



ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ (Α.Π.Θ.)

ΗΥ0901 ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΕΣ ΚΑΙ ΠΕΡΙΦΕΡΕΙΑΚΑ

Εργασία Arduino

Αντωνιάδης Δημήτριος (8462): akdimitri@auth.gr
Δημητριάδης Βασίλειος (**AEM**): dimvasdim@auth.gr

15 Μαΐου 2019

Περιεχόμενα

1 Εισαγωγή.	2
2 Εργασία στον μικροελεγκτή <i>Arduino</i>.	2
2.1 Λειτουργία προγράμματος.	2
2.2 Τλικό και συνδεσμολογία.	2
2.3 Πηγαίος κώδικας.	3
2.4 Αποτελέσματα εργασίας του μικροελεγκτή <i>Arduino</i>	7
3 Εργασία στον μικροελεγκτή <i>Wemos D1 R2</i>.	8
3.1 Λειτουργία προγράμματος.	8
3.2 Τλικό και συνδεσμολογία.	8
3.3 Πηγαίος κώδικας.	9
3.4 Αποτελέσματα εργασίας του μικροελεγκτή <i>Wemos D1 R2</i>	14
4 Επίλογος.	16

1 Εισαγωγή.

Το παρόν έγγραφο αποτελεί την αναφορά της εργασίας *Arduino* που πραγματοποιήθηκε στο πλαίσιο του μαθήματος *Mικροεξεργαστές και Περιφερειακά* του 8^{ου} εξαμήνου. Σκοπός της εργασίας ήταν η δημιουργία ενός έξυπνου θερμομέτρου-θερμοστάτη. Πιο συγκεκριμένα, με τη χρήση του μικροελεγκτή *Arduino* σε συνδυασμό με αισθητήρες θερμότητας και εγγύτητας έπρεπε να αναπτυχθεί ένα σύστημα το οποίο θα μετρούσε την θερμοκρασία του χώρου ανά τακτά χρονικά διαστήματα και θα την παρουσίαζε τοπικά μέσω μιας LCD οιδόνης.

Παραπάνω έγινε μία σύντομη περιγραφή της εργασίας. Στην επόμενη (2^η) ενότητα παρουσιάζεται η λειτουργία του προγράμματος στον μικροελεγκτή *Arduino*. Εντός της δεύτερης ενότητας παρουσιάζονται ο τρόπος λειτουργίας του προγράμματος, τα εξαρτήματα που χρησιμοποιήθηκαν και η συνδεσμολογία τους. Ακόμη, στην ενότητα αυτή παρουσιάζεται ο κώδικας που χρησιμοποιήθηκε για την εκτέλεση του προγράμματος και τα αποτελέσματα των μετρήσεων. Στην τρίτη (3^η) ενότητα παρουσιάζεται η λειτουργία του προγράμματος στον μικροελεγκτή *Wemos D1 R2*. Ομοίως με τη δεύτερη ενότητα παρουσιάζονται οι αντίστοιχες πτυχές. Στην τέταρτη (4^η) ενότητα παρουσιάζονται τα συμπεράσματα της εργασίας αυτής και ο επίλογος.

Η δεύτερη υλοποίηση στον μικροελεγκτή *Wemos D1 R2* πραγματοποιήθηκε με σκοπό την εκπλήρωση του υποερωτήματος που αφορούσε την αποστολή e-mail για την άνοδο της θερμοκρασίας πάνω από μία ορισμένη τιμή. Και αυτό γιατί ο μικροελεγκτής *Arduino* που διατίθεται δεν είχε τη δυνατότητα σύνδεσης στο διαδίκτυο. Ωστόσο, ο μικροελεγκτής *Wemos D1 R2* έχει μόλις 8 pins GPIO και για το λόγο αυτό στη δεύτερη υλοποίηση δεν περιλαμβάνονται οι ενέργειες αναφορικά με τα LEDs.

Ο συνολικός κώδικας πέρα από τα δύο βασικά αρχεία που παρουσιάζονται στην εργασία αυτή περιλαμβάνεται στον εξής σύνδεσμο:

<https://github.com/akdimitri/Microprocessors-and-Peripherals-Arduino-Project.git>

2 Εργασία στον μικροελεγκτή *Arduino*.

Η παρούσα ενότητα περιγράφει την υλοποίηση την εργασίας σε μικροελεγκτή *Arduino*.

2.1 Λειτουργία προγράμματος.

Το πρόγραμμα που εκτελείται στον μικροελεγκτή *Arduino* μετράει κάθε 5 δευτερόλεπτα τη θερμοκρασία του περιβάλλοντος. Αν η θερμοκρασία αυτή είναι μεγαλύτερη από μία συγκεκριμένη τιμή (**extremeHigh**) τότε ανάβει το **κόκκινο** LED. Αντίστοιχα, αν είναι μικρότερη από μία τιμή (**extremeLow**) τότε ανάβει το **μπλέ** LED. Σε κάθε περίπτωση τυπώνεται το αντίστοιχο μήνυμα στην οιδόνη LCD.

Κάθε 24 μετρήσεις της θερμοκρασίας περιβάλλοντος, δηλαδή κάθε δύο (2) λεπτά, υπολογίζεται η μέση τιμή των 24 αυτών μετρήσεων. Η μέτρηση αυτή τυπώνεται στην οιδόνη LCD και παραμένει για δέκα (10) δευτερόλεπτα στην οιδόνη και δεν μπορεί να τυπωθεί κάτι άλλο κατά τη διάρκεια αυτών των δέκα δευτερολέπτων. Αν η μέση τιμή είναι μεγαλύτερη από μία συγκεκριμένη τιμή **extremeFan**, τότε ανάβει το **κόκκινο** LED. Το LED αυτό προσομοιώνει τη λειτουργία ενός ανεμιστήρα.

