CHENG-CHIH LEE

LightSwarm

CHENG-CHIH LEE

2025/11/02

#Summary

This project is built upon the previous lab and creates a swarm to connect at most three ESP32's and a Raspberry Pi (as a data logger). The RPi will have a button and four LEDs (Red, Green, Yellow, White). The ESP will have an analog light sensor (photoresistor in the package).

The following links show the reference code for lightswarm logic.

- **ESP 8266 LightSwarm.ino Download LightSwarm.ino(obtained from Prof. Shovic's github)**
- **Raspberry Pi LightSwarm.py Download LightSwarm.py(obtained from Prof. Shovic's github)**
  - **Python 3 version: github.com/switchdoclabs/SDL_Pi_LightSwarmLinks to an external site.**

Each of the three ESP32's in the swarm is identical. There are no software differences and no hardware differences. They can communicate with each other by broadcasting messages to exchange sensor readings. One of them, the ESP32 with the highest reading, will become the "Master" and will forward the readings to the RPi for data logging.

***I teared down the legacy code and fit it into FreeRTOS architecture.**

(key words: ESP32, Raspberry Pi 5, UDP, Photocell Sensor, GPIO, FreeRTOS, threading)

#OUTLINE

Version History

Version	*Date*	*Comment*	*Known Issue/Fix*
v01_Beta	11/04/2025	Beta version	

#Components used

1. **Photocell with a 10k pull-down resistor.**
2. **External LEDs. Need resistors (330 Ohm).**
3. **A button. Need a pull-down resistor (10k Ohm), digital input and with interrupt mode set to that pin.**

#Pin Definition for *ESP32*

Pin Name	Definition	Usage and comments
LED_BUILTIN	GPIO 2	Built in onboard LED (fixed)
PHOTO_CELL	GPIO 34	ADC Sampling pin for Photocell sensor

Table.1 Pin Definition

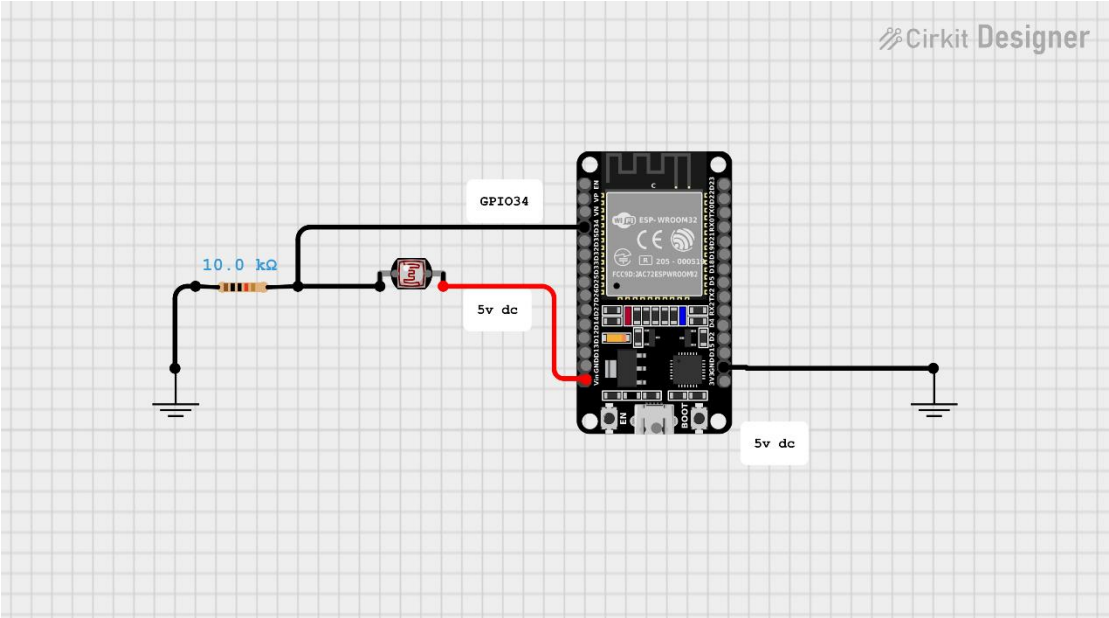


Fig.1 Schematics

#Pin Definition for *Raspberry Pi*

Pin Name	Definition	Usage and comments
LED_R	GPIO 26	Pin for Red External LED
LED_Y	GPIO 13	Pin for Red External YELLOW
LED_G	GPIO 6	Pin for Red External GREEN
LED_W	GPIO 5	Pin for Red External WHITE
BTN	GPIO 16	Pin for External Push Button

