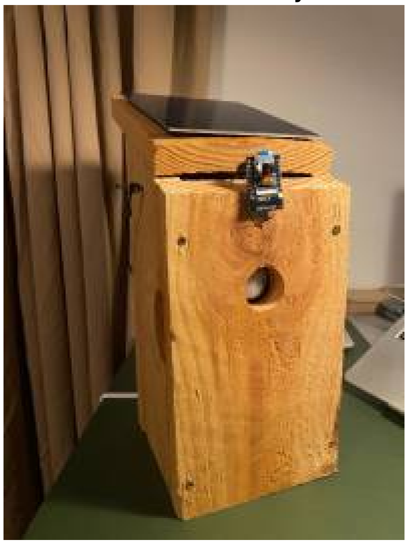
Bird House: Surveillance system

1. Introduction

EU The Birds Directive (Directive 79/409/EEC) The European Union adopted the birds directive in 1979 in order to conserve the wild birds in the EU by setting regulations for their protection, management and control. The directive covers the protection of their eggs, nests and habitats. The member countries have the responsibility to monitor, maintain or restore the population of endangered species and ensure the biodiversity of the species of birds in their native ecosystems. Therefore the monitoring systems are important for preservation efforts and monitoring and tracking of bird species and population are considered basic requirements and carry certain ecological, scientific and cultural significance. Because birds, with their fascinating beauty and astounding powers, have long captivated both nature enthusiasts and scientists. Avian species continue to fascinate us with their extraordinary movements and behaviors, from the epic migrations of great raptors across continents to the subtle foraging behaviors of small songbirds in local environments.
Our project is based on the idea to monitor birds in the environment with a simple arduino kit. This main idea of the project is to build an inexpensive monitoring system with simplified components which can be set up in the local ecosystems of the birds non-intrusively. The system automatically takes pictures when triggered by the PIR motion detection sensor at intervals of five minutes. It is powered by a 9V battery which is in turn being charged by PV modules on the roof of housing. The low power design means the camera is in deep sleep mode unless the PIR signal turns it on and it goes right back to sleep after the operation cycle which include the image capture, the wifi and MQTT connection and the upload to local storage(SD card) and MQTT server is completed. The additional features include the access from "Home Assistant" as a video stream and additional control which can be added as necessary.

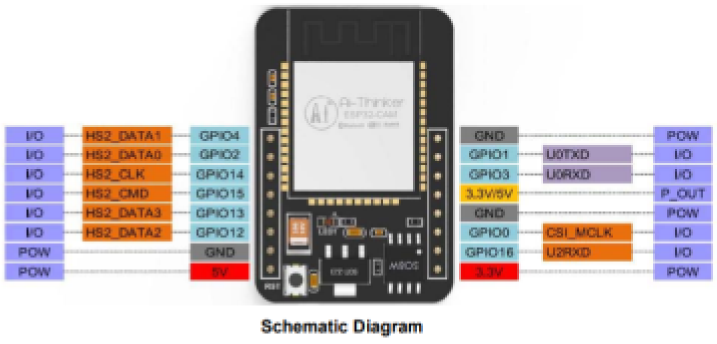


2.Material and method

2.1 Material

● ESP 32-Cam

ESP 32-Cam is a low cost development board with an onboard camera. It is suitable for our application as it supports WiFi, Bluetooth and SD card. Even though on the datasheet, it supports an SD card of 4GB storage for our part it still functions on 16GB. The specification can be assessed here in the datasheet below.



Esp32 cam can be powered by either 3.3V or 5V but when flashing on 3.3 V, esp32 cam error 0x20004 with camera probe failure was received. Therefore to be on the safe side, it should be powered with 5V. Gpio 3 and 1 are serial pins and transfer data and used to upload codes to the camera module. Esp32 In flashing mode:GPIO 0 is important when uploading codes because to set the camera to flashing mode it has to be grounded. Afte the code is uploaded, remove the connection and unground the GPIO 0, for the camera module to start working. /\\

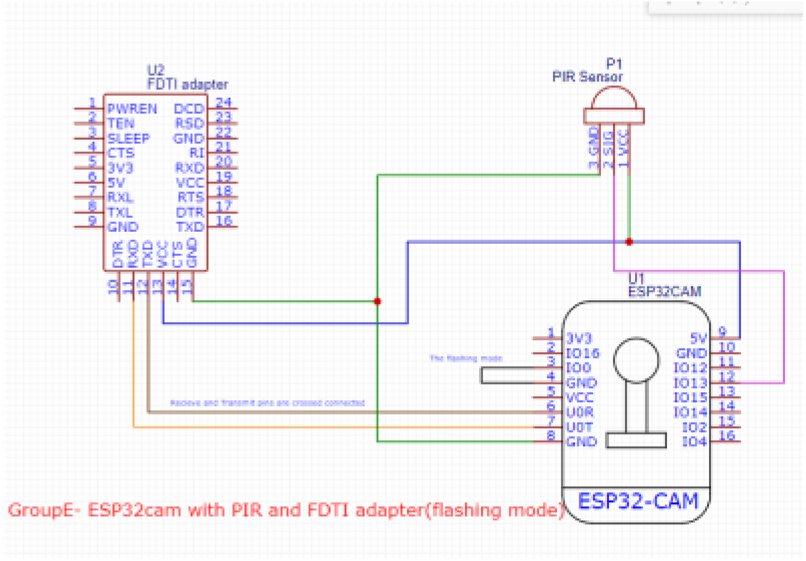


Fig3: Esp32 cam on flashing mode(source: Wai Lin)

