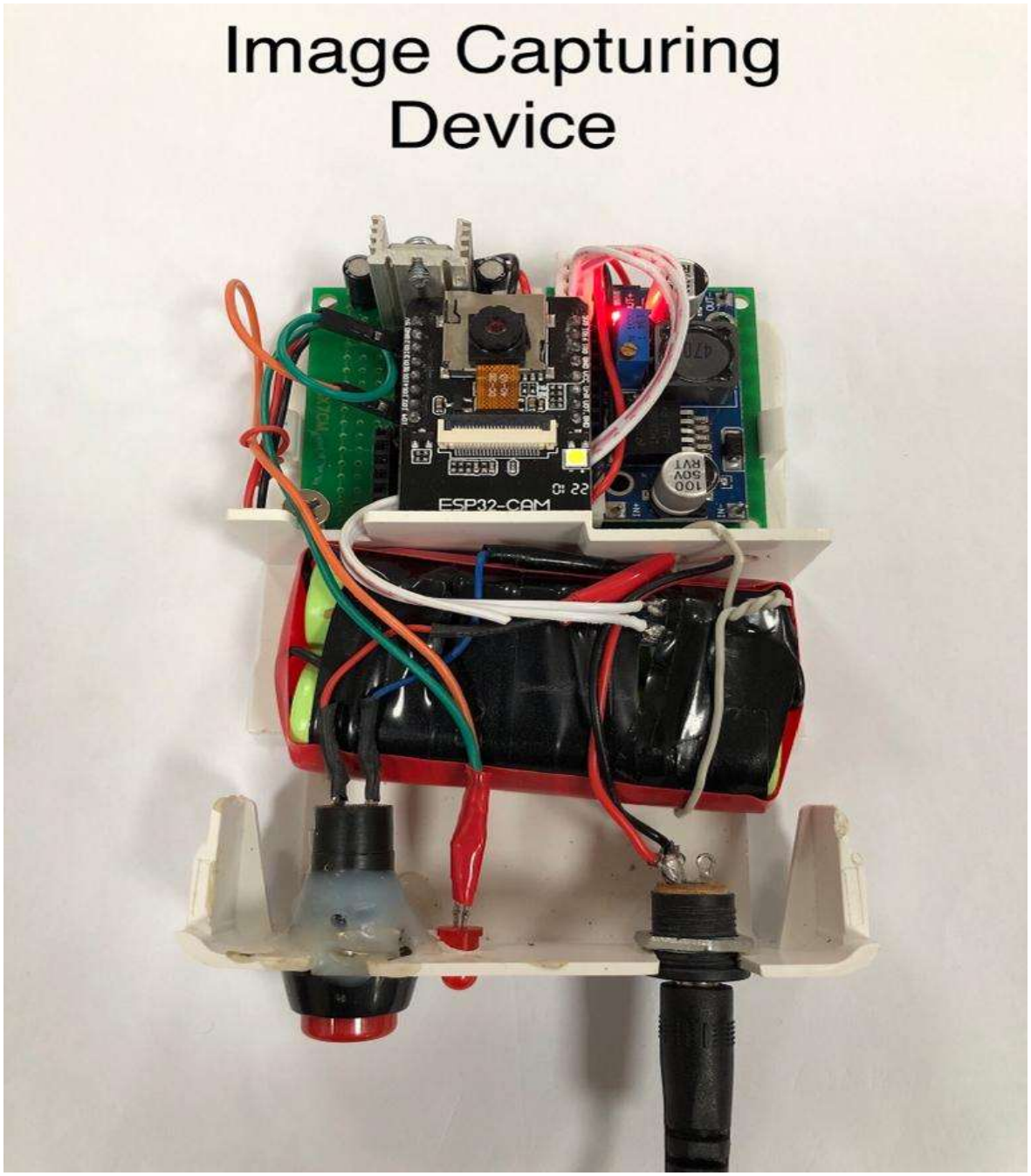
(5) DC-DC Buck Converter

(6) ESP32-CAM