Table.2 Pin Definition

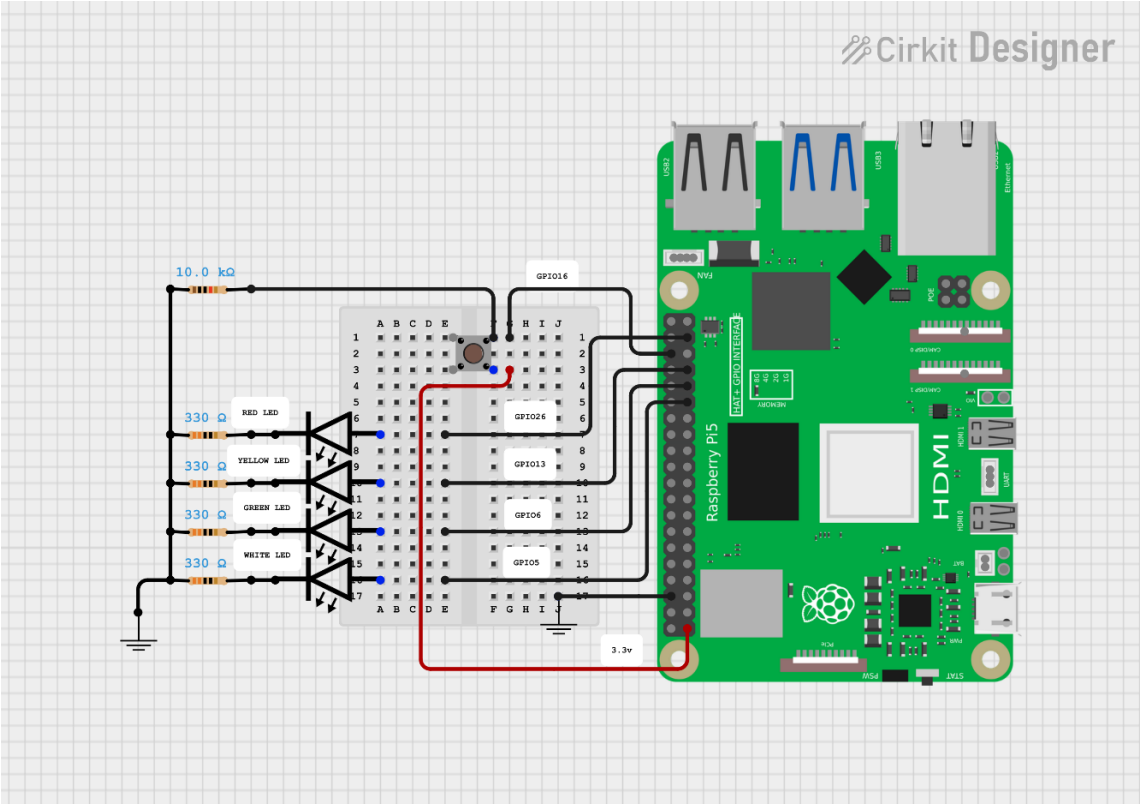