One of the disadvantages of the ESP32-cam is that it has no USB connection and cannot be directly connected to the computer. Hence FDTI adapter is used which supports UART serial connection with the camera module.

//media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf>/ESP32 cam-Datasheet from digikey data sheet

● UartSBee V5 UartSBee V5 is FTDI cable compatible USB to Serial adapter equipped with BEE socket(20pin 2.0mm). The integrated FT232RL can be used for programming or communicating with MCUs. UARTSBEE by seeedstudio

● PIR Sensor

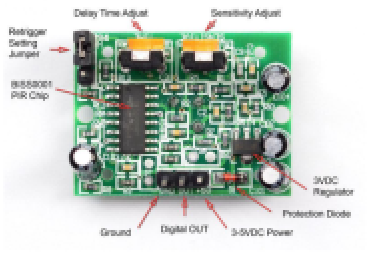


Fig 5: PIR sensor description (source: Adafruit)

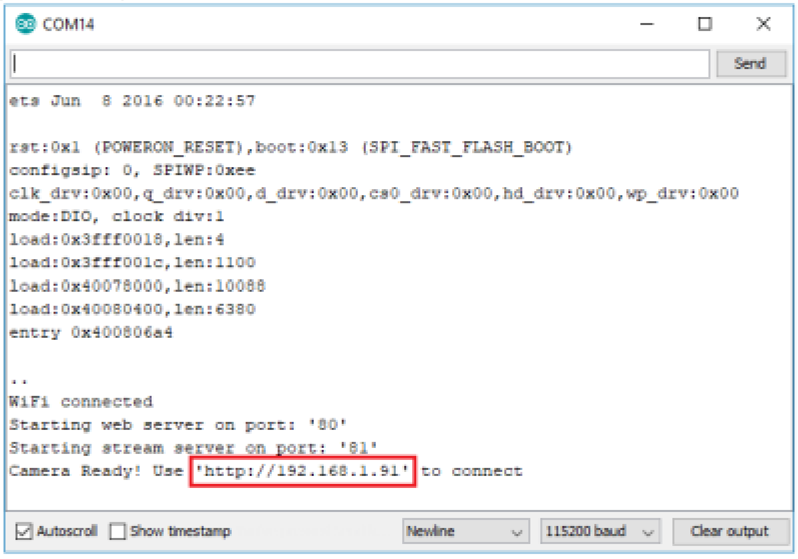
PIR referred to as “Passive Infrared”, “Pyroelectric”, or “IR motion” sensors, they are small, low power and inexpensive motion sensors. It has a sensitivity range up to 6 meters high and a range of 110° to 70°. There are two IR sensitive slots in 1 PIR sensor, when heat signature such as animals and human passes by, a positive differential change is created, when it leaves the detection area, a negative differential change is then created, which creates an output signal. When motion is detected, the digital output will be high (3V), when there is no motion, digital output will be low again. Since the sensor only detects heat signatures, it is perfect for bird houses, as it will not be triggered by other motions such as falling leaves. Adafruit PIR sensor datasheet

- Solar Panel
- Battery 9V
- Power Control Module
- Female-to-female jumper wires
- Female-to-male jumper wires
- Micro SD card

2.2 Method

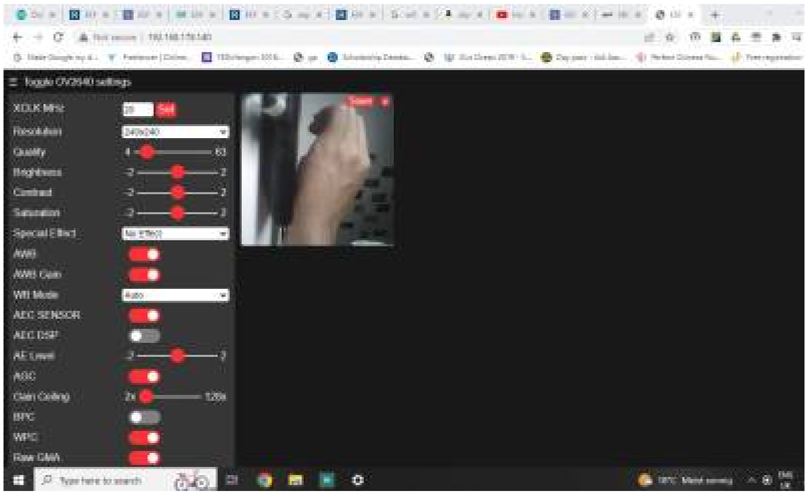
ESP32 Library :If you have not installed ESp32 boards library on your ardunio programmer, Go to File> Prerfercences (Clt + Comma) and insert theLink in additional bords URL. It takes some time to download for us 5 minutes. And you are free to continue.

