Tervezés mikrovezérlőkkel

Sapientia EMTE
2021, Marosvásárhely

Vezető tanár: dr. Losonczi Lajos
Készítette: Magyarosi Roland

Tartalom

Bevezető 3
Felhasználói platform 3
Kapcsolási rajzok 5
Elvi megvalósítás: 5
Schematic: 5
Gyakorlati megvalósítás: 6
Alkatrészlista 6
Alkatrészleírások 6
1. Vízálló hőmérsékletmérő szenzor: 6
2. LED 7
3. Ellenállások 7
4. Huzalok 7
5. Breadboard 7
Felhasználói követelmények 7
Funkcionális követelmények 7
Nem funkcionális követelmények 8
Beágyazott vezérlő programok 8
Felhasználói interfész programok 13
Összegzés 15
Jövőbeli tervek 15
Bibilográfia: 15

Bevezető

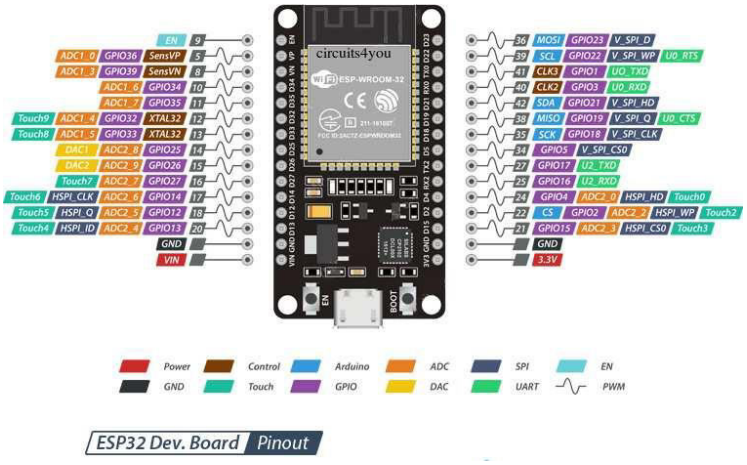
A mai kényelmes világban minden megoldás a zsebünkben, vagy éppen a laptopunkon rejtőzik. A legtöbb mindennapokban használt eszköz elérhető a telefonunkról, elég csak az olyan alapvető tárgyakra gondolni, mint az óra, a fényképezőgép, vagy éppen a tükör. A legtöbb lakásban fellelhető egy hőmérő, amely manapság inkább digitális, mint klasszikus, a rendszerem ezt az eszközt is a zsebünkbe varázsolja a WEB-en keresztül. Az ESP32-nek hála a lokális Wi-Fi-hez csatlakozott eszközök elérik a lokális weboldalt, amely 10 másodpercenként frissülve 2 tizedesnyi pontossággal megmondja a szoba hőmérsékletét. A szenzor kádba, vagy medencébe helyezésével pár másodpercen belül a telefonunkon, vagy okos eszközünkön megfigyelhetjük az aktuális hőmérsékletet.

A közeljövőben elengedhetetlen kellék lesz az okosotthonokban az én módszeremhez hasonló megközelítés, lassan a digitális hőmérő is idejét múlta, amely a falnak támasztva csak akkor közöl információt számunkra, ha elé ballagunk, az alábbi megoldással a Wi-Fi hatótávján belül bárhol megfigyelhetjük a szenzor által közölt hőmérséklet értékeket, amelyek meglehetősen pontosak.

Felhasználói platform

Az általam használt platform összetevői az ESP-WROOM-32 chip, amely egy ESP32 DevKitC lapra van felszerelve az Espressif által.

Az ESP32-DevKitC egy kis méretű ESP32 alapú fejlesztőtábla. Az ESP-WROOM-32 köré épített rendszerfejlesztő tábla optimális teljesítményt ér el Wi-Fi, Bluetooth és rádió megoldásokkal. Az ESP32-DevKitC tartalmazza az ESP-WROOM-32 teljes támogatási áramkörét, beleértve az USB-UART hidat, az EN és a BOOT mód gombjait, az LDO szabályozót és a mikro-USB csatlakozót. Az LDO (low-dropout) szabályozó egy egyenáramú lineáris feszültségszabályozó, amely akkor is szabályozza a kimeneti feszültséget, ha a tápfeszültség nagyon közel van a kimeneti feszültséghez. Az alaplapon lévő GPIO (General-purpose input/output) portok lehetővé teszik a kártya csatlakoztatását más perifériákhoz.