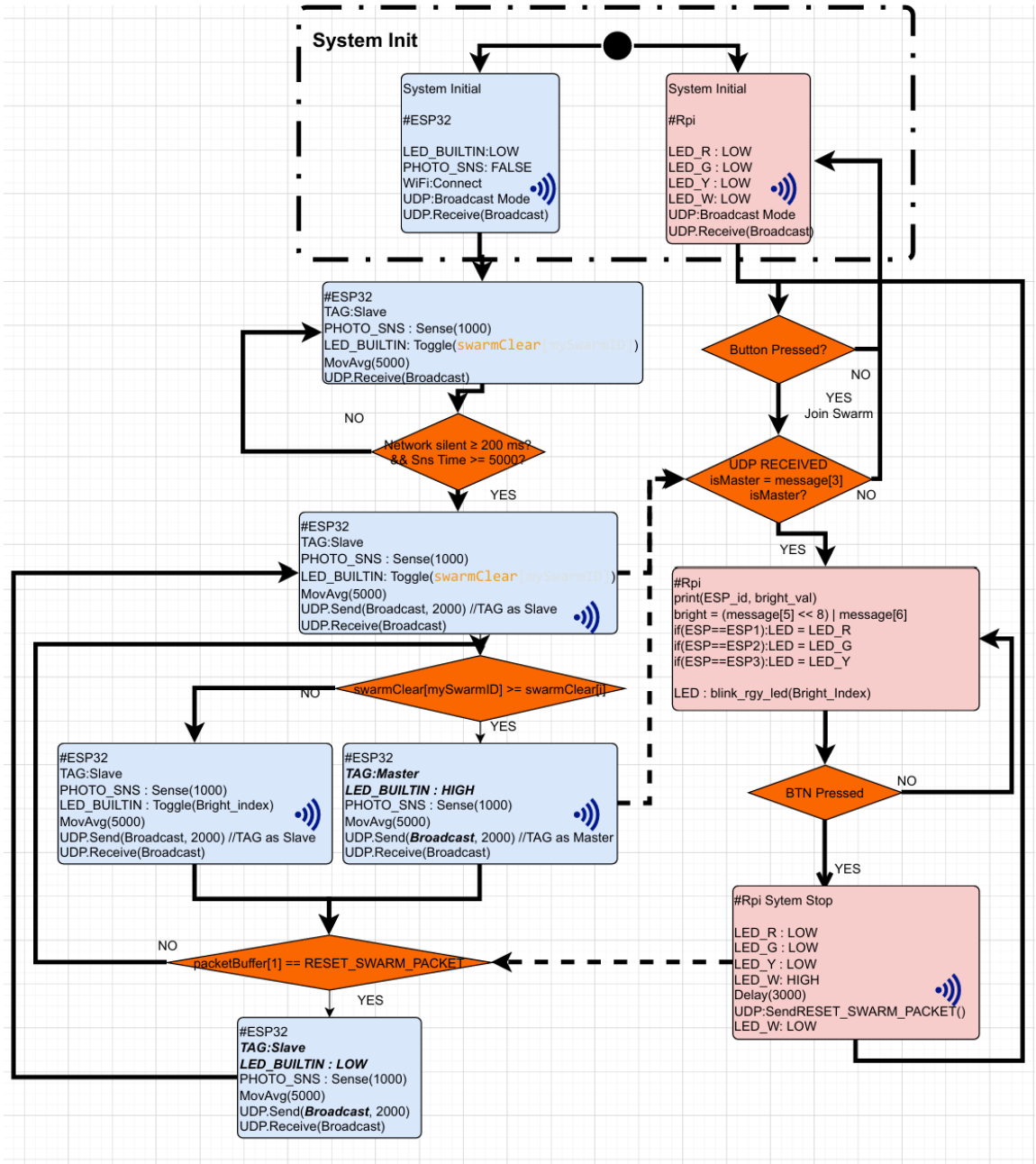
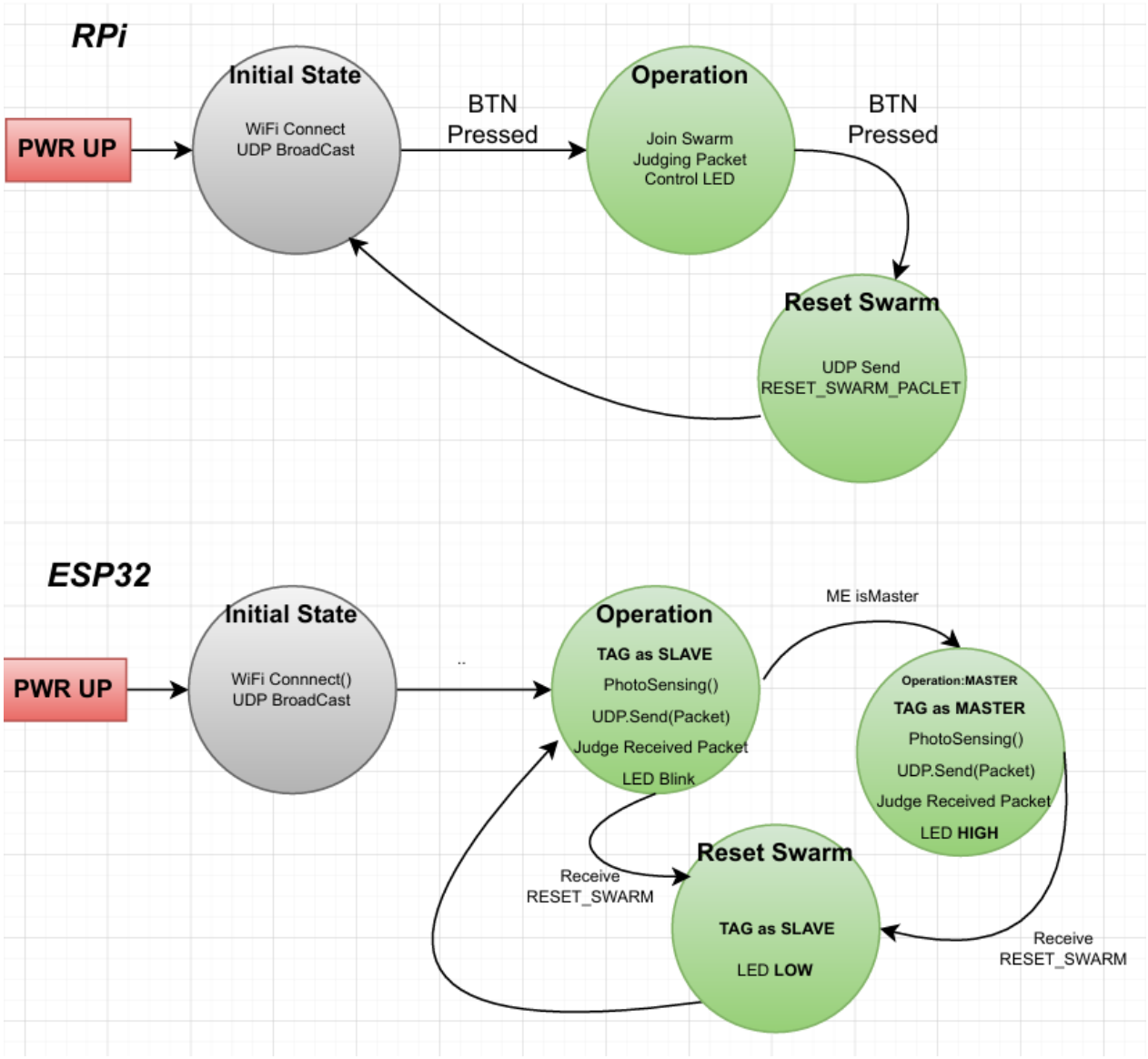


