University of Asia Pacific

Department of Computer Science and Engineering

CSE 316: Microprocessors and Microcontrollers Lab

LAB REPORT CODE

Experiment Title: **Batwave Object Detective Car (B.O.D): A Line-Following Surveillance Robot with Object Detection, Alerts, and CCTV Recording**

Date of Submission: **23.10.2025**

Submitted by:
Name : **Shadman Sarwer**
H.M. Tahsin Sheikh
Humayra Jihan Arpita

Student ID : **22201242**
22201243
22201244

Section : **E-1**

Submitted to:
Zaima Sartaj Taheri
Lecturer,
Department of Computer Science and Engineering

B.O.D Line Following using 6 IR sensors (Arduino Code):

```
// Libraries
#include <NewPing.h>
#include <Servo.h>
#include <AFMotor.h>

// Ultrasonic sensor pins
#define TRIGGER_PIN A4
#define ECHO_PIN A5
#define MAX_DISTANCE 50

// IR sensor pins (4-channel, using A0–A3)
#define ir2 A0
#define ir3 A1
#define ir4 A2
#define ir5 A3

// Motor setup (L298D Motor Shield using AFMotor library)
AF_DCMotor motor1(1, MOTOR12_1KHZ); // M1 terminal
AF_DCMotor motor2(2, MOTOR12_1KHZ); // M2 terminal

Servo servo;
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

int distance = 0;
int leftDistance;
int rightDistance;
boolean object;
```

```
// Threshold for IR sensors (adjust after testing)
int threshold = 800; // 0–1023 range

void setup() {
Serial.begin(9600);

// Servo
servo.attach(10);
servo.write(90);

// Motor speed
motor1.setSpeed(100);  // adjust 0–255
motor2.setSpeed(100);
}

void loop() {
// Read IR sensors as analog, convert to digital (1=black, 0=white)
int d2 = (analogRead(ir2) < threshold) ? 0 : 1;
int d3 = (analogRead(ir3) < threshold) ? 0 : 1;
int d4 = (analogRead(ir4) < threshold) ? 0 : 1;
int d5 = (analogRead(ir5) < threshold) ? 0 : 1;

// Debug print
Serial.print("S2:"); Serial.print(d2);
Serial.print(" S3:"); Serial.print(d3);
Serial.print(" S4:"); Serial.print(d4);
Serial.print(" S5:"); Serial.println(d5);

// First check obstacle
```

```
distance = getDistance();
if (distance <= 15) {
objectAvoid();
} else {
// Line following logic
if(d4 == 0 && d3 == 0){

moveForward();
}
else if(d4 == 0 && d3 == 1){

moveLeft();

}
else if (d4 == 1 && d3 == 0){
moveRight();
}
else if (d4 == 1 && d3 == 1){
Stop();
}
}

delay(50);
}

// ---------------- OBSTACLE AVOIDANCE ----------------
void objectAvoid() {
Stop();
distance = getDistance();
```

```
if (distance <= 15) {
Stop();
lookLeft();
lookRight();
delay(100);
if (rightDistance <= leftDistance) {
object = true;
turn();
} else {
object = false;
turn();
}
delay(100);
}
else {
moveForward();
}
}

int getDistance() {
delay(50);
int cm = sonar.ping_cm();
if (cm == 0) cm = 100;
return cm;
}

int lookLeft () {
servo.write(150);
delay(500);
```

```
leftDistance = getDistance();
delay(100);
servo.write(90);
Serial.print("Left:"); Serial.println(leftDistance);
return leftDistance;
}

int lookRight() {
servo.write(30);
delay(500);
rightDistance = getDistance();
delay(100);
servo.write(90);
Serial.print("Right:"); Serial.println(rightDistance);
return rightDistance;
}

// ---------------- MOTOR CONTROL ----------------
void Stop() {
motor1.run(RELEASE);
motor2.run(RELEASE);
}

void moveForward() {
motor1.run(FORWARD);
motor2.run(FORWARD);
}

void moveBackward() {
```

```
motor1.run(BACKWARD);
motor2.run(BACKWARD);
}

void moveRight() {
motor1.run(FORWARD);
motor2.run(BACKWARD);
}

void moveLeft() {
motor1.run(BACKWARD);
motor2.run(FORWARD);
}

void turn() {
if (object == false) {
moveLeft();
delay(700);
moveForward();
delay(800);
moveRight();
delay(900);
}
else {
moveRight();
delay(700);
moveForward();
delay(800);
moveLeft();
```

delay(900);

}

}

Camera website access by ESP32 CAM module:

#include "esp_camera.h"

#include <WiFi.h>

#include <U8g2lib.h>

// ===========================

// Select camera model in board_config.h

// ===========================

#include "board_config.h"

