ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

Iványi Bálint László
T008139/FI12904/K

OE-KVK
2022.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Iványi Bálint László
Szakdolgozat száma: SZD22030109186987
Törzskönyvi száma: T008139/FI12904/K **Neptun kódja:** [redacted]

Szak: villamosmérnöki (magyar nyelven)
Specializáció: Műszer-automatika specializáció

A dolgozat címe: Multimédiás megoldás ESP32-vel
A dolgozat címe angolul: Multimedia solution with ESP32
A feladat részletezése: Készítsen audiójellel vezérelt fénytechnikai rendszert WEB-es technológiával kiegészítve!

Intézményi konzulens neve: Sándor Tamás

A kiadott téma elévülési határideje: 2025. március 1.
Beadási határidő: 2022. 05. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2022. 03. 21.

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikus Kommunikációs Rendszerek Intézet
P.H

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom.

.....
belső konzulens

.....
külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZETI
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 2022.05.13.

Iványi Bálint László

.....

hallgató aláírása

Tartalomjegyzék

1. Bevezető 3

2. Specifikáció 5

3. Rendszer felépítése 6

3.1. Vezérlő eszközök 7

3.2. Hangtechnika 8

3.3. Fénytechnika 9

3.4. Tápfeszültségellátás 12

4. Rendszer működése 13

4.1. Audió jel mérése 13

4.2. Spektrum analízis számítása 16

4.3. Eszközök kommunikációja 19

4.4. Mátrix (címezhető) LED szalag vezérlése 20

4.5. DMA technológia használata 24

5. Rendszerhez tartozó kezelőfelület 26

5.1. Webszerver működése (backend) 26

5.2. Kliens oldal működése (frontend) 28

5.3. Beállítható paraméterek 30

5.4. Kezelőfelület megjelenése 31

6. Szoftveres háttér 33

6.1. ARM mikrokontroller szoftver működése 33

6.2. ESP32 mikrokontroller szoftver működése 35

6.3. Kliens oldali frontend szoftver működése 36

7. Összefoglalás 38

7.1. Elkészült projekt 38

7.2. Fejlesztési lehetőségek 39

8. Irodalomjegyzék 40

8.1. Felhasznált elektronikus jegyzetek 40

8.2. Tájékozódásra használt internetes források 40

8.3. Felhasznált eszközökhöz tartozó adatlapok 40

1. Bevezető

Egy zeneszerkesztő szoftverben megismerkedtem azzal, hogy különböző hangokat milyen jelgenerátorok segítségével lehet előállítani, akár hangszerek hangzását hogyan lehet elektronikus módon generálni. Ezzel tudomásomra jutott, hogy milyen érdekes összefüggő harmonikus jelekből tud összeállni egy zenei hang. Így elkezdtem azzal foglalkozni, hogy ezeket a jeleket megmutatni hogyan lehetne. Adott hangsávban milyen mértékben, mely jelek vannak jelen egyszerre.

Tehát az audió jel feldolgozása és vizualizációja a célom, mellyel már projekt I-II. kurzusaim óta is kísérleteztem több módon. Eleinte az ismereteim csak a digitális szűrőkre terjedtek ki. Így első próbálkozásomkor 3db kimenetet szerettem volna, hogy a jelben épp mekkora mértékben vannak jelen a mély, közép és magas frekvenciájú komponensek. Ezeket alul-, sáv- és felüláteresztő digitális szűrőkkel tudtam megvalósítani. Ezeknek a határait Matlab program segítségével tudtam beállítani rendesen. Matlabból kigenerált szűrők paramétereivel ekkor még csak a 8bites T-Bird fejlesztői panelen lévő Atmega-128 mikrokontrolleren futó algoritmus számolt. A digitális szűrőket megvalósító algoritmus kimeneteit szerettem volna vizuálisan is megjeleníteni. RGB LED szalag itt jött a képbe, mert a 3db kimenet pont az R, G, B csatornák számával azonos.

Az egyetemen a tanulmányaim előrehaladtával megismertem az FFT algoritmusát. Továbbá nagyobb számítási teljesítményű 32bites mikrokontroller (STM32 ARM Cortex-M7-es család) programozását is. Így újabb jelfeldolgozási módszerekre tettem szert. Spektrumképet megjeleníteni hagyományos RGB LED szalagon nem lenne egyszerű, így került a képbe pixelenként vezérelhető RGB LED szalag, amin már ez kivitelezhető. Ezt legegyszerűbben úgy lehet kivitelezni, hogy egy-egy pixel a LED szalagon egymást követő spektrumvonalak, csak a LED fényereje fogja megmutatni az adott tartományban jelen lévő komponensek intenzitását. Ez az effekt lett az egyik, ami audió vizualizációt valósít meg.

Attól a ponttól kezdve, hogy ezt a LED szalagot beépítettem egy régi hangfalba, úgy már nem is az audió vizualizáció lett minden egyes effektnél a célom. Végül így ebben az elkészült projektben a látványosság mellett a *„diszkós”* fények, effektek leutánzására is fektettem némi hangsúlyt. A kezelőfelületen látható effektek között például a beállítható stroboszkóp is ezért kapott helyet.

Továbbá szeretném megmutatni, hogy ESP32-vel irányított projektben Androidra megírt applikáció nélkül is meg lehet valósítani vezeték nélküli távolról elérhető vezérelhetőséget. Webes technológiával például olyan előnyökre is szert tehetünk, hogy lényegében bármilyen platformról elérhetővé tesszük a projektünket, aminek csak egyszerűen annyit kell tudnia, hogy betudjon tölteni egy weboldalt.

1. ábra: teljes hosszában bekapcsolt LED szalag

2. Specifikáció

A hangfalba beépített mátrix RGB LED szalagon audió vizualizáció megjelenítése. Ehhez szükséges analóg audió jel mérése és digitális jellé konvertálása a feldolgozáshoz elegendő értéken fenntartott jel/zaj viszonnyal és kellően nagy felbontásban.

A LED szalagon történő megjelenítés a következők alkalmazását foglalhatja magában:

- szín módosítása adott LED-en, vagy több LED-et tartalmazó szakaszon,
- fényerő változtatása adott LED-en, vagy több LED-et tartalmazó szakaszon,
- futófény gyorsaságának növelése, csökkentése (akár irány változtatásával),
- felsoroltak közül tetszőlegesen bármennyi kombinációja egymással

Megjelenített effektek működése az alábbiakban felsoroltaktól függhetnek:

- audió jel hangereje,
- audió jel spektrumképe,
- kezelőfelületen beállított paraméter

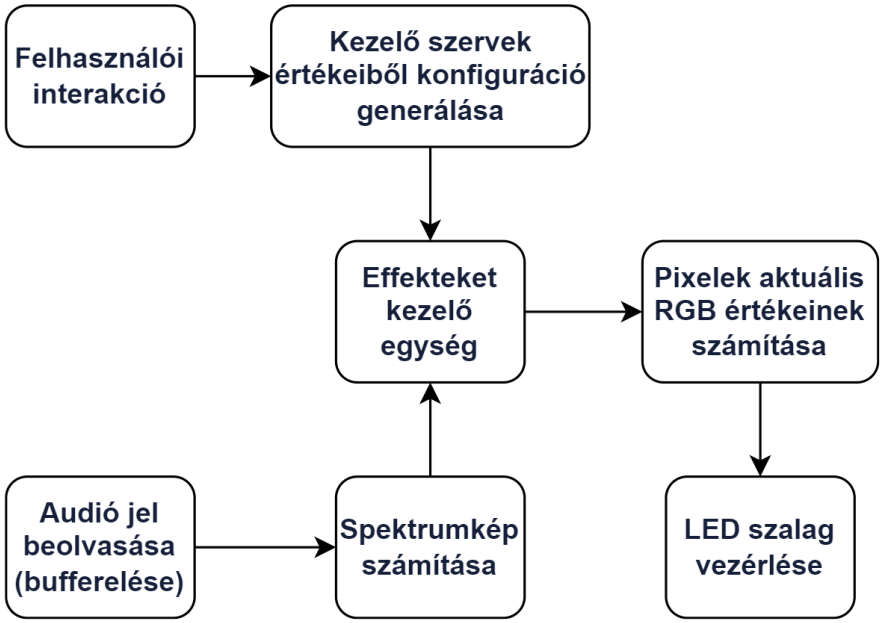
Rendszer megépítése ESP32-vel úgy, hogy a fő vezérlő egységet kiegészítse, mint egy bővítő modul. Az ESP32 megvalósítson belső kommunikációt a fő vezérlő egységgel, amin keresztül felkonfigurálhatja paraméterekkel a megjeleníteni kívánt szín- és fényjátékot.

Egy olyan rendszer megalkotása az ESP32-vel, mely vezeték nélküli kapcsolaton keresztül tudja a paramétereket bekonfigurálni. Ne kelljen alkalmazásokat telepíteni, minél jobban platform független tudjon maradni. Bármilyen új eszköz a lehető legegyszerűbben tudjon a meglévő rendszerrel kapcsolatba kerülni.

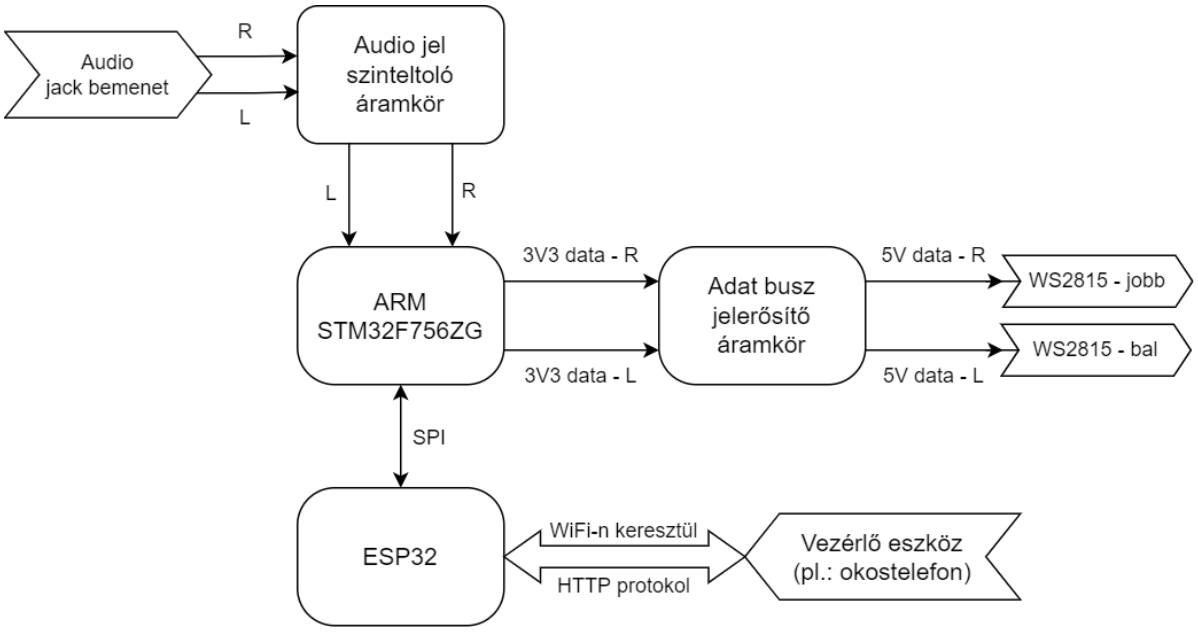
Kezelőfelülete illeszkedjen a megjelenítésre szolgáló képernyőre, ne legyen nehézkes a kezelő szervek használata se érintő képernyős eszközökön, se a hagyományosan egér és billentyűzettel használt eszközökön.