Testing your ESP32-Cam
Before programming the ESP 32 cam for our purposes can be started, the connection and the camera module can be tested. This step takes only a few minutes and it can save a lot of unnecessary troubleshooting along the way. The source code from esp32 library was utilized for this step. (source- Wai Lin)
We chose AI thinker camera configuration. If everything goes well you will receive the IP address of your image stream on serial monitor on baud rate 115200.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4

..
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.1.91' to connect
```

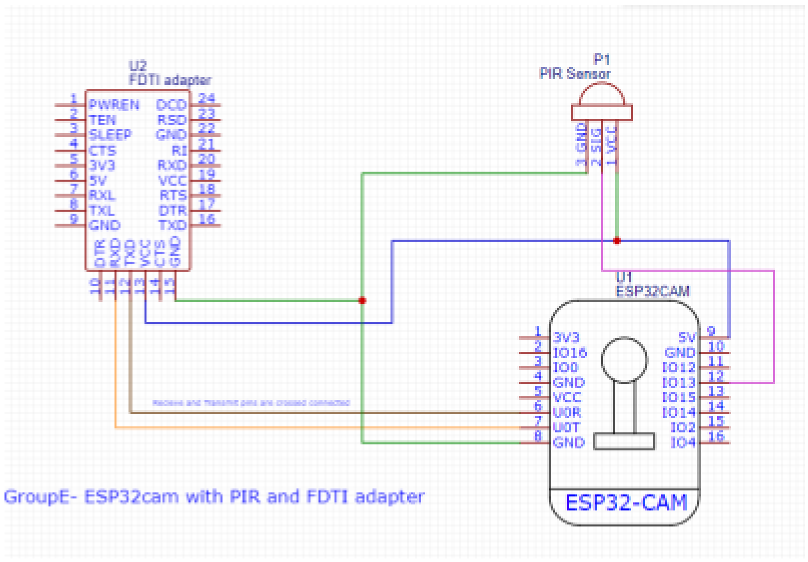
After inserting the camera Ip address. you will get assess to your camera stream



Toggle OV2640 settings

XCLK MHz 20 Set
Resolution 240x240
Quality 4 — 63
Brightness -2 — 2
Contrast -2 — 2
Saturation -2 — 2
Special Effect No Effect
AWB
AWB Gain
WB Mode Auto
AEC SENSOR
AEC DSP
AE Level -2 — 2
AGC
Gain Ceiling 2x — 128x
BPC
WPC
Raw GMA

Camera module and PIR sensor connection



(source-Wai Lin)

When motion is detected by the PIR sensor, GPIO13 goes high, the board will then wake up, and take a photo through the onboard camera. In addition, the system will take a photo every 5 minutes in order to make sure that the set up is still functioning well. The photos taken will be published to Flespi via MQTT as a message and also saved on the SD card.

====Sleeping Modes of ESP32====
The ESP32 microcontroller utilized in the project supports a range of general sleeping modes depending on which can reduce the RAM and CPU power and the working Wifi capabilities. Table 1.2 depicts a range of sleeping modes and their features with power supply. Table 1.2: The sleeping modes supported by ESP32 (source:espressif) Power mode Description Description Power consumption Active (RF working) Wi-Fi/BT Rx and listening Wi-Fi/BT Rx and listening 78 mA ~ 90 mA without communication

For TX RX more info in the next table Modem -sleep The CPU is powered on. 240 MHz * 30 mA ~ 68 mA Modem -sleep The CPU is powered on. 160 MHz * 27 mA ~ 44 mA Modem -sleep The CPU is powered on. 160 MHz * 27 mA ~ 34 mA Modem -sleep The CPU is powered on. Normal speed: 80 MHz 20 mA ~ 31 mA Modem -sleep The CPU is powered on. Normal speed: 80 MHz 20 mA ~ 25 mA Light-sleep – – 0.8 mA Deep- sleep The ULP co-processor is powered on. The ULP co-processor is powered on. 150 µA 100 µA @1% duty 10 µA Deep- sleep ULP sensor-monitored pattern ULP sensor-monitored pattern 150 µA 100 µA @1% duty 10 µA Deep- sleep RTC timer + RTC memory RTC timer + RTC memory 150 µA 100 µA @1% duty 10 µA Hiberna tion RTC timer only RTC timer only 5 µA Power off

CHIP_PU is set to low level, the chip is powered off. CHIP_PU is set to low level, the chip is powered off. 1 µA ESP32 power saving: modem and light sleep – 2 – Renzo Mischianti
Wake Up sources : The RTC controller is a built-in timer which can be used to wake up the microcontroller after a predefined amount of time. For our part we are using the external sensor(PIR) to wake up the camera. The Gpio 13 which supports the power on <https://mischianti.org/2021/03/10/esp32-power-saving-modem-and-light-sleep-2/> and off with external signal; the initial level 1 means when the signal is high from the sensor, the waking up of the system takes place.
esp_sleep_enable_ext0_wakeup(gpio_13, int level=1)

MQTT Protocol*
MQTT (message queuing telemetry transport) is a messaging protocol that was designed to create a reliable standard for machine-to-machine (m2m) communication. It is a publish-and- subscribe protocol, meaning that instead of communicating with a server, client devices and applications publish and subscribe to topics handled by a broker. It typically uses TCP/IP (Transmission Control Protocol/Internet Protocol) as its transport but can also use other bi- directional transports. It has become the de facto standard for IoT communication because of its efficiency and flexibility. u-blox uses this to overlay various radio networks (2G-4G cellular and LoRa) and protocols (USSD, UDP) providing developers with a familiar and simple experience. It allows devices and systems (clients) to communicate by sending messages. Messages are not sent directly from client to client but are published by a client to a topic* stored in an MQTT broker. Clients receive messages by subscribing to one or more topics and will receive messages from that point forwards. Topics are like street addresses - they form a tree which becomes more specific the further down you travel.

3.Result
3.1 Code

Final code.ino

