ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

UNIVERSITY OF WEST ATTICA

Σχολή Μηχανικών

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διαδίκτυο των Αντικειμένων

Τελική Εργασία

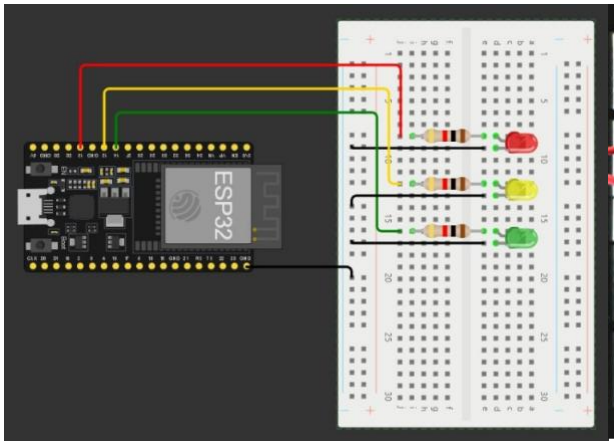
ΚΟΝΤΟΥΛΗΣ ΔΗΜΗΤΡΙΟΣ 21390095

ΜΕΝΤΖΕΛΟΣ ΑΓΓΕΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 21390132

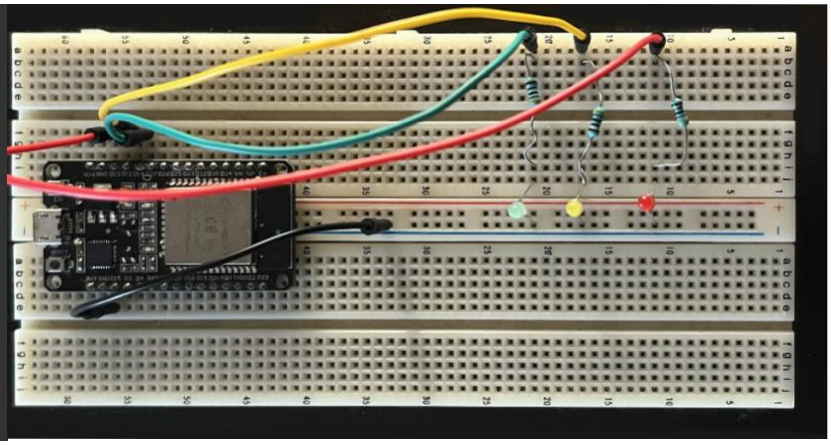
ΕΥΦΡΟΣΥΝΗ ΒΑΡΣΟΥ 21390021

Εισαγωγή

Η εφαρμογή αποτελείται από έναν φωτεινο σηματοδότη (3 LED διαφορετικού χρώματος) του οποίου η λειτουργία ορίζεται από τον μικροελεγκτή ESP32 και απεικονίζεται στην πλατφόρμα IoT ThingSpeak. Η λειτουργικότητα της εφαρμογής αναλύεται σε τρία στάδια που θα αναλύσουμε στην συνέχεια σε κάθε δραστηριότητα. Ειδικότερα, στην πρώτη δραστηριότητα περιγράφεται πως γίνεται ο προγραμματισμός εναλλαγής των LED καθώς και η αποστολή της καταστασής τους σε κανάλι του ThingSpeak. Στην δευτερη και τρίτη δραστηριότητα απλά προσθέτουμε έξτρα λειτουργίες (μεταβλητή ειδοποιήσης) που αφορούν στην πλατφόρμα του ThingSpeak.



