

Önálló laboratórium dokumentáció

Wágner Árpád (O2OFFX)

2019/20 - 2. félév

1. Bevezetés

Ez a dokumentum tartalmazza az önálló laboratórium tárgy keretében elvégzett munkám dokumentációját.

A munka rám háruló része két felé osztható: az első részben feladatom volt egy bizonyos ESP32 lapka megismerése és lehetőségeinek felderítése. A második részben pedig egy, a későbbiekben diagnosztikai eszközként használandó mikrofonnal foglalkoztam.

Megjegyzés: A laboratórium során létrejött forráskódok [ezen](https://github.com/awrrpad/Onlab)¹ a GitHub repository-n elérhetőek.

¹A repository URL-je: <https://github.com/awrrpad/Onlab>

2. „Nagykijelzős” ESP32

2.1. Bevezetés

Mint azt a bevezetőben említettem, a félév első részében egy ESP32-re épülő fejlesztői lapkával foglalkoztam. Az ESP32 egy kedvező árú, de erős mikrokontroller-család, melyről érdemes tudni, hogy beépített Wi-Fi-vel és Bluetooth-szal rendelkezik. A szóban forgó eszköz ezen funkcionalitáson túl rendelkezik egy beépített MPU9250-el (giroszkóp, gyorsulásmérő, iránytű), SD kártya olvasóval, egy egyszerű hangszóróval és kijelzővel illetve három előlapi gombbal.

Az eszköz programzása Arduino IDE-n keresztül lehetséges, de az ESP32 alaplapkönyvtár hozzáadása szükséges (ennek a mikéntjéről például [itt](#)² található egy leírás). Ha ez megtörtént, az **Eszközök > Alaplap** menüpontnál a felkínált lehetőségek közül válasszuk az **ESP 32 Pico Kit**-et. Így a megírt programunkat már feltölthetjük az eszközre.

2.2. Tapasztalatok

Előjáróban szeretném megjegyezni, hogy a következőkben megemlített könyvtárak használatához szükséges egyes dolgokat beállítani az eszköz sajátosságainak figyelembevételével, ilyen beállítások a bevezetőben említett GitHub repository-ban lévő **sound2.ino**³ fájlban találhatóak. Ebben a fájlban az itt ismertetett funkciók nagyrésze is megtalálható.

A tapasztalatok részletesebb leírását a **kijelzővel**⁴ kezdeném, hisz ez talán az eszköz legérdekesebb pontja. A kijelző használatához ajánlott egy már meglévő könyvtárat használni, én az Adafruit_ST7735-öt⁵ használtam, ha ezt a könyvtárat az Arduino IDE-n keresztül telepítjük (**Eszközök > Könyvtárak kezelése... > {könyvtár megkeresése a keresőben} > Telepítés**), automatikusan letölti az Adafruit_GFX könyvtárat és minden egyebet, amikre ezek építenek. Az első dolog ami a kijelző kapcsán eszünkbe juthat, hogy szeretnénk egy egységes színt beállítani. Ezt a **fillScreen()** metódussal valósítható meg. Ez egy argumentumot vár, mégpedig a színt, amivel ki szeretnénk tölteni a képernyőt. Ehhez

²Az útmutató URL-je: <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

³A fájl URL-je: <https://github.com/awrpad/Onlab/blob/master/Arduino%20ESP/src/sound2/sound2.ino>

⁴A kijelzőt a használat előtt inicializálni kell, mely megtehető ezzel a kódsorral: `Adafruit_ST7735 tft = Adafruit_ST7735(16, 17, 23, 5, 9);`

⁵Adafruit_ST7735: <https://github.com/adafruit/Adafruit-ST7735-Library> Az itt található fájlok közül a Adafruit_ST7735.h fájlt „inklúdoltam”.