3. Rendszer felépítése

3 fő egységre lehet bontani a megépített rendszert. Ami az egész vázaként szolgál, az a hangfalpár, mint hangtechnika. Az ebbe a két hangfalba beépített LED szalag, mint a fénytechnika. Továbbá még a vezérlő eszközök, amik ezt az egészet vezérlik, a fő mikrokontroller, és a WiFi kapcsolódást lehetővé tevő modul.



2. ábra: Logikai rendszerterv



3. ábra: Fizikai rendszerterv

3.1. Vezérlő eszközök

Fő vezérlő egységnek egy Nucleo-144 fejlesztői boardon lévő 32bites STM32F756ZG ARM Cortex M7 mikrokontrollert használok. WiFi-s ESP32 mikrokontrollert pedig egy NodeMCU-ESP-32S nevű fejlesztőeszközön használom, amiben Espressif WROOM típusú ESP-32 chipet építettek be.

A bevezetőben említett 8bites Atmega-128 mikrokontroller, mint feldolgozó egység azért vált célszerűvé, hogy lecseréljem, mivel a gyors Fourier transzformációhoz (FFT – Fast Fourier Transform) –a minél pontosabb eredmények érdekében– nagyobb számítási kapacitás szükséges. Az algoritmus a számoláshoz lebegő pontos (float) típusú komplex számokat tárol le. Ez a változó a memóriában több, mint 8bit, így a 8bites mikrokontroller nem lesz annyira hatékony. Továbbá 16Mhz az órajele. Rendelkezésemre állt az említett ARM mikrokontroller, ami maximálisan 216Mhz órajelen képes dolgozni, és 32bites mikrokontrollerként sokkal alkalmasabb erre a feladatra. Ezért választottam inkább az utóbbit, és építettem be a rendszerbe.

Az ARM-nak szerettem volna meghagyni a lehető legtöbb számítási kapacitást arra, hogy minél pontosabb spektrum képet tudjon számolni. A rendszerbe beépítésre került ESP32 az ARM-nak kizárólag már az egyből vezérléshez felhasználható paramétereket küldi el SPI-on keresztül, hogy minél kevesebb dolga legyen már vele.

Az ESP32 két vezeték nélküli kommunikációra is képes: Bluetooth és WiFi. A Bluetooth helyett azért választottam a WiFi-t, mivel a Bluetooth-on való kommunikációhoz olyan egyedi csomagok küldésére van szükség, mint az effektek felparaméterezése. Ilyen protokollt nem tud egy átlagos Bluetooth-os eszköz, ehhez kell írni egy alkalmazást, ami képes ezeknek a csomagoknak a küldésére/fogadására. Ha változatlanul mindig egy eszközről történik az irányítás, akkor ezzel nem is lehet gond. Viszont a WEB-es technológiával arra szeretnék rávilágítani, hogy mennyivel felhasználó barátabb alkalmazásokat lehet létrehozni. Bármilyen eszköz, mely képes egy weboldal betöltésére, és az ESP32 által sugárzott WiFi hálózatra képes valamilyen módon csatlakozni, akkor elérhető lesz onnan is a kezelőfelület. Nem kell alkalmazást telepíteni egy új felhasználónak, sőt, ha azt akarjuk, hogy több platformon is (pl.: Android, Windows) elérhető legyen az alkalmazásunk minden operációs rendszerre le kell fejleszteni az irányításra szolgáló alkalmazást. WEB-es technológia ezzel szemben teljesen platform független, elég egyetlen weboldalt lefejleszteni a kezelőfelülethez.

A 3. ábrán látható blokkvázlatban a bemeneten és a kimeneten lévő szinteltoló/jelerősítő áramkörök azért kaptak helyet, mert a vezérlő eszközöknek a megfelelő működés érdekében szükségük van kiegészítő áramkörökre. Erről bővebben a rendszer működésénél, a jelfeldolgozás, illetve a LED szalag vezérlésénél ejtek szót.

3.2. Hangtechnika

Az általam átépítésre került hangfalpár 3utas Videoton DC 2550 A típusú. Olyan régi gyártmányúak már ezek, hogy a bennük lévő összes hangszóró membránja körül a gumiperemek teljesen kiporladtak a helyükről. Mivel már úgy is fel kellett őket újítani, kicsit fel lettek dobva egy csipet RGB LED szalaggal. Tehát a 3-3 hangszórót, mind a mélysugárzót, középsugárzót, és magassugárzót is kicseréltem ehhez a hangfalhoz utángyártott új hangszórókra.

A hangfalban lévő eredeti hangváltót úgy véltem mai napig működő képes, így azt nem cseréltem ki. Más elektronika pedig nincs is a dobozokban, mivel ezek 8Ω ellenállású passzív hangfalak. Ami azt jelenti, hogy kell hozzájuk meghajtó erősítő végfok.

A hangfalak hagyományos zárt doboz kialakításúak. Erre azért kellett odafigyelni, mert nem hiába vannak „végtelen falba” bezárva a hangszórók. Ennek célja az akusztikus rövidzár elkerülése, mely leginkább a mély hangoknál jelentkezik. Ami annyit tesz, hogy a hangszóró membránja által mozgatott légtömeg vissza szökik a hangszóró mögé, így olyan mintha nem is mozgatná a levegőt, nem gerjeszt hanghullámokat. Tehát a LED szalagok beépítését úgy kell véghez vinni, hogy minden keletkező lyukat, rést, ahol a levegő ki-be szökhet azt meg kell szüntetni.

A gyártó alapból a hangszórók karimája, és a fa doboz közé gyurma szerű tömítő anyagot használt. Szintén ezt az anyagot használtam a hangfal fedlapjának, és az oldalsó lapok találkozásánál lévő rések betömítésére. A LED szalaghoz tartozó vezetékeknek kifúrt lyukat pedig egyszerű szilikon tömítővel nyomtam ki, ami egyben tehermentesíti a vezetékeket attól, hogy közvetlen a LED szalagra rögzítve lógjanak ki a dobozból, így erősebb behatás következtében akár a LED szalagot is kitépve a helyéről.

4. ábra: Első tesztelések alatt lévő állapot, még a teljes összeszerelés előtt

3.3. Fénytechnika

Az általam használt RGB LED szalag 12V-os WS2815 típusú. A hangfalpár elhelyezhetőségét sem akartam rontani azzal, hogy csak nagyon rövid vezetékeket használjak fel. Az is szempont volt, hogy ne kelljen több vezérlőt alkalmaznom, hanem egy központi vezérlő inkább akkor hosszabb vezetéken keresztül, de egy helyről tudja vezérelni.

Erre a célra pedig egy 12V feszültségen üzemelő LED szalag alkalmas, mivel a több mint 5V-nál nagyobb feszültségnek köszönhetően kevesebb lesz a feszültségesés. Így garantálható, hogy színhű tud maradni a LED szalag legutoljára megtáplált pixel-je is. Mivel az alacsonyabb feszültségen üzemelő LED szalagok hátránya, hogy ha túl hosszú vezetéken keresztül jut csak el a tápfeszültség az adott pixelhez, akkor nem biztos, hogy mind az R, G és B LED is maximális fényerővel tud világítani. Ez miatt pedig módosul a színárnyalat, az eddigi tapasztalataim alapján minden esetben így a LED szalag végéhez érve egyre jobban besárgult a fénye.

Ezt a nem kívánt hatást még azzal lehet csökkenteni, ha az egyik vége helyett több ponton van megtáplálva a LED szalag. A beépítést követően tehát így is tettem, ahány darab szakaszt ragasztottam fel, mind annyi helyen kaptak is tápfeszültséget.

Összesen 5m LED szalagot (300 pixel) használtam fel. Mivel két hangfal van, értelemszerűen fele-fele (150-150 pixel) arányban építettem be. Elsődlegesen a hangszórók köré tekerve, a karimájuk szélét lekövetve szerettem volna elhelyezni, de mivel maradt még belőle, így még beleépítettem a maradékot is.
A szakaszok pedig amiket elhelyeztem egy-egy hangfalon belül:

- magas sugárzó körül: 22 pixel,
- közép sugárzó körül: 22 pixel,
- mély sugárzó körül: 55 pixel,
- hangdoboz széleire (felső széle + egyiknél bal, másiknál jobb széle): 51pixel

