Dokumentáció az IoT alapok (BMI1504L) tárgyhoz elkészített ESP32 prototípushoz

Készítették: Rinyu László, Kovács Kristóf, Fodor Géza

A projekt leírása, célkitüzés

A választott projektünk egy ajtónyitást, illetve általános mozgást érzékelő és riasztó prototípus, mely PIR szenzor segítségével figyeli a környezetet, s mozgás esetén többféle módon is visszajelzéssel szolgál. A rendszer nyugalmi állapotban „nincs mozgás” üzenetet ír az OLED kijelzőn. Riasztás esetén a LED folyamatosan pirosan világít, megszólal egy piezo hangjelző, s figyelmeztető üzenet jelenik meg.

A projekt során azt a célt tűztük ki magunk elé, hogy egy olyan ESP32 prototípust alkossunk, mely működő, szemléltetésre megfelelő, s alkalmas egy ajtó vagy kisebb helyiség felügyeletére. Ugyanakkor felépítésében kellően moduláris és bővíthető legyen ahhoz, hogy később akár kiegészíthető legyen, például WIFI-n keresztüli értesítésekkel vagy naplózással.

A prototípust Rinyu László alkotta meg, a vizuális szemléltetést Fodor Géza készítette, a dokumentációt Kovács Kristóf készítette.

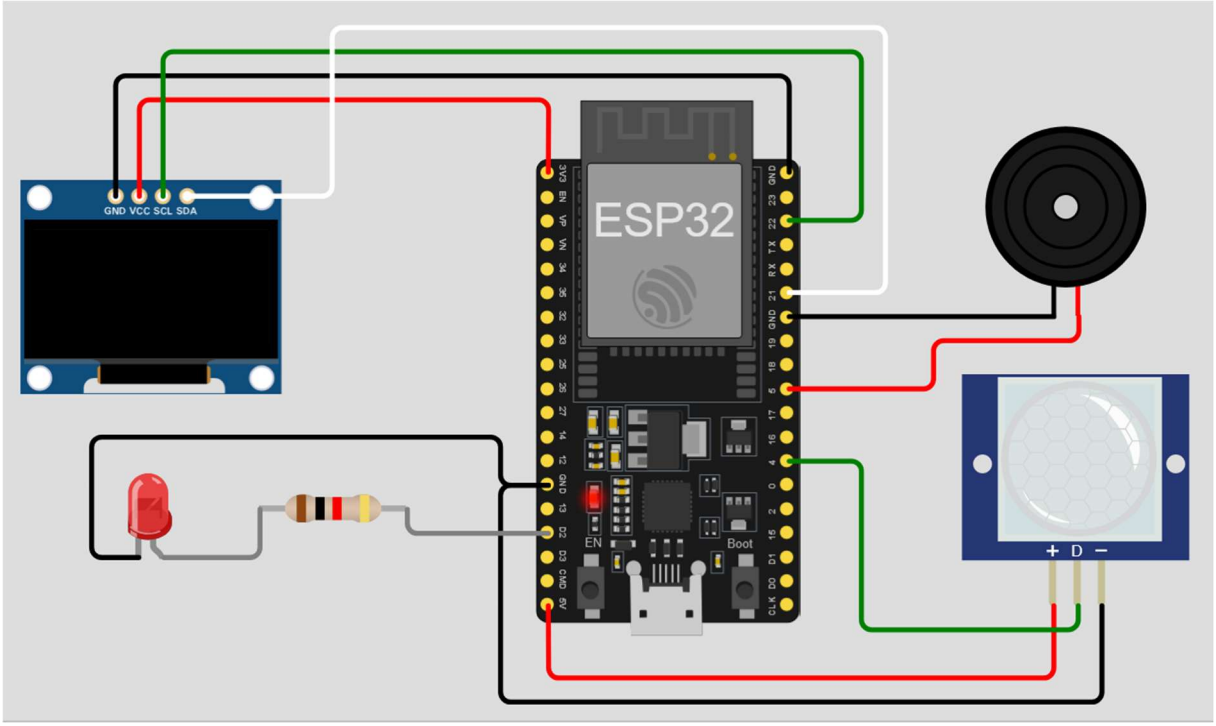
A modell fizikai felépítése

A projekthez egy ESP32 Starter Kit került megvásárlásra. A modell megépítéséhez a csomag tartalmából azokat válogattuk össze, amelyek közvetlenül szükségesek a kívánt funkciók megvalósításához.