Fig.2 Schematics

#Overall Flow-chart



#State Machine



#Protocols

During my development, UDP broadcasting is configured. That means all devices in the network should be able to receive all packets sent. No IP is hardcoded in any device. Port is set as 1996.

Laptop：IP ----, Port 1996
RPi　　：IP ----, Port 1996
ESP32：IP ----, Port 1996

Byte	Field	Value	Description
0	0xF0	Constant	**Start-of-frame marker**. Helps receivers validate the packet type.
1	LIGHT_UPDATE_PACKET (=0)	0	**Packet type ID** — distinguishes from reset, blink, etc.
2	localIP[3]	e.g., 123	**Node ID** — the last octet of its IP (unique in local subnet). Used as swarm identifier.
3	masterState	0 or 1	**Role flag:** 1 = master, 0 = slave.
4	VERSIONNUMBER	e.g., 28	**Firmware version** for compatibility tracking.
5–6	clearColor (high, low)	sensor value	**Clear (luminance)** 16-bit reading from the photocell (moving average).
7–8	redColor (high, low)	Not used	Placeholder for **red channel intensity**. (Unused here since no color sensor.)
9–10	greenColor (high, low)	Not used	Placeholder for green.
11–12	blueColor (high, low)	Not used	Placeholder for blue.
13	0x0F	0x0F	**End-of-frame marker** (symmetrical with start marker).