Επιπλέον, το πρόγραμμα αυτό περιλαμβάνει και την υλοποίηση ενός **Overflow Interrupt Vector** του **Timer2**. Συγκεκριμένα, το Interrupt αυτό χρησιμοποιείται για τη μέτρηση της απόστασης του κοντινότερου αντικείμενου από τον αισθητήρα εγγύτητας κάθε 0.5 δευτερόλεπτα. Αν η απόσταση αυτή είναι μικρότερη από μία συγκεκριμένη τιμή (10 cm) τότε τυπώνεται στην οιδόνη LCD η προηγούμενη μέση τιμή της θερμοκρασίας και η τελευταία μέτρηση της θερμοκρασίας που πραγματοποιήθηκε.

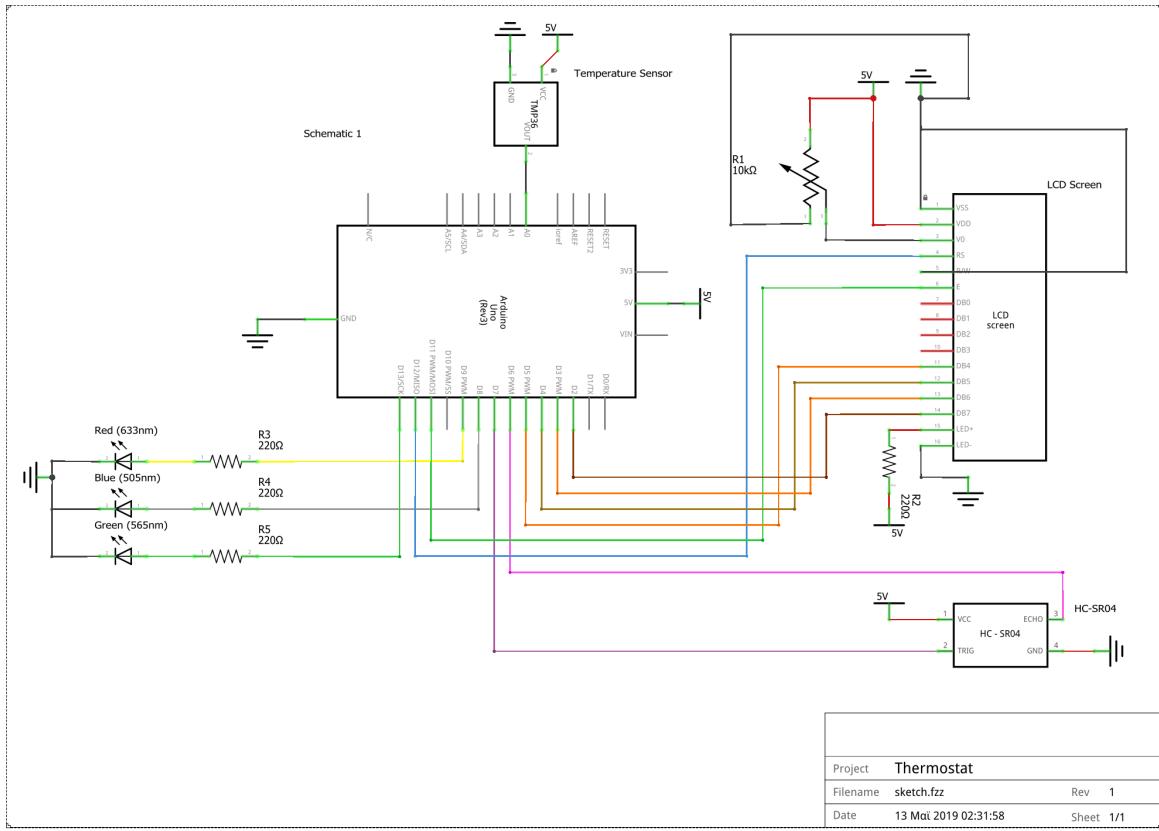
2.2 Υλικό και συνδεσμολογία.

Για την υλοποίηση του παραπάνω προγράμματος χρησιμοποιήθηκαν τα εξής εξαρτήματα:

- Arduino UNO
- Αισθητήρας θερμότητας TMP36 [1]
- Οιδόνη LCD διαστάσεων 16x2 JHD659 M10 1.1 [2]

- Αισθητήρας εγγύτητας HC-SR04 [3]
- Ποτενσιόμετρο 10 kΩ
- 3 LEDs

Τα παραπάνω εξαρτήματα συνδέθηκαν σύμφωνα με την εξής συνδεσμολογία:



Σχήμα 1: Συνδεσμολογία εξαρτημάτων με μικροελεγκτή Arduino.

Με βάση τη συνδεσμολογία αυτή υλοποιήθηκε και ο πηγαίος κώδικας που εκτελεί το πρόγραμμα.

2.3 Πηγαίος κώδικας.

Στην υποενότητα αυτή παρουσιάζεται ο πηγαίος κώδικας. Με πράσινο είναι τα σχόλια, με ροζ οι εντολές και οι τύποι δεδομένων και με μοβ τα Strings.

```

1  /*
2   * Authors: 1. Dimitrios Antoniadis
3   *          2. Vasileios Dimitriadis
4   *
5   * email:   1. akdimitri@auth.gr
6   *          2. dimvasdim@auth.gr
7   *
8   * University: Aristotle University of Thessaloniki (AUTH)
9   * Semester: 8th
10  * Subject:   Microprocessors and Peripherals
11  *
12  * Parts required:
13  * - one Arduino UNO
14  * - one TMP36 temperature sensor
15  * - one LCD 16x2 JHD659 M10 1.1
16  * - one HC-SR04 proximity sensor
17 */
18
19 // include the library code:
20 #include <LiquidCrystal.h>
21