static const unsigned char image_WarningDolphinFlip_bits[] PROGMEM = {

0x00,0x00,0x00,0x00,0x41,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x41,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x21,0x80,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x21,0x60,0x00,0x0c,0x00,0x00,0x00,0x00,0x00,0x00,0x11,0x18,0x00,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x11,0x04,0x00,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x09,0x01,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x02,0x00,0x00,0x00,0x00,0x00,0x80,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x00,0x80,0x87,0x00,0xc0,0x03,0x04,0x00,0x00,0x00,0x00,0x40,0x58,0x00,0x20,0x04,0x04,0x00,0x00,0x00,0x00,0x20,0x60,0x00,0xd0,0x0b,0x08,0x00,0x00,0x00,0x00,0x20,0x80,0x01,0x68,0x17,0x08,0x00,0x00,0x00,0x00,0xe0,0x01,0x06,0x28,0x17,0x08,0x00,0x00,0x00,0x00,0xc0,0x07,0x18,0xe8,0x17,0x08,0x00,0x00,0x00,0x00,0x80,0x0f,0x00,0xe8,0x17,0x10,0x00,0x00,0x00,0x00,0x00,0x3f,0x00,0xd0,0x0b,0x10,0x00,0x00,0x00,0x00,0x00,0x7e,0x00,0xf0,0x05,0x10,0x00,0x00,0x00,0x00,0x00,0xfc,0x00,0x08,0x02,0x10,0x00,0x00,0x00,0x00,0x00,0xf8,0x01,0x00,0x04,0x10,0x00,0x00,0x00,0x00,0x00,0xf0,0x03,0x00,0x00,0x10,0x00,0x00,0x00,0x00,0xc0,0x3f,0x0c,0x00,0x00,0x10,0x00,0x00,0x00,0x00,0x20,0x0f,0x30,0x00,0x0c,0x10,0x00,0x00,0x00,0x00,0x20,0xf0,0xff,0x03,0x12,0x10,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x00,0x00,0x12,0x30,0x00,0x00,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x00,0x12,0x30,0x00,0x00,0x00,0x00,0x00,0x80,0x01,0x00,0x00,0x00,0x12,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x0e,0x00,0x00,0x00,0xe2,0x61,0x00,0x00,0x00,0x00,0x00,0xf0,0x1f,0x00,0x12,0xe2,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0x07,0x00,0x0e,0xe4,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x00,0x09,0xc4,0x01,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x00,0x09,0xc5,0x01,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x00,0x09,0xc5,0x03

```
,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0xf1,0x84,0x03,0x00,0x00,0x00,0x00,0x20,0
x00,0x00,0xe2,0x88,0x03,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x04,0x88,0x03,0x0
0,0x00,0x00,0x00,0x10,0x00,0x00,0x08,0x10,0x03,0x00,0x00,0x00,0x00,0x10,0x00,
0x00,0x08,0x10,0x03,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x08,0x20,0x03,0x00,0x
00,0x00,0x00,0x10,0x00,0x00,0x08,0x20,0x02,0x00,0x00,0x00,0x00,0x10,0x00,0x00
,0x08,0x40,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0
x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0
x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
// ============================
// Enter your WiFi credentials
// ============================
const char *ssid = "**********";
const char *password = "**********";

//U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);
U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(
U8G2_R0, /* clock=*/ 14, /* data=*/ 15, /* reset=*/ U8X8_PIN_NONE);

void startCameraServer();
void setupLedFlash();

void setup() {
Serial.begin(115200);
Serial.setDebugOutput(true);
Serial.println();

//Display
```

```
u8g2.begin();



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_sccb_sda = SIOD_GPIO_NUM;
config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_UXGA;
config.pixel_format = PIXFORMAT_JPEG;  // for streaming
//config.pixel_format = PIXFORMAT_RGB565; // for face detection/recognition
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
```

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

// if PSRAM IC present, init with UXGA resolution and higher JPEG quality
//                      for larger pre-allocated frame buffer.
if (config.pixel_format == PIXFORMAT_JPEG) {
if (psramFound()) {
config.jpeg_quality = 10;
config.fb_count = 2;
config.grab_mode = CAMERA_GRAB_LATEST;
} else {
// Limit the frame size when PSRAM is not available
config.frame_size = FRAMESIZE_SVGA;
config.fb_location = CAMERA_FB_IN_DRAM;
}
} else {
// Best option for face detection/recognition
config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
config.fb_count = 2;
#endif
}

#if defined(CAMERA_MODEL_ESP_EYE)
pinMode(13, INPUT_PULLUP);
pinMode(14, INPUT_PULLUP);
#endif

// camera init
```

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

sensor_t *s = esp_camera_sensor_get();
// initial sensors are flipped vertically and colors are a bit saturated
if (s->id.PID == OV3660_PID) {
s->set_vflip(s, 1);       // flip it back
s->set_brightness(s, 1);   // up the brightness just a bit
s->set_saturation(s, -2);  // lower the saturation
}
// drop down frame size for higher initial frame rate
if (config.pixel_format == PIXFORMAT_JPEG) {
s->set_framesize(s, FRAMESIZE_QVGA);
}

#if defined(CAMERA_MODEL_M5STACK_WIDE) || defined(CAMERA_MODEL_M5STACK_ESP32CAM)
s->set_vflip(s, 1);
s->set_hmirror(s, 1);
#endif

#if defined(CAMERA_MODEL_ESP32S3_EYE)
s->set_vflip(s, 1);
#endif

// Setup LED FLash if LED pin is defined in camera_pins.h
```

```
#if defined(LED_GPIO_NUM)
setupLedFlash();
#endif

WiFi.begin(ssid, password);
WiFi.setSleep(false);

Serial.print("WiFi connecting");
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

startCameraServer();

Serial.print("Camera Ready! Use 'http://");
Serial.print(WiFi.localIP());
Serial.println("' to connect");
}

void loop() {

u8g2.clearBuffer();
u8g2.setFontMode(1);
u8g2.setBitmapMode(1);
u8g2.setFont(u8g2_font_timR24_tr);
u8g2.drawStr(28, 64, "BOD");
```

```
u8g2.drawXBM(8, 0, 74, 52, image_WarningDolphinFlip_bits);

u8g2.sendBuffer();

// Do nothing. Everything is done in another task by the web server
delay(10000);
}
```