a feljebb említett könyvtár tartalmaz színdefiníciókat, így a teljes utasítás nézhet ki például így: `tft.fillScreen(ST7735_BLACK);`. Itt azonban meg kell jegyezni, hogy az itt definiált színek sajnos nem feltétlenül működnek tökéletesen. A `ST7735_CYAN` használatával például a sárga színnel lesz kitöltve a képernyő (de a `ST7735_BLACK` például a várt módon feketével teszi ugyanezt). Az első képernyő átszínezés után észre fogjuk venni, hogy ez a módszer lassú. Szemre azt lehet mondani, hogy úgy egy másodperc környéki időt vesz igénybe. Nyilvánvaló, hogy ez a sebesség erősen korlátozza a modul felhasználhatóságát. Például, ha egy értéket szeretnénk rendszeresen frissítve megjeleníteni a képernyőn, a törlés és kiírás ideje könnyen lehet túlzottan lassú. Az ilyen és ehhez hasonló helyzetek megoldására használhatjuk például a `fillRect()` metódust, mely egy megadott téglalapot a megadott színnel kitölt. Ilyen módon nem szükséges az egész kijelzőt frissítenünk, lehetséges csupán a megadott terület egy színnel való kitöltése. Ezt kombinálva például a `tft.print("uzenet")` metódussal aránylag gyorsan is képesek vagyunk változó értékeket a képernyőn megjeleníteni. Az egyetlen problémát az így létrejövő villódzás jelenti, de az értékek még ilyen módon is könnyen leolvashatóak maradnak (amint ez látszik akkor is, ha a fent említett vázlatot futtatjuk a panelen). Természetesen az így megjelenő szöveg színe illetve mérete is befolyásolható a `tft.setTextSize(1)` illetve `tft.setTextColor(0x5FCC)` metódusokkal. Ha nem megfelelő a szöveg elhelyezése, kiírás előtt a `tft.setCursor(0, 0)` metódussal a kurzort a képernyőn belül szabadon bárhova elhelyezhetjük. A kijelzőhöz kapcsolódóan érdemes még megjegyezni, hogy annak rotációját is beállíthatjuk. Ha ezt a kódban nem tesszük meg, alapértelmezés szerint „portré” módba lesz az eszköz orientációja beállítva (mint egy okostelefon: allul gombok, felfül a képernyő a hosszabb oldalakat bal/jobbs oldalra használva). Ezt módosítani a `tft.setRotation(1)` metódussal lehet, ami például az itt használt 1-es értéket megkapva a kijelző tájkép orientációjú lesz, a gombokkal a jobb oldalon.

A másodikként bemutatásra kerülő funkció a **hangszóró**. Ennek a használatához először is szükség van egy könyvtárra, melyet a következőképp használhatunk: `#include "esp32-hal-ledc.h"` (Figyelem! Az idézőjelek használata relációs jelek helyett nem véletlen, a megfelelő alaplap kiválasztása után ez a könyvtár így használatos). Hogy megfelelően használhassuk, a setup részben be kell állítani a hangszórót, ezt az `ledcSetup` illetve `ledcAttachPin` metódusok megfelelően paraméterezett hívásával tehetjük meg. A megfelelő paraméterek a mellékelt forráskódban megtalálhatóak. A beállítás után az `ledcWriteTone(TONE_PIN_CHANNEL, 440)` metódushívással játszhatunk le hangot, első paraméterként a megfelelő (és beállításnál korábban már használt) csatornát, míg második paraméterként a lejátszani kívánt frekvenciát

kell átadni Hz-ben (a példa tehát egy zenei A hangot fog lejátszani a hangszórón). A megadott hang lejátszása egészen addig folytatódik, amíg meg nem adunk egy újabb lejátszandó hangot. Ilyen módon ha nem akarjuk tovább játszani az addigi hangot, meghívhatjuk az imént említett metódust 0 frekvenciaértéket átadva. Mivel több ilyen hang egymás utáni lejátszása nagyon repetitív forráskódot eredményezne, ráadásul úgy, hogy egy paraméter mindig fix, írtam egy egyszerű függvényt `void playTone(int freq)` interfésszel, mely az átadott frekvenciát 200 miliszekundumig játsza le szálblokkoló módon.