```

```

22 // functions declaration
23 void checkExtremeValues( float temperature, int i);
24 void printLCD( String line1, String line2);
25 void checkExtremeFan( float averageTemperature);
26 int checkProximity();
27
28 // initialize the LCD function with the numbers of the interface pins
29 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
30
31 /* Global Variables Declaration */
32 const int sensorPin = A0, BLUE = 8, RED = 9;
33 const int GREEN = 13, trigPin = 7, echoPin = 6;
34 const float extremeLow = 21.5, extremeHigh = 22.5, extremeFan = 23;
35 float temperature[24], averageTemperature, sum = 0;
36 int i, isSomeoneClose = 0, counter = 0;
37 unsigned long duration, cm, inches;
38 bool FAN = false;
39
40
41
42 /* setup(): This function is executed only once at the POWER ON
43 * or at RESET
44 */
45 void setup() {
46
47     /* Define PINs Mode */
48     pinMode(BLUE, OUTPUT);
49     pinMode(RED, OUTPUT);
50     pinMode(GREEN, OUTPUT);
51     digitalWrite(RED, LOW);
52     digitalWrite(BLUE, LOW);
53     digitalWrite(GREEN, LOW);
54
55     /* Setup LCD */
56     lcd.begin(16, 2);
57     lcd.print("ARDUINO");
58     lcd.setCursor(0, 1);
59     lcd.print("PROJECT");
60
61     /* Setup HC-SR04 */
62     pinMode(trigPin, OUTPUT);
63     pinMode(echoPin, INPUT);
64
65     /* Enable TIMER2 OVERFLOW INTERRUPT */
66     TIMSK2 = (TIMSK2 & B11111110) | 0x01;           // Arduino UNO has an ATmega328P CPU
67     TCCR2B = (TCCR2B & B00111000) | 0x07;           // Thus, ENABLE bit TIMSK:0 and SET prescaler to clk
68         /1024
69
70     /* Setup Serial Communication */
71     Serial.begin(9600);
72
73     /* Enable Interrupts Globally */
74     sei();
75 }
76
77 /* loop(): this function is the main function. It executes
78 * continuously.
79 */
80 void loop() {
81
82     sum = 0;                                         // this variable holds the sum of temperatures
83     Serial.println("\n\nNew Measurement");
84     for( i = 0; i < 24; i++){
85         delay(5*1000);
86
87         if( i == 1){                                // 10 seconds have elapsed since avg. temp. print on LCD
88             lcd.clear();
89             lcd.setCursor(0, 0);
90             lcd.print("CALCULATING... ");
91         }
92
93         // convert the ADC reading to voltage
94         // convert the voltage to temperature in degrees C
95         // the sensor changes 10 mV per degree
96         // the datasheet says there's a 500 mV offset
97         // ((voltage - 500 mV) times 100)
98         int sensorVal = analogRead(sensorPin);      // read TMP36 value in Volts
99         float voltage = (sensorVal / 1024.0) * 5.0;

```

```

100 temperature[i] = (voltage - .5) * 100;
101 String printLine = String(String(i) + ". sensor Value: " + String(sensorVal) + ", Volts: " +
102 String(voltage, 3) + ", degrees C: " + String(temperature[i], 3));
103 Serial.println(printLine);
104
105 sum = sum + temperature[i];
106
107 checkExtremeValues( temperature[i], i);
108 }
109
110 /* Calculate Average Temperature */
111 averageTemperature = averageTemperature/24;
112 String line1 = String("Avg. Temp.");
113 String line2 = String(averageTemperature, 3);
114 printLCD( line1, line2);
115 Serial.println("\n\n\n Average Temperature: " + String(averageTemperature, 3));
116
117 //check for fan
118 checkExtremeFan( averageTemperature);
119 }
120
121 ****
122 * Sub-routines *
123 ****
124
125
126 /* printLCD(): this function is responsible for
127 *           printing average temperature to
128 *           LCD screen.
129 */
130 void printLCD( String line1, String line2){
131 lcd.clear();
132 lcd.setCursor(0, 0);
133 lcd.print(line1);
134 lcd.setCursor(0, 1);
135 lcd.print(line2);
136 }
137
138 /* checkExtremeValues: this function is responsible for cheking wether
139 *           temperature is Greater than a high value or
140 *           Lower than a low value. Accordingly, this function
141 *           turns on/off BLUE/RED LEDS and prints respectively
142 *           on LCD the suitable message. LCD is used only after
143 *           i > 0 and that's because only then 10 secs have elapsed
144 *           since previous average value print.
145 */
146 void checkExtremeValues( float temperature, int i){
147 String line1, line2;
148 if( temperature < extremeLow && i > 0){
149 line1 = String("Warning: LOW");
150 line2 = String( "Temp      " + String(temperature, 3));
151 printLCD( line1, line2);
152 digitalWrite(BLUE, HIGH);
153 digitalWrite(RED, LOW);
154 }
155 else if( temperature > extremeHigh && i > 0){
156 line1 = String("Warning: HIGH");
157 line2 = String( "Temp      " + String(temperature, 3));
158 printLCD( line1, line2);
159 digitalWrite(RED, HIGH);
160 digitalWrite(BLUE, LOW);
161 }
162 else{
163 digitalWrite(RED, LOW);
164 digitalWrite(BLUE, LOW);
165 if( i > 0)
166 printLCD("Calculating", " ");
167 }
168 }
169
170 /* checkExtremeFan(): this function is responsible
171 *           for checking whether fan is needed
172 *           to be turned on
173 */
174 void checkExtremeFan( float averageTemperature){
175
176 // check fan
177 if( averageTemperature > extremeFan){