App httpd.cpp Code:

```
// Copyright 2015-2016 Espressif Systems (Shanghai) PTE LTD
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "img_converters.h"
#include "fb_gfx.h"
#include "esp32-hal-ledc.h"
```

```
#include "sdkconfig.h"
#include "camera_index.h"
#include "board_config.h"

#if defined(ARDUINO_ARCH_ESP32) && defined(CONFIG_ARDUHAL_ESP_LOG)
#include "esp32-hal-log.h"
#endif

// LED FLASH setup
#if defined(LED_GPIO_NUM)
#define CONFIG_LED_MAX_INTENSITY 255

int led_duty = 0;
bool isStreaming = false;

#endif

typedef struct {
  httpd_req_t *req;
  size_t len;
} jpg_chunking_t;

#define PART_BOUNDARY "123456789000000000000987654321"
static const char *_STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char *_STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char *_STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\nX-Timestamp: %d.%06d\r\n\r\n";
```

```
httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

typedef struct {
  size_t size;   //number of values used for filtering
  size_t index;  //current value index
  size_t count;  //value count
  int sum;
  int *values;  //array to be filled with values
} ra_filter_t;

static ra_filter_t ra_filter;

static ra_filter_t *ra_filter_init(ra_filter_t *filter, size_t sample_size) {
  memset(filter, 0, sizeof(ra_filter_t));

  filter->values = (int *)malloc(sample_size * sizeof(int));
  if (!filter->values) {
    return NULL;
  }
  memset(filter->values, 0, sample_size * sizeof(int));

  filter->size = sample_size;
  return filter;
}

#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
static int ra_filter_run(ra_filter_t *filter, int value) {
```

```
  if (!filter->values) {
    return value;
  }
  filter->sum -= filter->values[filter->index];
  filter->values[filter->index] = value;
  filter->sum += filter->values[filter->index];
  filter->index++;
  filter->index = filter->index % filter->size;
  if (filter->count < filter->size) {
    filter->count++;
  }
  return filter->sum / filter->count;
}
#endif

#if defined(LED_GPIO_NUM)
void enable_led(bool en) {  // Turn LED On or Off
  int duty = en ? led_duty : 0;
  if (en && isStreaming && (led_duty > CONFIG_LED_MAX_INTENSITY)) {
    duty = CONFIG_LED_MAX_INTENSITY;
  }
  ledcWrite(LED_GPIO_NUM, duty);
  //ledc_set_duty(CONFIG_LED_LEDC_SPEED_MODE,
CONFIG_LED_LEDC_CHANNEL, duty);
  //ledc_update_duty(CONFIG_LED_LEDC_SPEED_MODE,
CONFIG_LED_LEDC_CHANNEL);
  log_i("Set LED intensity to %d", duty);
}
#endif
```

```
static esp_err_t bmp_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  uint64_t fr_start = esp_timer_get_time();
#endif
  fb = esp_camera_fb_get();
  if (!fb) {
    log_e("Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }

  httpd_resp_set_type(req, "image/x-windows-bmp");
  httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.bmp");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

  char ts[32];
  snprintf(ts, 32, "%lld.%06ld", fb->timestamp.tv_sec, fb->timestamp.tv_usec);
  httpd_resp_set_hdr(req, "X-Timestamp", (const char *)ts);

  uint8_t *buf = NULL;
  size_t buf_len = 0;
  bool converted = frame2bmp(fb, &buf, &buf_len);
  esp_camera_fb_return(fb);
  if (!converted) {
    log_e("BMP Conversion failed");
```

```
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }
  res = httpd_resp_send(req, (const char *)buf, buf_len);
  free(buf);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  uint64_t fr_end = esp_timer_get_time();
#endif
  log_i("BMP: %llums, %uB", (uint64_t)((fr_end - fr_start) / 1000), buf_len);
  return res;
}

static size_t jpg_encode_stream(void *arg, size_t index, const void *data, size_t len) {
  jpg_chunking_t *j = (jpg_chunking_t *)arg;
  if (!index) {
    j->len = 0;
  }
  if (httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK) {
    return 0;
  }
  j->len += len;
  return len;
}

static esp_err_t capture_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
```

```
  int64_t fr_start = esp_timer_get_time();
#endif

#if defined(LED_GPIO_NUM)
  enable_led(true);
  vTaskDelay(150 / portTICK_PERIOD_MS);  // The LED needs to be turned on ~150ms before the call to esp_camera_fb_get()
  fb = esp_camera_fb_get();              // or it won't be visible in the frame. A better way to do this is needed.
  enable_led(false);
#else
  fb = esp_camera_fb_get();
#endif

  if (!fb) {
    log_e("Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }

  httpd_resp_set_type(req, "image/jpeg");
  httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

  char ts[32];
  snprintf(ts, 32, "%lld.%06ld", fb->timestamp.tv_sec, fb->timestamp.tv_usec);
  httpd_resp_set_hdr(req, "X-Timestamp", (const char *)ts);

#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
```

```
  size_t fb_len = 0;
#endif
  if (fb->format == PIXFORMAT_JPEG) {
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    fb_len = fb->len;
#endif
    res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
  } else {
    jpg_chunking_t jchunk = {req, 0};
    res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk) ? ESP_OK : ESP_FAIL;
    httpd_resp_send_chunk(req, NULL, 0);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    fb_len = jchunk.len;
#endif
  }
  esp_camera_fb_return(fb);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  int64_t fr_end = esp_timer_get_time();
#endif
  log_i("JPG: %uB %ums", (uint32_t)(fb_len), (uint32_t)((fr_end - fr_start) / 1000));
  return res;