(1. ábra)

Az első ábrán szemléltetve vannak a lap lábai, a modult kétféle képpen lehet betáplálni:

- USB bementeten, ahol a chip felprogramozása is történik, ha pedig már nem programozzuk, akkor elegendő egy 5V-os adapter is.
- VIN porton keresztül, amely 1000mA-es erőssű áramot képes elviselni, maximálisan 20V feszültséget képes kezelni.

Az alaplap az ESP32, egy olcsó, alacsony energiafogyasztású rendszer, amely egy chip mikrovezérlőn intergrált Wi-Fi-vel és kettős módú Bluetooth-tal rendelkezik. Az ESP32-t az Espressif Systems hozta létre és fejlesztette ki, amely egy Sanghaj-i székhelyű vállalat. Az ESP32 elődje az ESP8266 mikrovezérlő, amely 2014-ben került a piacra.

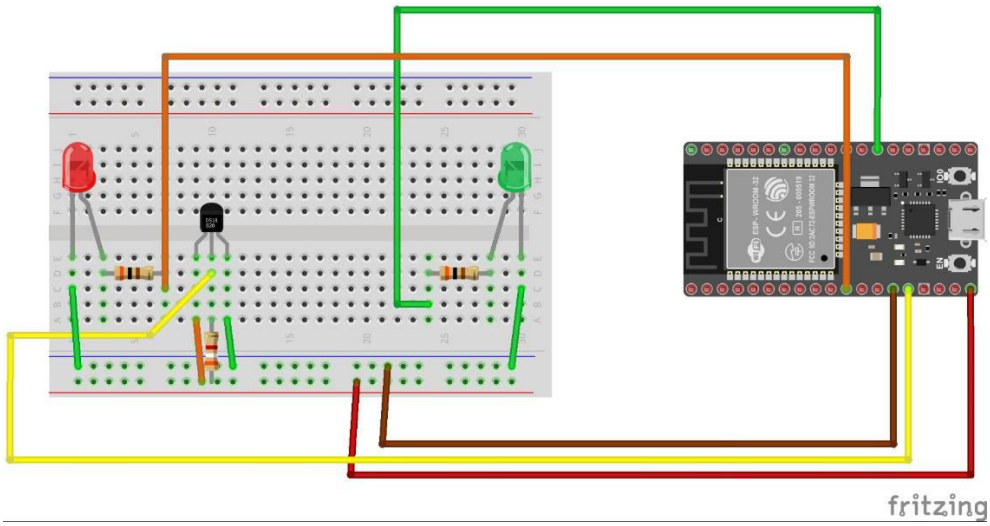
Az általam használt ESP32 programozásához használt programozási nyelv az Arduino IDE volt, de számos más platform és környezet is létezik. Az Arduino IDE egyaránt elérhető Windows, macOS, illetve Linux operációs rendszereken is, támogatja a C, C++ nyelveket. A fejlesztői környezet segítségével programoztam fel a rendszerem funkcionalitásait az Arduino kompatibilis táblámra.

Az ESP32 jellemzői:

- Processzorok:
  - Xtensa két, vagy egymagos 32 bites LX6 mikroprocesszor, 160, vagy 240 MHz-en és legfeljebb 600 DMIPS-en működik
  - ULP (Ultra alacsony fogyasztású) társprocesszor.
- Memória:
  - 520 KiB SRAM
- Vezeték nélküli kapcsolat:
  - Wi-Fi 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR és BLE (megosztja a rádióval)
- Energia menedzselés:
  - Belső alacsony esésszabályozó
  - Egyéni energia tartomány az RTC számára
  - 5 μA alvási áram
  - Ébresztés GPIO megszakításból, időzítőből, ADC mérésekből, kapacitív érintésérzékelő megszakításból.