```

```

178 if( FAN == false){
179     Serial.println("1.FAN is " + String(FAN) + ": TURN it ON");
180     Serial.println("FAN is OFF: TURN it ON");
181     digitalWrite(GREEN, HIGH);
182     FAN = true;
183 }
184 else{
185     Serial.println("2.FAN is " + String(FAN) + ": KEEP it ON");
186 }
187 }
188 else{
189     digitalWrite(GREEN, LOW);
190     FAN = false;
191     Serial.println("3.FAN is " + String(FAN));
192 }
193 }
194
195
196 /* checkProximity(): this function is responsible for acknowledging whether
197 * someone is close or not.
198 */
199 int checkProximity(){
200 // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
201 // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
202 digitalWrite(trigPin, LOW);
203 delayMicroseconds(5);
204 digitalWrite(trigPin, HIGH);
205 delayMicroseconds(15);
206 digitalWrite(trigPin, LOW);
207
208 // Read the signal from the sensor: a HIGH pulse whose
209 // duration is the time (in microseconds) from the sending
210 // of the ping to the reception of its echo off of an object.
211 if( duration = pulseIn(echoPin, HIGH)){
212     // Convert the time into a distance
213     cm = (duration/2) / 29.1;      // Divide by 29.1 or multiply by 0.0343
214     inches = (duration/2) / 74;    // Divide by 74 or multiply by 0.0135
215
216     Serial.print(inches);
217     Serial.print("in, ");
218     Serial.print(cm);
219     Serial.print("cm");
220     Serial.println();
221     if( cm < 10){
222         Serial.println("Somebody is close: " + String(cm) + "cm" );
223         return 1;
224     }
225     else{
226         return 0;
227     }
228 }
229 else{
230     Serial.println("Nobody Nearby");
231     return 0;
232 }
233 }
234
235 /* TIMER2 OVERFLOW INTERRUPT VECTOR: this function checks periodically every 0.5 seconds
236 * if someone is nearby. Then it prints average Temperature
237 * and the latest temperature measurement.
238 */
239 ISR(TIMER2_OVF_vect){
240     counter++;
241     String line1, line2;
242     if( counter == 30){
243         // check if someone is close
244         isSomeoneClose = checkProximity();
245
246         if( isSomeoneClose == 1){
247             line1 = String("AVG: " + String( averageTemperature , DEC));
248             if(i == 0)
249                 line2 = String("Last TEMP: " + String(temperature[24] , DEC));
250             else
251                 line2 = String("Last TEMP: " + String(temperature[i-1], DEC));
252
253             printLCD( line1, line2);
254             counter = 0;
255         }
256     }

```

2.4 Αποτελέσματα εργασίας του μικροελεγκτή *Arduino*.

3 Εργασία στον μικροελεγκτή *Wemos D1 R2*.

Η παρούσα ενότητα περιγράφει την υλοποίηση την εργασίας σε μικροελεγκτή *Wemos D1 R2*.

3.1 Λειτουργία προγράμματος.

Το πρόγραμμα που εκτελείται στον μικροελεγκτή *Wemos D1 R2* μετράει κάθε 5 δευτερόλεπτα τη θερμοκρασία του περιβάλλοντος. Αν η θερμοκρασία αυτή είναι μεγαλύτερη από μία συγκεκριμένη τιμή (**extremeHigh**) ή αν είναι μικρότερη από μία τιμή (**extremeLow**) τότε τυπώνεται το αντίστοιχο μήνυμα στην οθόνη LCD.

Κάθε 24 μετρήσεις της θερμοκρασίας περιβάλλοντος, δηλαδή κάθε δύο (2) λεπτά, υπολογίζεται η μέση τιμή των 24 αυτών μετρήσεων. Η μέτρηση αυτή τυπώνεται στην οθόνη LCD και παραμένει για δέκα (10) δευτερόλεπτα στην οθόνη και δεν μπορεί να τυπωθεί κάτι άλλο κατά τη διάρκεια αυτών των δέκα δευτερολέπτων. Αν η μέση τιμή είναι μεγαλύτερη από μία συγκεκριμένη τιμή **extremeFan**, τότε αποστέλλεται ε-μαίλ το οποίο πληροφορεί τον παραλήπτη για την ενεργοποίηση του ανεμιστήρα.

Επιπλέον, το πρόγραμμα αυτό περιλαμβάνει και την υλοποίηση ενός Software Interrupt με τη χρήση της βιβλιοθήκης *Ticker.h*. Η βιβλιοθήκη αυτή κάνει χρήση των *interrupts* του *ESP8266* για να προσεγγίσει τη λειτουργία ενός Timer Overflow Interrupt. Έτσι κατασκευάζεται μία συνάρτηση η οποία να καλείται κάθε 0.5 δευτερόλεπτα και να διαχορίζει το κυρίος πρόγραμμα για να εκτελέσει κάποιες λειτουργίες. Συγκεκριμένα, αυτή χρησιμοποιείται για τη μέτρηση της απόστασης του κοντινότερου αντικείμενου από τον αισθητήρα εγγύτητας. Αν η απόσταση αυτή είναι μικρότερη από μία συγκεκριμένη τιμή (10 cm) τότε τυπώνεται στην οθόνη LCD η προηγούμενη μέση τιμή της θερμοκρασίας και η τελευταία μέτρηση της θερμοκρασίας που πραγματοποιήθηκε.

Για την αποστολή του e-mail χρησιμοποιήθηκε ο **SMTP Server** που παρέχει η *Google*. Πιο συγκεκριμένα, δημιουργήθηκε ο λογαριασμός mcu.2019.wemos@gmail.com. Οι πληροφορίες του λογαριασμού αυτού τοποθετήθηκαν στον πηγαίο κώδικα και το πρόγραμμα ρυθμίστηκε ώστε να εκτελούνται οι απαραίτητες ενέργειες.