Part 1-1 Raspberry Pi WiFi setup and packet delivery

Files: talk_v03.py, UDP_v03.py, state_machine_v03.py, LightSwarm.py

In this project, the LightSwarm functionality is modified/integrated into the previous "we_need_to_talk" project. talk_v03 works as the main file, initializing some threads. UDP_v03 works as the UDP rx/tx library module. The state machine module is modified to keep only two states, including operation and reset Swarm. And the RPi is connected to my personal hotspot named IPhoneKL. WiFi related functionalities are developed in the ***"UDP_ v03.py"*** module.

#WiFi Setup and UDP

For the UDP setup, it's set to broadcast mode, thus sending to 255.255.255.255. port is 1996.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
#Kevin Lee 10/11/2025
#v03 Kevin Lee 11/03/2025 Modified for LightSwarm Project
import socket
print("####################################")
print("!!!!!!!!!!!!!!!!!!Hi!!!!!>0<!!!!!!!!")
print("####################################")
#UDP_IP = input("Enter Listening IP or Enter to default 0.0.0.0: ") or "0.0.0.0"
#UDP_PORT = 8386
#Laptop_UDP_IP = input("Enter Laptop IP or Enter to default 172.20.10.4: ") or "1
#Laptop_UDP_PORT = 823
#ESP32_UDP_IP = input("Enter ESP32 IP or Enter to default 172.20.10.2: ") or "172
#ESP32_UDP_PORT = 1996
UDP_IP   = "255.255.255.255"
UDP_PORT = 1996

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)   # ? allow broadcast
sock.bind((UDP_IP, UDP_PORT))
```

In the main file (***talk_ v03.py***), I put UDP_Receive() and processPacket() to ***thread*** to make sure it keeps listening.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
#Kevin Lee 10/11/2025
#v03 Kevin Lee 11/03/2025 Modified for LightSwarm Project
import time
import threading
import UDP_v03 as UDP
import state_machine_v03 as STATE
import LightSwarm as LS

if __name__=='__main__':

    STATE.gpio_setup()
    receiver_thread = threading.Thread(target=UDP.UDP_Receive, daemon=True)
    receiver_thread.start()# Put UDP listening to thread to ensuring listening
    processPacket_thread = threading.Thread(target=LS.processPacket, daemon=True)
    processPacket_thread.start()
    print("#####################################################")
    print("System Up! Listening to LightSwarm Packets!!")
    print("#####################################################")

    try:
        while True:
            time.sleep(0.5)  # keep main thread alive
    except KeyboardInterrupt:
        print("Exiting...")
```

This UDP module also works as a layer for state_machine and LightSwarm to exchange the LSCommand. The set functionset ***LSCommand(cmd)*** is for writing to this layer; ***get*** functions are for accessing values.

```
40  def get_message_r():
41      #for getting the received UDP message
42      return message_r
43
44  def get_new_msg_cnt():
45      #for getting the UDP message incoming status
46      return new_msg_cnt
47
48  def getLSCommand():
49      global LScmd
50      return LScmd
51
52  def setLSCommand(cmd):
53      global LScmd
54      LScmd = cmd
```

For UDP **sending**, I developed ***UDP_Send()*** method to send messages to both my laptop and ESP32 for the beta version. This method will later be called by the ***state_maching*** module.

For UDP **receiving**, I developed ***UDP_Receive()*** method to receive messages. Now RPi is listening to all UDPs with this method put to thread. Upon receiving UPDs, it also updated the ***new_msg_cnt*** for checking message incoming status.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
19
20  message_r = "", ""
21  new_msg_cnt = 0
22  LScmd = ""
23
24  def UDP_Send(message):
25      #BroadCast UDP
26      sock.sendto(message, (UDP_IP, UDP_PORT))#.encode()
27
28      print(f"BroadCasting UDP Message:'{message}' to {UDP_IP}:{UDP_PORT}")
29
30  def UDP_Receive():
31      global message_r
32      global new_msg_cnt
33
34      while True:
35          data, addr = sock.recvfrom(1024)
36          message_r = (data, addr)  #store tuple (bytes, address)
37          new_msg_cnt += 1
38          #print(f"Received {len(data)} bytes from {addr}: {list(data)}")
39
40  def get_message_r():
41      #for getting the received UDP message
42      return message_r
43
44  def get_new_msg_cnt():
45      #for getting the UDP message incoming status
46      return new_msg_cnt
47
48  def getLSCommand():
49      global LScmd
50      return LScmd
51
52  def setLSCommand(cmd):
53      global LScmd
54      LScmd = cmd
```

Part 1-2 Rpi reacts to received packets

#State Machine

In the State machine, 4 state transitions are developed. The ***button_callback()*** is attached to external push button input interrupt for triggering state change.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
28  blink_stop      = threading.Event() #for stop led blink
29  blink_rgy_stop  = threading.Event() #for stop led rgy blink
30  photosns_stop   = threading.Event() #for stop photo sense
31
32  def button_callback(channel):
33      state_machine()
34
35  def state_machine():
36      global sys_state
37
38      if sys_state == 0: #from init to operation
39          m_operation()
40
41      elif sys_state == 1:  #from norm to down
42          photosns_stop.set()
43          blink_rgy_stop.set()
44          m_pwr_dwn() #to pwr down
45          sys_state = 0 #put back to initial stat
46
47      elif sys_state == 2: #from err
48          # Stop blinking
49          blink_stop.set()
50          sys_state = 0
51
```