Kapcsolási rajzok

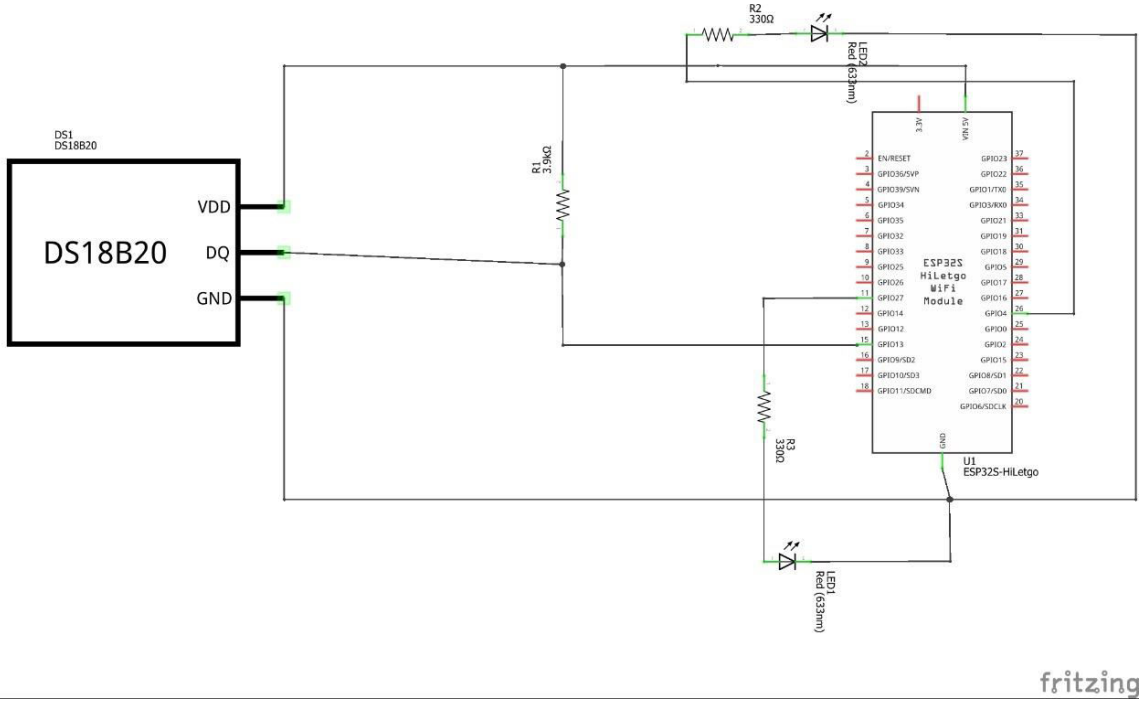
Elvi megvalósítás:



(2. ábra)