```
Please refer to the end of the page
```

3.1.2 Connect to WiFi

Insert your network credentials

```
<const char* ssid = "iotlab";
const char* password = "****"; >

**3.1.3 Use Flespi MQTT broker to receive pictures**
 <file c++ setting up mqtt broker client>
const char* mqtt_server = "mqtt.flespi.io";
const int mqtt_port = 1883;
const char*  mqtt_user  =
"Ydl9IsQMA5NqDfGWgQt98ebuHLbm1gsPkDNQTtqho2xP51CbHTaGX9YuIZEz3Xdd";
const char* mqtt_password = "";
const char* mqtt_TopicName = "esp32/birds";
```

3.1.4 Use PIR sensor to trigger wake up the board

Use PIR sensor to trigger wake up the board

```
   if (SLEEP_DELAY == 0) {
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 1);
    delay(1000);
    esp_deep_sleep_start();
  }

  if (SLEEP_DELAY > 0) {
    delay(SLEEP_DELAY);
    delay(30000);
```

3.1.5 Save photos to SD card

saving photos to sd card

```
EEPROM.begin(EEPROM_SIZE);
  pictureNumber = EEPROM.read(0) + 1;
  // Path where new picture will be saved in SD Card
  String path = "/picture" + String(pictureNumber) +".jpg";
  fs::FS &fs = SD_MMC;
  Serial.printf("Picture file name: %s\n", path.c_str());
   File file = fs.open(path.c_str(), FILE_WRITE);
  if(!file){
    Serial.println("Failed to open file in writing mode");
  }
  else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
    EEPROM.write(0, pictureNumber);
    EEPROM.commit();
  }
  file.close();
```

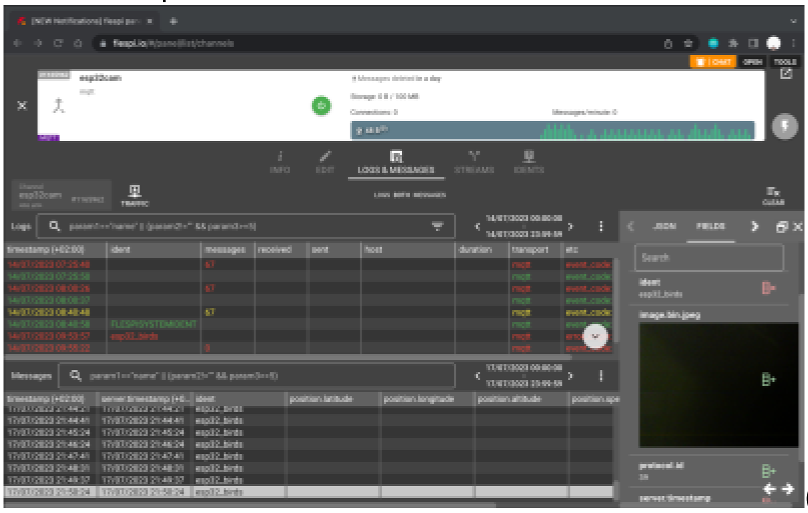
3.1.6 compression of JPEG and convert the picture to UTF-8 bits

Compressiong of jpeg files ino