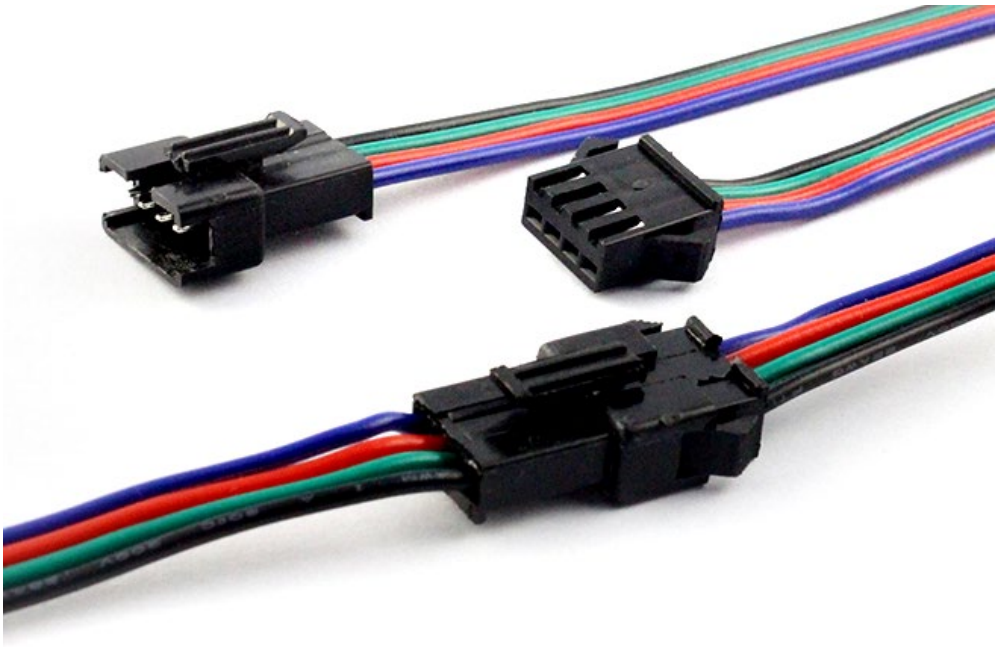
A hangfalba kerülő LED szalag látványosságát azzal fokoztam még, hogy úgynevezett „végtelen tükrös" kialakításban építettem be. Ami annyit tesz, hogy két tükör van egymással szemben. Pontosabban, hogy kívülről is lehessen látni, a felső rétegen egy félig átlátszó tükrös felületet kellett kialakítani. A két tükör között van a LED szalag, így fog tudni hatni a fényére ez az effektus. Erre került rá egy plexi lap. Mivel ez a fedlap, csak erre tudnak ráülni a hangszórók, ezért ennek kellően merevnek és vastagnak kell lennie ahhoz, hogy elbírja deformálódás nélkül a hangszórók hozzá való lefogatását. Ezzel a vastagsággal pedig, amivel a plexi lap rendelkezik + a LED szalag szélessége, annyival bentebb kellett süllyeszteni a hangfal belsejébe ezeket, hogy elférjenek, ne lógjanak ki a doboz síkjából.

Az eredeti fedlapja a hangfalnak tehát mai napig meg van, csak a hangfalba mélyebbre került. Továbbá ez a régi fedlap szolgál a végtelen tükrös effekt ellendarabjának, mivel erre lett ráragasztva egy teljesen tükrös fólia. Ennek a tükörnek, és a plexi belső felére ragasztott félig áteresztő tükörnek köszönhető az, hogy úgy látszik kívülről, mintha nem csak 1db LED szalag húzódna meg a dobozban, hanem végtelen sok (vagy legalább is egynél biztosan több).

A végtelen tükrös effekt annál jobb, minél tökéletesebb a tükör felülete, minél jobban visszaveri a fényeket. Én egyszeri otthoni módszerekkel közel sem tudtam felvinni a fóliákat úgy, hogy tökéletes síktükrös felületük legyen. Továbbá a fényerőtől is függ az effekt erőssége, minél halványabban világít egy LED, annál „kevesebb LED lesz a dobozba zárva” – avagy annál kevesebbszer verődik vissza a fénye, egyre gyengébb a tükrös hatás. Ezt a különbséget a nagy és kicsi fényerők között a lentebb látható 5. ábra szemlélteti.

5. ábra: Nagy és kicsi fényerő összehasonlítása

A címezhető LED szalagok eredetileg 4 pin-es csatlakozókkal szereltek (lásd 6. ábrán). Ezek a csatlakozók csak egy féleképpen illeszkednek egymásba, így nem tudnak összekeveredni az összeköttetések. Ezekből plusz darabokat szereztem be, és a hangfalból kilógó vezetékeket krimpeltem fel ugyanezzel a típusú csatlakozóval. Ebbe lehet csatlakoztatni a toldó kábelt, aminek másik végén is van a csatlakozó ellendarabja, mely pedig így a vezérlővel, és táppal köti össze a LED szalagot. Ezzel lett biztosítva a hordozhatósága a szettnek, hogy se a hangfalból, se a vezérlőből nem lógnak ki méteres kábelek, hanem azt külön összetekerve könnyedén szállítható. Mivel ez is fontos szempont volt, hogy a bevetési helyszínéhez könnyedén lehessen szállítani, és egyszerűen összerakni is.



6. ábra: 4 pin-es egy féleképpen egymásba illeszthető csatlakozók

3.4. Tápfeszültségellátás

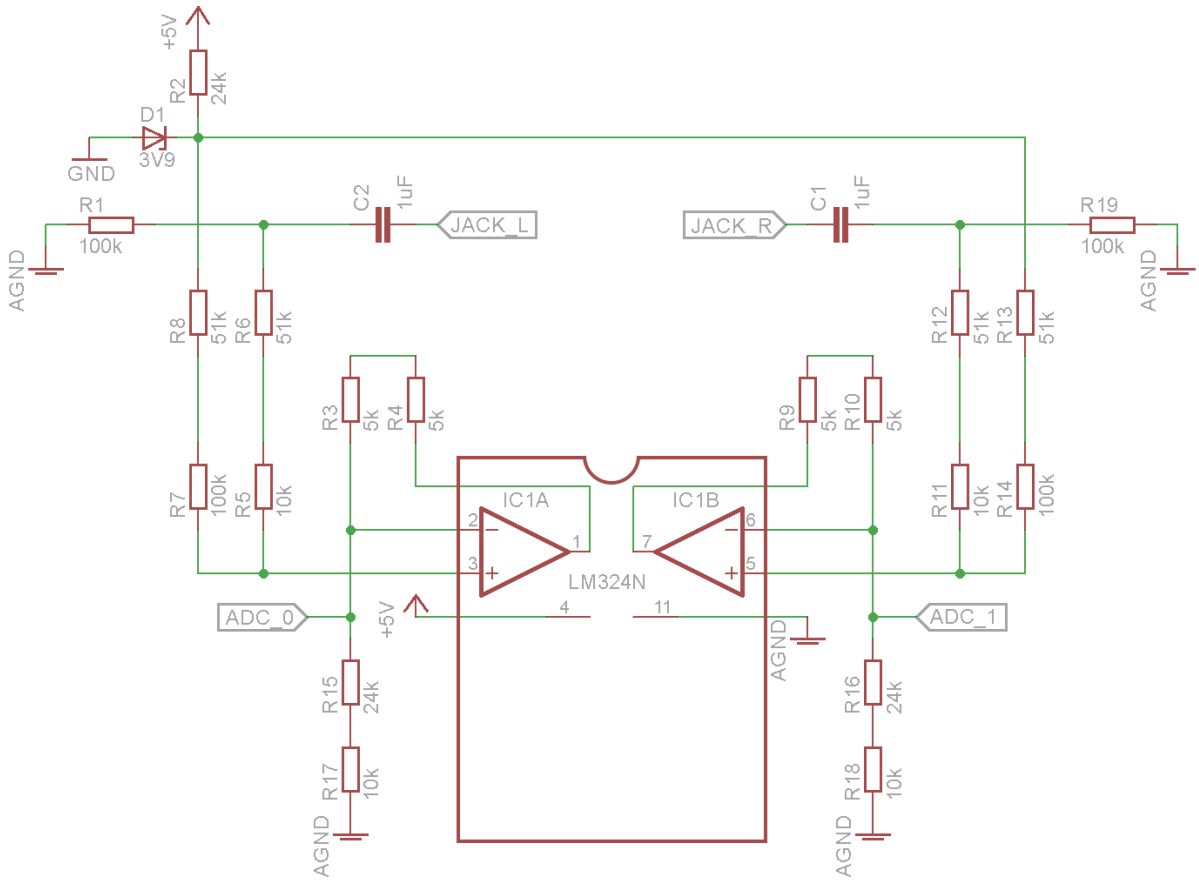
Mindkét vezérlő 5V tápfeszültséget igényel. Amik micro USB csatlakozón keresztül történnek, melyekbe akár a programozó számítógép, akár hálózati adapter USB kábele lehet csatlakoztatva. WS2815 típusú LED szalag az a leírása szerint 12V feszültséget, és 5m (300pixel) hosszon 90W teljesítményű tápellátást igényel. Erre megfelelő hatásfokkal üzemelő 90W teljesítmény leadására biztosan képes 12V-os kapcsolóüzemű tápegységet használok. A hangtechnikához tartozó erősítő pedig 150W maximális teljesítményt képes leadni a hangfalak meghajtásához.

4. Rendszer működése

4.1. Audió jel mérése

STM32-ben egy SAR (Successive-approximation) típusú ADC (Analog Digital Converter) van. Mely egy jelösszehasonlításos eljárás. A bemeneti jeltartománya 0V és 3,3V között van. Ha a tartomány felétől nagyobb a jel, akkor az első bit 1, igaz értéket vesz fel. Ahány bites a SAR típusú ADC, annyiszor osztja mindig újra és újra felére az aktuális tartományt, melynek az értékét a következő bitbe állítja be. Rendelkezésemre álló ADC pedig 12bites, így 0 - 4096 felbontásnak megfelelő pontosságban tud mérni.

A jack csatlakozón érkező audió jel -1,7V és 1,7V közé esik, amit el kell tolni az ADC mérési tartományába. Erre a célra egy LM324 műveleti erősítőt használó neminvertáló kapcsolást építettem, hogy ne rontsa el az audió jelet a mérő áramkör, mely a 7. ábrán látható. [1]