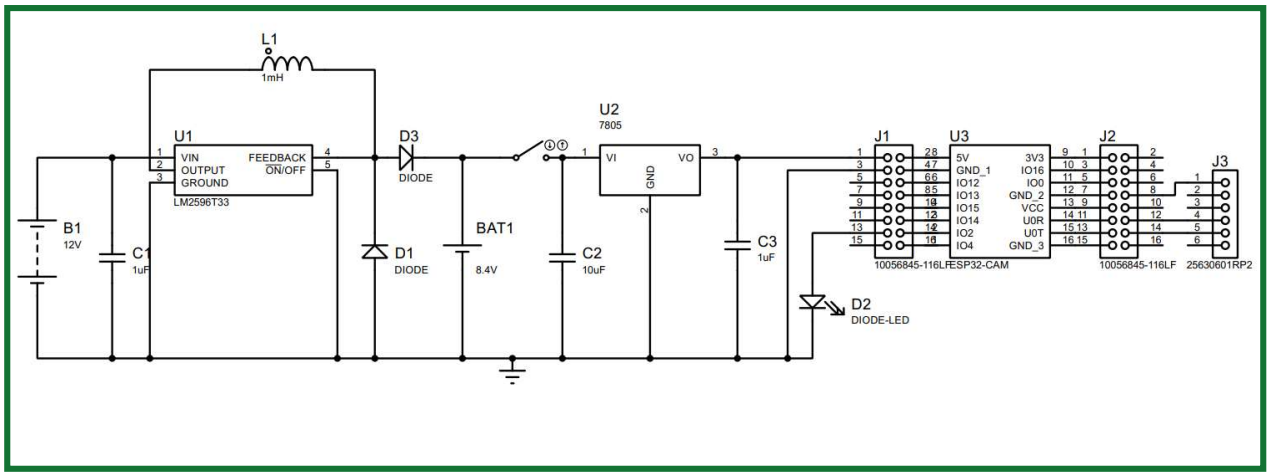
(7) Diode



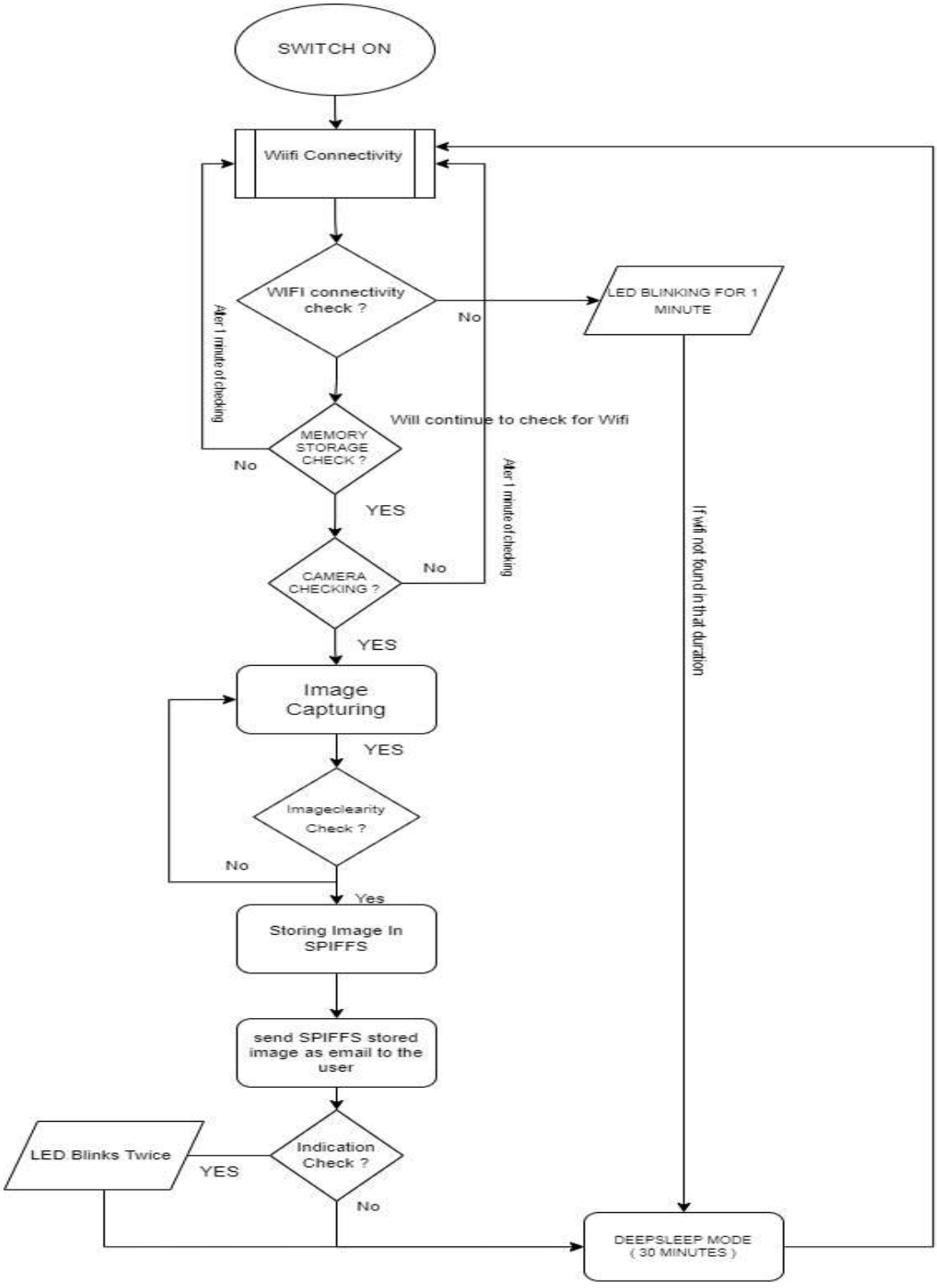
Internal Integration:



Circuit Diagram:



Block Diagram:



Further Modifications:

- Remote Configuration: configure the sent email address and time interval of image capturing.
- EEPROM: We can use EEPROM which is a built-in controller that works as a non-volatile memory we can use it for default and current configuration purposes
- Charging LED indicators: We can show the charging of the batteries with the help of different LEDs or LCDs, which will inform us about the percentage of the charging of the batteries.
- Flash Light: We can also add Flash Light for the night which can be controlled by any I/O pin.
- With a little modification, this can be used as face recognizer
- Can integrate gsm module to send data through wifi or gsm

Limitations and Precautions:

- Requires Wifi with stable internet connectivity.
- Requires memory card and RAM.
- Requires camera.
- Require 12 volts power supply.
- Should be prevented from water.
- Can not work efficiently on higher temperatures (greater than 55°C)
- Should be placed in a specific position with correct angle

Applications:

- Agriculture Purpose
- Meter Reading
- Remote Attendance

Testings and Results:

- [ ] Power Cut Off Switch
- [ ] LED indicating
- [ ] Charging of battery through BMS (Battery Management system)
- [ ] Regulation of Regulator IC
- [ ] Power Delivery of Bug Convertor
- [ ] At complete battery drainage voltage cut-off
- [ ] Working under hot temperature
- [ ] Battery backup up to one day (when one cell is weak)
- [ ] Charging and Operation of device at a particular instant
- [ ] Normal operations of ESP-32 controller takes 0.2 mA

Reference:

- https://randomnerdtutorials.com/program-upload-code-esp32-cam/
- https://randomnerdtutorials.com/esp32-cam-send-photos-email/
- https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/
- https://app.diagrams.net/#G1eBnD3pUr8RUZQPrGkUhp_P8xqFHlZiyt

Appendices:

Appendix 1: Main Code

```
/*
 *  main.ino
 *
 *  Created on: 06/09/2022
 *      Author: Muhammad Danish
 */

#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>


//------------------WIFI---------------
// REPLACE WITH YOUR NETWORK CREDENTIALS
//const char* ssid = "RCAI";
//const char* password = "RCAIned@123";
const char* ssid = "Extensity";
const char* password = "password1";
#define NWT_TIMEOUT 1*60*1000 //trying for 1 min

//-----------------------------------




//-------------------------ESP-CAM-----------
//sending picture to gmail at particular time interval
//unsigned long tm_now = -5 * 60 * 1000;
//unsigned long Alert_tm = 5 * 60 * 1000; //1 min * 60 second * 1000 ms = 1
min
// ledPin refers to ESP32-CAM GPIO 4 (flashlight)
#define Network_led 2
#define FLASH_GPIO_NUM 4

// To send Emails using Gmail on port 465 (SSL), you need to create an app
password: https://support.google.com/accounts/answer/185833
```

```
#define emailSenderAccount    "engrmuhammaddanish001@gmail.com"
#define emailSenderPassword   "ntcxyfmngcwwoxay"
#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ESP32-CAM Photo Captured"
#define emailRecipient        "smartdanish96@gmail.com"
//#define emailRecipient        "engrmuhammaddanish001@gmail.com"
#define CAMERA_MODEL_AI_THINKER

#if defined(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
#else
#error "Camera model not selected"
#endif

bool run_mode = false; //use for capture image or not
// The Email Sending data object contains config and data to send
SMTPData smtpData;

// Photo File Name to save in SPIFFS
#define FILE_PHOTO "/photo.jpg"
//-----------------------------------------------------------
```

```
//---------------------------------DEEP-SLEEP-----------------
//deep sleep variable
#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to
seconds */
#define TIME_TO_SLEEP  30*60    /* set to 30 min: Time ESP32 will go to
sleep (in seconds) */

RTC_DATA_ATTR int bootCount = 0;

/*
  Method to print the reason by which ESP32
  has been awaken from sleep
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external
signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external
signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer");
break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by
touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP
program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",
wakeup_reason); break;
  }
}

//------------------------------------------------------------------------

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector
```

```
  // initialize digital pin ledPin as an output
  //  pinMode(FLASH_GPIO_NUM, PULL_UP);
  pinMode(Network_led,OUTPUT);

  Serial.begin(115200);
  Serial.println();

  Serial.print(millis());
  Serial.println("ms: start Time");
  //------------DEEP SLEEP MODE----------

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  /*
    First we configure the wake up source
    We set our ESP32 to wake up every 5 seconds
  */
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP)
+
                 " Seconds");

  //------------DEEP SLEEP MODE END-----------------

  //--------------------WIFI-CONNECTIVITY-----------
  //Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED && millis() <= NWT_TIMEOUT) { //if
wifi not found then trying for 1 min
    digitalWrite(Network_led,!digitalRead(Network_led));
    Serial.print(".");
    delay(500);
  }
```

```
if (WiFi.status() != WL_CONNECTED) {
  Serial.println();
  Serial.println("WIFI NOT FOUND");
  digitalWrite(Network_led,HIGH);
  run_mode = LOW;
} else {
  // Print ESP32 Local IP Address
  Serial.print("IP Address: http://");
  Serial.println(WiFi.localIP());
  digitalWrite(Network_led,LOW);
  run_mode = HIGH;
}
Serial.println();

//--------------------WIFI-CONNECTIVITY-END-----------

if(run_mode){ //IF RUN MODE IS ACTIVATE THEN CAPTURE THE IMAGE
//-------------------ESP-CAM-MODE------------
if (!SPIFFS.begin(true)) {
  Serial.println("An Error has occurred while mounting SPIFFS");
  ESP.restart();
}
else {
  delay(500);
  Serial.println("SPIFFS mounted successfully");
}



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;
  //    config.jpeg_quality = 10;
  //    config.fb_count = 2;
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 10;
  config.fb_count = 1;
} else {
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}

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

//------------CAPTURED IMAGE SEND TO SERVER----------
delay(200);
//    digitalWrite(FLASH_GPIO_NUM, HIGH);
capturePhotoSaveSpiffs();
//    digitalWrite(FLASH_GPIO_NUM, LOW);
sendPhoto();

//------------CAPTURED IMAGE SERVER END-----------
```

```
  }


  //------------DEEP SLEEP MODE----------


  /*
    Now that we have setup a wake cause and if needed setup the
    peripherals state in deep sleep, we can now start going to
    deep sleep.
    In the case that no wake up sources were provided but deep
    sleep was started, it will sleep forever unless hardware
    reset occurs.
  */
  Serial.println("Going to sleep now");
  digitalWrite(Network_led,LOW);
  delay(200);
  Serial.flush();
  //  digitalWrite(FLASH_GPIO_NUM, LOW);
  esp_deep_sleep_start();
  Serial.println("This will never be printed");

  //------------DEEP SLEEP MODE END------------------



}

void loop() {

}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
  File f_pic = fs.open( FILE_PHOTO );
  unsigned int pic_sz = f_pic.size();
  return ( pic_sz > 100 );
}

// Capture Photo and Save it to SPIFFS
```

```
void capturePhotoSaveSpiffs( void ) {
  camera_fb_t * fb = NULL; // pointer
  bool ok = 0; // Boolean indicating if the picture has been taken
correctly

  do {
    // Take a photo with the camera
    Serial.println("Taking a photo...");

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

    // Photo file name
    Serial.printf("Picture file name: %s\n", FILE_PHOTO);
    File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

    // Insert the data in the photo file
    if (!file) {
      Serial.println("Failed to open file in writing mode");
    }
    else {
      file.write(fb->buf, fb->len); // payload (image), payload length
      Serial.print("The picture has been saved in ");
      Serial.print(FILE_PHOTO);
      Serial.print(" - Size: ");
      Serial.print(file.size());
      Serial.println(" bytes");
    }
    // Close the file
    file.close();
    esp_camera_fb_return(fb);

    // check if file has been correctly saved in SPIFFS
    ok = checkPhoto(SPIFFS);
  } while ( !ok );
}
```

```
void sendPhoto( void ) {
  // Preparing email
  Serial.println("Sending email...");
  // Set the SMTP Server Email host, port, account and password
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);

  // Set the sender name and Email
  smtpData.setSender("ESP32-CAM-D1", emailSenderAccount);

  // Set Email priority or importance High, Normal, Low or 1 to 5 (1 is
highest)
  smtpData.setPriority("High");

  // Set the subject
  smtpData.setSubject(emailSubject);

  // Set the email message in HTML format
  smtpData.setMessage("<h2>Photo captured with ESP32-CAM-DEVICE-1 and
attached in this email.</h2>", true);
  // Set the email message in text format
  //smtpData.setMessage("Photo captured with ESP32-CAM and attached in this
email.", false);

  // Add recipients, can add more than one recipient
  smtpData.addRecipient(emailRecipient);
  //smtpData.addRecipient(emailRecipient2);

  // Add attach files from SPIFFS
  smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
  // Set the storage type to attach files in your email (SPIFFS)
  smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

  smtpData.setSendCallback(sendCallback);

  // Start sending Email, can be set callback function to track the status
  if (!MailClient.sendMail(smtpData)){
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
  }else{
     for(int i=0; i<2 ; i++){
```

```
      digitalWrite(Network_led,HIGH);
      delay(250);
      digitalWrite(Network_led,LOW);
      delay(250);
     }
  }
  // Clear all data from Email object to free memory
  smtpData.empty();

  //delete created file
   if(SPIFFS.remove(FILE_PHOTO)){
        Serial.println("- file deleted");
    } else {
        Serial.println("- delete failed");
    }
}

// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {
  //Print the current status
  Serial.println(msg.info());
}
```