Εικόνα 1: Προσομοιωτικη Απεικόνιση Κυκλώματος



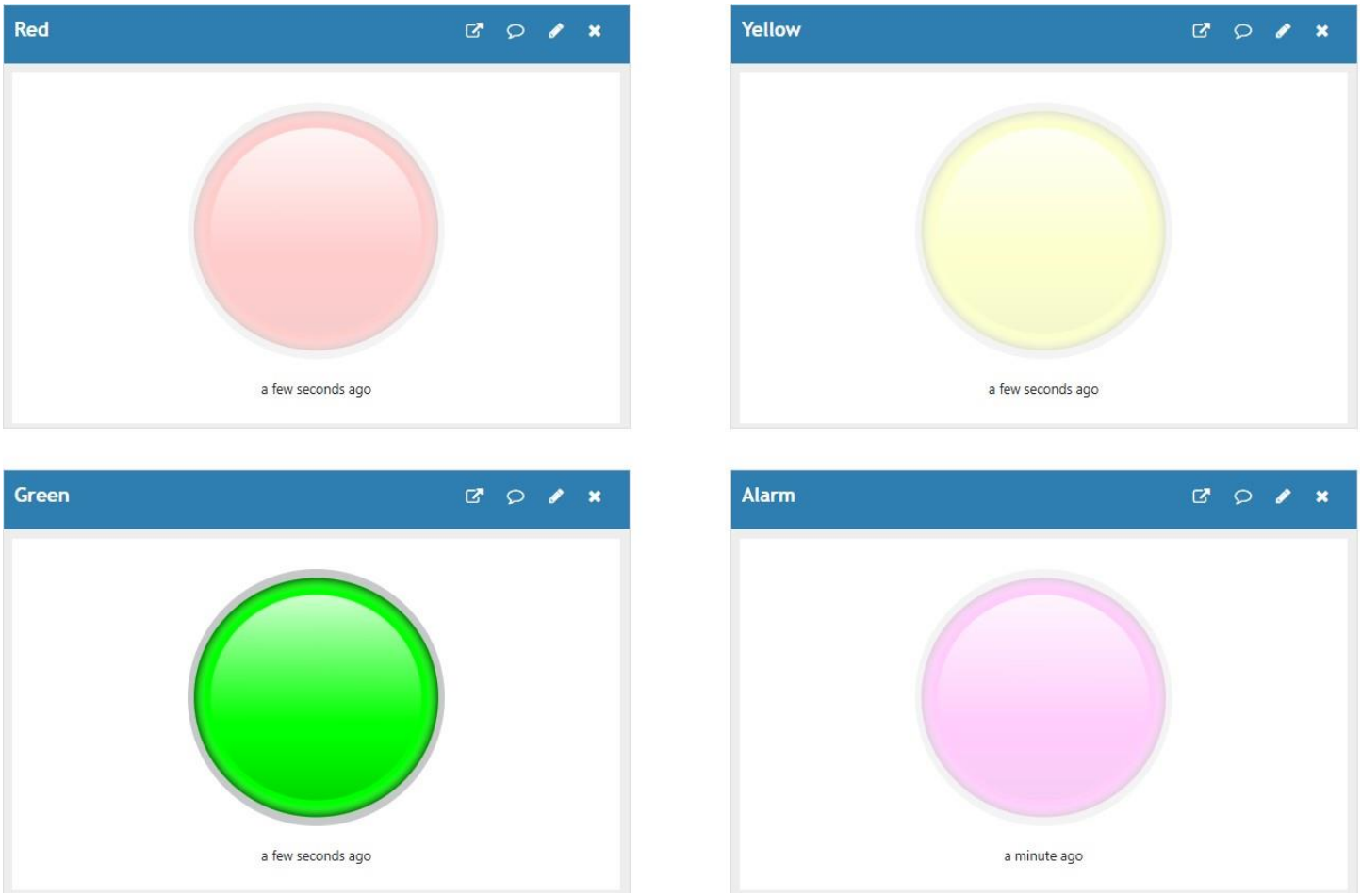
Εικόνα 2: Πραγματική Απεικόνιση Κυκλώματος