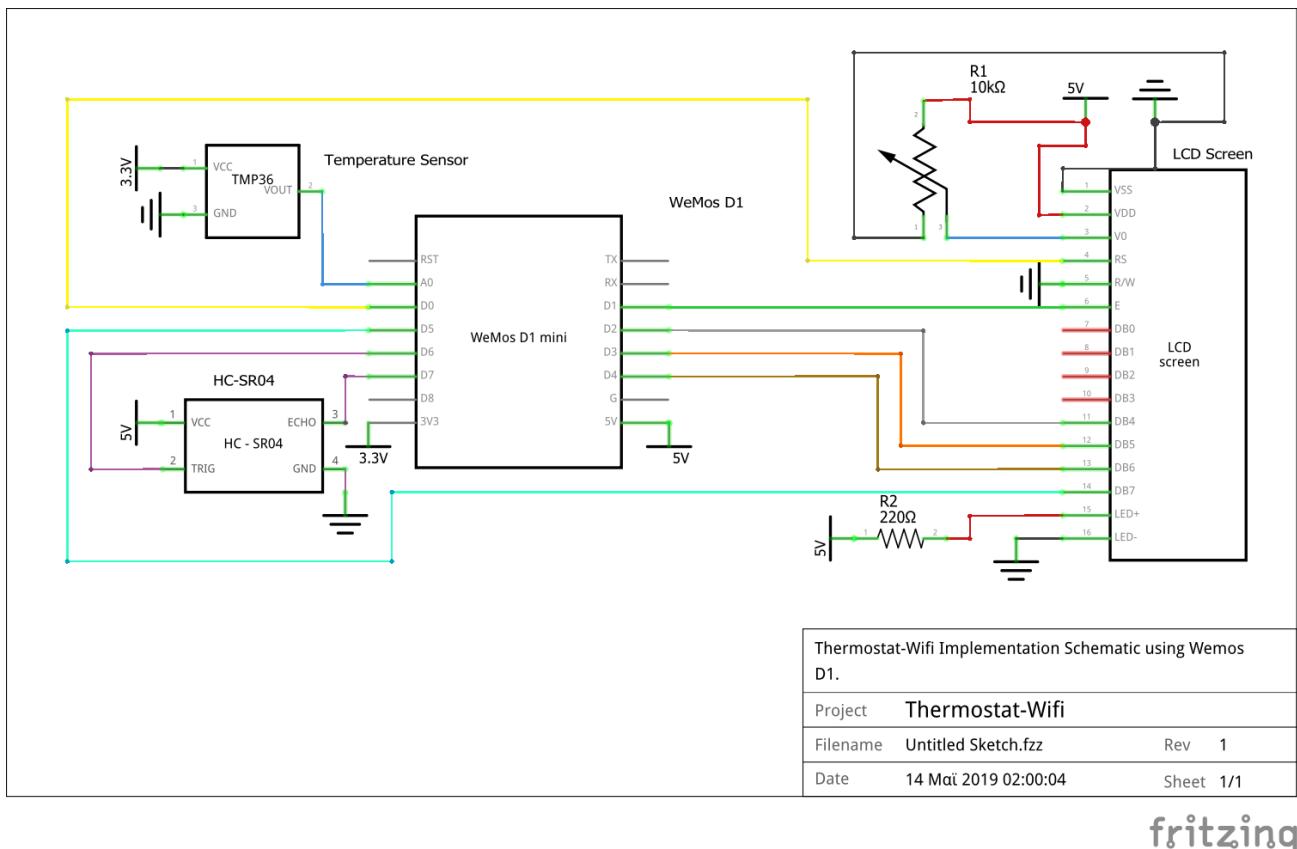
Το πρόγραμμα γενικά είναι παρόμοιο με αυτό που εκτελείται στον μικροελεγκτή *Arduino* με τη διαφορά ότι δεν περιλαμβάνονται στο πρόγραμμα αυτό οι λειτουργίες των LEDs.

3.2 Υλικό και συνδεσμολογία.

Για την υλοποίηση του παραπάνω προγράμματος χρησιμοποιήθηκαν τα εξής εξαρτήματα:

- Wemos D1 R2(ESP8266EX)
- Αισθητήρας θερμότητας TMP36 [1]
- Οθόνη LCD διαστάσεων 16x2 JHD659 M10 1.1 [2]
- Αισθητήρας εγγύτητας HC-SR04 [3]
- Ποτενσιόμετρο 10 kΩ

Τα παραπάνω εξαρτήματα συνδέθηκαν σύμφωνα με την εξής συνδεσμολογία:



fritzing

Σχήμα 2: Συνδεσμολογία εξαρτημάτων με μικροελεγκτή *Weemos D1 R2*.

Με βάση τη συνδεσμολογία αυτή υλοποιήθηκε και ο πηγαίος κώδικας που εκτελεί το πρόγραμμα.

3.3 Πηγαίος κώδικας.

Στην υποενότητα αυτή παρουσιάζεται ο πηγαίος κώδικας ο οποίος εκτελείται ακριβώς με τον ίδιο τρόπο τις λειτουργίες όπως αυτές περιγράφηκαν στο διάγραμμα ροής.

Το πρόγραμμα περιλαμβάνει 3 αρχεία. Το αρχείο *thermostat-Wifi.ino* περιλαμβάνει το κύριο πρόγραμμα ενώ τα άλλα δύο αρχεία *Gsender.cpp*, *Gsender.h* υλοποιούν τις ρυθμίσεις για την επικοινωνία με τον **SMTP Server**. Τα δύο τελευταία αρχεία πάρθηκαν από τη σελίδα [4] και τροποποιήθηκαν σε πρόσφατες ρυθμίσεις διότι δεν ήταν πλέον λειτουργικά.

```

1 /*
2  * Authors: 1. Dimitrios Antoniadis
3  *          2. Vasileios Dimitriadis
4  *
5  * email:   1. akdimitri@auth.gr
6  *          2. dimvasdim@auth.gr
7  *
8  * University: Aristotle University of Thessaloniki (AUTH)
9  * Semester: 8th
10 * Subject:    Microprocessors and Peripherals
11 *
12 * Parts required:
13 * - one Wemos D1 R2 (ESP8266)
14 * - one TMP36 temperature sensor
15 * - one LCD 16x2 JHD659 M10 1.1
16 * - one HC-SR04 proximity sensor
17 */
18
19 /* include Libraries */
20 #include <Ticker.h>
21 #include <ESP8266WiFi.h>
22 #include <LiquidCrystal.h>
23 #include "Gsender.h"
24