A rendszer központja az ESP32, amelyet a mellékelt micro-USB kábellel csatlakoztattunk a számítógéphez, így biztosítva a tápellátást és a program feltöltését is. A mozgás érzékelését a HC-SR501 PIR szenzor végzi, ennek kimeneti jelét egy digitális bemenetre kötöttük, így az ESP32 egyszerűen, logikai szinten (LOW vagy HIGH) tudja megkülönböztetni a „nincs mozgás” és a „mozgás érzékelve” állapotokat. A felhasználó felé irányuló vizuális visszajelzést

egy piros LED biztosítja, amelyet egy 1kΩ-os soros ellenálláson keresztül egy ESP32-es kimenetre kapcsoltunk, így megakadályozva a túl nagy áramfelvételt és a LED, illetve a mikrokontroller kimenetének károsodását. A hangjelzésért az aktív piezo hangjelző felel. Ennek előnye, hogy a benne lévő elektronika miatt elegendő csupán egy digitális jelet ráadni, és automatikusan hallható sípoló hangot generál. A figyelmeztető szövegek megjelenítésére a mellékelt OLED kijelzőt választottuk, amely az I2C buszon keresztül, a 21-es (SDA) és 22-es (SCL) lábak segítségével kommunikál az ESP32-vel.

Felhasznált alkatrészek listája: ESP32, PIR szenzor, OLED kijelző, aktív piezo hangjelző, 1 darab piros LED, 1 darab ellenállás, próbapanel, vezetékek, micro USB kábel.



A rendszer logikai felépítése

A programot az Arduino IDE segítésével írtuk és töltöttük fel az ESP32-re.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_ADDR 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Pins
int pirPin = 4;    // D4
int ledPin = 2;    // D2
int buzzerPin = 5; // D5 (aktív piezo)
int stableState = LOW;
unsigned long lastChange = 0;
const unsigned long debounceTime = 500;

void setup() {
Serial.begin(115200);
pinMode(pirPin, INPUT);
pinMode(ledPin, OUTPUT);
pinMode(buzzerPin, OUTPUT);

Wire.begin(21, 22); // SDA, SCL
if(!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)) {
Serial.println("Nem találtam OLED kijelzőt!");
for(;;);
}

display.clearDisplay();
display.setTextSize(2);          // NAGY betűméret
```

```
display.setTextColor(SSD1306_WHITE);
display.setCursor(0, 10);
display.println("PIR indul...");
display.display();

Serial.println("PIR mozgás monitor indul...");
}

void loop() {
int pirReading = digitalRead(pirPin);

if (pirReading != stableState && (millis() - lastChange > debounceTime)) {
stableState = pirReading;
lastChange = millis();

display.clearDisplay();
display.setTextSize(2);        // NAGY betűméret
display.setCursor(0, 10);

if(stableState == HIGH) {
display.println("Mozgas!");
Serial.println("Mozgas!");
digitalWrite(ledPin, HIGH);
digitalWrite(buzzerPin, HIGH); // aktív piezo
delay(300);                    // sípolás hossza 1000 ms = 1 sec
digitalWrite(buzzerPin, LOW);

} else {
display.println("Nincs\nMozgas");
Serial.println("Nincs mozgas.");
digitalWrite(ledPin, LOW);
digitalWrite(buzzerPin, LOW);  // piezo le
}
```