}

static esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  struct timeval _timestamp;
  esp_err_t res = ESP_OK;
```

```
  size_t _jpg_buf_len = 0;
  uint8_t *_jpg_buf = NULL;
  char *part_buf[128];

  static int64_t last_frame = 0;
  if (!last_frame) {
    last_frame = esp_timer_get_time();
  }

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if (res != ESP_OK) {
    return res;
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  httpd_resp_set_hdr(req, "X-Framerate", "60");

#if defined(LED_GPIO_NUM)
  isStreaming = true;
  enable_led(true);
#endif

  while (true) {
    fb = esp_camera_fb_get();
    if (!fb) {
      log_e("Camera capture failed");
      res = ESP_FAIL;
    } else {
      _timestamp.tv_sec = fb->timestamp.tv_sec;
```

```
    _timestamp.tv_usec = fb->timestamp.tv_usec;
    if (fb->format != PIXFORMAT_JPEG) {
      bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
      esp_camera_fb_return(fb);
      fb = NULL;
      if (!jpeg_converted) {
        log_e("JPEG compression failed");
        res = ESP_FAIL;
      }
    } else {
      _jpg_buf_len = fb->len;
      _jpg_buf = fb->buf;
    }
  }
  if (res == ESP_OK) {
    res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
  }
  if (res == ESP_OK) {
    size_t hlen = snprintf((char *)part_buf, 128, _STREAM_PART, _jpg_buf_len, _timestamp.tv_sec, _timestamp.tv_usec);
    res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
  }
  if (res == ESP_OK) {
    res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
  }
  if (fb) {
    esp_camera_fb_return(fb);
    fb = NULL;
```

```
      _jpg_buf = NULL;
    } else if (_jpg_buf) {
      free(_jpg_buf);
      _jpg_buf = NULL;
    }
    if (res != ESP_OK) {
      log_e("Send frame failed");
      break;
    }
    int64_t fr_end = esp_timer_get_time();

    int64_t frame_time = fr_end - last_frame;
    last_frame = fr_end;

    frame_time /= 1000;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    uint32_t avg_frame_time = ra_filter_run(&ra_filter, frame_time);
#endif
    log_i(
      "MJPG: %uB %ums (%.1ffps), AVG: %ums (%.1ffps)", (uint32_t)(_jpg_buf_len), (uint32_t)frame_time, 1000.0 / (uint32_t)frame_time, avg_frame_time,
      1000.0 / avg_frame_time
    );
  }

#if defined(LED_GPIO_NUM)
  isStreaming = false;
  enable_led(false);
```

```
#endif

  return res;
}

static esp_err_t parse_get(httpd_req_t *req, char **obuf) {
  char *buf = NULL;
  size_t buf_len = 0;

  buf_len = httpd_req_get_url_query_len(req) + 1;
  if (buf_len > 1) {
    buf = (char *)malloc(buf_len);
    if (!buf) {
      httpd_resp_send_500(req);
      return ESP_FAIL;
    }
    if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
      *obuf = buf;
      return ESP_OK;
    }
    free(buf);
  }
  httpd_resp_send_404(req);
  return ESP_FAIL;
}

static esp_err_t cmd_handler(httpd_req_t *req) {
  char *buf = NULL;
  char variable[32];
```

```
  char value[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "var", variable, sizeof(variable)) != ESP_OK ||
httpd_query_key_value(buf, "val", value, sizeof(value)) != ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int val = atoi(value);
  log_i("%s = %d", variable, val);
  sensor_t *s = esp_camera_sensor_get();
  int res = 0;

  if (!strcmp(variable, "framesize")) {
    if (s->pixformat == PIXFORMAT_JPEG) {
      res = s->set_framesize(s, (framesize_t)val);
    }
  } else if (!strcmp(variable, "quality")) {
    res = s->set_quality(s, val);
  } else if (!strcmp(variable, "contrast")) {
    res = s->set_contrast(s, val);
  } else if (!strcmp(variable, "brightness")) {
    res = s->set_brightness(s, val);
  } else if (!strcmp(variable, "saturation")) {
```

```
  res = s->set_saturation(s, val);
} else if (!strcmp(variable, "gainceiling")) {
  res = s->set_gainceiling(s, (gainceiling_t)val);
} else if (!strcmp(variable, "colorbar")) {
  res = s->set_colorbar(s, val);
} else if (!strcmp(variable, "awb")) {
  res = s->set_whitebal(s, val);
} else if (!strcmp(variable, "agc")) {
  res = s->set_gain_ctrl(s, val);
} else if (!strcmp(variable, "aec")) {
  res = s->set_exposure_ctrl(s, val);
} else if (!strcmp(variable, "hmirror")) {
  res = s->set_hmirror(s, val);
} else if (!strcmp(variable, "vflip")) {
  res = s->set_vflip(s, val);
} else if (!strcmp(variable, "awb_gain")) {
  res = s->set_awb_gain(s, val);
} else if (!strcmp(variable, "agc_gain")) {
  res = s->set_agc_gain(s, val);
} else if (!strcmp(variable, "aec_value")) {
  res = s->set_aec_value(s, val);
} else if (!strcmp(variable, "aec2")) {
  res = s->set_aec2(s, val);
} else if (!strcmp(variable, "dcw")) {
  res = s->set_dcw(s, val);
} else if (!strcmp(variable, "bpc")) {
  res = s->set_bpc(s, val);
} else if (!strcmp(variable, "wpc")) {
  res = s->set_wpc(s, val);
```

```
  } else if (!strcmp(variable, "raw_gma")) {
    res = s->set_raw_gma(s, val);
  } else if (!strcmp(variable, "lenc")) {
    res = s->set_lenc(s, val);
  } else if (!strcmp(variable, "special_effect")) {
    res = s->set_special_effect(s, val);
  } else if (!strcmp(variable, "wb_mode")) {
    res = s->set_wb_mode(s, val);
  } else if (!strcmp(variable, "ae_level")) {
    res = s->set_ae_level(s, val);
  }
#if defined(LED_GPIO_NUM)
  else if (!strcmp(variable, "led_intensity")) {
    led_duty = val;
    if (isStreaming) {
      enable_led(true);
    }
  }
#endif
  else {
    log_i("Unknown command: %s", variable);
    res = -1;