```

```

25 /* Global Variable Declaration */
26 Ticker INTERRUPT;
27 const char* ssid = "Dimitris2"; // WIFI network name
28 const char* password = "69451070306945558718"; // WIFI network password
29 uint8_t connection_state = 0; // Connected to WIFI or not
30 uint16_t reconnect_interval = 10000; // If not connected wait time to try again
31 const int sensorPin = A0;
32 int i, isSomeoneClose = 0;
33 float sum, temperature[24], averageTemperature;
34 const float extremeLow = 21.5, extremeHigh = 22.5, extremeFan = 23;
35 const int trigPin = 12, echoPin = 13;
36 unsigned long duration, cm, inches;
37 bool FAN = false;
38
39 // initialize the LCD function with the numbers of the interface pins
40 // D0 -> GPIO16
41 // D1 -> GPIO5
42 // D2 -> GPIO4
43 // D3 -> GPIO0
44 // D4 -> GPIO2
45 // D5 -> GPIO14
46 // D6 -> GPIO12
47 // D7 -> GPIO13
48 LiquidCrystal lcd(16, 5, 4, 0, 2, 14);
49
50
51 /* WiFiConnect: this function is responsible for connecting
52 *          MCU to WiFi.
53 */
54 uint8_t WiFiConnect(const char* nSSID = nullptr, const char* nPassword = nullptr)
55 {
56     static uint16_t attempt = 0;
57     Serial.print("Connecting to ");
58     if(nSSID) {
59         WiFi.begin(nSSID, nPassword);
60         Serial.println(nSSID);
61     } else {
62         WiFi.begin(ssid, password);
63         Serial.println(ssid);
64     }
65
66     uint8_t i = 0;
67     while(WiFi.status() != WL_CONNECTED && i++ < 50)
68     {
69         delay(200);
70         Serial.print(".");
71     }
72     ++attempt;
73     Serial.println("");
74     if(i == 51) {
75         Serial.print("Connection: TIMEOUT on attempt: ");
76         Serial.println(attempt);
77         if(attempt % 2 == 0)
78             Serial.println("Check if access point available or SSID and Password\r\n");
79         return false;
80     }
81     Serial.println("Connection: ESTABLISHED");
82     Serial.print("Got IP address: ");
83     Serial.println(WiFi.localIP());
84     return true;
85 }
86
87
88 /* Awaits(): this function tries continuously to connect to WiFi.
89 *          It stops when connection has been established.
90 */
91 void Awaits()
92 {
93     uint32_t ts = millis();
94     while(!connection_state)
95     {
96         delay(50);
97         if(millis() > (ts + reconnect_interval) && !connection_state){
98             connection_state = WiFiConnect();
99             ts = millis();
100        }
101    }
102}

```

```

103
104
105 /* checkProximity(): this function is responsible for acknowledging whether
106 * someone is close or not.
107 */
108 int checkProximity(){
109     // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
110     // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
111     digitalWrite(trigPin, LOW);
112     delayMicroseconds(5);
113     digitalWrite(trigPin, HIGH);
114     delayMicroseconds(15);
115     digitalWrite(trigPin, LOW);
116
117     // Read the signal from the sensor: a HIGH pulse whose
118     // duration is the time (in microseconds) from the sending
119     // of the ping to the reception of its echo off of an object.
120     if( duration = pulseIn(echoPin, HIGH)){
121         // Convert the time into a distance
122         cm = (duration/2) / 29.1;           // Divide by 29.1 or multiply by 0.0343
123         inches = (duration/2) / 74;        // Divide by 74 or multiply by 0.0135
124
125         //Serial.print(inches);
126         //Serial.print("in, ");
127         //Serial.print(cm);
128         //Serial.print("cm");
129         //Serial.println();
130         if( cm < 10){
131             Serial.println("Somebody is close: " + String(cm) + "cm" );
132             return 1;
133         }
134         else{
135             return 0;
136         }
137     }
138     else{
139         Serial.println("Nobody Nearby");
140         return 0;
141     }
142 }
143
144
145 /* printLCD(): this function is responsible for
146 *               printing average temperature to
147 *               LCD screen.
148 */
149 void printLCD( String line1 , String line2){
150     lcd.clear();
151     lcd.setCursor(0, 0);
152     lcd.print(line1);
153     lcd.setCursor(0, 1);
154     lcd.print(line2);
155 }
156
157
158 /* interruptVector(): this function checks periodically every 0.5 seconds
159 *                     if someone is nearby. Then it prints average Temperature
160 *                     and the latest temperature measurment.
161 */
162 void interruptVector(){
163
164     // check if someone is close
165     isSomeoneClose = checkProximity();
166     String line1 , line2;
167     if( isSomeoneClose == 1){
168         line1 = String("AVG: " + String( averageTemperature , 3));
169         if(i == 0)
170             line2 = String("Last TEMP: " + String(temperature[24] , 3));
171         else
172             line2 = String("Last TEMP: " + String(temperature[i-1], 3));
173
174         printLCD( line1 , line2);
175     }
176 }
177 /* checkExtremeValues: this function is responsible for cheking wether
178 *                     temperature is Greater than a high value or
179 *                     Lower than a low value. Accordingly, this function
180 *                     turns on/off BLUE/RED LEDS and prints respectively
181 *                     on LCD the suitable message. LCD is used only after