```
display.display();
}

delay(50);
}
```

A kód elején a szükséges könyvtárak betöltése történik, mely biztosítja az I2C kommunikációt és a kijelző grafikus kezelését. Rögzítjük a kijelző felbontását, meghatározzuk az I2C címet. Ezután globális változók segítségével rögzítjük az ESP32 lábkiosztását, mivel ezekhez a GPIO-khoz van csatlakoztatva a PIR szenzor kimenete, a LED, illetve az aktív piezo hangjelző. Azért emellett az elrendezés mellett döntöttünk, mert így a kód elején, egy helyen látszik minden fontos lábkötés, s ha később módosítani kell a bekötést, elég ebben a részben szerkeszteni a kódot. Debounce-ként a stableState és a lastChange változókat használtuk. A stableState a PIR szenzor utolsó stabil állapotát tárolja (HIGH vagy LOW), a lastChange az utolsó állapotváltás időbélyegét tárolja milliszekundumban kifejezve, s a debounceTime 500-ra van állítva, mely meghatározza, hogy hány milliszekundumnak kell eltelnie két érdemi állapotváltás között, hogy azt a program valódinak tekintse.

A setup függvényben először inicializáljuk a hardver és a kommunikációs perifériákat. Beállítjuk a soros kommunikáció sebességét, a pinMode sorokkal pedig megadjuk, hogy melyik láb bemenet, illetve kimenet, így a PIR jelet betudjuk olvasni, ezzel vezérelve a LED-et és a piezót. A Wire.begin-ben megadjuk hogy az I2C busz SDA vonala a GPIO21-hez, az SCL pedig a GPIO22-höz kapcsolódik. A display.begin inicializálja az OLED kijelzőt. Hibakezelésként, amennyiben ez sikertelen, a program végtelen ciklusba lép, s a monitor kiírja, hogy „Nem találtam OLED kijelzőt!”. A kijelző inicializálása után töröljük a képernyőt, beállítjuk a karakterek méretét és színét, s meghatározzuk a kezdőpozíciót. A „PIR indul…” és a „PIR mozgás monitor indul…” üzenetekkel adunk visszajelzést arról, hogy a rendszer indul.

A loop függvény valósítja meg a folyamatos működést. A pirReading beolvassa a PIR szenzor akutális logikai állapotát (0 vagy 1, tehát LOW vagy HIGH). Ezután debounce történik: „if (pirReading != stableState && (millis() - lastChange > debounceTime))”. Ez a feltétel két részből áll. Az első, pirReading != stableState, azt vizsgálja, hogy megváltozott-e a szenzor állapota az utolsó stabil értékhez képest. A második, millis() - lastChange > debounceTime, azt ellenőrzi, hogy a legutóbbi állapotváltás óta eltelt-e már legalább 500 milliszekundum. A millis() függvény az ESP32 bekapcsolása óta eltelt időt adja vissza. Ha mindkettő feltétel