  }

  if (res < 0) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
```

```
  return httpd_resp_send(req, NULL, 0);
}

static int print_reg(char *p, sensor_t *s, uint16_t reg, uint32_t mask) {
  return sprintf(p, "\"0x%x\":%u,", reg, s->get_reg(s, reg, mask));
}

static esp_err_t status_handler(httpd_req_t *req) {
  static char json_response[1024];

  sensor_t *s = esp_camera_sensor_get();
  char *p = json_response;
  *p++ = '{';

  if (s->id.PID == OV5640_PID || s->id.PID == OV3660_PID) {
    for (int reg = 0x3400; reg < 0x3406; reg += 2) {
      p += print_reg(p, s, reg, 0xFFF);  //12 bit
    }
    p += print_reg(p, s, 0x3406, 0xFF);

    p += print_reg(p, s, 0x3500, 0xFFFF0);  //16 bit
    p += print_reg(p, s, 0x3503, 0xFF);
    p += print_reg(p, s, 0x350a, 0x3FF);   //10 bit
    p += print_reg(p, s, 0x350c, 0xFFFF);  //16 bit

    for (int reg = 0x5480; reg <= 0x5490; reg++) {
      p += print_reg(p, s, reg, 0xFF);
    }
```

```
  for (int reg = 0x5380; reg <= 0x538b; reg++) {
    p += print_reg(p, s, reg, 0xFF);
  }

  for (int reg = 0x5580; reg < 0x558a; reg++) {
    p += print_reg(p, s, reg, 0xFF);
  }
  p += print_reg(p, s, 0x558a, 0x1FF);  //9 bit
} else if (s->id.PID == OV2640_PID) {
  p += print_reg(p, s, 0xd3, 0xFF);
  p += print_reg(p, s, 0x111, 0xFF);
  p += print_reg(p, s, 0x132, 0xFF);
}

p += sprintf(p, "\"xclk\":%u,", s->xclk_freq_hz / 1000000);
p += sprintf(p, "\"pixformat\":%u,", s->pixformat);
p += sprintf(p, "\"framesize\":%u,", s->status.framesize);
p += sprintf(p, "\"quality\":%u,", s->status.quality);
p += sprintf(p, "\"brightness\":%d,", s->status.brightness);
p += sprintf(p, "\"contrast\":%d,", s->status.contrast);
p += sprintf(p, "\"saturation\":%d,", s->status.saturation);
p += sprintf(p, "\"sharpness\":%d,", s->status.sharpness);
p += sprintf(p, "\"special_effect\":%u,", s->status.special_effect);
p += sprintf(p, "\"wb_mode\":%u,", s->status.wb_mode);
p += sprintf(p, "\"awb\":%u,", s->status.awb);
p += sprintf(p, "\"awb_gain\":%u,", s->status.awb_gain);
p += sprintf(p, "\"aec\":%u,", s->status.aec);
p += sprintf(p, "\"aec2\":%u,", s->status.aec2);
p += sprintf(p, "\"ae_level\":%d,", s->status.ae_level);
```

```
  p += sprintf(p, "\"aec_value\":%u,", s->status.aec_value);
  p += sprintf(p, "\"agc\":%u,", s->status.agc);
  p += sprintf(p, "\"agc_gain\":%u,", s->status.agc_gain);
  p += sprintf(p, "\"gainceiling\":%u,", s->status.gainceiling);
  p += sprintf(p, "\"bpc\":%u,", s->status.bpc);
  p += sprintf(p, "\"wpc\":%u,", s->status.wpc);
  p += sprintf(p, "\"raw_gma\":%u,", s->status.raw_gma);
  p += sprintf(p, "\"lenc\":%u,", s->status.lenc);
  p += sprintf(p, "\"hmirror\":%u,", s->status.hmirror);
  p += sprintf(p, "\"vflip\":%u,", s->status.vflip);
  p += sprintf(p, "\"dcw\":%u,", s->status.dcw);
  p += sprintf(p, "\"colorbar\":%u", s->status.colorbar);
#if defined(LED_GPIO_NUM)
  p += sprintf(p, ",\"led_intensity\":%u", led_duty);
#else
  p += sprintf(p, ",\"led_intensity\":%d", -1);
#endif
  *p++ = '}';
  *p++ = 0;
  httpd_resp_set_type(req, "application/json");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, json_response, strlen(json_response));
}

static esp_err_t xclk_handler(httpd_req_t *req) {
  char *buf = NULL;
  char _xclk[32];

  if (parse_get(req, &buf) != ESP_OK) {
```

```
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "xclk", _xclk, sizeof(_xclk)) != ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int xclk = atoi(_xclk);
  log_i("Set XCLK: %d MHz", xclk);

  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_xclk(s, LEDC_TIMER_0, xclk);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t reg_handler(httpd_req_t *req) {
  char *buf = NULL;
  char _reg[32];
  char _mask[32];
  char _val[32];

  if (parse_get(req, &buf) != ESP_OK) {
```

```
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "reg", _reg, sizeof(_reg)) != ESP_OK || httpd_query_key_value(buf, "mask", _mask, sizeof(_mask)) != ESP_OK
      || httpd_query_key_value(buf, "val", _val, sizeof(_val)) != ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int reg = atoi(_reg);
  int mask = atoi(_mask);
  int val = atoi(_val);
  log_i("Set Register: reg: 0x%02x, mask: 0x%02x, value: 0x%02x", reg, mask, val);

  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_reg(s, reg, mask, val);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t greg_handler(httpd_req_t *req) {
  char *buf = NULL;
```

```
  char _reg[32];
  char _mask[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "reg", _reg, sizeof(_reg)) != ESP_OK || httpd_query_key_value(buf, "mask", _mask, sizeof(_mask)) != ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int reg = atoi(_reg);
  int mask = atoi(_mask);
  sensor_t *s = esp_camera_sensor_get();
  int res = s->get_reg(s, reg, mask);
  if (res < 0) {
    return httpd_resp_send_500(req);
  }
  log_i("Get Register: reg: 0x%02x, mask: 0x%02x, value: 0x%02x", reg, mask, res);

  char buffer[20];
  const char *val = itoa(res, buffer, 10);
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, val, strlen(val));
}
```

```
static int parse_get_var(char *buf, const char *key, int def) {
  char _int[16];
  if (httpd_query_key_value(buf, key, _int, sizeof(_int)) != ESP_OK) {
    return def;
  }
  return atoi(_int);
}

static esp_err_t pll_handler(httpd_req_t *req) {
  char *buf = NULL;

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }

  int bypass = parse_get_var(buf, "bypass", 0);