Υλοποιήση Κυκλώματος Υλικού

Για την υλοποίηση του κυκλώματος επιλέξαμε να χρησιμοποιήσουμε τον μικροελεγκτή ESP32 καθώς έχει ενσωματωμένη την δυνατότητα σύνδεσης Wi-Fi καθιστώντας έτσι εύκολα την σύνδεση στην πλατφόρμα IoT. Επίσης, δεν χρειαζόταν να αλλάξουμε πλατφόρμα προγραμματισμού κάθως το ESP32 χρησιμοποιέι και αυτό την πλατφορμα Arduino IDE. Η υλοποιήση του κυκλώματος είναι σχετικά απλή (Εικόνα 1,2). Κάθε LED συνδέεται με μία έξοδο GPIO του ESP32 και μια αντίσταση σε σειρά, η οποία περιορίζει το ρεύμα που περνάει μέσα από το LED και το αρνητικό άκρο του κάθε LED συνδέεται με το GND.

Κανάλι Thingspeak

Για την προβολή τον LED από το κανάλι του ThingSpeak, αρχικά στις ρυθμίσεις καναλιού(Channel Settings) ενεργοποιήσαμε τα πρώτα τρία field (Field1, Field2, Field3) και στην συνέχεια δημιουργήσαμε τρία Widgets «Lamp Indicators» αναθέτοντας για κάθε field ένα widget με διαφορετικό χρώμα (Εικόνα 3). Επιπλέον, όπως θα δούμε στις δραστηριότητες 2,3 έχουμε και ένα τέταρτο widget σε χρώμα μωβ που αντιστοιχει στο field8 το οποίο αποτελεί την μεταβλητη ειδοποιήσης.



Εικόνα 3: Κανάλι Thingspeak

Δραστηριότητα 1. Ηλεκτρικός φωτεινός σηματοδότης

```
#include <WiFi.h>
#include <HTTPClient.h>
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_PASSWORD";
String api = "YOUR_API_KEY";

// LED pin setup
int greenLED = 14;  // Pin for green LED
int yellowLED = 12; // Pin for orange LED
int redLED = 13;   // Pin for red LED

void setup() {
  Serial.begin(115200);
  // Set up LED pins
  pinMode(greenLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(redLED, OUTPUT);

  WiFi.begin(ssid,password);
  Serial.print("Connecting to WiFi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");
}
```

```
void sendData(String green, String orange, String red) {
  String url = "http://api.thingspeak.com/update?api_key=" + api + "&field1=" + green + "&field2=" + orange +
"&field3=" + red;
  if (WiFi.status() == WL_CONNECTED) { // Check if connected to WiFi
    HTTPClient http;
    http.begin(url); // Specify the URL

    int httpResponseCode = http.GET(); // Send the GET request

    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
    } else {
      Serial.print("Error on sending request: ");
      Serial.println(httpResponseCode);
    }

    http.end(); // Free resources
  } else {
    Serial.println("WiFi Disconnected");
  }
}

void controlTrafficLight(String state) {
  if (state == "green") {
    digitalWrite(greenLED, HIGH);
    digitalWrite(yellowLED, LOW);
    digitalWrite(redLED, LOW);
  } else if (state == "orange") {
    digitalWrite(greenLED, LOW);
    digitalWrite(yellowLED, HIGH);
    digitalWrite(redLED, LOW);
  } else if (state == "red") {
    digitalWrite(greenLED, LOW);
    digitalWrite(yellowLED, LOW);
    digitalWrite(redLED, HIGH);
  }
}

void loop() {
  // Simulate traffic light control
  controlTrafficLight("red");
  sendData("0", "0", "1");
  delay(30000);

  controlTrafficLight("green");
  sendData("1", "0", "0");
  delay(30000);

  controlTrafficLight("orange");
  sendData("0", "1", "0");
  delay(20000);
}
```

Σε αυτήν την δραστηρίοτητα, ορίζουμε την βασική δομή της εφαρμογής η οποία προγραμματίζει τις εναλλαγες των LED και στέλνει την κατασταση τους στο κανάλι ThingSpeak. Αρχικά, ορίζουμε το Wifi SSID και Password που θα συνδέσουμε το ESP32 καθώς και το Write API Key του καναλιού Thingspeak στο οποίο θα ενημερώνουμε τις μεταβλητές.