#Operation Mode *m_operation()*

This state is triggered when status flag ***sys_state == 0*** and the **button is pressed**. The sys_state is then set to 1 indicating it's in operation mode. The phototsensing and RGY led indicating tasks will be put to threading.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
129 def m_operation():
130     global sys_state
131     global message_r
132     err = 0
133     
134     sys_state = 1
135     GPIO.output(led_w, GPIO.HIGH)
136     print("##################")
137     print("In operation mode")
138     print("##################")
139 
140     photosns_stop.clear() #clear the stop event
141     photosns_thread = threading.Thread(target=photo_sns, daemon=True)
142     photosns_thread.start()
143     
144     blink_rgy_stop.clear() #clear the stop event
145     blink_RGY_thread = threading.Thread(target=blink_rgy_led, daemon=True)
146     blink_RGY_thread.start()
147 
148     
149     time.sleep(1)
```

#Photocell value process *photo_sns()*

When photo_sns() is put to thread, with ***sys_state==1 and the thread is not stopped***. This method checks if the packet is from a master device or not, clear led indicator thread stop flag and it also shuts down leds accordingly.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
57  def photo_sns():
58      global new_msg_cnt
59      global pre_msg_cnt
60      global led_ind
61      global bright_value
62      value = 0
63      print("#################")
64      print("In photo sns now")
65      print("#################")
66
67      while sys_state==1 and not photosns_stop.is_set():
68          pre_msg_cnt = new_msg_cnt
69          device_id, isMaster, value = LS.getLSMasterBright()
70          #print("photo_sns Receiving Packet. device_id, isMaster, value
71          new_msg_cnt = UDP.get_new_msg_cnt()
72
73          if(isMaster):
74              blink_rgy_stop.clear()
75              bright_value = value
76              print("bright_value = ",bright_value, ", value= ", value)
77              if device_id is 0:
78                  led_ind = led_g
79                  #print("LED indicator is:", led_g, '\n')
80                  GPIO.output(led_r, GPIO.LOW)
81                  GPIO.output(led_y, GPIO.LOW)
82              elif device_id is 1:
83                  led_ind = led_y
84                  #print("LED indicator is:", led_y, '\n')
85                  GPIO.output(led_r, GPIO.LOW)
86                  GPIO.output(led_g, GPIO.LOW)
87              elif device_id is 2:
88                  led_ind = led_r
89                  #print("LED indicator is:", led_r, '\n')
90                  GPIO.output(led_y, GPIO.LOW)
91                  GPIO.output(led_g, GPIO.LOW)
92
93          time.sleep(0.5)
```

#Power Down mode *m_pwr_dwn()*

Works as reset Swarm task

During normal operation, when the external button is pressed, the state machine will generate the photo_sns() and blink_rgy_stop() thread stopping event and sent **UDP.setLSCommand("RESETSWARM")**.