teljesül, akkor a programnak ez egy valódi állapotváltás, frissíti a stableState változót, s elmenti az időt a lastChange-be, és a frissíti a kijelzők és a kimenetek állapotát.

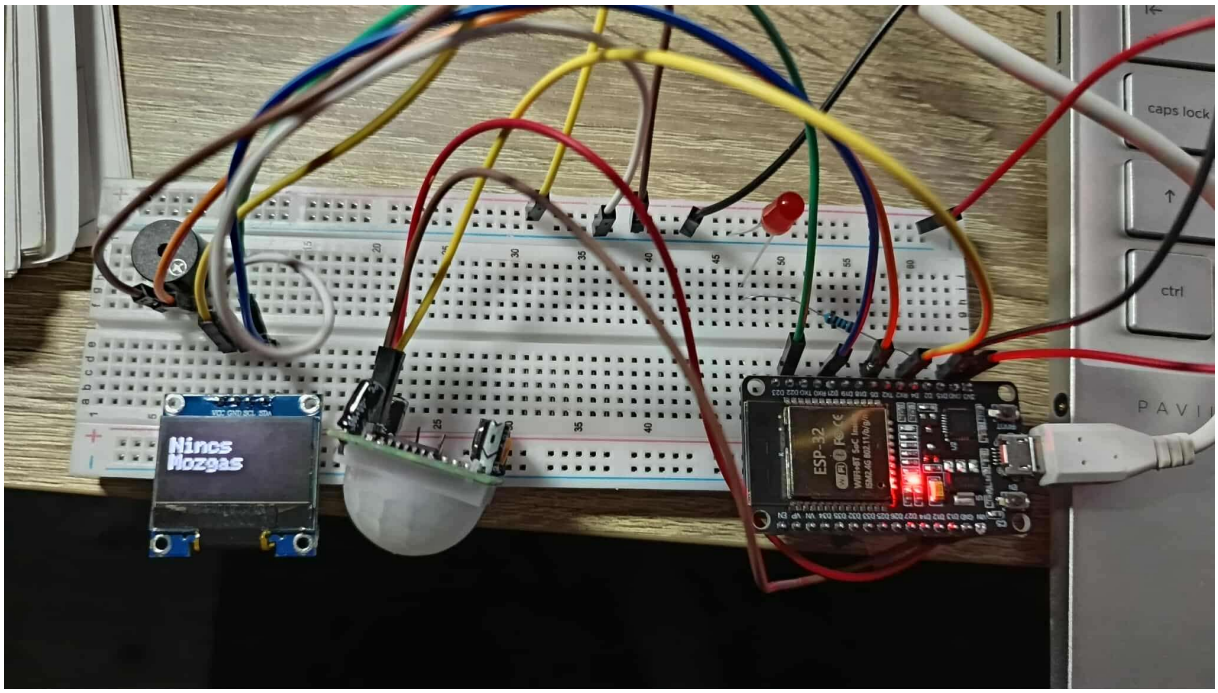
Állapotváltásnál először a kijelzőről töröljük a régi üzenetet, és pozícionáljuk az új üzenetet. Amennyiben stableState == High, akkor a PIR szenzor mozgást jelez. A program az OLED panelre kiírja a „Mozgas!” szöveget, s a soros monitorra is kiírja ugyanezt, ezután bekapcsolódik a LED, és megszólal az aktív piezo hangjelző. 300ms-ig késleltetjük a programot, ez idő alatt sípol a piezo. 300ms után lekapcsoljuk a hangjelzőt. A LED világít mindaddig, amíg a PIR nem jelez mozgást.

Amennyiben a stableState LOW-ra vált, tehát nincs mozgás, a kijelző és a konzolon kiírja a „Nincs Mozgas” feliratot. Ezután lekapcsol a LED, és a piezo lekapcsolását ismét megtesszük.

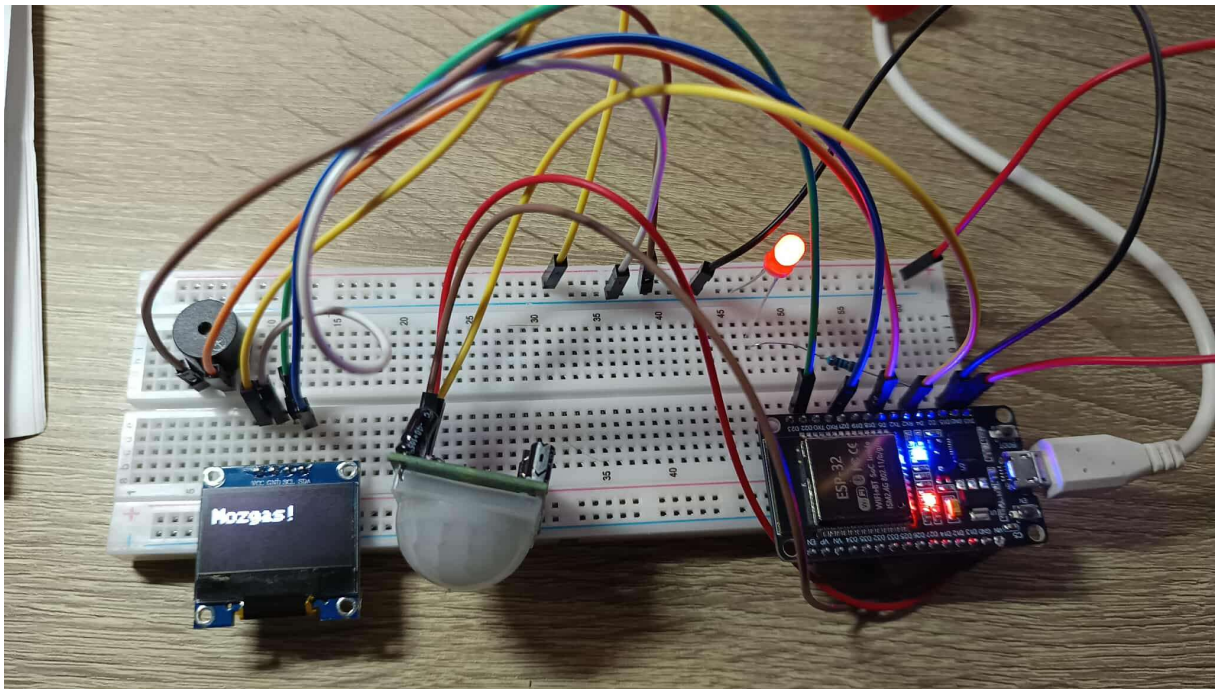
A prototípus tesztelése

A modell tesztelése otthoni, szobakörülmények között történt, szimulálva egy ajtónyitás-érzékelő tipikus használati környezetét. PIR szenzort úgy került elhelyezésre, hogy a megfigyelni kívánt terület egy ajtó előtti sáv legyen, így tehát amikor valaki belép a helyiségbe vagy kinyitja az ajtót, a mozgása a szenzor látómezőjébe essen.

A tesztelés első fázisában elsősorban a rendszer alapfunkciói kerültek ellenőrizésre. Nyugalmi állapotban, amikor a szenzor látóterében nem tartózkodott senki, a LED nem világított, az aktív piezo hangjelző néma maradt, az OLED kijelzőn pedig a „Nincs Mozgas” felirat jelent meg. Ezzel párhuzamosan a soros monitoron is megjelent a „Nincs mozgas.” üzenet, ami segített annak ellenőrzésében, hogy a szenzor kimenete valóban alacsony szinten van, és a program helyesen értelmezi az állapotot.