```
if (fb->format != PIXFORMAT_JPEG) {
      Serial.println("Compressing");
      bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &frame_size);
```

3.2 Results on Flespi

We use the platform named as flespi, which allows pictures to be published via MQTT. Once messages flow into flespi, the platform transforms them into the unified JSON format and saves them in the database. Flespi token is a 64-byte randomly generated key-string used to access the data on the flespi platform via API, In MQTT API a token is used as an MQTT connection Username.Token configuration for your project is set up as shown in figure. In order to receive pictures taken in JSON messages, an MQTT channel is needed to receive messages from the broker, then a Device is needed to save the pictures.



Logs param1=="name" || (param2!="" && param3==5) 14/07/2023 00:00:00 – 14/07/2023 23:59:59

timestamp (+02:00)	ident	messages	received	sent	host	duration	transport	etc
14/07/2023 07:25:48		67					mqtt	event_code
14/07/2023 07:25:58							mqtt	event_code
14/07/2023 08:08:26		67					mqtt	event_code
14/07/2023 08:08:37							mqtt	event_code
14/07/2023 08:48:48		67					mqtt	event_code
14/07/2023 08:48:58	FLESPISYSTEMIDENT						mqtt	[illegible]
14/07/2023 09:53:57	esp32_birds						mqtt	[illegible]
14/07/2023 09:55:22		0					mqtt	event_code

Messages param1=="name" || (param2!="" && param3==5) 17/07/2023 00:00:00 – 17/07/2023 23:59:59

timestamp (+02:00)	server.timestamp (+0…	ident	position.latitude	position.longitude	position.altitude	position.spe…
17/07/2023 21:44:21	17/07/2023 21:44:21	esp32_birds				
17/07/2023 21:44:41	17/07/2023 21:44:41	esp32_birds				
17/07/2023 21:45:24	17/07/2023 21:45:24	esp32_birds				
17/07/2023 21:46:24	17/07/2023 21:46:24	esp32_birds				
17/07/2023 21:47:41	17/07/2023 21:47:41	esp32_birds				
17/07/2023 21:48:31	17/07/2023 21:48:31	esp32_birds				
17/07/2023 21:49:37	17/07/2023 21:49:37	esp32_birds				
17/07/2023 21:58:24	17/07/2023 21:58:24	esp32_birds				

(source- Wang)

On flespi, all photos taken can be shown under Messages, together with the time of shooting. This page also includes other useful information such as number of messages received per minute, connection status, and possible error message of MQTT broker.