state_machine_v03.py | talk_v03.py | UDP_v03.py | LightSwarm.py

```
152 def m_pwr_dwn():
153     global sys_state
154     sys_state = 3
155     
156     UDP.setLSCommand("RESETSWARM")#setting LS cmd to UDP layer
157     print("###################")
158     print("Reset Swarm")
159     print("###################")
160     GPIO.output(led_r, GPIO.HIGH)
161     GPIO.output(led_y, GPIO.HIGH)
162     GPIO.output(led_g, GPIO.HIGH)
163     GPIO.output(led_w, GPIO.HIGH)
164     time.sleep(0.5)
165     GPIO.output(led_r, GPIO.LOW)
166     time.sleep(0.5)
167     GPIO.output(led_y, GPIO.LOW)
168     time.sleep(0.5)
169     GPIO.output(led_g, GPIO.LOW)
170     time.sleep(1.5)
171     GPIO.output(led_w, GPIO.LOW)
172     #Enter pwr_dwn procedure
```

The operation method will then put RPi back to initial state to wait for the button pressed to join the Swarm again.

Part 2-1 ESP WiFi setup and packet delivery

ESP is connected to my hotspot. It is also developed to UDP broadcasting.

LightSwarm.ino

```
12  char ssid[] = "iPhoneKL";  //  your network SSID (name)
13  char pass[] = "[redacted]"; // your network password
14
15  unsigned int localPort = 1996; // local port to listen for UDP packets
16  IPAddress serverAddress = IPAddress(255, 255, 255, 255); // default no IP Address
```

```
114  Serial.print("Connecting to ");
115  Serial.println(ssid);
116  WiFi.begin(ssid, pass);
117
118  // initialize Swarm Address - we start out as swarmID of 0
119
120  while (WiFi.status() != WL_CONNECTED) {
121    delay(500);
122    Serial.print(".");
123  }
124  Serial.println("");
125
126  Serial.println("WiFi connected");
127  Serial.println("IP address: ");
128  Serial.println(WiFi.localIP());
129
130  Serial.println("Starting UDP");
131
132  udp.begin(localPort);       // bind & listen on port
133  //udp.setBroadcast(true);     // allow broadcast TX
```

Swarm data array is initialized here.

```
137  // initialize light sensor and arrays
138  int i;
139  for (i = 0; i < SWARMSIZE; i++)
140  {
141    swarmAddresses[i] = 0;
142    swarmClear[i] = 0;
143    swarmTimeStamp[i] = -1;
144  }
145  swarmClear[mySwarmID] = 0;
146  swarmTimeStamp[mySwarmID] = 1;   // I am always in time to myself
147  clearColor = swarmClear[mySwarmID];
148  swarmVersion[mySwarmID] = VERSIONNUMBER;
149  swarmState[mySwarmID] = masterState;
150  Serial.print("clearColor =");
151  Serial.println(clearColor);
152  // set SwarmID based on IP address
153  localIP = WiFi.localIP();
154  swarmAddresses[0] =  localIP[3];
155  mySwarmID = 0;
156  Serial.print("MySwarmID=");
157  Serial.println(mySwarmID);
```

#Tasks

After WiFi connection, six tasks are created. **UDP receiving Task** for UDP receiving, **Task_PhotoSns** for Photocell sensing, **Task_UDP_Send** for sending UDP packet. **Task_Swarm** for Swarm logic implementation. **Task_LED_Blink** for controlling Built-In led blinking behavior. **Task_Monitor** for checking stack watermark.

```
159  /*Mutesx to protect shared swarm data field*/
160  swarmMutex = xSemaphoreCreateMutex();
161  /*Task Creation*/
162  xTaskCreatePinnedToCore(Task_PhotoSns, "Task_PhotoSns", 4096, NULL, 1, &xHandle_Task_PhotoSns, 1);
163  xTaskCreatePinnedToCore(Task_UDP_Receive, "Task_UDP_Receive", 4096, NULL, 2, &xHandle_Task_UDP_Receive, 1);
164  xTaskCreatePinnedToCore(Task_UDP_Send, "Task_UDP_Send", 4096, NULL, 1, &xHandle_Task_UDP_Send, 1);
165  xTaskCreatePinnedToCore(Task_Swarm, "Task_Swarm", 4096, NULL, 1, &xHandle_Task_Swarm, 1);
166  xTaskCreatePinnedToCore(Task_LED_Blink, "Task_LED_Blink", 4096, NULL, 1, &xHandle_Task_LED_Blink, 1);
167  xTaskCreatePinnedToCore(Task_Monitor, "Task_Monitor", 4096, NULL, 1, NULL, 1);
```

