Security system based on ESP32

2021/2022

Microprocessors and Embedded Systems

Project documentation

Roman Janiczek (xjanic25)

December 13, 2021

Contents

1 Introduction

This project aims to create a simple security system that consists of one PIR sensor and a camera. The system should also have WiFi AP capabilities to view captured pictures on the webserver running in this internal network.

For this, we will be using `WeMos D1 R32 UNO ESP32`, `AM312 PIR Motion sensor` and `AI-Thinker's ESP32-CAM`.

1.1 WeMos D1 R32 UNO ESP32

`WeMos D1 R32 UNO ESP32` is par of ESP32 series. This is a series low-cost, low-power MCUs with integrated Wi-Fi and Bluetooth.[1]

1.2 AI-Thinker's ESP32-CAM

`AI-Thinker's ESP32-CAM` is ESP32-based small sized camera with integrated Wi-Fi and support for TF cards.[2]

2 Design

The system will consist of two parts that will be interconnected.

- Motion detection
- Camera and AP

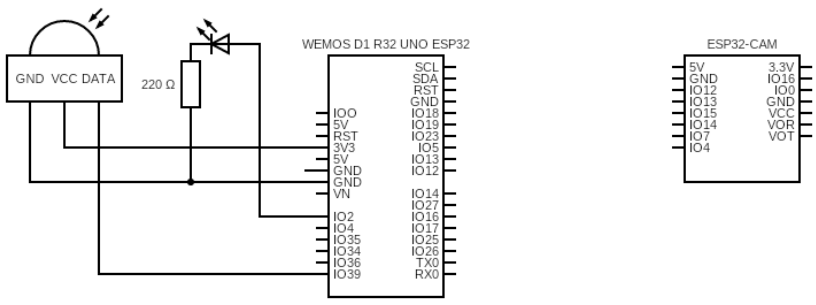

Figure 1: Connection diagram

[1] https://en.wikipedia.org/wiki/ESP32
[2] https://docs.ai-thinker.com/en/esp32-cam

2.1 Motion detection

Motion detection will be realized using `AM312 PIR Motion sensor` and a `WeMos D1 R32 UNO ESP32`.

The motion detection model attempts to connect to a local WiFi created by a camera module as a first step. This will allow it to send information about a detected motion to the camera.

Detected motion sends a signal from a PIR sensor to an ESP32 unit that creates an HTTP request for a second ESP32 unit (Camera), informing it about this event.

2.1.1 Process of detecting movement

- Wait for a movement (loop waiting)
- Receive signal from a PIR sensor
- Blink debug LED to let us know about detection
- Send signal over HTTP to camera
- Process response and go back to waiting

2.2 Camera and AP

The camera module is responsible for creating a local WiFi network . This module also serves as a web server that displays the latest captured photo.

Another functionality of the camera module and its primary purpose is capturing and saving a photo after receiving the signal from a motion detection module.

2.2.1 Process of capturing a photo

- Receive signal over HTTP about a movement
- Capture a photo (do not retry if failed)
- Save a photo to a memory
- Send a response to the motion detection module (information about the status of photo capture)

2.2.2 Process of presenting a photo

- Listens for requests to a web server
- Receive a request for a latest photo
- Load the photo from a memory
- Display it on the main page (or send to a requester depends on request URL)

3 Implementation

3.1 Used technologies

Implementation of this system was realized using ESP-IDF [3] , programming language C [4] and PlatformIO in VSCode [5].

3.2 Motion detection

3.2.1 PIR sensor

GPIO first had to be chosen and configured as an input to use PIR sensor. For this, GPIO39 was selected. At the same time, debug LED output was configured on GPIO2.

```
#include <driver/gpio.h>

#define LED_GPIO GPIO_NUM_2
#define PIR_GPIO GPIO_NUM_39

// Setup OUTPUT
gpio_pad_select_gpio(LED_GPIO);
gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);
gpio_set_level(LED_GPIO, 0);
// Setup INPUT
gpio_pad_select_gpio(PIR_GPIO);
gpio_set_direction(PIR_GPIO, GPIO_MODE_INPUT);

```

After correct configuration of INPUT GPIO it is left running in infinite while loop where it checks if a signal has been received. If it detects signal it blinks with debug LED and calls function to send HTTP request to camera.

```
while(1) {
    if (gpio_get_level(PIR_GPIO)) {
        timestamp = esp_timer_get_time();
        gpio_set_level(LED_GPIO, 1);
        send_request_to_camera();
        vTaskDelay(delay);
        gpio_set_level(LED_GPIO, 0);
    } else {
        gpio_set_level(LED_GPIO, 0);
        vTaskDelay(delay);
    }
}