Στην συνάρτηση **setup()**, αναθέτουμε κάθε LED με τα αντίστοιχα GPIO pins που έχουμε συνδέσει στο κύκλωμα. Στην συνέχεια, αρχικοποιηούμε την σύνδεση Wifi με την συνάρτηση **Wifi.begin()** και μέσω της επανάληψης **while (WiFi.status() != WL_CONNECTED)** τσεκάρουμε αν έχουμε επιτυχής σύνδεση και όταν βρεθεί βγαίνουμε από την while έτσι **έχοντας** εξασφαλίσει επιτυχής σύνδεση στο Internet.

Στην συνάρτηση **sendData(),** δημιουργούμε το URL που θα χρησιμοποιήσουμε για να στείλουμε τα δεδομένα στο ThingSpeak. Το URL περιλαμβάνει το Write API key του καναλιού και τις τιμές των LED που θέλουμε να ενημερώσουμε. Ελέγχουμε αν είμαστε συνδεδεμένοι στο Wifi και αν ναι, δημιουργούμε ένα αντικείμενο HTTPClient και χρησιμοποιούμε την begin() για να ορίσουμε το URL. Στην συνέχεια, στέλνουμε το αίτημα GET και ελέγχουμε τον κωδικό απόκρισης για να δούμε αν το αίτημα ήταν επιτυχές και αν ναι το τυπώνουμε. Τέλος, καλούμε την μέθοδο end() για να ελευθερώσουμε τους πόρους που χρησιμοποιήθηκαν.

Η συνάρτηση **controlTrafficLight()**, είναι υπεύθυνη για τον έλεγχο των κατάστασεων των LED. Δέχεται ως παράμετρο ένα string που καθορίζει ποιο LED είναι σε κατάσταση ON ("green", "orange", "red"). Ανάλογα με την τιμή της, η συνάρτηση ενεργοποιεί το αντίστοιχο LED και απενεργοποιεί τα υπόλοιπα. Αυτό επιτυγχάνεται με τη χρήση της συνάρτησης digitalWrite() που ορίζει την κατάσταση των GPIO pins στα οποία είναι συνδεδεμένα τα LED.

Η συνάρτηση **loop()** είναι υπεύθυνη για την επαναλαμβανόμενη εκτέλεση των εντολών που ελέγχουν την εναλλαγή των LED και στέλνουν τα δεδομένα στο ThingSpeak. Αρχικά, καλούμε τη συνάρτηση controlTrafficLight() με την παράμετρο "red" για να ενεργοποιήσουμε το κόκκινο LED και να απενεργοποιήσουμε τα υπόλοιπα. Στη συνέχεια, καλούμε τη συνάρτηση sendData() με τις τιμές "0", "0", "1" για να ενημερώσουμε το ThingSpeak ότι το κόκκινο LED είναι ενεργοποιημένο. Μετά από μια καθυστέρηση 20 δευτερολέπτων (20000 milliseconds), επαναλαμβάνουμε τη διαδικασία για το πράσινο και το πορτοκαλί LED, με τις αντίστοιχες τιμές για το ThingSpeak και τις αντίστοιχες καθυστερήσεις.

Δραστηριότητα 2. Αποστολή δεδομένων σε κανάλι άλλης εφαρμογής

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_PASSWORD";
String api = "YOUR_WRITE_API_KEY"; // Write Thingspeak Channel API Key
String sec_api = "OTHER_WRITE_API_KEY"; // Write Thingspeak Channel API Key for the other app

// LED pin setup
int greenLED = 14;  // Pin for green LED
int yellowLED = 12; // Pin for orange LED
int redLED = 13;   // Pin for red LED

void setup() {
  Serial.begin(115200);
  // Set up LED pins
  pinMode(greenLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(redLED, OUTPUT);

  WiFi.begin(ssid,password);
  Serial.print("Connecting to WiFi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");
  // Set field8 to 0 for both channels
  setField8("0", sec_api);
  setField8("0", api);
}
```