7. ábra: Audió jel szinteltoló áramkör

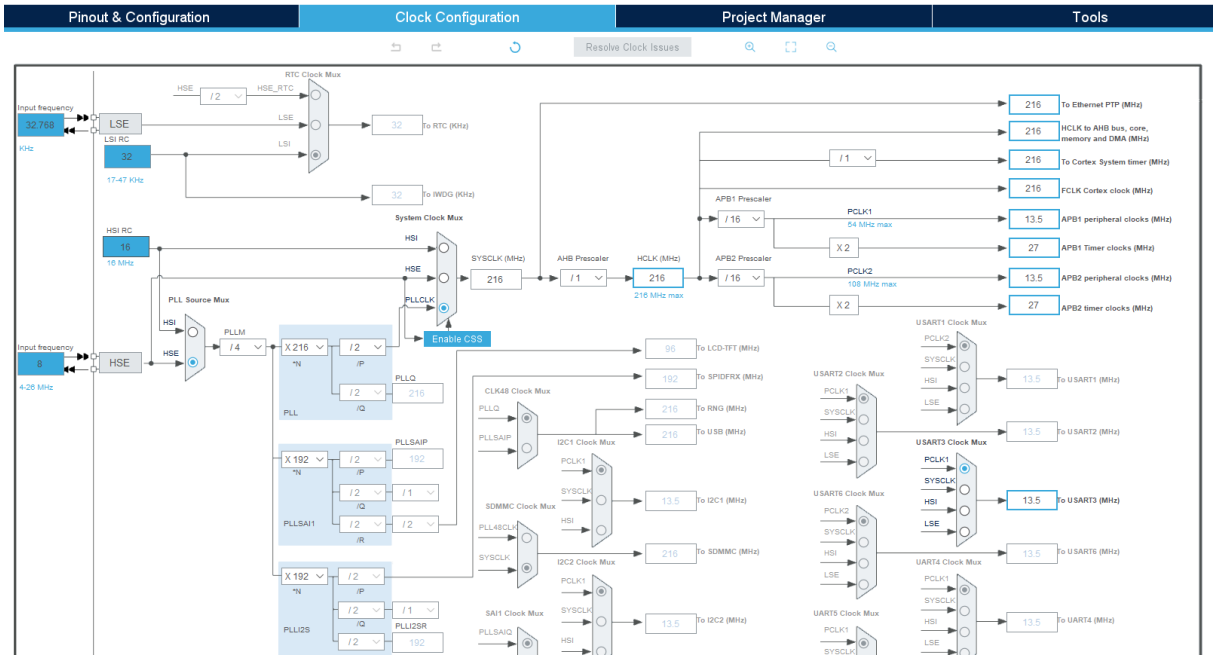
Így gyengítetlenül tovább tud terjedni a jel a hanglejátszó eszközhöz az alkalmazott műveleti erősítő nagy bemeneti ellenálla miatt. Sztereó jelfeldolgozás miatt jobb és bal csatornára is meg kellett építeni ugyanazt az áramkört.

ADC időzítéseit tekintve pedig azt kellett figyelembe venni, hogy a mérés megfeleljen a Nyquist mintavételezési tételének. Mely szerint a mintavételezési frekvencia legalább kétszerese kell legyen a jelben előforduló legnagyobb frekvenciának. Ennek a feltételnek a megszegése esetén a digitálisan mintavételezett jelben átlapolódó frekvenciájú komponensek léphetnek fel.

Mivel az ember nagyjából 20kHz frekvenciájú hangot hall még (ami idős korára egyre csak romlik), így audió jelben is 20kHz lehet az előforduló legnagyobb frekvencia. Az én mérési tapasztalataim szerint egyébként a zenékben 15kHz fölött nagyon ritka, hogy előfordul bármilyen jel. Viszont az ADC lehetségesen bekonfigurálható órajelét úgy beállítani, hogy a standard audió jelből biztosan mindent tudjon mérni a rendszer, így legalább a 20kHz duplája legyen a mintavételezési frekvencia, és ne legyen a rendszert fölöslegesen terhelő túlságosan nagyobb órajel, úgy tudtam csak beállítani, hogy 50kHz közeli mintavételezési frekvenciát lőttem be.

Az ARM órajelét a lehető legnagyobb értékre (max 216MHz érhető el) választottam meg, hogy minél gyorsabban (minél valósabb idejű legyen) tudja elvégezni a spektrum analízis algoritmust. Ehhez a legnagyobb leosztást, ami csak elérhető volt (16) választottam meg a belső perifériáinak biztosított órajelre. Ebből kettő is van, a PCLK1 és PCLK2. Ezt az órajelet az ADC 2, 4, 6, 8-cal tudja leosztani. Továbbá a mintavételezési időt is többféle érték közül lehet bekonfigurálni.

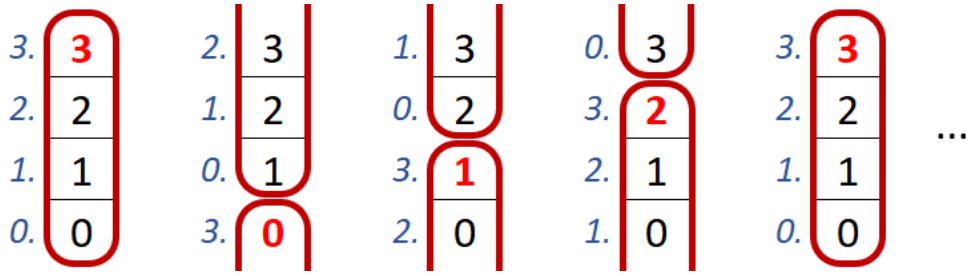
Az 50kHz közeli mintavételi frekvenciát eredményező beállítást úgy értem el, hogy PCLK2-őt 2-es leosztásra állítottam be az ADC órajelére, majd a mintavételezési időnél az 56 ADC ciklusnyi időt választottam, így 49.632,7 Hz frekvenciával mintavételez 1-1 csatornát. Mivel 2 csatornát használok, így kétszer gyorsabban történik a tényleges mintavételezés, csak váltakozva a két csatorna közül egyszer egyiket (jobb audió hangsáv), utána másikat (bal audió hangsáv) olvassa be.



8. ábra: ARM órajeleinek konfigurációja

Külön-külön jobb és bal csatornák utoljára mért értékeiből 2048 darabot őriz meg a memóriájában az ARM. Ha egy közös tömböt alkalmaznék arra, amiben minden egyes mérés végeztével folyamatosan átíródnak az értékek és ebből közvetlen számolna az analízist végző algoritmus, akkor az nem vezetne helyes eredményre, mivel két mérés között eltelt idő alatt az algoritmus nem képes végezni a számítással. Ezért szükséges egy átmeneti buffer tömböt létrehozni, amibe folyamatosan kerülnek az új értékek. Ezután, amikor elindul az analízist végző algoritmus, egy saját belső tömbbe másolja át az értékeket –mintha egy pillanatképet készítene–, ahol számol vele tovább. Ezzel a megoldással viszont az a probléma, hogy ezeket a 2048 méretű tömböket mire feltölti az ARM értékekkel, eltelik a mérési frekvencia periódusideje megszorozva a 2048-cal idő. Ez nagyjából 40ms, tehát az eredmények 1 másodperc alatt csak 25-ször frissülnek „25Hz képfrissítéssel”. Ennyi idő alatt viszont bőven végez a számítással az analízist végző algoritmus is. Tehát ezekhez a kissé darabosan frissülő eredményekhez képest el lehet érni gyorsabban frissülő, folyamatosabbnak tekinthető eredményeket is.

Ennek a problémának a kiküszöbölésére olyan megoldást alkalmaztam, hogy mindkét tömböt további 4 résztömbre osztottam. Ez a 4 darab negyed tömb egymás után forog úgy, hogy épp a sorban melyik negyed tömb következik, amibe kerüljenek a frissen utoljára mért értékek. Ami miatt, még memóriából memóriába másolás műveletből is kevesebb szükséges. Továbbá csak időrendi sorrendben kell feldolgozni a negyed tömböket úgy, hogy az éppen utoljára frissült legyen értelemszerűen az utolsó is a sorban, és így csak ezt az egy tömböt kell átmásolni.



9. ábra: Forgó tömbök (fekete a tömb sorszáma, piros az utoljára frissült tömb sorszáma, dőlt kék a helyreállított, feldolgozás szerinti sorrend)

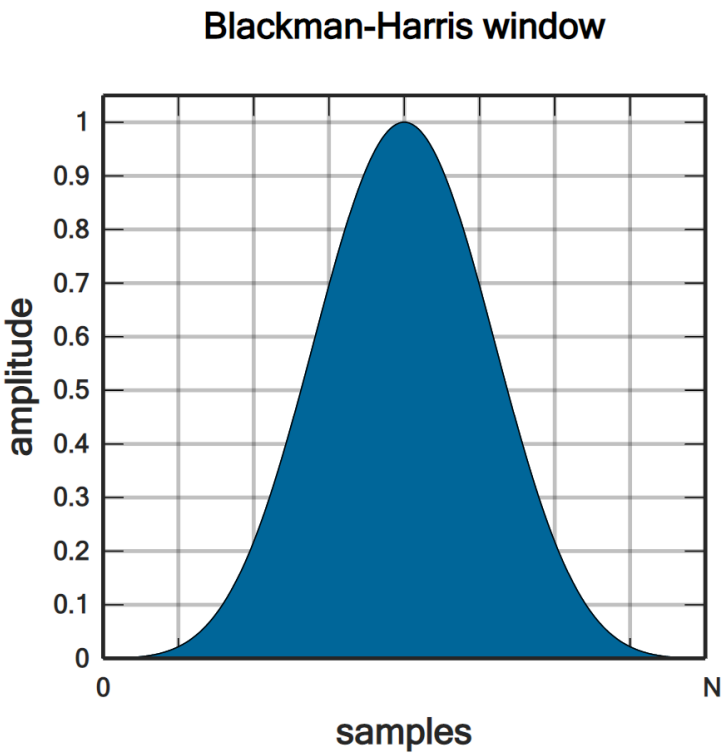
4.2. Spektrum analízis számítása

A spektrum analízis algoritmusa, amit használok az a Cooley-Tukey fajta FFT (Fast Fourier Transform). Ennek az első megkötése, hogy a minták száma csak 2 hatványaival megegyező számúak lehetnek. Mert ebben az esetben lecsökkenthető a műveletek száma. Az eredménye pedig a minták darabszámának a felével egyenlő hosszúságú adatsor lesz. Melynek első eleme a jelben található DC komponens nagyságát adja vissza, a további értékek pedig az amplitúdó nagyságai a jelben lévő frekvenciáknak. A pontosabb eredmény érdekében a mintavételi ablakra Blackman-Harris ablakozást alkalmazok. Ez elősegíti a tényleges frekvenciák kiemelését, és kissé elnyomja a jelben lévő zaj frekvencia komponenseit.

$$w[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N}\right) + a_2 \cos\left(\frac{4\pi n}{N}\right) - a_3 \cos\left(\frac{6\pi n}{N}\right)$$