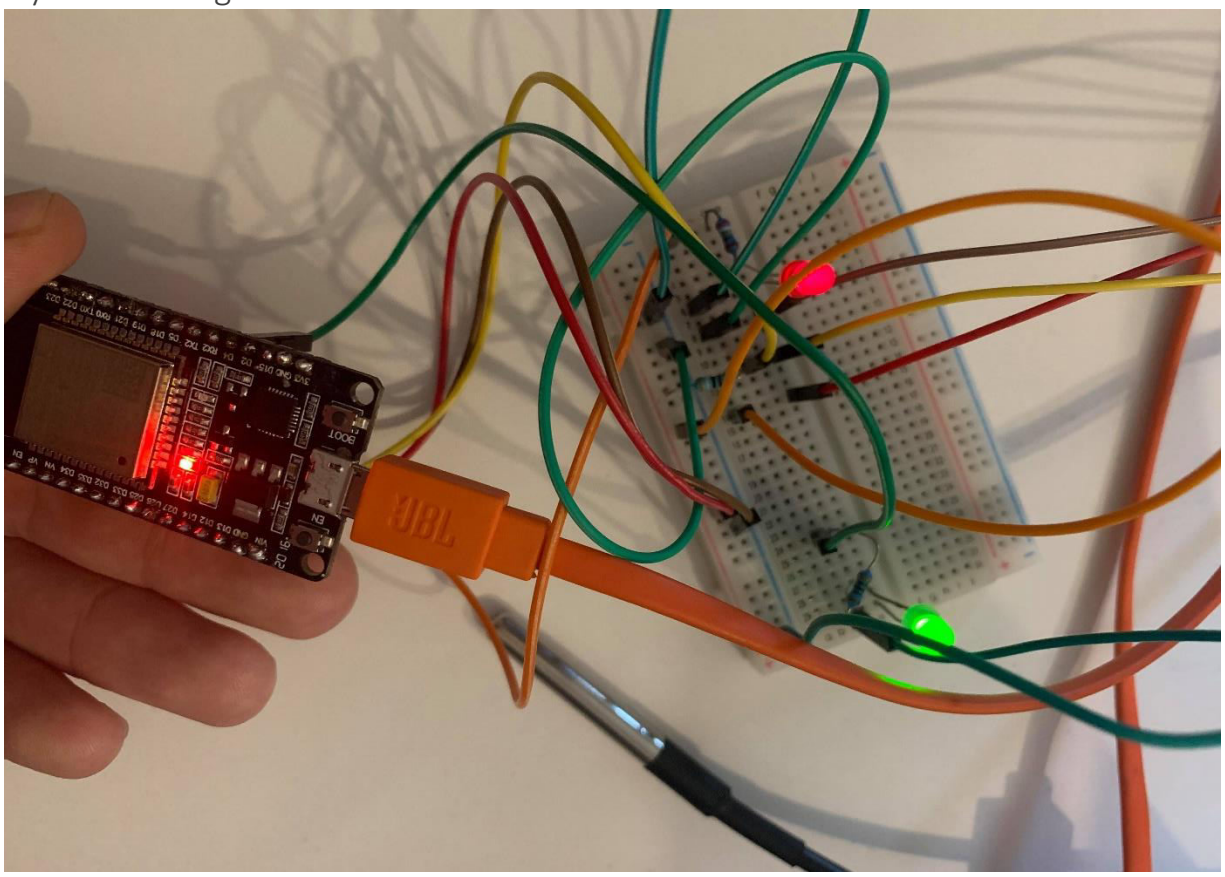
A 3. ábrán a kapcsolási rajzom látható, amelyet a Fritzing Béta verziós ingyenes szoftverével valósítottam meg. A képen látható az általam használt piros, illetve zöld színű LED, a huzalok színei megegyeznek a gyakorlati megvalósításban alkalmazott színekkel, továbbá az ellenállások mérete is azonos ugyanúgy, mint maga az ESP32 lap, amely az nem szerepelt, ám kevés utánajárással könnyedén be tudtam importálni.

Schematic:



(3. ábra)

Gyakorlati megvalósítás:



(4. ábra)

A szemléltető ábrán (4) tökéletesen látszik az áramköröm, amely a 3. illetve a 2. ábrán is megtekinthető. Jelen esetben mindkét LED világít, a piros LED mögött elhúzódó három szál (barna, sárga, piros) a hőmérsékletmérő szenzor kábelei.

Alkatrészlista

- ESP32
- Vízálló hőmérsékletmérő szenzor (DS18B20)
- Ledek (5mm)
- 330 Ohm-os ellenállás
- 3.9k Ohm-os ellnállás
- Huzalok
- Breadboard

Alkatrészleírások

1. Vízálló hőmérsékletmérő szenzor: A projektemben egy hőmérsékletmérő szenzor segítségével kellett információt közölnöm, a választásom a DS18B20 nevű érzékelőre

esett a vízállósága miatt, ugyanis ezt könnyen hűthető és fűthető is, leegyszerűsítette a szoftver tesztelési fázisának idejét. Az érzékelő akkor igazán hasznos, ha vízben akarunk hőmérsékletet mérni, vagy egy a rendszertől távoli helyen, ugyanis hosszú kábelezése van és a nedvesség sem árthat neki. A szenzor azonos kódú társa egy három lábú kis műszer, amely nem vihető távolra a rendszertől és a nedvességet sem bírja. A gyári adatok szerint a -55-ös Celsius foktól egészen +125-ig mér, azonban ajánlott 100 fok alatt tartani a PVC borítás miatt. Fél fokos hibával dolgozik -10 és +85 C fokon az érzékelő, amely meglehetősen jónak mondható a piacon. Úgy 3, mint 5 volton használható, én az utóbbi mellett döntöttem, egy 3.9 Ohmos ellenállást használva. Három szálnak kell helyet kapnunk az áramkörünkben, a piros a tápkábel, a fekete a földelés, a sárgán közli az adatot a szenzor. A nem vízálló társnak szürkék a lábai, így abban az esetben az írással szemben fordítva fekete, sárga, piros a sorrend.