```
void setField8(String value, String api) {
  // Construct and send the HTTP GET request
  String url = "http://api.thingspeak.com/update?api_key=" + api + "&field8=" + value;
   if (WiFi.status() == WL_CONNECTED) { // Check if connected to WiFi
    HTTPClient http;
    http.begin(url); // Specify the URL

    int httpResponseCode = http.GET(); // Send the GET request

    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
    } else {
      Serial.print("Error on sending request: ");
      Serial.println(httpResponseCode);
    }

    http.end(); // Free resources
  } else {
    Serial.println("WiFi Disconnected");
  }
}
```

Σε αυτήν την δραστηριότητα, τροποποιούμε την υπάρχουσα δομή της πρώτης δραστηριότητας και προσθέτουμε την δυνατότητα ορισμού τιμής για μια μεταβλητή άλλου καναλιού. Ειδικότερα, θα θέσουμε στην μεταβλητή Field8 (Μωβ Widget Alarm) του καναλιού με API Key == sec_api την τιμή 0 καθώς και στο κανάλι μας με API Key == api. Οι συναρτήσεις **sendData(), controlTrafficLight()** και **loop()** παραμένουν ίδιες με την πρώτη δραστηρίοτητα και για αυτό το λόγο δεν θα αναλυθούν ξανά.

Η συνάρτηση **setup()** παραμένει ίδια με την πρώτη δραστηριότητα, με μόνη διαφορά την κλήση της συνάρτησης setField8() και για τα δύο κανάλια, που αρχικοποιεί την τιμή του field8 σε 0. Η συνάρτηση **setField8()** είναι υπεύθυνη για την αποστολή της τιμής value (0 ή 1) στο field8 του καναλιού με παράμετρο api.Η λειτουργία της είναι ανάλογη της sendData() με την διαφορά μόνο στο URL που θα κληθει για το GET request. Το URL που καλεί αλλάζει την μεταβλητή field8 για όποιο κανάλι έχει δοθεί ως παράμετρος.

Δραστηριότητα 3. Ανάγνωση δεδομένων σε κανάλι άλλης εφαρμογής

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
String api = "YOUR_WRITE_API_KEY"; // Write Thingspeak Channel API Key
String read_api_key = "YOUR_READ_API_KEY"; // Read API key for ThingSpeak
String sec_api = "OTHER_WRITE_API_KEY"; // Write Thingspeak Channel API Key for the other app
String channel_id = "YOUR_CHANNEL_ID"; // Channel ID for ThingSpeak

// Traffic light durations
unsigned long redTime = 30000;       // Red light duration time
unsigned long greenTime = 30000;     // Green light duration time
unsigned long orangeTime = 20000;    // Orange light duration time

// Variables
unsigned long previousMillis = 0;            // used to track last light for the normal traffic light cycle
unsigned long previousUpdateMillis = 0;      // used to track update timer
unsigned long previousFieldCheckMillis = 0;  // used to track field 8 value check timer
unsigned long updateInterval = 600000;       // Update interval (every 10 minutes)
unsigned long alertDuration = 60000;         // Alert mode duration (1 minute)
unsigned long checkInterval = 5000;          // Read ThingSpeak Channel interval
int currentLight = 0;                        // used to control traffic light cycles. 0 = Red, 1 = Green, 2 = Orange
```

```
void setup() {
  Serial.begin(115200);

  // Setup LED pins
  pinMode(greenLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(redLED, OUTPUT);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected");

  delay(2000);

  setField8(0, sec_api);
  setField8(0, api); // Initially set field8 value to 0
  delay(15000); //! Overcome Thingspeak API limit

  // Initialize first stage of traffic light cycle
  controlTrafficLight("red");
  sendData("0", "0", "1");