  int mul = parse_get_var(buf, "mul", 0);
  int sys = parse_get_var(buf, "sys", 0);
  int root = parse_get_var(buf, "root", 0);
  int pre = parse_get_var(buf, "pre", 0);
  int seld5 = parse_get_var(buf, "seld5", 0);
  int pclken = parse_get_var(buf, "pclken", 0);
  int pclk = parse_get_var(buf, "pclk", 0);
  free(buf);

  log_i("Set Pll: bypass: %d, mul: %d, sys: %d, root: %d, pre: %d, seld5: %d, pclken: %d, pclk: %d", bypass, mul, sys, root, pre, seld5, pclken, pclk);
  sensor_t *s = esp_camera_sensor_get();
```

```
  int res = s->set_pll(s, bypass, mul, sys, root, pre, seld5, pclken, pclk);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t win_handler(httpd_req_t *req) {
  char *buf = NULL;

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }

  int startX = parse_get_var(buf, "sx", 0);
  int startY = parse_get_var(buf, "sy", 0);
  int endX = parse_get_var(buf, "ex", 0);
  int endY = parse_get_var(buf, "ey", 0);
  int offsetX = parse_get_var(buf, "offx", 0);
  int offsetY = parse_get_var(buf, "offy", 0);
  int totalX = parse_get_var(buf, "tx", 0);
  int totalY = parse_get_var(buf, "ty", 0);  // codespell:ignore totaly
  int outputX = parse_get_var(buf, "ox", 0);
  int outputY = parse_get_var(buf, "oy", 0);
  bool scale = parse_get_var(buf, "scale", 0) == 1;
  bool binning = parse_get_var(buf, "binning", 0) == 1;
  free(buf);
```

```
  log_i(
    "Set Window: Start: %d %d, End: %d %d, Offset: %d %d, Total: %d %d, Output: %d %d, Scale: %u, Binning: %u", startX, startY, endX, endY, offsetX, offsetY,
    totalX, totalY, outputX, outputY, scale, binning  // codespell:ignore totaly
  );
  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_res_raw(s, startX, startY, endX, endY, offsetX, offsetY, totalX, totalY, outputX, outputY, scale, binning);  // codespell:ignore totaly
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t index_handler(httpd_req_t *req) {
  httpd_resp_set_type(req, "text/html");
  httpd_resp_set_hdr(req, "Content-Encoding", "gzip");
  sensor_t *s = esp_camera_sensor_get();
  if (s != NULL) {
    if (s->id.PID == OV3660_PID) {
      return httpd_resp_send(req, (const char *)index_ov3660_html_gz, index_ov3660_html_gz_len);
    } else if (s->id.PID == OV5640_PID) {
      return httpd_resp_send(req, (const char *)index_ov5640_html_gz, index_ov5640_html_gz_len);
    } else {
```

```
    return httpd_resp_send(req, (const char *)index_ov2640_html_gz, index_ov2640_html_gz_len);
   }
  } else {
   log_e("Camera sensor not found");
   return httpd_resp_send_500(req);
  }
}

void startCameraServer() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.max_uri_handlers = 16;

  httpd_uri_t index_uri = {
   .uri = "/",
   .method = HTTP_GET,
   .handler = index_handler,
   .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
   ,
   .is_websocket = true,
   .handle_ws_control_frames = false,
   .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t status_uri = {
   .uri = "/status",
   .method = HTTP_GET,
```

```
    .handler = status_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t cmd_uri = {
    .uri = "/control",
    .method = HTTP_GET,
    .handler = cmd_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t capture_uri = {
    .uri = "/capture",
    .method = HTTP_GET,
    .handler = capture_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
```

```
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t stream_uri = {
    .uri = "/stream",
    .method = HTTP_GET,
    .handler = stream_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t bmp_uri = {
    .uri = "/bmp",
    .method = HTTP_GET,
    .handler = bmp_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
```

```
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t xclk_uri = {
    .uri = "/xclk",
    .method = HTTP_GET,
    .handler = xclk_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t reg_uri = {
    .uri = "/reg",
    .method = HTTP_GET,
    .handler = reg_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };
```

```
  httpd_uri_t greg_uri = {
    .uri = "/greg",
    .method = HTTP_GET,
    .handler = greg_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t pll_uri = {
    .uri = "/pll",
    .method = HTTP_GET,
    .handler = pll_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t win_uri = {
    .uri = "/resolution",
```

```
    .method = HTTP_GET,
    .handler = win_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  ra_filter_init(&ra_filter, 20);

  log_i("Starting web server on port: '%d'", config.server_port);
  if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
    httpd_register_uri_handler(camera_httpd, &status_uri);
    httpd_register_uri_handler(camera_httpd, &capture_uri);
    httpd_register_uri_handler(camera_httpd, &bmp_uri);

    httpd_register_uri_handler(camera_httpd, &xclk_uri);
    httpd_register_uri_handler(camera_httpd, &reg_uri);
    httpd_register_uri_handler(camera_httpd, &greg_uri);
    httpd_register_uri_handler(camera_httpd, &pll_uri);
    httpd_register_uri_handler(camera_httpd, &win_uri);
  }

  config.server_port += 1;
```

```
  config.ctrl_port += 1;
  log_i("Starting stream server on port: '%d'", config.server_port);
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }
}

void setupLedFlash() {
#if defined(LED_GPIO_NUM)
  ledcAttach(LED_GPIO_NUM, 5000, 8);
#else
  log_i("LED flash is disabled -> LED_GPIO_NUM undefined");
#endif
}
```