A következő funkció a beépített **MPU9250** szenzor. Amint azt a bevezetőben is említettem, ennek a szenzornak három különböző funkcionalitása van:

- Giroszkóp
- Gyorsulásmérő
- Iránytű

Mint eddig, most is akkor lesz a legegyszerűbb a dolgunk, ha egy könyvtárat használunk. Ehhez a szenzorhoz több könyvtár is lehet találni, melyek közül próbáltam többet is használni, nekem a `MPU9250_asukiaaa`⁶ elnevezésűt sikerült működésre bírnom, a többivel különböző problémáim adódtak. Az MPU9250 használatához ezen kívül szükség van a `Wire.h` könyvtárra is, aminek beállítása szintűgy szükséges. Ezt a `Wire.begin(SDA_PIN, SCL_PIN);` kódsorral tehetjük meg⁷. Amint ezekkel a beállításokkal elkészültünk, a szenzor könyvtárát „össze kell kötni” a `Wire` library-vel, ezt a következőképp lehet: `mpu.setWire(&Wire);`. Továbbá szükséges még a használni kívánt funkciók inicializálása, amit a `begin...` alakú metódushívásokkal tehetünk meg. A következő kódrészlet például megteszi ezt a gyorsulásmérővel és az iránytűvel:

```
mpu.beginAccel();  
mpu.beginMag();
```

Miután sikeresen beállítottunk mindent, a szenzort a következőképp használhatjuk: először a megfelelő `...Update` végződésű metódus hívásával frissítjük a szenzor megfelelő adatait, majd a `...<[X][Y][Z]>` alakú függvénnyel kiolvassuk a kívánt adatot. A következő kódrészlet például frissíti és kiolvassa a gyorsulásmérő szenzor adatait a három tengely mentén:

⁶A felhasznált könyvtár és dokumentációja elérhető itt: https://github.com/asukiaaa/MPU9250_asukiaaa

⁷Amint az a fájlban is látható, az `SDA_PIN` értéke 19, míg az `SCL_PIN` konstanst 18-ra kell állítani.

```
// After setting up the MPU9250 and the screen
float aX, aY, aZ;

mpu.accelUpdate();
aX = mpu.accelX();
aY = mpu.accelY();
aZ = mpu.accelZ();
```

Hasonló módon kell dolgozni a giroszkóppal is (*gyro*) és az iránytűvel (*mag*) is.

Hátra van még az **SD kártya foglalat** bemutatása⁸. Ahogy eddig, most is érdemes egy könyvtárat használni. A lapka dokumentációja alapján a `mySD.h` fájl használata javasolt, azonban ezt telepítenünk kell. Ráadásul sajnos nem lehetséges a szokásos módon a „Könyvtárak kezelése” menüpontnál importálni, hanem le kell tölteni GitHub-ról: <https://github.com/nhatuan84/esp32-micro-sdcard>. Itt töltsük le az egész repository-t ZIP állományként, ezután az Arduino IDE-ben: Vázlat > Könyvtár tartalmazása... > .ZIP könyvtár hozzáadása... > *válasszuk ki a letöltött ZIP fájlt* > Open. Ezzel a fejlesztőkörnyezet megcsinál minden szükséges lépést és innentől kezdve a könyvtárat használhatjuk a vázlateinkhoz. Ebben az esetben is az inicializálással kell kezdeni, amelyet jelen esetben a `SD.begin(13, 15, 2, 14)`⁹ függvényhívással tehetünk meg. Mely tájékoztat is minket a sikességről: ha sikerült az SD modul inicializálása, `true` értékkel tér vissza, különben `false`-al.