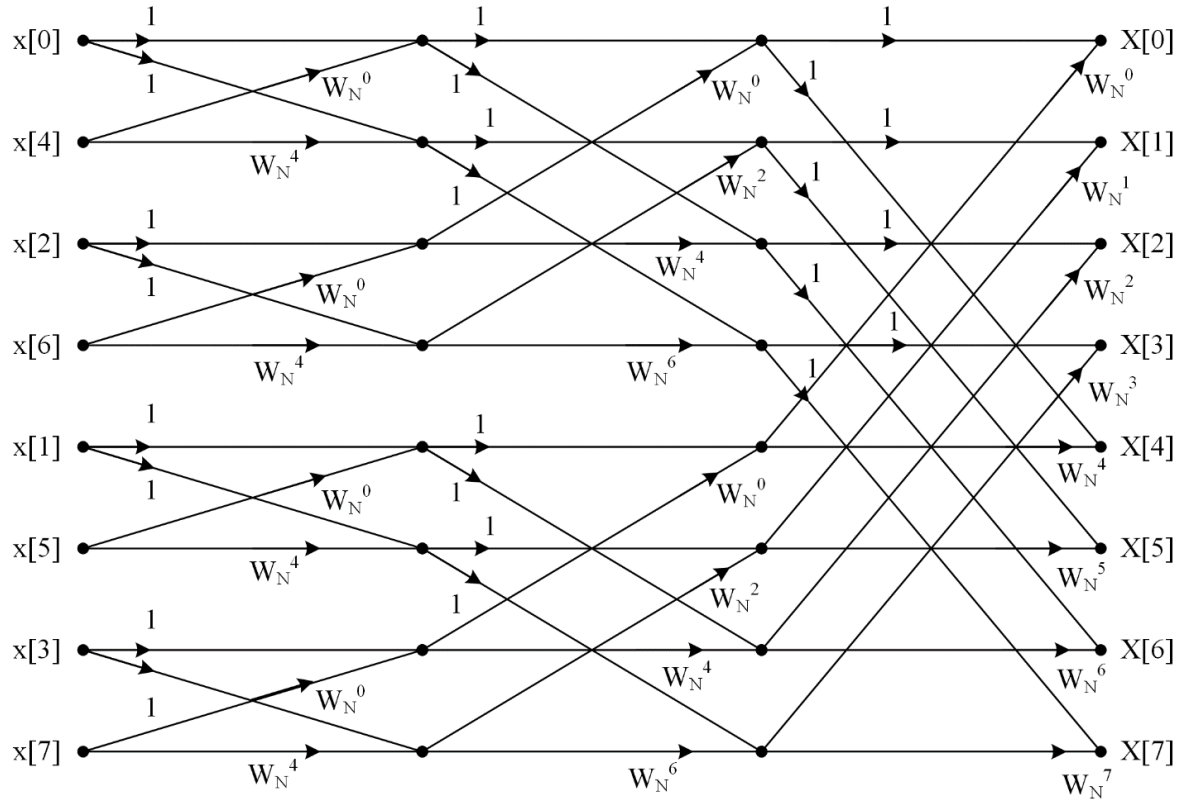
$$a_0 = 0.35875; \quad a_1 = 0.48829; \quad a_2 = 0.14128; \quad a_3 = 0.01168.$$

10. ábra: Blackman-Harris ablakozásnál a súlytényezők kiszámításához alkalmazott összefüggés képlete



11. ábra: Ablakozásnál a súlytényezők értékeinek szemléltetése grafikonon

Az algoritmus lépéseiben az ablakozás után a következő már maga az FFT. A mért értékeken egy bizonyos módszerrel kell műveletet elvégezni. Nem elég egy egyszerű ciklus, ami a jelet reprezentáló tömb elemein egymás után sorba végig lépked. Úgynevezett pillangó művelet (butterfly, vagy radix-2) formájában történik meg. A művelet során 2 bemeneti értékből 2 kimeneti érték keletkezik, és a bemenő értékek az algoritmus futása közben többé nem kerülnek felhasználásra, így lecserélhetőek az új kimenet értékeire. Ez miatt szokás úgy hívni, hogy ez egy helybenszámoló algoritmus. A lentebb látható 12. ábrán pedig egy 8 elemű minta esetében megfigyelhető, hogyan történnek ezek a műveletek. A művelet közben történik egy twiddle-faktor értékkel történő szorzás is, melynek a jele a W_N. [3] [4]



12. ábra: FFT algoritmusban történő pillangó műveletek vázlata

Az algoritmus utolsó lépése az úgynevezett bit reverzálás, ami azért szükséges, mert az eddig kapott eredmények nem jó sorrendben helyezkednek el a tömbben. A fentebb látható ábrán ez az algoritmus elején történik meg, ahol az ábra bal oldalán a bemeneti értékeknél jól látszódik is, hogy nem egymást követő sorrendben vannak a hozzá tartozó indexek számai. A sorrend helyre állítása pedig úgy történik, hogy megtükrözi az elemek indexeként szolgáló sorszámban lévő biteket (pl.: 001-ből 100 lesz). Az ilyen műveleteket elegendő csak egyszer elvégezni, és a kapott eredményeket csak egy tömbbe elmenteni. Ilyen optimalizációkra is törekedtem a fejlesztés során, hogy minél kevesebb számítási kapacitásba kerüljön egy FFT ciklusideje. Még egy ilyen optimalizálás, amit megtettem, hogy az ablakozás művelethez sem kellett más, csupán súlytényezőket előre kiszámolni, és letárolni egy tömbbe, majd csak egyszerűen a kész értékeket kivenni onnan. Továbbá a pillangó műveletnél történő twiddle-faktorral való beszorzás esetén is ez az érték egyedül a minták számától függ, amely mivel nem változik futás közben meg, szintén előre kiszámolhatók egy tömbbe, és onnan hivatkozhatók csak index alapján.

4.3. Eszközök kommunikációja

Két eszköz közötti kommunikációra SPI kapcsolatot hoztam létre. Az SPI kommunikáció úgy történik, hogy van egy master, és 1, vagy több slave akik között közös buszon történik az adatcsere. A közös adatvezetékek pedig MISO (Master In Slave Out), MOSI (Master Out Slave In). A master választja ki, hogy melyik slave-el szeretne adatot cserélni. Ezt az SS (Slave Select) vezetéken teszi, és innen tudják a slave-ek, hogy épp mikor küldhetnek/fogadhatnak. Az adat csere pedig shiftregiszterekből történik, ezért is van két vezeték, mert full-duplex (kétirányú) a kommunikáció egy időben egyszerre. Az első bitet átshifteli a master a slave utolsó bitjére. Eközben a slave szintén a nála lévő első bitet átshifteli a master utolsó bitjére. Majd ez újra és újra, míg átforog az adat egyik shiftregiszterből a másikba.

Én esetemben a masternek az ESP32-őt választottam ki, és egyetlen egy slave van, ami az ARM lett. Ezt azért gondoltam logikus választásnak, mert alapvetően az ARM csak fogadja a beérkező beállításokat, és nem ő fogja így kezdeményezni úgyse a kommunikációt. ESP32 így szerintem alkalmasabb arra, hogy master legyen, mivel minden egyes WiFi-n beérkező új beállításra rögtön tud szólni a slave select vonalon az ARM-nak, hogy akkor már indítja is a kommunikációt és küldi az adatokat neki mi változott. Vagy ha esetleg az ESP32 szeretne kérni valami adatot az ARM-tól, akkor is csak egyszerűen tud szólni SPI-on egy csomag azonosítóval, ami után csak egyszerűen hallgat az ESP32, hogy mit válaszol az ARM.

Én úgy alakítottam ki a beszélgetést az SPI-on, hogy csomag azonosítókkal kezdődik minden adatcsere, amiket elkezdenek küldeni egymásnak az eszközök. Az azonosító 1db byte, tehát 256 féle csomagot tudok létrehozni így, ami egy ilyen rendszerben bőven elegendőnek bizonyul, ahol csak effekteket akarok felparaméterezni. Amint fogadta az egyik eszköz a másiktól jövő azonosítót, akkor onnantól kezdve az általam előre definiált adott csomag byteokban mért hosszáig a master fenntartja a kommunikációt a slave-el. Ahogy küldésre/fogadásra került a csomag, utána a slave újra egy csomag azonosítót fog várni. A master pedig semmilyen adatot addig nem fog elküldeni, amíg újra nem küldi el az egyik csomag azonosítóját. Azt a két esetet alakítottam ki, hogy az ESP32 küld egy azonosítót minden esetben először az ARM-nak. Ebből az ARM az azonosító alapján tudja milyen csomagról van szó, és a két eset az lehet, hogy ha ez egy olyan csomag azonosítója, ahol az ARM-nak kell küldenie adatot, akkor addig az ESP32 csak szimplán hallgatja az érkező adatokat, amíg le nem telik a csomag hossza. Másik eset pedig amikor az azonosító egy olyan csomagé, ahol az ARM-nak küld adatokat az ESP32, addig pedig értelemszerűen az ARM fogja beolvasni az érkező

adatokat, amíg az ESP32 szorgosan küld pont akkora hosszúságú csomagot, amennyi a csomag előre ledefiniált hossza is ténylegesen.

Felfedezhető, hogy ugyan nem használom ki a full-duplex adottságát, hogy egy időben egyszerre mindkét irányba küldhetnék adatokat. Viszont azért esett a választásom mégis az SPI-ra, mert jó tulajdonsága, hogy egyik leggyorsabb módja a mikrokontrollerek közötti kommunikációnak. Továbbá, hogy ha esetleg későbbiekben több fényre vezérelhető akár csak lámpát, vagy még több RGB LED szalagot szeretnék a rendszerbe építeni, akkor az ARM már nem bírná el ennyi dolog vezérlését úgy, hogy real-time maradjon az audió vizualizáció. Így lehetőségem marad arra, hogy több slave-t helyezhessek el, akiknek küldheti az ESP32, hogy most épp milyen konfiguráció alapján vezéreljen fényeket az adott slave.

4.4. Mátrix (címezhető) LED szalag vezérlése

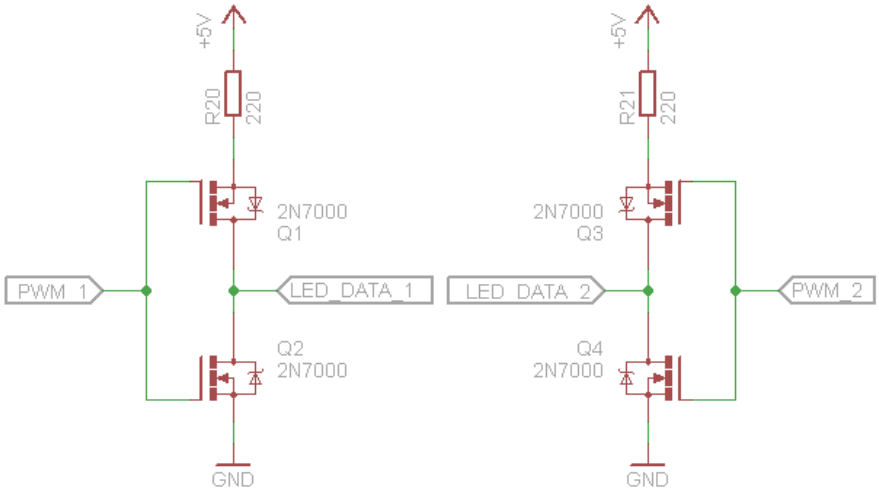
A beolvasott jelek alapján lehetséges a hangfalba épített RGB LED szalag audió vizualizációja fényekkel, színekkel. Az ARM a kimenetein 3,3V feszültséget ad ki, az adatbemenete egy címezhető LED szalagnak pedig 5V-os. Ezzel a feszültség kimenettel pedig éppen határon van, amit már HIGH-nak tud érzékelni egy pixel. De a vezeték hosszából eredő ellenálláson, meg a zavarok miatt elég sokszor volt, hogy nem igazán vett be minden HIGH értéket HIGH-nak a WS2815. Ezért mindenképp szükséges volt egy 5V-ra szinteltoló áramkörre, ami elég gyors ahhoz, hogy a 800kHz közeli PWM jelet megfelelően átalakítsa a nagyobb feszültségre.