3.3 Troubleshooting Logic Analyser

In order to understand how the effective communication between our modules and sensors is taking place, a logic analyser used to assess the logic curves of the system. And to test the reliability of our PIR sensor was tested with different condition of motions and calibrated to the desired sensitivity. Fig 7 describes the Universal unsynchronized communication between the esp32 camera and on Channel 2 the PIR sensor state shows high when a motion is detected.

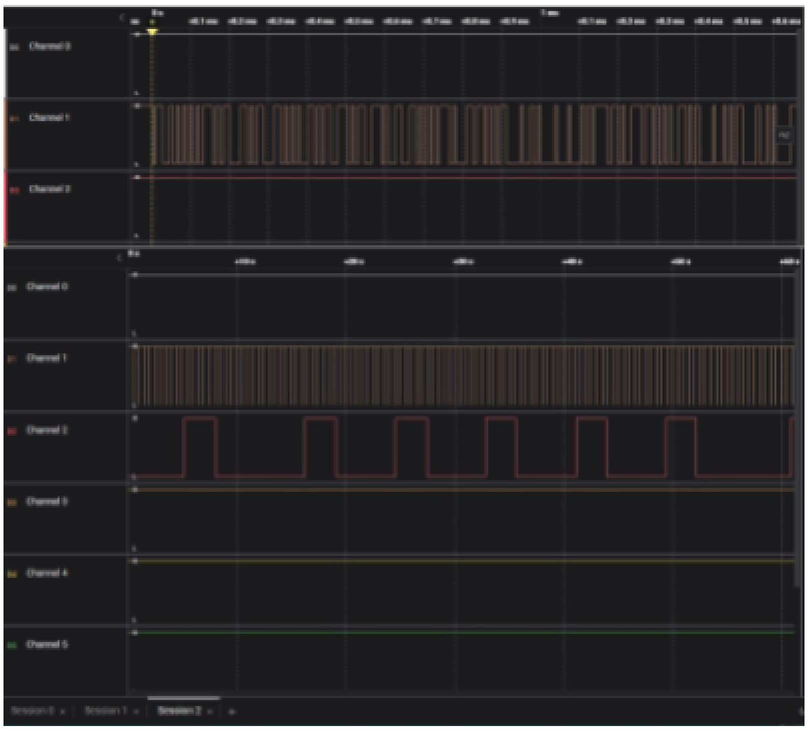


Fig 7: The capture of logic analyser (source -Wai Lin)

3.3 Power Profile

In order to test the prower profile of the system we utilized the Power Profile Kit 2 to test the efficiency of the code and whether the battery can support the system only being charged by the PV module which comprises four cells. The power output on average is measured on average 45mA in direct sunlight conditions and therefore the power consumption is less than that for 2 minutes of 4 image captures and uploading to local SD card and MQTT server. Typically the system would go to sleep for 5 minutes after taking one cycle but here in order to test the reproducibility of the power consumption we run 4 cycles in sequences and they are observed to be similar power curves.

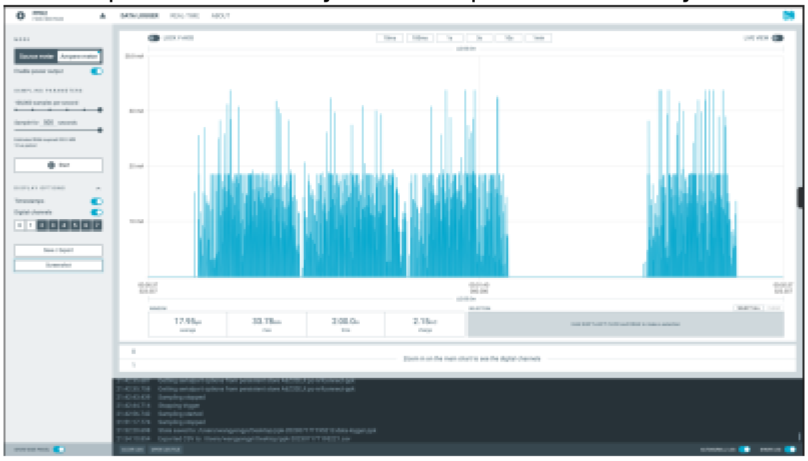
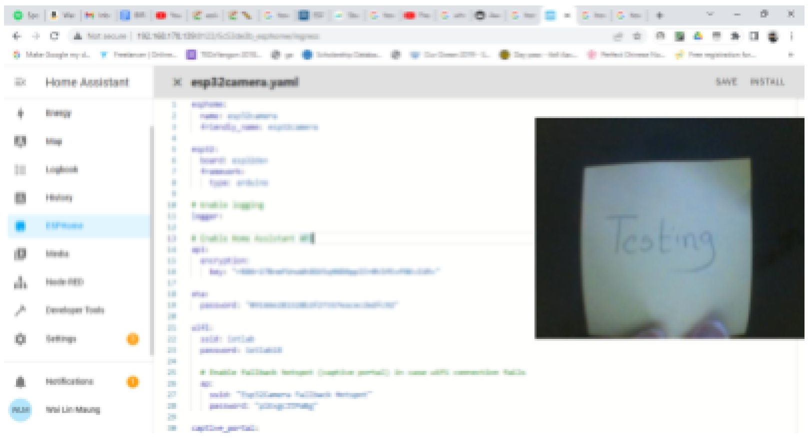


Fig 8: The power curve of 4 photos tanken successively (source - Wang)

3.4 Home assistant integration

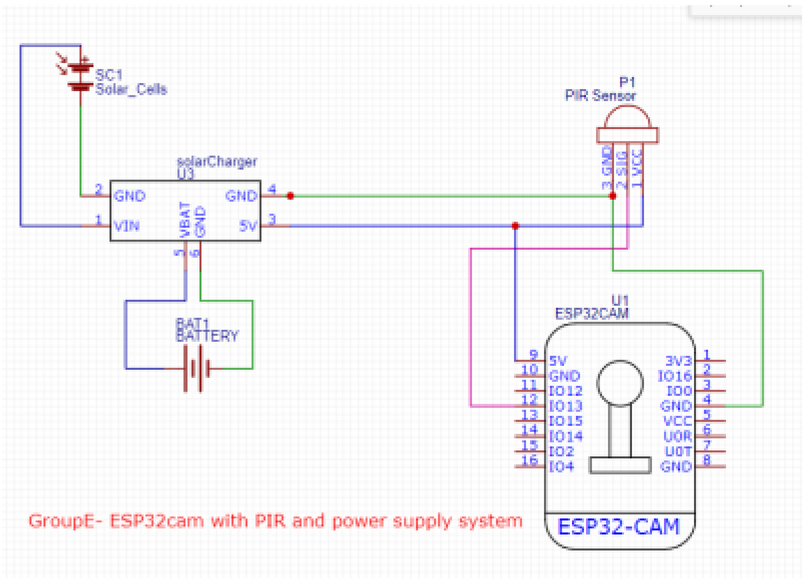
Home assistant is an open source home automation that can control a whole host of devices with custom dashboards. It can host a range of add ons such as Node red, Mqtt dockers, microcontrollers and sensors. The Home assistant server can be run on a Raspberry Pi or a local server. In this project home assistant server was hosted on a docker environment namely Oracle.This intergration allows this project to branch out into numerous possibility OpenCV can be hosted on home assistant to object detect the images taken by our camera modules. A whole range of controls can also be made