```

```

182 * i > 0 and that's because only then 10 secs have elapsed
183 * since previous average value print.
184 */
185 void checkExtremeValues( float temperature , int i){
186     String line1 , line2 ;
187     if( temperature < extremeLow && i > 0){
188         line1 = String("Warning: LOW");
189         line2 = String( "Temp      " + String(temperature , 3));
190         printLCD( line1 , line2 );
191         //digitalWrite(BLUE, HIGH);
192         //digitalWrite(RED, LOW);
193     }
194     else if( temperature > extremeHigh && i > 0){
195         line1 = String("Warning: HIGH");
196         line2 = String( "Temp      " + String(temperature , 3));
197         printLCD( line1 , line2 );
198         //digitalWrite(RED, HIGH);
199         //digitalWrite(BLUE, LOW);
200     }
201     else{
202         //digitalWrite(RED, LOW);
203         //digitalWrite(BLUE, LOW);
204         if( i > 0)
205             printLCD("Calculating" , " ");
206     }
207 }
208
209 /* checkExtremeFan(): this function is responsible
210 *          for checking whether fan is needed
211 *          to be turned on
212 */
213 void checkExtremeFan( float averageTemperature){
214
215     // check fan
216     if( averageTemperature > extremeFan){
217         if( FAN == false){
218             Serial.println("1.FAN is " + String(FAN) + ": TURN it ON");
219             //digitalWrite(GREEN, HIGH);
220             Gsender *gsender = Gsender::Instance(); // Getting pointer to class instance
221             String subject = "FAN = ON!";
222             String value = String(averageTemperature , 3);
223             String message = String("FAN TURNED ON : " + value);
224             if(gsender->Subject(subject)->Send("mitsos1996@yahoo.com" , message)) {
225                 Serial.println("Message send.");
226             } else {
227                 Serial.print("Error sending message: ");
228                 Serial.println(gsender->getError());
229             }
230             FAN = true;
231         }
232         else{
233             Serial.println("2.FAN is " + String(FAN) + ": KEEP it ON");
234         }
235     }
236     else{
237         FAN = false;
238         Serial.println("3.FAN is " + String(FAN));
239         //digitalWrite(GREEN, LOW);
240     }
241 }
242
243
244 /* setup(): This function is executed only once at the POWER ON
245 *          or at RESET
246 */
247 */
248 void setup()
249 {
250     Serial.begin(115200);
251     connection_state = WiFiConnect();
252     if(!connection_state) // if not connected to WIFI
253         Awaits(); // constantly trying to connect
254
255     /* Setup LCD */
256     lcd.begin(16, 2);
257     lcd.print("ARDUINO");
258     lcd.setCursor(0, 1);
259     lcd.print("PROJECT");
260 }
```

```

261
262 // HC-SR04
263 pinMode(trigPin, OUTPUT);
264 pinMode(echoPin, INPUT);
265
266 /* Setup ticker function */
267 INTERRUPT.attach( 0.5, interruptVector );
268
269 /* Setup Serial Communication */
270 Serial.begin(115200);
271
272 }
273
274 /* loop(): this function is the main function. It executes
275 *      continuously.
276 */
277 void loop(){
278     sum = 0;                                // this variable holds the sum of temperatures
279     Serial.println("\n\n\nNew Measurement");
280     for( i = 0; i < 24; i++){
281         delay(5*1000);
282
283         if( i == 1){                         // 10 seconds have elapsed since avg. temp. print on LCD
284             lcd.clear();
285             lcd.setCursor(0, 0);
286             lcd.print("CALCULATING... ");
287         }
288
289         // convert the ADC reading to voltage
290         // convert the voltage to temperature in degrees C
291         // the sensor changes 10 mV per degree
292         // the datasheet says there's a 500 mV offset
293         // ((voltage - 500 mV) times 100)
294         int sensorVal = analogRead(sensorPin);    // read TMP36 value in Volts
295         float voltage = (sensorVal / 1024.0) * 3.1;           // 3.1 is used after calibration. Normally
296         3.3 should be used
297         temperature[i] = (voltage - .5) * 100;            // since 3.3 V is the Operation Vlotage of
298         MCU.
299         String printLine = String(i) + ". sensor Value: " + String(sensorVal) + ", Volts: " +
300         String(voltage, 3) + ", degrees C: " + String(temperature[i], 3));
301         Serial.println(printLine);
302
303         sum = sum + temperature[i];
304
305         checkExtremeValues( temperature[i], i );
306     }
307
308     /* Calculate Average Temperature */
309     averageTemperature = sum/24;
310     String line1 = String("Avg. Temp.");
311     String line2 = String(averageTemperature, 3);
312     printLCD( line1, line2 );
313     Serial.println("\n\n\n Average Temperature: " + String(averageTemperature, 3));
314
315 }

```

3.4 Αποτελέσματα εργασίας του μικροελεγκτή Wemos D1 R2.

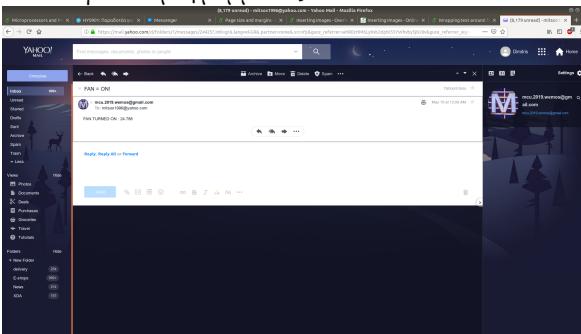
```

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
}

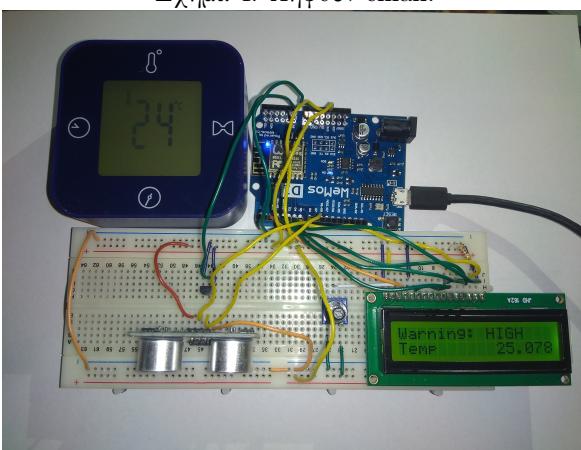
void loop() {
  // read the data from the DHT sensor
  float temp = dht.readTemperature();
  // print the results to the serial monitor
  Serial.print("Temperature: ");
  Serial.println(temp);
}

```

Σχήμα 3: Παράθυρα σειριακής επικοινωνίας κατά την εκτέλεση του προγράμματος.



Σχήμα 4: Ληφθέν email.



Σχήμα 5: Ειδοποίηση Θερμοκρασίας extremeHigh.