#UDP Receiving *UDP_Receive(void)*

In the UDP_Receive() function, UDP incoming messages are processed by calling the Packet_Helper function. There's a mutex set to protect the incoming packet.

```
356  void Task_UDP_Receive(void *pvParameters){
357
358    for(;;){
359      int cb = udp.parsePacket();
360      if (cb) {
361        udp.read(packetBuffer, PACKET_SIZE);
362        xSemaphoreTake(swarmMutex, portMAX_DELAY);
363  // update swarm arrays and clearColor
364        Incoming_Packet_Helper(packetBuffer);
365        xSemaphoreGive(swarmMutex);
366      }
367      vTaskDelay(100 / portTICK_PERIOD_MS);
368    }
369  }
```

#UDP Sending *UDP_Send(const char* msg)*

In the UDP_Send() function, broadcastARandomUpdatePacket() is called and sendLightUpdatePacket() is called accordingly.

```
371  void Task_UDP_Send(void *pvParameters){
372    for(;;){
373      vTaskDelay(1);                    // let RX run first
374      broadcastARandomUpdatePacket();
375      vTaskDelay(1000 / portTICK_PERIOD_MS);
376    }
377  }
```

```
206  void broadcastARandomUpdatePacket()
207  {
208    int sendToLightSwarm = 255;
209    int randomDelay;
210    randomDelay = random(0, MAXDELAY);
211    Serial.print("Delay = ");
212    Serial.print(randomDelay);
213    Serial.print("ms : ");
214
215    vTaskDelay(randomDelay / portTICK_PERIOD_MS);
216
217    IPAddress sendSwarmAddress(255, 255, 255, sendToLightSwarm); // my Swarm Address
218    sendLightUpdatePacket(sendSwarmAddress);
219  }
```

Part 2-2 ESP correctly reacts to received packets

#Task_Swarm

The Task_Swarm is for checking the main logic for Light Swarm by calling checkAndSetIfMaster().

```
379 void Task_Swarm(void *pvParameters){
380   for(;;){
381     // Check to see if I am master!
382     checkAndSetIfMaster();
383     vTaskDelay(500 / portTICK_PERIOD_MS);
384   }
385 }
```

```
221 void checkAndSetIfMaster()
222 {
223   int i;
224   int howLongAgo;
225   for (i = 0; i < SWARMSIZE; i++)
226   {
227
228 #ifdef DEBUG
229     Serial.print("swarmClear[");
230     Serial.print(i);
231     Serial.print("] = ");
232     Serial.print(swarmClear[i]);
233     Serial.print("  swarmTimeStamp[");
234     Serial.print(i);
235     Serial.print("] = ");
236     Serial.println(swarmTimeStamp[i]);
237 #endif
238     Serial.print("#");
239     Serial.print(i);
240     Serial.print("/");
241     Serial.print(swarmState[i]);
242     Serial.print("/");
243     Serial.print(swarmVersion[i]);
244     Serial.print(":");
245     // age data
246     howLongAgo = millis() - swarmTimeStamp[i]
```

#Photocell sensing function

The ***Photo_Sns(void)*** function is developed to execute sensing with analog reading. A moving average function **MovAvg(uint16_t input, uint16_t *buffer)** is used to derive 5 sec sliding window average.

LightSwarm.ino

```
530
531  int Photo_Sns(void){
532    int val = 0u;
533
534    val = analogRead(PHOTO_SNS_PIN);
535    val = MovAvg(val, &PHOTO_READINGS[0]);/*Moving average*/
536    data_cnt+=1;
537    sprintf(photo_sns_buffer, "%d", val);  /*convert int → text*/
538    Serial.print("Photo Sns Read=");
539    Serial.print(photo_sns_buffer);
540    Serial.println(" ");
541    return val;
542  }
543
544  uint16_t MovAvg(uint16_t input, uint16_t *buffer){
545    /*Buffer with 5 elements*/
546    float avg = 0.0f;
547
548    buffer[4] = buffer[3];
549    buffer[3] = buffer[2];
550    buffer[2] = buffer[1];
551    buffer[1] = buffer[0]; /*pop out old element and push new*/
552    buffer[0] = input;
553
554    avg = (buffer[0]+buffer[1]+buffer[2]+buffer[3]+buffer[4])/5;
555
556    return avg;
557  }
```