```
esphome:
  name: esp32camera
  friendly_name: esp32camera

esp32:
  board: esp32dev
  framework:
    type: arduino

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "[illegible]"

ota:
  password: "[illegible]"

wifi:
  ssid: [illegible]
  password: [illegible]

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Esp32Camera Fallback Hotspot"
    password: "[illegible]"

captive_portal:
```

in one go in home assistant environment.
Fig9: ESP Home configuration assistant (source:Wai Lin)
Via ESP Home add-on, our camera module can be integrated into Home assistant environment. The camera can also be controlled by Mqtt protocol(mosquitto) which can be hosted on HA and by publishing on specific topics the commands can be given.

====4. Discussion===
4.1 Limitations
The flashlight on ESP 32 cam is an indicator for successful photo taking, however, it is too bright for a birdhouse. Animals will be scared away by the flash light. Since birds' vision can see blue, green, red, and UV light. Replacing the flashlight on board by Infrared LED can solve the problem. With IR LED, the light will not be visible by both birds, and thus taking photos of birds without disturbing them.

Moreover, the PIR sensor is based on sense heat signatures, their sensitivity will decrease under warm environments (especially above 35°C). Therefore, IR sensor is not suitable for hot summers and microwave sensor are recommended for such conditions. Further one problem the esp32 cam has with FDTI adaptors is when flashing the device, it goes into time out mode when not reset in timely manner. In order to avoid this, ESP32-Cam- MB module can be utilized, it allows the board to directly via Micro-USB without a programming device.
5. Conclusion


(source-Wai Lin) In the end, the surveillance system achieves the set criteria of inexpensive, low power, reliable design which can be set up and operated without much knowledge prior. The system can be utilized for the purposes of monitoring birds and since the images uploaded are time lapsed, a whole host local bird species can be archived in spacial and time. This project has so much potential since esp32 cam can be utilized for many more purposes such as image recognition and tracking systems and can be taken as far as one wishes to go.

Final code.ino

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "soc/soc.h"           // Disable brownour problems
#include "soc/rtc_cntl_reg.h"  // Disable brownour problems
#include "driver/rtc_io.h"
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include <EEPROM.h>            // read and write from flash memory
#include <PubSubClient.h>
#include <base64.h>
#include <libb64/cencode.h>

// define the number of bytes you want to access
#define EEPROM_SIZE 1

constexpr int kCameraPin_PWDN   =  32;
constexpr int kCameraPin_RESET  =  -1;  // NC
constexpr int kCameraPin_XCLK   =   0;
constexpr int kCameraPin_SIOD   =  26;
constexpr int kCameraPin_SIOC   =  27;
constexpr int kCameraPin_Y9     =  35;
constexpr int kCameraPin_Y8     =  34;
constexpr int kCameraPin_Y7     =  39;
constexpr int kCameraPin_Y6     =  36;
constexpr int kCameraPin_Y5     =  21;
constexpr int kCameraPin_Y4     =  19;
constexpr int kCameraPin_Y3     =  18;
constexpr int kCameraPin_Y2     =   5;
constexpr int kCameraPin_VSYNC  =  25;
constexpr int kCameraPin_HREF   =  23;
constexpr int kCameraPin_PCLK   =  22;

int pictureNumber = 0;


const char* ssid = "FRITZ!Box 7530 MT";
const char* password = "14649735070293466500";


const char* mqtt_server = "mqtt.flespi.io";
const int mqtt_port = 1883;
const char* mqtt_user =
"Ydl9IsQMA5NqDfGWgQt98ebuHLbm1gsPkDNQTtqho2xP51CbHTaGX9YuIZEz3Xdd";
const char* mqtt_password = "";
const char* mqtt_TopicName = "esp32/birds";


framesize_t resolution_ = FRAMESIZE_QVGA;


#define SLEEP_DELAY 0
#define FILE_PHOTO "/photo.jpg"

// OV2640 相机模组的针脚定义
#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 CAMERA_MODEL_AI_THINKER


WiFiClient mqttClient;
PubSubClient client(mqttClient);

const int LED_BUILTIN = 4;


void setup_camera() {

     // OV2640 camera module
     camera_config_t config;
    config.pin_pwdn     = kCameraPin_PWDN;
    config.pin_reset    = kCameraPin_RESET;
    config.pin_xclk     = kCameraPin_XCLK;
    config.pin_sscb_sda = kCameraPin_SIOD;
    config.pin_sscb_scl = kCameraPin_SIOC;
    config.pin_d7       = kCameraPin_Y9;
    config.pin_d6       = kCameraPin_Y8;
    config.pin_d5       = kCameraPin_Y7;
    config.pin_d4       = kCameraPin_Y6;
    config.pin_d3       = kCameraPin_Y5;
    config.pin_d2       = kCameraPin_Y4;