Appendix 2: Library header file

```
/*
 * Copyright (c) 2013 Adam Rudd.
 * See LICENSE for more information
 * https://github.com/adamvr/arduino-base64
 */
#ifndef _BASE64_H
#define _BASE64_H

/* b64_alphabet:
 *          Description: Base64 alphabet table, a mapping between integers
 *                               and base64 digits
 *          Notes: This is an extern here but is defined in Base64.c
 */
extern const char b64_alphabet[];

/* base64_encode:
 *          Description:
 *                Encode a string of characters as base64
 *          Parameters:
 *                output: the output buffer for the encoding, stores the
encoded string
 *                input: the input buffer for the encoding, stores the
binary to be encoded
 *                inputLen: the length of the input buffer, in bytes
 *          Return value:
 *                Returns the length of the encoded string
 *          Requirements:
 *                1. output must not be null or empty
 *                2. input must not be null
 *                3. inputLen must be greater than or equal to 0
 */
int base64_encode(char *output, char *input, int inputLen);

/* base64_decode:
 *          Description:
 *                Decode a base64 encoded string into bytes
```

```
 *          Parameters:
 *                output: the output buffer for the decoding,
 *                            stores the decoded binary
 *                input: the input buffer for the decoding,
 *                         stores the base64 string to be decoded
 *                inputLen: the length of the input buffer, in bytes
 *          Return value:
 *                Returns the length of the decoded string
 *          Requirements:
 *                1. output must not be null or empty
 *                2. input must not be null
 *                3. inputLen must be greater than or equal to 0
 */
int base64_decode(char *output, char *input, int inputLen);

/* base64_enc_len:
 *          Description:
 *                Returns the length of a base64 encoded string whose
decoded
 *                form is inputLen bytes long
 *          Parameters:
 *                inputLen: the length of the decoded string
 *          Return value:
 *                The length of a base64 encoded string whose decoded form
 *                is inputLen bytes long
 *          Requirements:
 *                None
 */
int base64_enc_len(int inputLen);

/* base64_dec_len:
 *          Description:
 *                Returns the length of the decoded form of a
 *                base64 encoded string
 *          Parameters:
 *                input: the base64 encoded string to be measured
 *                inputLen: the length of the base64 encoded string
 *          Return value:
 *                Returns the length of the decoded form of a
 *                base64 encoded string
```

```
 *          Requirements:
 *                1. input must not be null
 *                2. input must be greater than or equal to zero
 */
int base64_dec_len(char *input, int inputLen);

#endif // _BASE64_H
```