Αριστερά στις δύο εικόνες φαίνεται η εκτέλεση του προγράμματος από την αρχή. Σημειώνεται ότι η τιμή **extremeFan** τέθηκε ίση με 23. Συνεπώς, για μέση τιμή θερμοκρασίας 23 και πάνω ενεργοποιείται ο ανεμιστήρας (**bool FAN**) και αποστέλλεται e-mail.

Στις δύο εικόνες αριστερά, αρχικά φαίνεται ότι επιτυγχάνεται η σύνδεση στο WiFi και τυπώνεται η IP που λαμβάνει ο μικροελεγκτής. Στη συνέχεια εκκινεί η διαδικασία μετρήσεων. Όλες οι εκτυπώσεις στην ουδόνη είναι αριθμημένες. Πραγματοποιούνται 24 μετρήσεις. Κάθε μέτρηση εκτυπώνει τον αριθμό που μετράει ο ADC Converter. Η μέτρηση αυτή κυμαίνεται από 0 έως 1023. Η μέτρηση αυτή μετατρέπεται σε Volts και αντίστοιχα τυπώνεται η θερμοκρασία. [1]

Μόλις πραγματοποιηθούν 24 μετρήσεις τυπώνεται η μέση τιμή. Στη συνέχεια ελέγχεται αν πρέπει να ενεργοποιηθεί ο ανεμιστήρας. Εφόσον ο ανεμιστήρας είναι απενεργοποιημένος και η μέση τιμή είναι μεγαλύτερη από **extremeFan** τότε ενεργοποιείται ο ανεμιστήρας και αποστέλλεται το αντίστοιχο e-mail.

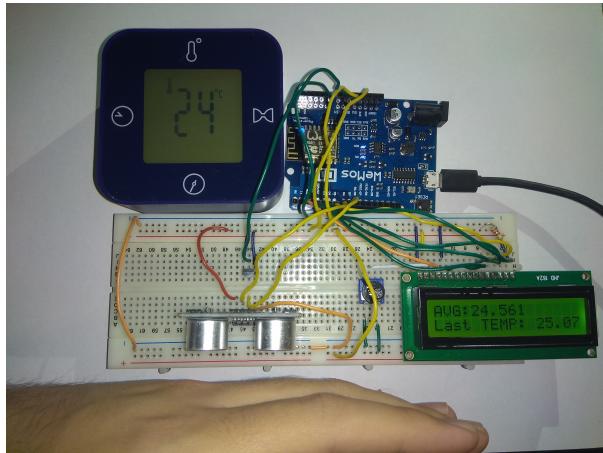
Αριστερά των εκτυπώσεων υπάρχει το αντίστοιχο timestamp. Τη χρονική στιγμή **00:09**, δηλαδή 12:09 π.μ. αποστέλλεται το σχετικό e-mail.

Στην επόμενη εικόνα φαίνεται το e-mail λήφθηκε στις **12:09** π.μ. Επίσης το e-mail περιλαμβάνει και τη μέση τιμή που υπολογίσθηκε. Το e-mail ενημερώνει τον παραλήπτη ότι ο ανεμιστήρας ενεργοποιήθηκε.

Στο Σχήμα 3, επιπλέον, φαίνεται ότι ενώ είναι ήδη ενεργοποιημένος ο ανεμιστήρας δεν αποστέλλεται νέο e-mail.

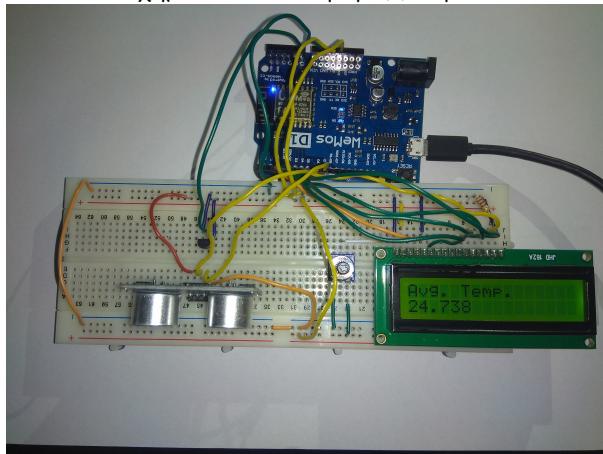
Ακόμη, ανάμεσα στις εκτυπώσεις, υπάρχουν και δύο εκτυπώσεις οι οποίες πληροφορούν ότι κάποιος βρίσκεται κοντά στον αισθητήρα εγγύτητας γεγονός που πυροδοτεί την εκτύπωση της προηγούμενης μέσης τιμής και της τελυταίας τιμής που διαβάστηκε.

Όλες οι μετρήσεις έχουν μεταξύ τους 5 δευτερόλεπτα όπως επιβεβαιώνεται και από το Σχήμα 3. Ακόμη, οι δύο εκτυπώσεις που πληροφορούν ότι κάποιος βρίσκεται κοντά έχουν μεταξύ τους χρονική απόσταση 0.5 δευτερόλεπτα, γεγονός που επιβεβαιώνει ότι λειτουργεί σωστά η συγκεκριμένη λειτουργία. Το γεγονός αυτό επιβεβαιώνεται και από το Σχήμα 6



Στο Σχήμα 7 φαίνεται η λειτουργία κατά την οποία εκτυπώνεται η μέση τιμή της θερμοκρασίας ύστερα από 24 μετρήσεις (2 λεπτά).

Σχήμα 6: Ειδοποίηση εγγύτητας.



Σχήμα 7: Ειδοποίηση μέσης τιμής.

4 Επίλογος.

Επίλογος.

Αναφορές

- [1] TMP35/TMP36/TMP37 datasheet. <https://www.arduino.cc/en/uploads/Main/TemperatureSensor.pdf>.
- [2] JHD659 M10 1.1 LCD datasheet. <https://www.arduino.cc/documents/datasheets/LCDscreen.PDF>.
- [3] HC-SR04 datasheet. <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>.
- [4] Boris Shobat. ESP8266 GMail Sender. <https://www.instructables.com/id/ESP8266-GMail-Sender/>.