    config.pin_d1       = kCameraPin_Y3;
    config.pin_d0       = kCameraPin_Y2;
    config.pin_vsync    = kCameraPin_VSYNC;
    config.pin_href     = kCameraPin_HREF;
    config.pin_pclk     = kCameraPin_PCLK;
    config.xclk_freq_hz = 20000000;
    config.ledc_timer   = LEDC_TIMER_0;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.pixel_format = PIXFORMAT_JPEG;
    config.frame_size   = FRAMESIZE_SVGA;
    config.jpeg_quality = 10;
    config.fb_count     = 1;

  esp_err_t err = esp_camera_init(&config);
  Serial.printf("esp_camera_init: 0x%x\n", err);

  // sensor_t *s = esp_camera_sensor_get();
  // s->set_framesize(s, FRAMESIZE_QVGA);
  }

String msg;
int timeCount = 0;
void getimg(){//拍照分段发送到mqtt

    camera_fb_t *fb = esp_camera_fb_get();
    if (fb){
        Serial.printf("width: %d, height: %d, buf: 0x%x, len: %d\n",
fb->width, fb->height, fb->buf, fb->len);
        char data[4104];
        //client.publish(mqtt_TopicName, "0");
        for (int i = 0; i < fb->len; i++){

            sprintf(data, "%02X", *((fb->buf + i)));
            msg += data;
           // if (msg.length() == 4096){
           //     timeCount += 1;
           //     client.beginPublish(mqtt_TopicName, msg.length(), 0);
           //     client.print(msg);
           //     client.endPublish();
           //     msg = "";
           // }
        }
        if (msg.length() > 0){
            client.beginPublish(mqtt_TopicName, msg.length(), 0);
            client.print(msg);
            client.endPublish();
            msg = "";
        }
        //client.publish(mqtt_TopicName, "1");
        timeCount = 0;
        esp_camera_fb_return(fb);
    }
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address : ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  setup_camera();
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
      Serial.println("mqtt connected");
    }
}

void publishTelemetry(String data) {
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  boolean Status = client.publish_P( mqtt_TopicName, (const uint8_t*)
data.c_str(), data.length(), true);
  Serial.println(String(Status ? "Successful" : "Error") );
}

void capturePhoto( void ) {
  // Retrieve camera framebuffer
  camera_fb_t * fb = NULL;
  uint8_t* _jpg_buf = NULL;
  esp_err_t res = ESP_OK;
  size_t frame_size = 0;
  Serial.print("Capturing Image...");

  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  fb = esp_camera_fb_get();
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
  if (!fb) {
    Serial.println("Camera capture failed");
    res = ESP_FAIL;
  } else {
    Serial.println("Done!");
    Serial.println(String("Size of the image...") + String(fb->len));
    if (fb->format != PIXFORMAT_JPEG) {
      Serial.println("Compressing");
      bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &frame_size);
      esp_camera_fb_return(fb);
      fb = NULL;
      if (!jpeg_converted) {
        Serial.println("JPEG compression failed");
        res = ESP_FAIL;
      }
    } else {
      frame_size = fb->len;
      _jpg_buf = fb->buf;
      Serial.print("Publish photo...");

      publishTelemetry(base64::encode(_jpg_buf, fb->len));
      Serial.println("Done!");
      esp_camera_fb_return(fb);
    }
  }
  if (res != ESP_OK) {
    return;
  }
```

```
//Serial.println("Starting SD Card");
  if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
  }

  uint8_t cardType = SD_MMC.cardType();
  if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
    return;
  }

    // initialize EEPROM with predefined size
  EEPROM.begin(EEPROM_SIZE);
  pictureNumber = EEPROM.read(0) + 1;
  // Path where new picture will be saved in SD Card
  String path = "/picture" + String(pictureNumber) +".jpg";

  fs::FS &fs = SD_MMC;
  Serial.printf("Picture file name: %s\n", path.c_str());

  File file = fs.open(path.c_str(), FILE_WRITE);
  if(!file){
    Serial.println("Failed to open file in writing mode");
  }
  else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
    EEPROM.write(0, pictureNumber);
    EEPROM.commit();
  }
  file.close();
  esp_camera_fb_return(fb);

}


void loop() {
  Serial.println("PSRAM found: " + String(psramFound()));

  if (!client.connected()) {
    reconnect();
  }
  if (client.connected()) {
    capturePhoto();
  }
  client.loop();

  Serial.println("Going to sleep now");
  if (SLEEP_DELAY == 0) {
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 1);
    delay(1000);
    esp_deep_sleep_start();
  }

  if (SLEEP_DELAY > 0) {
    delay(SLEEP_DELAY);
    delay(30000);
  }
}
```