  previousMillis = millis();
}

int getField8() {
  String url = "http://api.thingspeak.com/channels/" + channel_id + "/fields/8/last?api_key=" + read_api_key;
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(url);
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      String payload = http.getString();
      http.end();
      delay(100);
      return payload.toInt();
    } else {
      Serial.println("Error reading ThingSpeak Channel");
      http.end();
      delay(100); // Delay to handle failed read
      return -1;
    }
  }
  return -1;
}

void setField8(int value, String api) {
  unsigned long currentMillis = millis();

  String url = "http://api.thingspeak.com/update?api_key=" + api + "&field8=" + String(value);
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(url);
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      Serial.print("Updated Field 8 to: ");
      Serial.println(value);
      delay(500); // Allow time for the ThingSpeak channel to process the update
    } else {
      Serial.println("Error updating Field 8");
      delay(500); // Handle error gracefully with a delay
    }
    http.end();
  } else {
    Serial.println("WiFi Disconnected");
  }
}
```

```
void loop() {
  unsigned long currentMillis = millis();

  // Update field8 every 10 minutes
  if (currentMillis - previousUpdateMillis >= updateInterval) {
    previousUpdateMillis = currentMillis;
    // Set field8 to 1 or 0 (depending on your logic)
    setField8(1); // Or setField8(0) depending on the condition
    delay(1000);
    Serial.println("Field 8 has been updated");
  }

  // Check for keystroke to enter alert mode
  if (Serial.available() > 0) {
    char input = Serial.read();
    if (input == 'e') {
      setField8(1); // Update field8 to 1
    }
  }

  // Check field8 value at the specified interval
  if (currentMillis - previousFieldCheckMillis >= checkInterval) {
    previousFieldCheckMillis = currentMillis;  // Update the last field check time

    int field8Value = getField8();
    Serial.print("Current Field 8 Value: ");
    Serial.println(field8Value);

    // If the field8 value is 1, enter alert mode
    if (field8Value == 1) {
      Serial.println();
      Serial.println("Field 8 value detected. Value changed to 1. Entering alert mode...");
      controlTrafficLight("orange"); // turn on orange light
      delay(15000); //! Overcome ThingSpeak API limit
      sendData("0", "1", "0");
      delay(alertDuration);
      Serial.println("Alert mode duration ended");
      setField8(0);  // Reset field8 to 0 after alert mode ends
      delay(15000);  //! Overcome ThingSpeak API limit
      return;  // Skip normal light cycle if in alert mode
    }
  }

  // Normal light cycle
  if (currentLight == 0 && currentMillis - previousMillis >= redTime) {
    currentLight = 1; // Red to Green
    previousMillis = currentMillis;
    controlTrafficLight("green");
    sendData("1", "0", "0");
  } else if (currentLight == 1 && currentMillis - previousMillis >= greenTime) {
    currentLight = 2; // Green to Orange
    previousMillis = currentMillis;
    controlTrafficLight("orange");
    sendData("0", "1", "0");
  } else if (currentLight == 2 && currentMillis - previousMillis >= orangeTime) {
    currentLight = 0; // Orange to Red
    previousMillis = currentMillis;
    controlTrafficLight("red");
    sendData("0", "0", "1");
  }
}
```