```

To prevent it from sending too much requests processing is delayed for 1s using function `vTaskDelay()` and const delay.

```
const TickType_t delay = 1000 / portTICK_PERIOD_MS;

```

[3] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
[4] https://www.iso.org/standard/74528.html
[5] https://docs.platformio.org/en/latest/integration/ide/vscode.html

3.2.2 Wireless connection (WiFi STA)

Connection to existing WiFi AP is realized using library `"esp_wifi.h"` and official example code[6]. Default SSID for AP is hardcoded. Our IP and a gateway are saved for further use later in a program during connection.

```
#define WIFI_SSID       "ESP32-Cam AP"
.
my_ip = event->ip_info.ip;
gateway = event->ip_info.gw;

```

3.2.3 HTTP requests

Sending of a signal to a camera is realized using HTTP request. This functionality can be implemented using library `"esp_http_client.h"`. For this project was used modified official example code[7]. Here comes a previously saved gateway address into play - in this system, the gateway is always the camera module. The gateway address is taken as a host and appended `/pir` path. HTTP client is configured with this newly created URL, initiated and called. (Called only on PIR event)

```
char temp_url[50];
sprintf(temp_url, "http://%d.%d.%d.%d/take-photo", IP2STR(&gateway)
    );

esp_http_client_config_t config = {
    .url = temp_url,
    .method = HTTP_METHOD_GET,

```

```
esp_http_client_handle_t client = esp_http_client_init(&config);
esp_err_t err = esp_http_client_perform(client);

```

An HTTP request uses the method GET due to problems encountered with the POST method.

[6] https://github.com/espressif/esp-idf/blob/5c33570524118873f7bd32490c7a0442fede4bf8/examples/wifi/fast_scan/main/fast_scan.c

[7] https://github.com/espressif/esp-idf/blob/5c33570524118873f7bd32490c7a0442fede4bf8/examples/protocols/esp_http_client/main/esp_http_client_example.c

3.3 Camera and AP

3.3.1 Wireless connection (WiFi AP)

Camera module servers as the main hub of the system as such it's this module that creates WiFi AP. Library `"esp_wifi.h"` is used here as well but this time during configuration we chose AP mode.

```
esp_netif_create_default_wifi_ap();

```

```
wifi_config_t custom_config = {
    .ap = {
        .ssid = WIFI_SSID,
        .ssid_len = strlen(WIFI_SSID),
        .channel = WIFI_CHAN,
        .max_connection = WIFI_MSCO,
        .authmode = WIFI_AUTH_OPEN      // Do not require password
    },
};
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &custom_config));
ESP_ERROR_CHECK(esp_wifi_start());

```

3.3.2 Storage

Due to problems encountered when trying to store pictures on an SD card or SPIFFS [8], I decided to implement the last taken photo storage as a temporary global variable that is rewritten when a new photo is taken. This decision also makes it easier to serve this photo on the webserver.

```
/**
 * Latest photo store
 */
size_t taken_photo_len = 0;
size_t taken_photo_width = 0;
size_t taken_photo_height = 0;
pixformat_t taken_photo_format = PIXFORMAT_JPEG;
uint8_t *taken_photo_buf = NULL;

```

```
if (taken_photo_buf != NULL) {
    free(taken_photo_buf);
}
taken_photo_buf = malloc(photo->len);
mempcpy(taken_photo_buf, photo->buf, photo->len);

taken_photo_format = photo->format;
taken_photo_len = photo->len;
taken_photo_width = photo->width;
taken_photo_height = photo->height;

```

[8] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/spiffs.html

3.3.3 Camera

To initialize the camera, a correct pin schema was needed. This was obtained from ESP-IDF GitHub[9]. I have also decided to enable flash when camera is taking pictures which allows for better visibility during night or in dark rooms.

```
gpio_set_level(4, 1);
vTaskDelay(delay);
camera_fb_t *photo = esp_camera_fb_get();
gpio_set_level(4, 0);

```

3.3.4 Webserver

Webserver as de facto main 'brain' of the whole system which allows the processing of PIR event and is also serving latest taken photo is implemented using `"esp_http_server.h"`[10] library from ESP-IDF.