2.3. Konklúzió

Általánosan elmondható, hogy maga a fejlesztői eszköz hasznos és sokrétű. Különböző alkalmazásokban gyakran előforduló elemek találhatók meg benne beépítve, ami fejlesztés során sok forrasztástól és kábelrengetegről kímélhet meg minket. Azonban sajnálatos a megfelelő dokumentáció hiánya, így a vele történő korai munka nehézkes lehet, plusz szerintem a GPIO pinek elhelyezése sem optimális. Azonban amint megismerkedtünk a lapkával a rendelkezésre álló mintakódok alapján, ha tisztában vagyunk a korlátaival, összességében egy sokrétűen és jól használható eszköz lesz a kezünkben.

⁸Ennek a résznek a mintakódja nem a korábban említett fájlban, hanem a <https://github.com/awrpad/Onlab/blob/master/Arduino%20BESP/src/esp32/esp32.ino>-ban található.

⁹A megadott számok ezen esetben nem példák, a felhasznált modulhoz pontosan ezek az értékek kellenek. Más eszköz használata esetén természetesen ezek változhatnak.

3. Mikrofon és spektrumanalizátor

3.1. Bevezetés

A félév során a másik nagyobb feladatom egy mikrofon modul használata volt, illetve egy az így beolvasott hangok folyamatos elemzésére alkalmas szoftver készítése.

3.2. A mikrofon modul

Ez a modul egy egyszerű eszköz, melynek csupán három lába van, ebből egy kell a 3.3 vagy 5 Voltnak, egy a földelésnek, a maradék egy lábon pedig a beolvasott értékeket kaphatjuk meg analóg módon. Ebből fakadóan csak olyan eszközzel használható közvetlenül, melyben van beépített A/D konverter (így például egy Raspberry Pi önmagában nem elég). Viszont egy ilyen miniszámítógép teljesítményére szükségünk van az adatok feldolgozásához, ennek áthidalására természetesen használhatunk egy külső analóg-digitális átalakítót, én azonban azt a megoldást úgy oldottam meg, hogy az adatokat egy Arduino UNO segítségével olvastam be és onnan továbbítottam a Raspberry felé.

Az ezen feladathoz kapcsolódó forráskódok két fájlban találhatóak meg, az Arduino-n futtatott kód a [soundComm.ino](https://github.com/awrrpad/Onlab/blob/master/Arduino%20BESP/src/soundComm/soundComm.ino)¹⁰-ban, míg a Raspberry Pi-n lévő kód a [spectrum_analyzer.py](https://github.com/awrrpad/Onlab/blob/master/RPi/src/spectrum_analyzer.py)¹¹ fájlban.

3.3. Adatok beolvasása

Amint az előző részben említettem, a mikrofon adatait analóg módon lehet beolvasni például egy Arduino lapkával. Mivel az adatokat egyesével átküldeni eléggé pazarló és lassú lenne, így azt a megoldást választottuk, hogy először magán az Arduinon beolvasunk és eltárolunk valamennyi adatot, majd ezeket egy csomagként küldjük át a Raspberry-re. Végül az 500 darab beolvasott érték / csomag méretet választottam, mert ez még kényelmesen elfért az Arduino UNO memóriájában (míg például 1000 darab már nem fért volna be). Egy ötlet lehetne, hogy az `int` típus helyett `short` típust használjunk, azonban ezen az Arduino típuson ez nem jelentene különbséget¹². Azonban

¹⁰A `soundComm` fájl URL-je: <https://github.com/awrrpad/Onlab/blob/master/Arduino%20BESP/src/soundComm/soundComm.ino>

¹¹A `spectrum_analyzer` fájl URL-je: https://github.com/awrrpad/Onlab/blob/master/RPi/src/spectrum_analyzer.py

¹²Vesd össze: <https://www.arduino.cc/reference/en/language/variables/data-types/int/> és <https://www.arduino.cc/reference/en/language/variables/data-types/short/>

az ennél kisebb, egy bájtos típus mérete pedig nem lenne elég, hiszen az `AnalogRead()` függvény által visszaadott értékek 10 bitesek.