Ζητούμενο της δραστηριότητας, αποτελεί η **ανάγνωση** του πεδίου 8 (Field 8) από το αντίστοιχο κανάλι της πλατφόρμας ThingSpeak. Ανάλογα με την τιμή του πεδίου αυτού θα πραγματοποιούνται αλλαγές στον τρόπο λειτουργίας του σηματοδότη. Για την ανάγνωση του έγινε ανάπτυξη συνάρτησης (getField8) η οποία αξιοποιεί το Read API του ThingSpeak μέσω του οποίου μπορούμε να στείλουμε αίτημα **HTTP GET** και να μας επιστρέψει τα δεδομένα που θα του απαιτήσουμε. Όπως στα προηγούμενα μέρη 1 & 2 χρησιμοποιήσαμε ένα URL για να γράψουμε δεδομένα στο Write Channel της πλατφόρμας, έτσι και εδώ το αξιοποιούμε έτσι ώστε να

ανακτήσουμε δεδομένα από το αντίστοιχο **Read Channel**. Επιπλέον, χρειαζόμαστε το **Channel ID** της πλατφόρμας το οποίο βρίσκεται στα Channel Settings του ThingSpeak.

Για την εμπλοκή της συνάρτησης με το υπόλοιπο πρόγραμμα έπρεπε να πραγματοποιηθούν μερικές αλλαγές, έτσι ώστε να επιλυθούν κάποια προβλήματα, κυρίως συγχρονισμού. Στα προηγούμενα μέρη έγινε χρήση της εντολής delay(), η οποία μπορεί να διακόψει την ροή του προγράμματος. Δεδομένο λοιπόν ότι η κατάσταση του σηματοδότη μπορεί να αλλάξει οποτεδήποτε με βάση την τιμή που έχει το Field 8, η εντολή delay καθιστά τον συνεχή έλεγχο του πεδίου αδύνατο. Αν δηλαδή η τιμή του πεδίου άλλαζε όσο το φαναρι εκτελεί τον κανονικό κύκλο του, το πρόγραμμα δεν θα μπορούσε να μεταβάλλει την κατάσταση του επειδή πρακτικά βρίσκεται “παγωμένο” λόγω της delay(). Για αυτό έπρεπε να γίνει αλλαγή στην λογική χρονισμού του σηματοδότη, έτσι ώστε να μπορεί να μεταβληθεί η λειτουργία του σε οποιαδήποτε χρονική στιγμή. Για την επίλυση του προβλήματος έγινε χρήση της εντολής **millis()**, η οποία επιστρέφει πόσα millisecond έχουν περάσει από την εκτέλεση κάποιου συμβάντος (π.χ την εκκίνηση του προγράμματος). Σε συνδυασμό με τις μεταβλητές **previousMillis, previousUpdateMillis, previousFieldCheckMillis,** καθώς και **updateInterval, alertDuration** και **checkInterval,** ουσιαστικά μπορούμε να δημιουργήσουμε **non-blocking timers**. Ας δούμε ένα παράδειγμα ώστε να γίνει κατανοητή η λειτουργία τους.

Τρόπος λειτουργίας των non-blocking timers

Τρέχοντας την εντολή millis, τότε αρχίζει η μέτρηση του χρόνου που έχει τρέξει το πρόγραμμα. Έχοντας αρχικοποιήσει μία βοηθητική μεταβλητή με μηδέν αλλά και τον χρόνο που επιθυμούμε να γίνει ο εκάστοτε έλεγχος (interval), μπορούμε να αφαιρέσουμε από τον συνολικό χρόνο που εκτελείται το πρόγραμμα την μεταβλητή που έχουμε αρχικοποιήσει, και να συγκρίνουμε το αποτέλεσμα αυτό με ένα διάστημα (interval) π.χ 5 sec. Οπότε αν περάσουν π.χ 5200 millisecond από την έναρξη του non-blocking timer, τότε 5200 - 0 >= 5000. Όταν η συνθήκη αυτή γίνει αληθής, τότε εντός της δομής ελέγχου, η βοηθητική μεταβλητή (έστω previousMillis) παίρνει την τιμή millis. Οπότε τώρα το timer θα φτάσει σε σημείο στο οποίο θα ισχύει η συνθήκη 10300 - 5000 >= 5000. Επόμενος παρατηρούμε ότι αναπτύσσεται ένα μοτίβο με το οποίο μπορούμε να ξέρουμε κάθε πότε περνάνε 5 δευτερόλεπτα, χωρίς την διακοπή του προγράμματος. Έτσι λοιπόν γίνεται η χρήση non-blocking timers, οι οποίοι είναι ιδιαίτερα σημαντικοί για το συγκεκριμένο μέρος, καθώς μπορεί ο κύκλος του φαναριού να μεταβληθεί οποτεδήποτε παρατηρηθεί αλλαγή στο πεδίο 8.