The webserver is served on the default IP address of the camera module `192.168.4.1` and allows for the following paths:

- `/` - main page which displays latest taken photo and automatically refreshes every 5 seconds
- `/latest-photo.jpg` - server latest photo captured by PIR event directly
- `/take-photo` - captures a new photo and displays it directly; this photo is not saved anywhere and does not replace the latest photo captured by the PIR event
- `/pir` - is used by motion sensor module as an endpoint for sending a signal about the PIR event

All of those endpoints are calling handler functions that are responsible for providing correct data.

```
httpd_uri_t index_get = {
    .uri      = "/",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};
httpd_register_uri_handler(server, &index_get);
httpd_uri_t latest_get = {
    .uri      = "/latest-photo.jpg",
    .method   = HTTP_GET,
    .handler  = img_handler,
    .user_ctx = NULL
};
httpd_register_uri_handler(server, &latest_get);
```

[9] https://github.com/espressif/esp32-camera/blob/master/examples/main/take_picture.c

[10] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_server.html

```
httpd_uri_t pir_post = {
    .uri      = "/pir",
    .method   = HTTP_GET,
    .handler  = pir_handler,
    .user_ctx = NULL
};
httpd_register_uri_handler(server, &pir_post);
httpd_uri_t take_post = {
    .uri      = "/take-photo",
    .method   = HTTP_GET,
    .handler  = take_handler,
    .user_ctx = NULL
};
httpd_register_uri_handler(server, &take_post);

```

4 Conclusion

In conclusion this project was a lot of fun as it allowed me to learn a lot about ESP32 lower lever programming as compared to Arduino framework. The only thing that should be fixed that I'm aware of is the lack of archiving older photos on SD card or some other media.

4.1 Notes

Due to problem with programmer for `AI-Thinker's ESP32-CAM` which cause following error to appear a lot even when following correct procedure, programming and testing anything on `AI-Thinker's ESP32-CAM` was incredibly difficult and time consuming.

```
Traceback (most recent call last):
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 4582, in <module>
    _main()
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 4575, in _main
    main()
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 4074, in main
    esp = esp or get_default_connected_device(ser_list, port=args.port, connect_attempts=args.connect_attempts,
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 121, in get_default_connected_device
    _esp.connect(before, connect_attempts)
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 632, in connect
    last_error = self._connect_attempt(mode=mode, esp32r0_delay=True, usb_jtag_serial=usb_jtag_serial)
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 596, in _connect_attempt
    self.sync()
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 486, in sync
    val, _ = self.command(self.ESP_SYNC, b'\x07\x07\x12\x20' + 32 * b'\x55',
File "C:\Users\roman\.platformio\packages\tool-esptoolpy\esptool.py", line 418, in command
    self._port.timeout = new_timeout
File "C:\Users\roman\.platformio\penv\lib\site-packages\serial\serialutil.py", line 372, in timeout
    self._reconfigure_port()
File "C:\Users\roman\.platformio\penv\lib\site-packages\serial\serialwin32.py", line 222, in _reconfigure_port
    raise SerialException(
serial.serialutil.SerialException: Cannot configure port, something went wrong. Original message: PermissionError(13, 'Access is denied.', None, 5)
*** [upload] Error 1