2. LED: A Light-Emitting Diode(fényt kibocsátó dióda) félvezető anyagból készült világító test, amely áram hatására fényt bocsát ki. A dióda által kibocsátott fény színe a félvezető anyag összetételétől, ötvözőitől függ. Az én projektemben a zöld LED irányítható a webszerverről, illetve a piros színű LED jelzi, ha a hőmérséklet meghaladta a 25 C fokot. Manapság már a LED-ek a leghatékonyabb és a legnagyobb teljesítménnyel rendelkező világítási eszközok, éppen ezért szinte mindenhol jelen vannak: forgalomirányító lámpák, járművilágítás, kültéri és beltéri világítás.
3. Ellenállások: Az ellenállás az elektronikai alkatrészek egyik fontos fajtája. Feladata, hogy megfelelő mértékű elektromos ellenállást biztosítson egy áramkör adott részén. Az ellenállások alapanyaga szén és fém vegyületek, ötvözetek. Egy LED működtetéséhez általában 330 Ohmos ellenállást használnak 5V-on. A szenzorom mellé egy 3.9K Ohmos felhúzó ellenállást használtam, ami azért szolgál, hogy a kommunikáció a legstabilabb legyen.
4. Huzalok: Olyan elektromos vezetékeket használtam, amelyeknek mindkét végén egy csatlakozó van, amelyek segítségével könnyen össze tudtam kötni az alkatrészeket az ESP32 lappal, illetve a breadboard-al. Mindez tökéletesen szemügyre vehető a gyakorlati megvalósításnál.
5. Breadboard: Egy áramköri lap, mely lehetővé teszi az áramkör prototípusának elkészítését. A modern szerelőlapok lyukacsosak, az alkatrészt forrasztás nélkül is tudják fogadni.

Felhasználói követelmények

Ahhoz, hogy a user használni tudja a rendszert szüksége van egy okoseszközre, ami csatlakozni tud az előre meghatározott Wi-Fi-hez, amelyen az ESP32 is kommunikál.

Funkcionális követelmények

Ahhoz, hogy a rendszer működőképes legyen az áramkör elemeinek tökéletesen illeszkedniük kell, a szenzornak adatot kell mérnie, megfelelő táplálásra van szüksége. A szenzor

emberi körülmények fölé, illetve alá terjeszti az érzékelő skáláját, ugyanis -55 Celsius foktól egészen 125-ig mér, ez nem jelent akadályt a rendszer működésében.

Amint beérkezett az adat a szenzortól és az feldolgozásra került ez megtekinthető az Arduino IDE kimeneti ablakában, illetve a létrejött weboldalon egyaránt. Minden 10 másodpercben új, friss adat kerül az ablakokba. Ezek mellett amikor a szenzor 25 fölötti hőmérsékletet észlel a rendszer figyelmezteti a felhasználót egy LED megvillanással. A LED kialszik, amikor 25 Celsius fok alá hűl a hőmérséklet. Egy másik LED-et lehet kapcsolgatni a webszerverről kedvünkre.

Nem funkcionális követelmények

Könyvtárak:

- WiFi.h: Ezen könyvtár teszi lehetővé a tábla csatlakozását az internethez. Szolgálhat akár bejövő kapcsolatokat elfogadó kiszolgálóként, de akár kimenő kapcsolatokat létrehozó kliensként.
- ESPAsyncWebServer: A webszerver felépítéséhez az ESPAsyncWebServer könyvtárat használtam, amely egyszerű módszert kínált az aszinkron webszerver felépítésére. Az aszinkron webszervernek számos előnye van, mint például: egyszerre több kapcsolatot kezelhet, amint választ akarok küldeni azonnal készen áll az egyéb kapcsolatok kezelésére, miközben egy hátsó szálon intézi a kérésemet és még egyéb más dolgokra is alkalmas.
- OneWire: Ezen könyvtár segítségével férhetek hozzá a Maxim/Dallas által gyártott eszközökhöz.
- DallasTemperature: Hőmérséklet-érzékelőknél a DallasTemperature könyvtár használható ezzel a fent említett OneWire könyvtárral.

Beágyazott vezérlő programok

Az Arduino IDE segítségével tudtam felprogramozni az alábbiakban részletezett kódot a mikrovezérlőmre.