Ezt követően különböző módokon az ajtónyitás került szimulálásra. A helyiségbe beléptek, elhaladtak a szenzor előtt, illetve gyorsabb és lassabb mozdulatokat végeztek. Mozgás esetén a rendszer a várakozásoknak megfelelően reagált. A PIR kimenete HIGH-ra váltott, az ESP32 riasztási módba lépett, a LED felgyulladt, az aktív piezo rövid, jól hallható sípoló hangot adott, az OLED kijelzőn pedig „Mozgas!” felirat jelent meg nagy betűmérettel. A soros monitoron ugyanekkor a „Mozgas!” szöveg volt olvasható, ami megerősítette, hogy a szoftveres logika a megfelelő pillanatban hajtódik végre.



A tesztelés során külön figyelem hárult arra, hogy a debounce mennyire hatékonyan csökkenti a téves riasztások számát. Ennek ellenőrzésére több olyan helyzet is szimulálásra került, amikor a szenzor a határán volt annak, hogy érzékelje-e a mozgást, illetve rövidebb mozdulatokat végeztünk (gyors ki-be mozgás a látómező szélén). A 500 ms-os minimális állapotváltási időnek köszönhetően a rendszer nem villogtatta a LED-et és nem frissítette feleslegesen a kijelzőt minden egyes rövid jelzésre. A riasztás csak akkor aktiválódott, amikor a mozgás egyértelmű és tartós volt. A gyakorlatban ez azt jelentette, hogy ha valaki ténylegesen belépett az ajtón, a riasztó megbízhatóan jelzett, viszont apró mozgásra nem reagált feleslegesen.

Továbbfejlesztési lehetőségek

A bemutatott prototípus számos irányban továbbfejleszthető. A beépített Wifi modult több funkcióval is kihasználható lenne. A rendszer képes lenne nem csak helyben jelezni, hanem saját vagy felhő-alapú kiszolgálón keresztül értesítést küldeni, például emailt, push értesítést, vagy éppen MQTT üzenetet, s akár lehetőség lenne távoli felügyeletre, vagy a riasztásokat időbélyegzővel ellátva naplózni.

Hardvert tekintve további modulok bevonásával még több funkcióval rendelkezne a modell. Hőmérséklet- és páratartalom-érzékelő integrálásával akár környezeti feltételekhez kötötten lehetne a riasztást állítani. Fényérzékelő modullal csak éjszakai módban is működhetne a prototípus.

A felhasználói élményt jelentősen javítaná, ha a készletben található nyomógombokkal a riasztót lehetne élesíteni vagy hatástalanítani. Akár PIN-kódos belépés vagy az OLED kijelzőn lehet egy menürendszer, melyben beállíthatók lennének az érzékenységi küszöbök, riasztási módok és a hálózati paraméterek.