```

4.2 Self-evaluation / Feedback

E	**F**	**Q**	**P**	**D**	Σ
0.5	4	2	1	4	10

5 Attachments

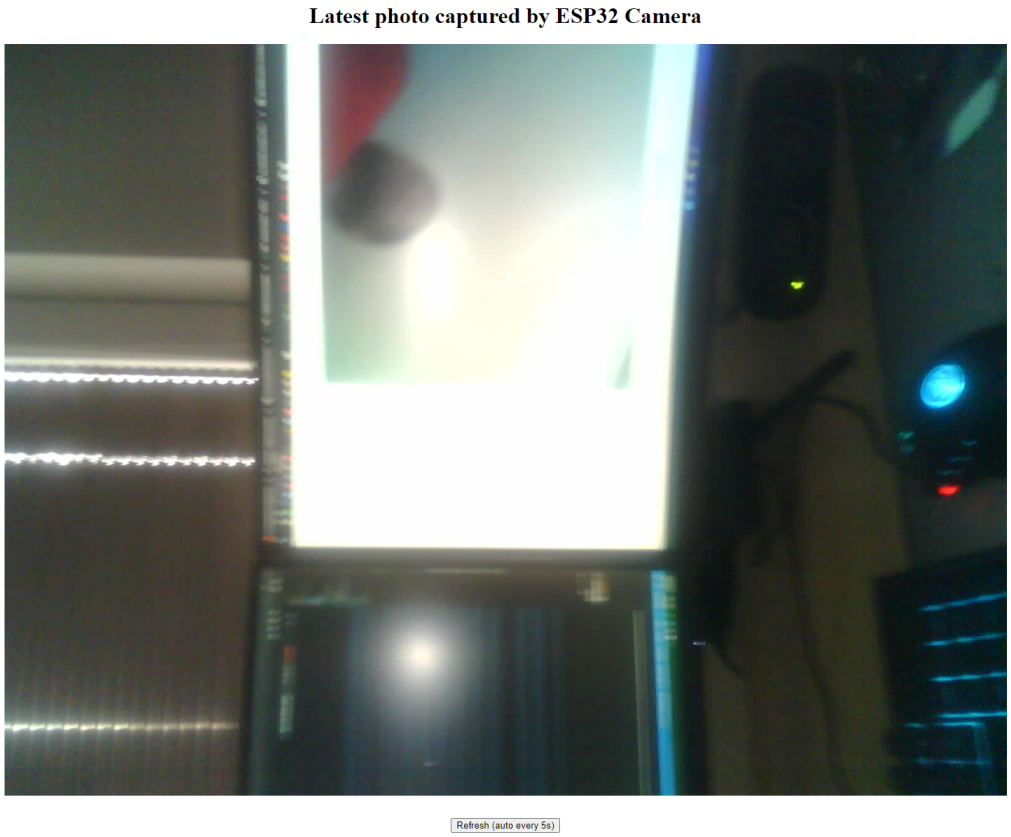

Figure 2: Main page of webserver

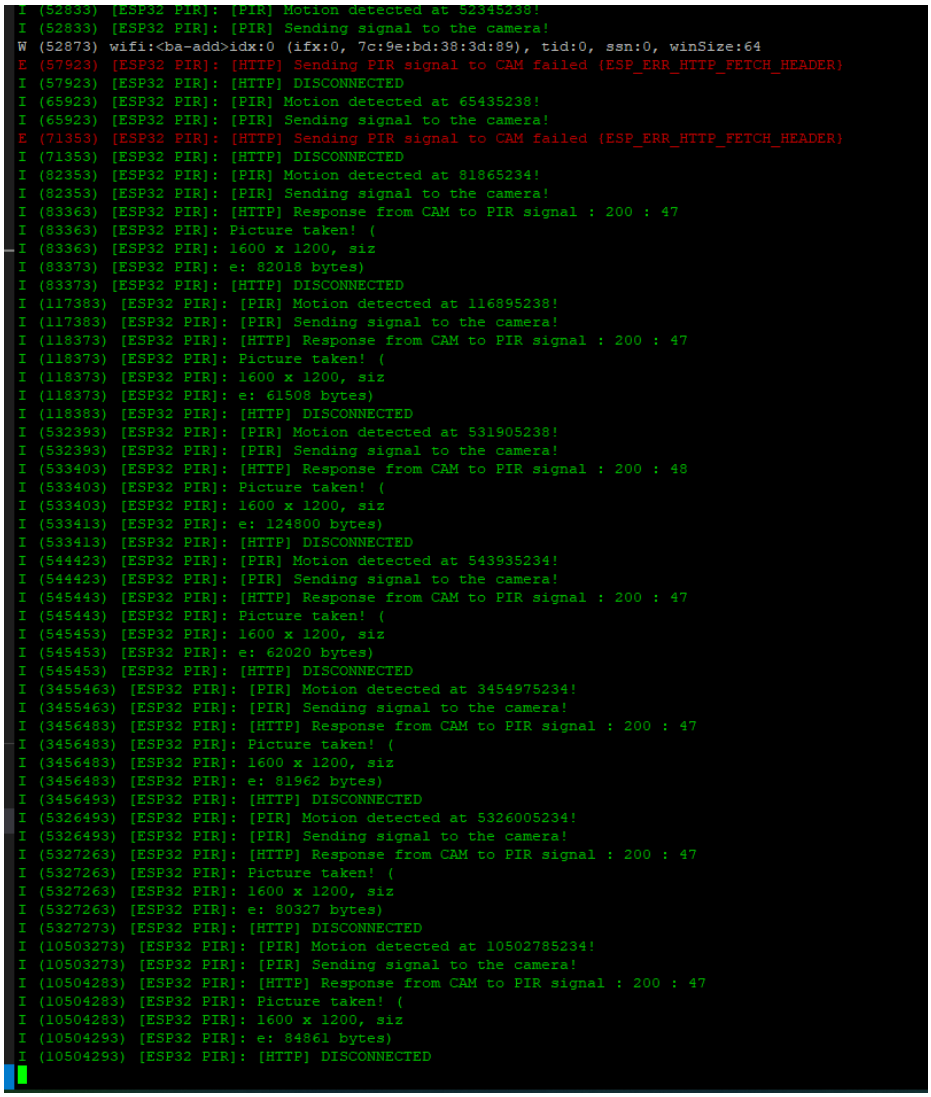

Figure 3: Logged PIR event on motion detection module