Kísérleteztem több módszer kipróbálásával, amik közül nyilván a legelső a jelerősítő áramkör nélküli meghajtás, amit fentebb említettem, hogy nem volt megfelelő. Következő próbára már egy 2N7000 N-csatornás MOSFET felhasználásával épített két irányú nem invertáló szintillesztő áramkört alkalmaztam. A felhúzó ellenállás miatt a HIGH jel már stabil lett így. Viszont a LED szalag bemeneti kapacitása miatt ebben az esetben a LOW szint lett problémás, mivel az áramkör LOW szint esetén nem húzta le a kimenetét.

Ezután egy LM393 műveleti erősítős komparátorral megépített szintillesztő open drain kimenetű áramkört próbáltam ki, ami a vezérlési sebességet nem tudta lekövetni, így se a LOW, se a HIGH szintet nem tudta olyan gyorsan kiadni, mint ahogy az megváltozott.

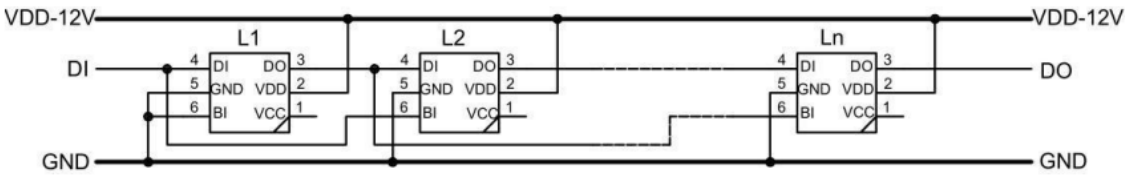
A sebesség miatt kikövetkeztethető, hogy a lehető legkevesebb alkatrész használatára kell törekedni, így végül, ami bevált kapcsolás, az egy egyszerű CMOS invertáló kapu lett. [2] Ez miatt még egy apró módosítást kellett tenni az ARM konfigurációjában. Ez pedig a kimenő PWM jel invertálása szoftveresen, hogy az invertáló kapcsolás miatt végül is a vissza invertált,

tehát az eredetileg akart kimenő jel jusson el a LED szalag bemenetére. Ehhez a kapcsoláshoz is 2N7000 MOSFET tranzisztort alkalmaztam, amiből a két külön elszeparált hangfalpár miatt szintén 2 darabot kellett felhasználni, mivel a két hangfalban lévő LED szalagok vezérlése külön-külön történik. Tehát ez a kapcsolás (lásd 13. ábrán) már stabil HIGH, és a LED szalag kapacitásainak kisütésére alkalmas LOW jelet biztosítva megfelelően működik.



13. ábra: Adat busz jelerősítő áramkör

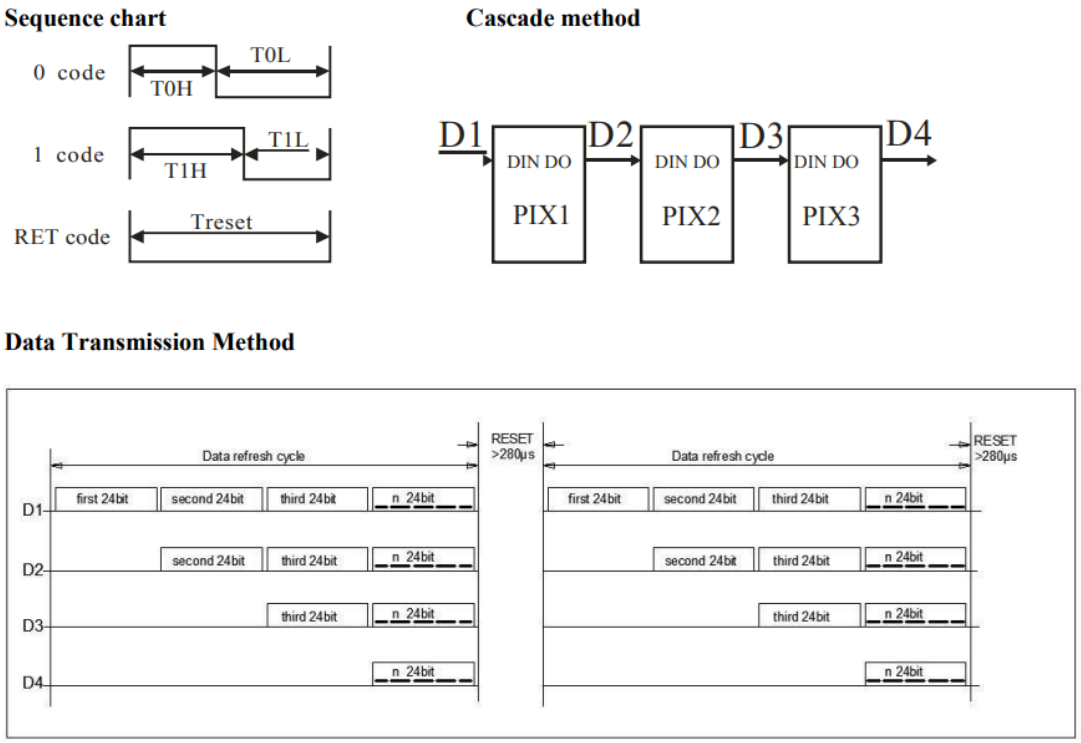
Pixelenként vezérelhető LED szalagoknak nincs szükségük másra, mint testre, tápfeszültségre, és adat vezetékre. Bekötésük így 3 vezetékből megoldható, az esetemben ez a 3 vezeték hossza, ami relatív nagy. A szalagon lévő pixelekben nem csak maga az RGB LED, hanem egy chip is található. Ez értelmezi a bejövő adatot, és ennek megfelelően állítja be a színt, fényerősséget. WS2815 egy 24bites színmélységgel rendelkezik, tehát piros, kék, és zöld mind 0-255 között vehet fel fényerősség értéket.



14. ábra: WS2815 RGB LED szalag felépítése

Az adatfolyam (lásd 15. ábra) pedig nem is áll másból, mint 24bitnyi adatcsoportokból egymás után küldve. Az első 24 bit az első pixelhez fog tartozni, majd a második 24bitet a 2. pixel fogja értelmezni, és így tovább ahány pixelig szeretnénk kiküldeni a színeket, annyiszor 24bitet kell egymás után küldeni. Ha már nem a soron következő pixelre szeretnénk küldeni színeket,

hanem újra az elsőtől kezdve állítanánk be a színeket, azt egy RESET kód küldésével érhetjük el, tehát az adatfolyam végét ezzel a kóddal tudjuk lezárni. A színkódot G (Green) R (Red) B (Blue) sorrendben, legnagyobb helyiértékű bitet elsőként, majd végig haladva a legkisebb helyiértékű bitig tartó sorrendben kell továbbítani. [11]



15. ábra: Címezhető RGB LED szalagok adatfolyama

Data Transfer Time

T0H	0-code, High-level time	220ns~380ns
T1H	1-code, High-level time	580ns~1.6μs
T0L	0-code, Low-level time	580ns~1.6μs
T1L	1-code, Low-level time	220ns~420ns
RES	Frame unit, Low-level time	> 280μs

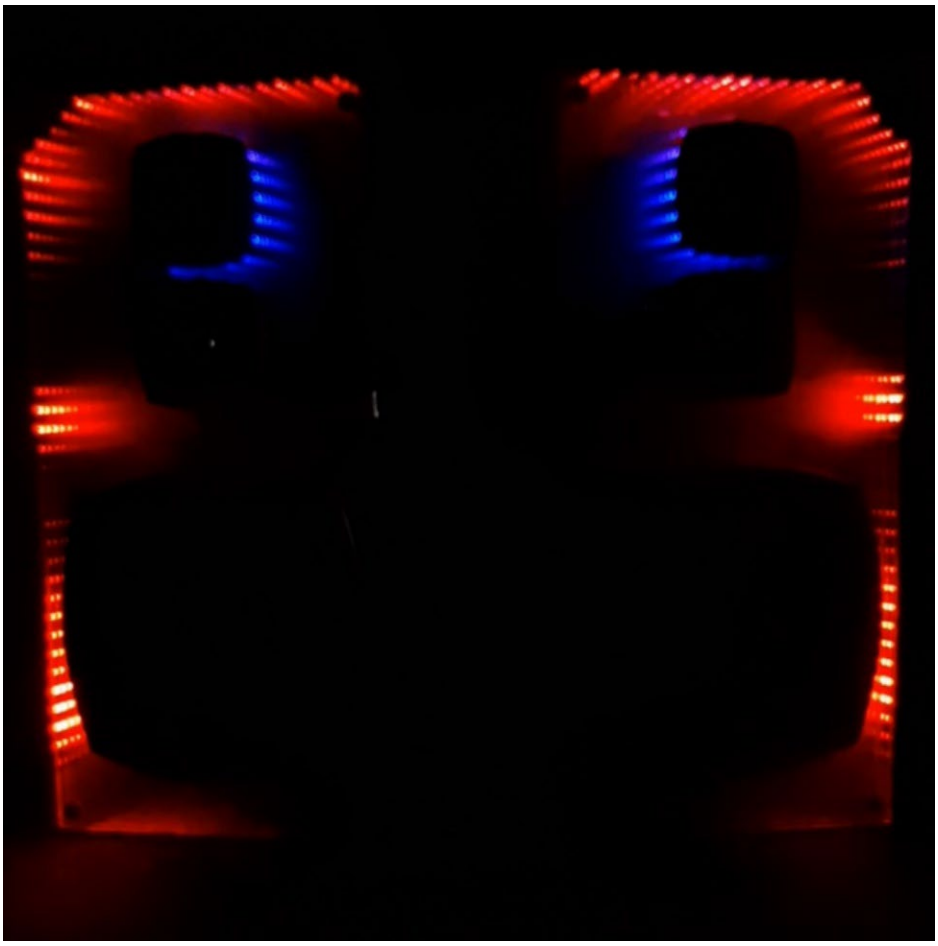
16. ábra: 0, 1, és RESET kódokban szereplő idő intervallumok

1 bit kódolása olyan formában történik meg, hogy ha 0 értékű, akkor T0H ideig magas, majd T0L ideig alacsony értéket kell tartani az adatvezetéken. Ha pedig 1 értékű, akkor ehhez hasonlóan T1H ideig magas, majd T1L ideig alacsony értéken kell tartani. RESET pedig a TRES ideig tartó alacsony értéken való jeltovábbítást jelenti. Ezeknek az időtartamoknak a dokumentáció alapján a 16. ábrán látható tartományban kell lenniük. Ez az adatfolyam lényegében így nem mást jelent, mint egy változó kitöltésű PWM jelet. Az ARM-ban pedig elérhetőek PWM kimenetek, melyeket Timer-ekhez lehet konfigurálni, mely beállított kitöltési tényezőjű PWM jelet generál.