Camera Pin Configuration:

```
#if defined(CAMERA_MODEL_WROVER_KIT)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  21
#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    19
#define Y4_GPIO_NUM    18
```

```
#define Y3_GPIO_NUM    5
#define Y2_GPIO_NUM    4
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM  23
#define PCLK_GPIO_NUM  22

#elif defined(CAMERA_MODEL_ESP_EYE)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  4
#define SIOD_GPIO_NUM  18
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    36
#define Y8_GPIO_NUM    37
#define Y7_GPIO_NUM    38
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    35
#define Y4_GPIO_NUM    14
#define Y3_GPIO_NUM    13
#define Y2_GPIO_NUM    34
#define VSYNC_GPIO_NUM 5
#define HREF_GPIO_NUM  27
#define PCLK_GPIO_NUM  25

#define LED_GPIO_NUM 22

#elif defined(CAMERA_MODEL_M5STACK_PSRAM)
#define PWDN_GPIO_NUM  -1
```

```
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  25
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    5
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    32
#define VSYNC_GPIO_NUM 22
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#elif defined(CAMERA_MODEL_M5STACK_V2_PSRAM)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  22
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    5
```

```
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    32
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#elif defined(CAMERA_MODEL_M5STACK_WIDE)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  22
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    5
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    32
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#define LED_GPIO_NUM 2

#elif defined(CAMERA_MODEL_M5STACK_ESP32CAM)
```

```
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  25
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    5
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    17
#define VSYNC_GPIO_NUM 22
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#elif defined(CAMERA_MODEL_M5STACK_UNITCAM)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  25
#define SIOC_GPIO_NUM  23

#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
```

```
#define Y5_GPIO_NUM    5
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    32
#define VSYNC_GPIO_NUM 22
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#elif defined(CAMERA_MODEL_M5STACK_CAMS3_UNIT)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM 21
#define XCLK_GPIO_NUM  11
#define SIOD_GPIO_NUM  17
#define SIOC_GPIO_NUM  41

#define Y9_GPIO_NUM    13
#define Y8_GPIO_NUM    4
#define Y7_GPIO_NUM    10
#define Y6_GPIO_NUM    5
#define Y5_GPIO_NUM    7
#define Y4_GPIO_NUM    16
#define Y3_GPIO_NUM    15
#define Y2_GPIO_NUM    6
#define VSYNC_GPIO_NUM 42
#define HREF_GPIO_NUM  18
#define PCLK_GPIO_NUM  12

#define LED_GPIO_NUM 14
```

```
#elif 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

// 4 for flash led or 33 for normal led
#define LED_GPIO_NUM   4

#elif defined(CAMERA_MODEL_TTGO_T_JOURNAL)
#define PWDN_GPIO_NUM  0
#define RESET_GPIO_NUM 15
#define XCLK_GPIO_NUM  27
#define SIOD_GPIO_NUM  25
#define SIOC_GPIO_NUM  23
```

```
#define Y9_GPIO_NUM    19
#define Y8_GPIO_NUM    36
#define Y7_GPIO_NUM    18
#define Y6_GPIO_NUM    39
#define Y5_GPIO_NUM    5
#define Y4_GPIO_NUM    34
#define Y3_GPIO_NUM    35
#define Y2_GPIO_NUM    17
#define VSYNC_GPIO_NUM 22
#define HREF_GPIO_NUM  26
#define PCLK_GPIO_NUM  21

#elif defined(CAMERA_MODEL_XIAO_ESP32S3)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  10
#define SIOD_GPIO_NUM  40
#define SIOC_GPIO_NUM  39

#define Y9_GPIO_NUM    48
#define Y8_GPIO_NUM    11
#define Y7_GPIO_NUM    12
#define Y6_GPIO_NUM    14
#define Y5_GPIO_NUM    16
#define Y4_GPIO_NUM    18
#define Y3_GPIO_NUM    17