Kezdetben a lokális Wi-Fi kapcsolat teremtéséhez megadtam a routerem SSID-ját, illetve a jelszót, hogy az ESP32 is használni tudja azt.

```
const char* ssid = "TP-LINK_18D4";
const char* password = "06397579";
```

A LED és a web közötti kapcsolathoz létrehoztam a következő két változót, az egyik a GPIO kimeneti értéke, a másik pedig az állapota. Erre azért volt szükség, hogy kezelni tudjam a weboldalról a zöld LED-et.

```
const char* PARAM_INPUT_1 = "output";
const char* PARAM_INPUT_2 = "state";
```

Egy kezdetleges HTML weboldal elkészítését követően CSS design elemeket adtam hozzá, majd funkciókkal láttam el, amelyekhez a JavaScript kódolási nyelvet használtam. az alábbi function helyet foglal egy placeholdernek, majd XML http kérést küld és ellenőrzi, hogy az elem, esetünkben a checkbox aktiválva lett-e, amennyiben igen az állapotot egyesre állítja és a breadboardon a LED kigyullad, amennyiben a checkbox értéke hamis lesz, a LED kialszik.

```
  %BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked){ xhr.open("GET", "/update?output="+element.id+"&state=1", true); }
  else {
    xhr.open("GET", "/update?output="+element.id+"&state=0", true);}
  xhr.send();
}
</script>
```

Ugyancsak JavaScriptben íródott a következő szkript, amely egy időzítőként szolgál, hogy a kliens, vagy a felhasználó tudja a következő frissítés pontos időpontját. Ez a visszaszámláló 10 másodpercről számlál vissza folyamatosan.

```
<p style="font-size:20px;"> The restart will begin in <span id="countdowntimer">10 </span> seconds</p>

<script type="text/javascript">
    var timeleft = 10;
    var downloadTimer = setInterval(function(){
    timeleft--;
    document.getElementById("countdowntimer").textContent = timeleft;
    if(timeleft == 0){
        timeleft = 10;
    }
    },1000);
```

A szkriptbe írt funkciók olvassák és küldik a szenzor által közölt értékeket.

```
<script>
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperaturec").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperaturec", true);
  xhttp.send();
}, 10000) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperaturef").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperaturef", true);
  xhttp.send();
}, 10000) ;
</script>
```

Az alábbi funkció szerepe annyi csupán, hogy figyeli a checkbox-ot, és visszatéríti az éppen aktuális állapotát.

```
String outputState(int output){
  if(digitalRead(output)){
    return "checked";
  }
  else {
    return "";
  }
}
```

Ezt követően megírásra került a readDSTempretureC függvény, ami arra szolgál, hogy leolvassa a hőmérsékletet a szenzorról, ezt követően visszatéríti az értéket Celsius fokban, az ehhez hasonló readDSTempretureF függvény Fahrenheit skála szerinti értéket ad vissza.

Lekezelésre került a -127 Celsius fokos, illetve -196 Fahrenheit fokos érték is, ugyanis amennyiben a függvény ezt az értéket adja vissza hiba történt a szenzor leolvasása közben.

```
String readDSTemperatureC() {

  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);

  if(tempC == -127.00) {
    Serial.println("Failed to read from DS18B20 sensor");
    return "--";
  } else {
    Serial.print("Temperature Celsius: ");
    Serial.println(tempC);
  }

  temp = tempC;

  return String(tempC);
}
```

A processor függvény lekezeli a HTML részben megalkotott placeholdereket, hogy a weboldalon a kész értékeket lássuk majd. Amikor a placeholder fel lett ismerve a helyére beilleszti a processor függvény a tényleges értéket.

```
String processor(const String& var){
  //Serial.println(var);
  if(var == "TEMPERATUREC"){
    return readDSTemperatureC();
  }
  else if(var == "TEMPERATUREF"){
    return readDSTemperatureF();
  }

  if(var == "BUTTONPLACEHOLDER"){
    String buttons = "";
    buttons += "<h4>Always on: </h4><label
    return buttons;
  }
  return String();
}
```

Ezután a setup függvény megírása következett. A setup függvény csak egyszer fut le, amikor az ESP először kapcsolatot teremt a kóddal.