A pixelek az utoljára kiküldött vezérlésnek megfelelően mindaddig megőrzik a beállításukat, amíg tápfeszültség alatt vannak, vagy pedig legalább egy RESET időtartamnyival később újabb beállítást nem kap. Mivel a RESET időtartamnak csak minimum ideje van, nincs maximuma, így elegendő csak akkor újra küldeni a bitsorozatot a pixelekre, ha meg kell változtatni legalább 1db LED aktuálisan megjelenített színkódját.

Az effektek különböző módokon vezérlik a pixeleket. Az effekteket úgy lehet beállítani, hogy az ESP32-re csatlakozott WiFi-s eszközön megjelenő vezérlőpultként szolgáló weblapon beállított konfigurációt elküldi az ESP32 az ARM-nak SPI-on keresztül. Az ARM pedig ez alapján frissíti az éppen módosítani kívánt effekt paramétereit. A vezérlőpult weblapról –más néven kezelőfelületről– a következő fejezetben ejtek szót.

Például az az effekt, ami igazából az audió vizualizációért felelős, az a spektrumkép effekt. Ami úgy működik, hogy egy-egy pixel egy frekvenciavonalnak felel meg. A pixelek pedig logaritmikus lépték szerint tartoznak egy-egy frekvenciavonalhoz. A vezérlés az adott frekvenciavonal nagyságával arányosan irányítja a hozzá tartozó pixel fényerejét. A LED szalagon tehát minél fényesebb lesz egy pixel, annál nagyobb amplitúdójú frekvencia komponenst tartalmazott a mintavételezett jel. A lentebb található 17. ábrán a hangfalpár felső-oldalsó szélein lévő szakaszokra piros színre állított spektrumkép effekt működése közben készített pillanatkép látható. Külön a hangfalak pozíciójának megfelelően a bal oldali az audió jel bal csatornájára, a jobb oldali pedig a jobb csatornájára van konfigurálva. Abban a sorrendben vannak megjelenítve a spektrumképek, hogy a hangfalak aljában a mély frekvenciák találhatók, amik a szakasz végére a hangfal tetejében a Nyquist frekvenciához érnek el.



17. ábra: LED szalagon megjelenített spektrumkép

4.5. DMA technológia használata

Egy beágyazott rendszerben gyakran előfordul, hogy a processzort olyan megszakítások (interrupt) terhelik, melyben csak egy érték regiszterből regiszterbe másolódik át. Ilyenkor ezért az egy műveletért rengeteg másik művelet hajtódik végre, mely a meghívásért, és az éppen futó ciklusba visszatérésért felelős. Ami azt eredményezi, hogy csökken a számítási teljesítménye a processzornak a túl sűrű össze-vissza ugrálás miatt a kódban. Ha valami tudná ezeket a regiszterből regiszterbe másolásokat helyettesíteni a processzor kikerülésével, az rengeteg plusz számítási kapacitáshoz tudná juttatni a processzort.

Erre a problémára tökéletes megoldást nyújt a DMA (Direct Memory Access). Ami az általam használt ARM mikrokontrollerben elérhető. Akár memóriából memóriába, akár egyik periféria regiszteréből memóriába, vagy memóriából perifériából másolásról van szó, ezekre a feladatokra mind bekonfigurálható a DMA. Beállítható a regiszter mérete, hogy hány bájtos memória területet másoljon, külön a forrás, és külön a célterület méretét. A beállított méretenként tud a következő memóriaterületre lépni ciklikus futás esetén. Ez is külön

konfigurálható, hogy a forrás vagy cél memóriacímét, vagy akár mindkettőt változtassa. Futásidőben pedig a DMA elindításakor lehet beállítani, hogy hányszor lépjen a következő memóriacímre, ha ezt a számot elérte, akkor abban az esetben, ha nem állítjuk le, újraindul és az első memóriacímre ugrik, majd újból végig lépked az adott memóriaterületen.

A rendszerben két helyen lett használva ez a funkció. Egyik, amikor az ADC méri a bemenő jelet, ahol az ARM-nak a mintavételezési frekvencia gyakoriságával természetesen nem kell egy megszakítást meghívnia és betenni az új értéket egy bufferbe. DMA úgy lett felkonfigurálva, hogy az ADC regiszterében lévő értéket másolja át a memóriába, amíg meg nem telik a mérési eredményeket tartalmazó buffer, majd csak ezután hívódik meg megszakítás. Ennek hála több idő marad a spektrum analízis algoritmus lefuttatására. Mert amíg lefut az analízis, addig a DMA szorgosan begyűjti az új adatokat a következő ciklusra.

Másik rész a rendszerben, ahol DMA segítségével optimalizálva lett a LED szalag címzésekor generált adatfolyamot tartalmazó PWM jel. Mivel a DMA vezérli a PWM jel kitöltési tényezőjének változtatását. Mely úgy jön létre, hogy előre a kitöltési tényezőket egymás után egy bufferbe ki kell számolnia, és betennie az algoritmusnak, majd kiadni a DMA PWM vezérlésnek, hogy 1-1 letelt periódus után automatikusan állítsa át a kitöltési tényezőt a bufferben lévő következő értéknek megfelelően. Ez a folyamat szintén addig tart, amíg a buffer végére nem ért a DMA, így kiadva ezt az adatsorozatot. Ilyenkor ez a DMA leállításra is kerül, ami csak akkor lesz újra elindítva, ha frissíteni kell a pixeleken megjelenő színeket, avagy meg kell változtatni legalább 1 darab pixel színárnyalatát.

5. Rendszerhez tartozó kezelőfelület

Felhasználói kezelőfelülethez minél egyszerűbb hozzáférhetőséget tűztem ki célomnak. Bármilyen eszköz, amiben van böngésző, és WiFi-t is tud, így bármilyen alkalmazás telepítése nélkül is meg tudja csak szimplán jeleníteni a kezelőfelületet. Ami nem más, mint egy weblap, ahol testre lehet szabni az effektek beállításait.

Az ESP32-re a paraméterek az effektek vezérléséhez egy WiFi-n csatlakozó eszközről tudnak jönni. Az ESP32 létrehoz egy WiFi hotspot-ot, amit „RGB hangfal" névre állítottam be jelszó védettség nélkül. Amint egy eszközre rácsatlakozunk erre a WiFi hálózatra, az eszköz azt kéri jelentkezzünk be a hálózatra. Amint ezt a Captive-Portal ablakot megnyitjuk, az ESP32-őn a WiFi hálózatán futó webszervernek küld az eszköz HTTP GET kérést, melyre a webszerver visszaküldi válaszul magát a weblapot. Ez a weblap szolgál az irányítópultként, melyen testre lehet szabni az effekteket.

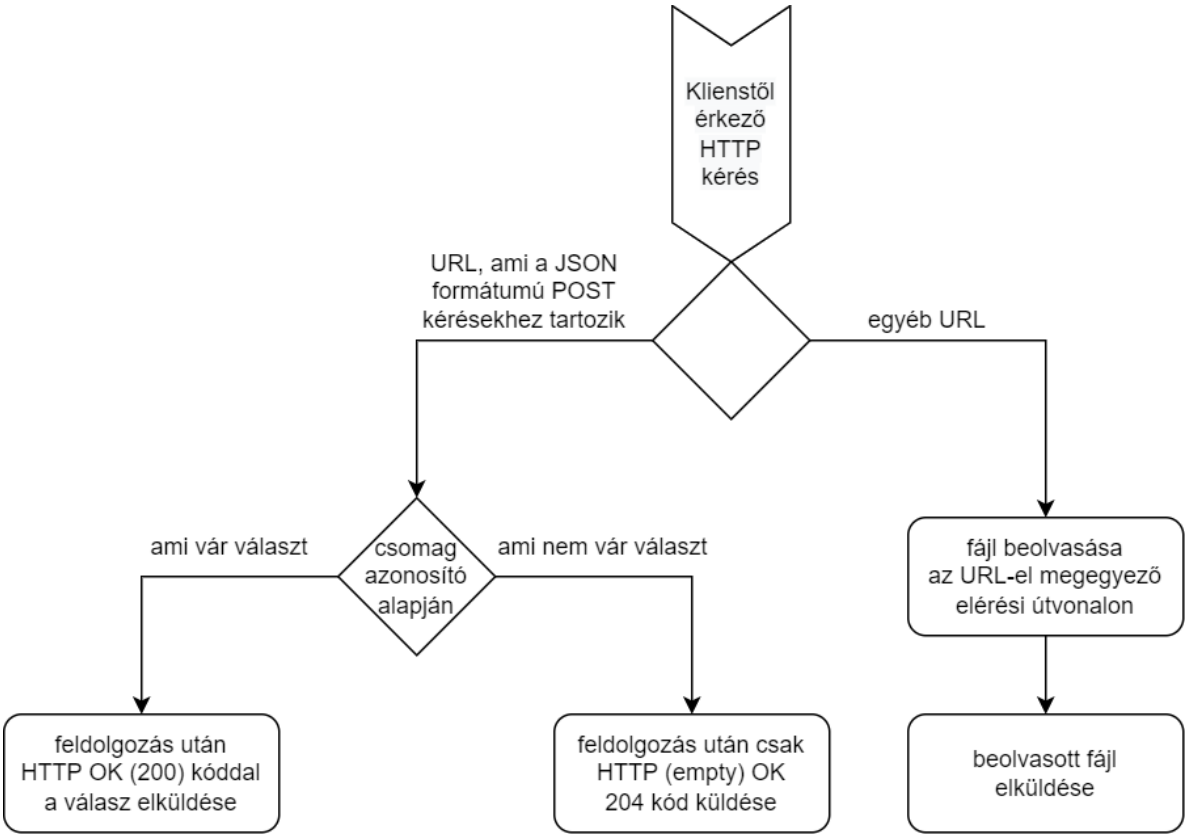
A weblap megírásakor HTML 5-ös verzióban teszteltem, tehát minden, ami tudja a HTML 5-ös verzióját kezelni, azon tud futni a weboldal (Pl.: Internet Explorer-en nem fog működni).