Τρόπος λειτουργίας των κύκλου σηματοδότη

Για την αλλαγή μεταξύ των διαφόρων καταστάσεων του κύκλου, χρησιμοποιείται η μεταβλητή **currentLight**. Εφόσον χρησιμοποιούνται **non-blocking timers** για την αλλαγή χρώματος του σηματοδότη, χωρίς την ύπαρξη της μεταβλητής, ο χρονισμός των φαναριών δεν θα ήταν σωστοί. Η εκκίνηση του κύκλου ξεκινάει από το κόκκινο φανάρι εντός της εντολής **setup()** και έπειτα συνεχίζει τελείως αυτόνομα εντός της **loop()** με τους προαναφερόμενους μηχανισμούς (**currentLight** & **non-blocking timers**).

Παρουσία και αντιμετώπιση λοιπών περιορισμών/προβλημάτων

Άλλο ένα πρόβλημα το οποίο υπό τις συνθήκες υλοποίησης της άσκηση φάνηκε αναπόφευκτο, είναι ο περιορισμός που έχει το Write API του ThingSpeak. Υπάρχει ένα διάστημα 15 δευτερολέπτων μεταξύ διαδοχικών αιτημάτων. Στην εφαρμογή αυτή πρέπει αρχικά να στείλουμε ένα αίτημα με το οποίο θα αλλάζουμε

την τιμή του πεδίου 8 σε από 0 σε 1. Όταν εκτελεστεί επιτυχώς το αίτημα αυτό, το πρόγραμμα θα εντοπίσει την αλλαγή αυτή και το φανάρι θα αλλάξει το state του σε Alert Mode.

Στο Alert Mode, το φανάρι κρατάει το φανάρι του πορτοκαλί μέχρι να ξαναπάρει την τιμή 0 (στη συγκεκριμένη περίπτωση έπειτα από το πέρας ενός χρονικού διαστήματος).

Όταν εντοπιστεί η αλλαγή αυτή, το API δεν θα μπορεί ακόμα να δεχθεί το αίτημα αλλαγής του σηματοδότη σε πράσινο, επειδή έχουμε ήδη στείλει με την εντολή setField αίτημα αλλαγή της τιμής του πεδίου 8. Για την επίλυση το προβλήματος αυτού, έγινε εισαγωγή καθυστέρησης με την εντολή **delay()** έτσι ώστε το πρόγραμμα να περιμένει έως ότου το API να είναι ξανά διαθέσιμο για χρήση. Αφού τελειώσει η καθυστέρηση, το πρόγραμμα στέλνει μέσω του API αίτημα αλλαγής του σηματοδότη σε πορτοκαλί χρώμα, ώστε να φανεί και στη πλατφόρμα του ThingSpeak.

Αφου γίνει αυτό, επόμενο βήμα αποτελεί η αλλαγή της τιμής του πεδίου 8 από 1 σε 0. Με την ίδια λογική, θα στείλουμε αίτημα στο ThingSpeak για την αλλαγή αυτή και έπειτα θα εισάγουμε μία καθυστέρηση ώστε να βεβαιωθούμε ότι μπορούμε να στείλουμε ξανά αίτημα εγγραφής μέσω του Write API. Η καθυστέρηση είναι απαραίτητη, διότι αφού αλλάξει η τιμή του πεδίου σε 0, το πρόγραμμα θα εντοπίσει την αλλαγή και θα ξεκινήσει τον κανονικό κύκλο του φαναριού. Όταν όμως εκτελείται ο κύκλος αυτός, τότε πραγματοποιείται αίτημα εγγραφής μέσω της **sendData()**, ώστε να αλλάξει χρώμα ο σηματοδότης και στο ThingSpeak. Η καθυστέρηση λοιπόν διαβεβαιώνει ότι έχει περάσει αρκετός χρόνος ώστε τα δύο αυτά αιτήματα να γίνουν με πλήρη επιτυχία και χωρίς να διακοπούν.