Appendix 3: Library C file

```
/*
 * Copyright (c) 2013 Adam Rudd.
 * See LICENSE for more information
 * https://github.com/adamvr/arduino-base64
 */
#if (defined(__AVR__))
#include <avr\pgmspace.h>
#else
#include <pgmspace.h>
#endif

const char PROGMEM b64_alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            "abcdefghijklmnopqrstuvwxyz"
            "0123456789+/";

/* 'Private' declarations */
inline void a3_to_a4(unsigned char * a4, unsigned char * a3);
inline void a4_to_a3(unsigned char * a3, unsigned char * a4);
inline unsigned char b64_lookup(char c);

int base64_encode(char *output, char *input, int inputLen) {
      int i = 0, j = 0;
      int encLen = 0;
      unsigned char a3[3];
      unsigned char a4[4];

      while(inputLen--) {
            a3[i++] = *(input++);
            if(i == 3) {
                  a3_to_a4(a4, a3);

                  for(i = 0; i < 4; i++) {
                        output[encLen++] =
pgm_read_byte(&b64_alphabet[a4[i]]);
                  }

                  i = 0;
            }
```

```
	}

	if(i) {
		for(j = i; j < 3; j++) {
			a3[j] = '\0';
		}

		a3_to_a4(a4, a3);

		for(j = 0; j < i + 1; j++) {
			output[encLen++] = pgm_read_byte(&b64_alphabet[a4[j]]);
		}

		while((i++ < 3)) {
			output[encLen++] = '=';
		}
	}
	output[encLen] = '\0';
	return encLen;
}

int base64_decode(char * output, char * input, int inputLen) {
	int i = 0, j = 0;
	int decLen = 0;
	unsigned char a3[3];
	unsigned char a4[4];


	while (inputLen--) {
		if(*input == '=') {
			break;
		}

		a4[i++] = *(input++);
		if (i == 4) {
			for (i = 0; i <4; i++) {
				a4[i] = b64_lookup(a4[i]);
			}

			a4_to_a3(a3,a4);
```

```
                for (i = 0; i < 3; i++) {
                        output[decLen++] = a3[i];
                }
                i = 0;
        }
    }

    if (i) {
        for (j = i; j < 4; j++) {
            a4[j] = '\0';
        }

        for (j = 0; j <4; j++) {
            a4[j] = b64_lookup(a4[j]);
        }

        a4_to_a3(a3,a4);

        for (j = 0; j < i - 1; j++) {
            output[decLen++] = a3[j];
        }
    }
    output[decLen] = '\0';
    return decLen;
}

int base64_enc_len(int plainLen) {
    int n = plainLen;
    return (n + 2 - ((n + 2) % 3)) / 3 * 4;
}

int base64_dec_len(char * input, int inputLen) {
    int i = 0;
    int numEq = 0;
    for(i = inputLen - 1; input[i] == '='; i--) {
        numEq++;
    }

    return ((6 * inputLen) / 8) - numEq;
```

```
}

inline void a3_to_a4(unsigned char * a4, unsigned char * a3) {
      a4[0] = (a3[0] & 0xfc) >> 2;
      a4[1] = ((a3[0] & 0x03) << 4) + ((a3[1] & 0xf0) >> 4);
      a4[2] = ((a3[1] & 0x0f) << 2) + ((a3[2] & 0xc0) >> 6);
      a4[3] = (a3[2] & 0x3f);
}

inline void a4_to_a3(unsigned char * a3, unsigned char * a4) {
      a3[0] = (a4[0] << 2) + ((a4[1] & 0x30) >> 4);
      a3[1] = ((a4[1] & 0xf) << 4) + ((a4[2] & 0x3c) >> 2);
      a3[2] = ((a4[2] & 0x3) << 6) + a4[3];
}

inline unsigned char b64_lookup(char c) {
      if(c >='A' && c <='Z') return c - 'A';
      if(c >='a' && c <='z') return c - 71;
      if(c >='0' && c <='9') return c + 4;
      if(c == '+') return 62;
      if(c == '/') return 63;
      return -1;
}
```