Első lépésben a LED-eket kezelem le, megadom a pin módját kimenetnek, valamint a zöld LED-nek LOW kezdőértéket adok, ami annyit jelent, hogy kezdő állapotban a LED-nek aludnia kell.

```
pinMode(LED_GREEN, OUTPUT);
digitalWrite(LED_GREEN, LOW);
pinMode(LED_RED, OUTPUT);
```

Ezt követően csatlakozok a DS18B20 szenzor könyvtárához és elindítom azt.

```
sensors.begin();
```

Majd csatlakozom a WiFi hálózathoz a fent megadott adatokkal és kiíratom a Serial Monitorra az adott IP címet amivel majd eltudom indítani a webszervert.

```
// Connect to Wi-Fi
WiFi.begin(ssid, password);
Serial.println("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println();

// Print ESP Local IP Address
Serial.println(WiFi.localIP());
```

Az alábbi kérésekkel tudok kapcsolatot teremteni a weboldal és a függvényeim között, itt hívom meg úgymond a megírt függvényeket. Az első ilyen kérés a processor meghívása, ami rendre helyettesíti a megfelelő kéréseket, a további funkciók frissítik a szenzorról lenyert adatokat.

```
// Route for root / web page, refreshing
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
});
server.on("/temperaturec", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", readDSTemperatureC().c_str());
});
server.on("/temperaturef", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/plain", readDSTemperatureF().c_str());
});
```

Az alábbi kérés a LED weboldalról való villogtatására szolgál, a változók itt kapnak értéket, majd itt kerülnek kiírásra a Serial Monitorra az információk a GPIO értékéről és aktuális állapotáról (1, vagy 0).

```
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {
  String inputMessage1;
  String inputMessage2;
  // GET input1 value on <ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
  if (request->hasParam(PARAM_INPUT_1) && request->hasParam(PARAM_INPUT_2)) {
    inputMessage1 = request->getParam(PARAM_INPUT_1)->value();
    inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
    digitalWrite(inputMessage1.toInt(), inputMessage2.toInt());
  }
  else {
    inputMessage1 = "No message sent";
    inputMessage2 = "No message sent";
  }
  Serial.print("GPIO: ");
  Serial.print(inputMessage1);
  Serial.print(" - Set to: ");
  Serial.println(inputMessage2);
  request->send(200, "text/plain", "OK");
});
```

A setup-ban kerül sor továbbá a szerver elindítására.

```
server.begin();
```

A loop folyamatosan fut és frissül, az én ciklusomban egy if feltétel található, ami folyamatosan nézi és ellenőrzi a szobahőmérsékletet, amennyiben eléri a 25 C fokot impulzust ad a piros lednek és az addig ég, amíg a hőmérséklet 25 fok alá nem esik.

```
void loop() {

  if(temp>=25.00)
    digitalWrite(LED_RED, HIGH);
  else
    digitalWrite(LED_RED, LOW);
}
```

Felhasználói interfész programok

A user interface (UI)-hez használt HTML weboldalt is Arduino IDE segítségével írtam meg, a **const char index_html[] PROGMEM = R"rawliteral** fejléctől kezdődően a **</html>)rawliteral";**-ig befejezve. Ebben a részben vannak a JavaScript funkciók is, illetve a CSS stílusjelek.