#define Y2_GPIO_NUM    15
#define VSYNC_GPIO_NUM 38
#define HREF_GPIO_NUM  47
```

```
#define PCLK_GPIO_NUM  13

#elif defined(CAMERA_MODEL_ESP32_CAM_BOARD)
// The 18 pin header on the board has Y5 and Y3 swapped
#define USE_BOARD_HEADER 0
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM   33
#define XCLK_GPIO_NUM    4
#define SIOD_GPIO_NUM    18
#define SIOC_GPIO_NUM    23

#define Y9_GPIO_NUM 36
#define Y8_GPIO_NUM 19
#define Y7_GPIO_NUM 21
#define Y6_GPIO_NUM 39
#if USE_BOARD_HEADER
#define Y5_GPIO_NUM 13
#else
#define Y5_GPIO_NUM 35
#endif
#define Y4_GPIO_NUM 14
#if USE_BOARD_HEADER
#define Y3_GPIO_NUM 35
#else
#define Y3_GPIO_NUM 13
#endif
#define Y2_GPIO_NUM    34
#define VSYNC_GPIO_NUM 5
#define HREF_GPIO_NUM  27
```

```
#define PCLK_GPIO_NUM  25

#elif defined(CAMERA_MODEL_ESP32S3_CAM_LCD)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  40
#define SIOD_GPIO_NUM  17
#define SIOC_GPIO_NUM  18

#define Y9_GPIO_NUM    39
#define Y8_GPIO_NUM    41
#define Y7_GPIO_NUM    42
#define Y6_GPIO_NUM    12
#define Y5_GPIO_NUM    3
#define Y4_GPIO_NUM    14
#define Y3_GPIO_NUM    47
#define Y2_GPIO_NUM    13
#define VSYNC_GPIO_NUM 21
#define HREF_GPIO_NUM  38
#define PCLK_GPIO_NUM  11

#elif defined(CAMERA_MODEL_ESP32S2_CAM_BOARD)
// The 18 pin header on the board has Y5 and Y3 swapped
#define USE_BOARD_HEADER 0
#define PWDN_GPIO_NUM    1
#define RESET_GPIO_NUM   2
#define XCLK_GPIO_NUM    42
#define SIOD_GPIO_NUM    41
#define SIOC_GPIO_NUM    18
```

```
#define Y9_GPIO_NUM 16
#define Y8_GPIO_NUM 39
#define Y7_GPIO_NUM 40
#define Y6_GPIO_NUM 15
#if USE_BOARD_HEADER
#define Y5_GPIO_NUM 12
#else
#define Y5_GPIO_NUM 13
#endif
#define Y4_GPIO_NUM 5
#if USE_BOARD_HEADER
#define Y3_GPIO_NUM 13
#else
#define Y3_GPIO_NUM 12
#endif
#define Y2_GPIO_NUM    14
#define VSYNC_GPIO_NUM 38
#define HREF_GPIO_NUM  4
#define PCLK_GPIO_NUM  3

#elif defined(CAMERA_MODEL_ESP32S3_EYE)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  15
#define SIOD_GPIO_NUM  4
#define SIOC_GPIO_NUM  5

#define Y2_GPIO_NUM 11
```

```
#define Y3_GPIO_NUM 9
#define Y4_GPIO_NUM 8
#define Y5_GPIO_NUM 10
#define Y6_GPIO_NUM 12
#define Y7_GPIO_NUM 18
#define Y8_GPIO_NUM 17
#define Y9_GPIO_NUM 16

#define VSYNC_GPIO_NUM 6
#define HREF_GPIO_NUM  7
#define PCLK_GPIO_NUM  13

#elif defined(CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3) || defined(CAMERA_MODEL_DFRobot_Romeo_ESP32S3)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  45
#define SIOD_GPIO_NUM  1
#define SIOC_GPIO_NUM  2

#define Y9_GPIO_NUM    48
#define Y8_GPIO_NUM    46
#define Y7_GPIO_NUM    8
#define Y6_GPIO_NUM    7
#define Y5_GPIO_NUM    4
#define Y4_GPIO_NUM    41
#define Y3_GPIO_NUM    40
#define Y2_GPIO_NUM    39
#define VSYNC_GPIO_NUM 6
```

```
#define HREF_GPIO_NUM  42
#define PCLK_GPIO_NUM  5

#else
#error "Camera model not selected"
#endif
```

Board Configuration.h :

```
#ifndef BOARD_CONFIG_H
#define BOARD_CONFIG_H

//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//
//            You must select partition scheme from the board menu that has at least 3MB APP space.

// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE  // Has PSRAM
//#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define  CAMERA_MODEL_M5STACK_V2_PSRAM  // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
```

```
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_CAMS3_UNIT  // Has PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
//#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
// ** Espressif Internal Boards **
//#define CAMERA_MODEL_ESP32_CAM_BOARD
//#define CAMERA_MODEL_ESP32S2_CAM_BOARD
//#define CAMERA_MODEL_ESP32S3_CAM_LCD
//#define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
//#define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
#include "camera_pins.h"

#endif  // BOARD_CONFIG_H
```