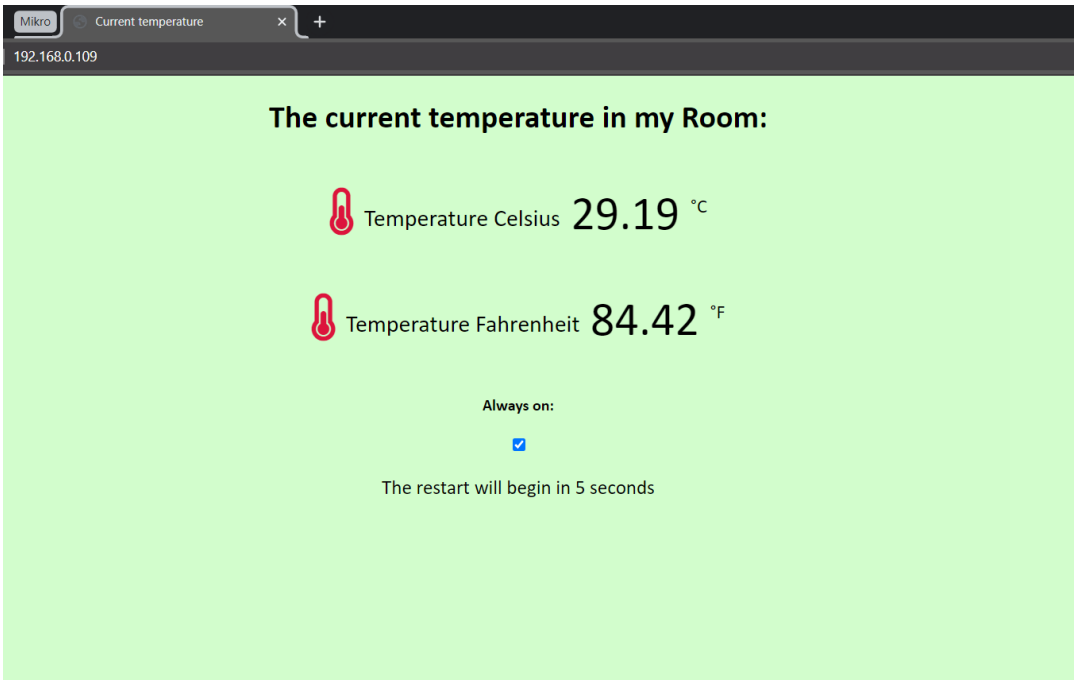
A weboldal alap stílusának a beállításai az alábbi kódrészlettel történtek:

```
<style>
  html {
   font-family: Calibri;
   display: inline-block;
   margin: 0px auto;
   text-align: center;
  }
  h2 { font-size: 3.0rem; }
  p { font-size: 3.0rem; }
  .units { font-size: 1.2rem; }
  .ds-labels{
    font-size: 1.5rem;
    vertical-align:middle;
    padding-bottom: 15px;
  }
  input:checked+.slider {background-color: #b30000}
  input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px)}
</style>
```

A placeholderek beállítása a következőképpen történt meg:

```
<body style="background-color:#d2fdcc;">
  <h1>The current temperature in my Room:</h1>
  <p>
    <i class="fas fa-thermometer-half" style="color:#DC143C;"></i>
    <span class="ds-labels">Temperature Celsius</span>
    <span id="temperaturec">%TEMPERATUREC%</span>
    <sup class="units">&deg;C</sup>
  </p>
  <p>
    <i class="fas fa-thermometer-half" style="color:#DC143C;"></i>
    <span class="ds-labels">Temperature Fahrenheit</span>
    <span id="temperaturef">%TEMPERATUREF%</span>
    <sup class="units">&deg;F</sup>
  </p>
  %BUTTONPLACEHOLDER%
```

A UI megvalósítva a következőképpen néz ki:



Összegzés

Az eddigiekben a szoftverek tervezése közben nem kellett figyelnem a hardver okozta nehézségekre, ismeretlen érzés volt, hogy a hibáimat nem csak a kódban, de az áramkörben is keresgélnem kellett.

Komoly problémát okozott a szenzorom bekonfigurálása, heteket töltöttem el a -127 Celsius probléma megoldásával, de amint sikerült ezt megoldanom nagy lépésekben kezdett el haladni a projekt. A hiba a tápáramra volt visszavezethető, megszokott módon a kódban kerestem a hibát, végső elkeseredettségemben mértem, ám ez vélhetően rutintalanságnak tudható be.

Összességében pozitívumként éltem meg projekttel töltött időt, a legtöbb részét élveztem, a legjobb az volt, hogy valami kézzel foghatót alkothattam a hardvernek köszönhetően.

Jövőbeli tervek

Mint ahogy az összegzőben említettem már élveztem a projekt elkészítésének minden pillanatát, rengeteget tanultam időközben az elektronikáról és a programozásról egyaránt.

Amint időm szabadul fel mindenképpen beruházok egy mikrokontrollerre, így a jelenlegi projektet is tovább fejleszthetem, például azzal, hogy a webszerver elérését globálissá teszem, vagy több szenzort használok, ezáltal komplexebbé téve a rendszert és a lefedettséget.

A mikrovezérlő tulajdonában új projektekbe is belekezdhetek, amelyekkel gyarapíthatom a tudásom és a tapasztalataim.

Bibilográfia:

https://eu.mouser.com/ProductDetail/Espressif-Systems/ESP32-DevKitC/?qs=chTDxNqvsyn3pn4VyZwnyQ==

https://www.adafruit.com/product/381

https://esp32.com/viewtopic.php?t=11904

https://www.arduino.cc/reference/en/

Programozható elektronikák, 2019 (PDF)

Learn ESP32 with Arduino IDE, 2020 (PDF)

ESP32 Web Server, 2019 (PDF)