5.1. Webszerver működése (backend)

Egy weboldal betöltéséhez szükség van egy webszerver futtatására. Ezt az ESP32-őn teljes mértékben ki lehet vitelezni. A felhasználó eszköze tehát az ESP32 által sugárzott WiFi hálózaton keresztül egyszerű HTTP protokollon keresztül küld kéréseket, melyeket az ESP32 dolgoz fel.

Webszerver futtatásához egy ESP32-höz elérhető könyvtár szolgál a működtetésre. Egy weboldal betöltése úgy működik, hogy a beérkező HTTP GET kérésre egy HTML stringet vár vissza a kliens. Ezt lehetne a kódban felvenni stringként, és csak ezt átadni a webszerver megfelelő moduljának, ami a választ küldi el. Én viszont azt a megoldást választottam, hogy az ESP32 flash memóriájába lehet tárolni adatokat, és magát a weblap fájljait a fejlesztői környezettel töltöm fel rá, és csak onnan olvassa ki.

Ehhez egy újabb könyvtár tartozik, mely a fájlokat kitudja olvasni a memóriájából, és magát a fájlt továbbítani a webszerveren keresztül a kliensnek. Itt olyan megkötéseket kell betartani, hogy a kliens által kért URL-nek megfelelő mappaszerkezetben a fájl elérési útvonala is ott legyen, ugyan azon az URL-en. Így nagyon egyszerűen le lehet kezelni a fájl kiolvasást, majd a fájlt válaszként elküldését, nem kell stringekkel bajlódni, hogy melyik URL-re, melyik változóban lévő stringet kelljen visszaküldeni.



18. ábra: Kliens - webszerver közötti kommunikáció

Továbbá ez a könyvtár aszinkron módon működik, nincsen a main ciklusban folyton lefuttatandó része. Tehát így esemény alapú, amikor érkezik egy HTTP kérés. Itt kell eldönteni, hogy az URL az egy valós fájlra mutat, amit vissza lehet küldeni. Ugyanis a POST kérésekre egy általam meghatározott URL-t használok, ahová a JS kódból fetch API-val küldök adatot, és ahol az aszinkron webszerver várja a JSON formátumú üzeneteket. Eddig szó volt a weblap statikus részéről, viszont ennél a pontnál lehet megemlíteni a dinamikus viselkedését a weboldalnak.

Ha egy olyan kérés jön a kliens felől tehát, ami az általam JSON formátumon működő beszélgetésre kijelölt URL-en történik, akkor nem fog a webszerver azon az elérési útvonalon egy fájlt keresni. Ekkor történik a JSON string beolvasása a HTTP kérésből, amit tovább feldolgoz az ESP32. Először megkeresi a csomag azonosítóját, ami szintén mindig belekerül a kliens felől érkező JSON string-be. Az azonosító alapján pedig a csomagnak megfelelő módon szimplán csak bejövő adatot feldolgozza, vagy pedig, ha olyan csomagról van szó, akkor visszaküldi válaszként.

A kliens mindig vár választ a webszervertől értelemszerűen a GET-re, mivel akkor konkrétan általában a weboldalt kéri le a kliens. De ez POST kérésnél is szintén meg van, hogy ha nem

kapna a kliens a webszervertől választ, akkor bizony a kliens ezt sikertelen kérésként értelmezi, és hibakezeléstől függően újra megpróbálja, vagy hibaüzenetet küld a felhasználónak. Tehát a webszervert úgy kellett felépíteni, hogy mindig kell választ küldenie a HTTP kérésekre a megfelelő működés érdekében. Amikor olyan a csomag, hogy beállításról van szó, akkor a sikeres beállítás gyanánt a kérésre válaszul egy „üres minden rendben” HTTP kód (204) elküldését állítottam be. Viszont, ha egy olyan csomagról van szó, ahol a kliens adatot szeretne lekérdezni, akkor pedig „minden rendben” HTTP OK kód (200) elküldésével, szintén az ESP32-őn összerakott JSON string formátumú adattal együtt válaszol vissza a kliensnek a webszerver.

Ekkor a kliens oldalon betöltött JS kódban egy hosszabb algoritmus szerepel a fetch API kérés után, ami a választ dolgozza fel. Ilyenkor szerkesztheti akár a HTML elemeket is a script, például módosít egy megjelenő szöveget az oldal egy részén. Ha a script egy olyan csomagot küld el, ami például csak egy színt állít be egy effektnél, akkor már HTML oldalon megtörtént a weboldal megjelenésében a változtatás a felhasználói interakció révén. Így a webszervertől érkező válasz esetén azon kívül, hogy tudomásul veszi az üzenet elküldésének sikerességét, más egyéb már nem történik.

5.2. Kliens oldal működése (frontend)

Kezelőfelület (kliens oldal) megjelenését a 19. és 20. ábra mutatja be, melyek a „Kezelőfelület megjelenése” alfejezetben találhatóak.

Dinamikus weboldalt hoztam létre, mely az alap HTML kódon felül (és CSS formázási fájlon kívül) JS (JavaScript) kódokat futtat le kliens oldalon a megjelenítő böngészővel. Ezek a JS kódok a felhasználói beavatkozások érvényesítéséhez a háttérben küldenek HTTP POST kéréseket a webszervernek. Erre a fetch API beépített funkciót használom. Mellyel JSON formátumban lehetőség van elég komplex struktúrájú adatok küldésére, melyeket az ESP32 fogad. Az adatok küldésére azért a JSON formátumot választottam, mert széles körben legtöbb helyen ezt alkalmazzák, és ráadásul ezt már ismertem is, tehát könnyen tudok vele boldogulni, meg egyébként is nagyon egyszerű a használata. Továbbá elérhetőek az ESP32-höz JSON-t kezelő könyvtárak, mellyel nagyon egyszerűen lehet vele dolgozni.

A JSON formátum felépítését tekintve egyébként egy közönséges string, melyben lehet „tárolni” bármilyen objektumot. Szintaxisa azonosító alapú, tehát minden objektumnak van egy azonosítója, amiket macskakörmök közé kell tenni minden esetben.
Például: {„colors”:[{„r”:”0”, „g”:”255”, „b”:”0”}, {„r”:”128”, „g”:”0”, „b”:”128”}]}

Az azonosító után kell lennie egy kettőspontnak, mely után:

- szintén macskakörmök között az objektum értéke lehet (pl.: szám esetén egy szám érték, string esetén pedig maga a szöveg),
- vagy pedig szögletes zárójelekben felsorolva egymás után értékek az előbb említett módon,
- vagy még egy lehetőség áll fent, amikor kapcsos zárójelek között egy újabb JSON objektum van.

A JSON objektumba pedig a fentebb felsorolt dolgokat lehet rakni, tehát JSON objektumok JSON objektumokból is állhatnak, és bármeddig lehet egymásba pakolni őket. Ez tehát egy nagy string, melyben például segít megkeresni egy azonosítóhoz tartozó értéket ez az ESP32-höz elérhető könyvtár. Stringként egy azonosító értéke mindig egy karaktersorozat lesz, hiszen az is a stringnek az azonosító után lévő része. Így, ha az egy szám, azt át kell konvertálni valóban integer/float/…stb típussá, ha számként szeretnénk értelmezni. Ezt is megcsinálja ez a könyvtár, hogy a kért változó típusának megfelelően adja vissza a kért azonosítóhoz tartozó értéket, így nem kell külön saját algoritmusokat írni erre a célra.

Ezt a JSON stringet pedig a HTML5 teljesen beépítve kezeli. Tehát a fetch API használatakor egy JS kóddal létrehozott objektumot egy belső függvénynek kell átadni, amit az át tud teljesen konvertálni JSON stringre, ami pedig elküldésre kerül a HTTP POST kéréssel az ESP32-nek.

Szükség volt így kialakítani szintén ilyen csomagokat, amik maguk a JSON stringek, melyekben az azonosítók nevét szükséges volt előre definiálni. Szóval egységes módon kezelje mind a kliens oldal, mind az ESP32 is, hogy ez működni tudjon. Itt már nem volt szükséges előre ledefiniálni a byte-hossz pontosan a csomagokat mi után minek kell jönnie, mint az ARM és ESP32 közötti SPI kommunikációban. Ez már egy rugalmasabb formátum, itt a sorrend se számít, a hossz is változhat, csak az azonosítók neve kell, hogy meglegyen mind a küldött, mind a fogadó félnek, ahogy el kell neveznie, és ahogy keresnie kelljen. Továbbá ezek az objektumok a JSON stringben az adott JSON objektum szintjében meg kell, hogy legyenek, és nem lehet benne lévő objektumnak az objektumában, mert akkor az már nem is hozzá tartozik.

Tehát a csomagok kialakításánál erre kellett figyelnem, az mellett, hogy az azonosító értéke is egy kötött dolog legyen, mert ha az értéknek egy számnak kellene lennie, de valamilyen szöveg, vagy egy komplett másik JSON objektum, amit a küldő fél nem helyesen írt be, akkor azt nem megfelelően fogja a küldő fél átkonvertálni számmá, mert az nem is egy szám. Ezeknek a megkötéseknek a betartásával tud csak minden működni az elvárt módon. Az ESP32 az ilyen módon beérkező adatokat pedig a fentebb említett SPI kommunikációval adja az ARM tudtára.

5.3. Beállítható paraméterek

Azt az alapkoncepciót dolgoztam ki a beállításra, hogy a hangfalba épített LED szalagnak adott szakaszait kiválasztva lehet rá alkalmazni egy effektet. Majd minden egyes típusú effektnek más beállítható paramétereit lehet változtatni. Ha már van jobb és bal hangfal, és külön adatvezetéken történik a LED szalagok vezérlése, így lehetővé tettem, hogy jobb/bal hangfalba épített LED szalagok más-más módon pompázzanak színeikben. Mivel az audió feldolgozás is sztereó jelből történik külön jobb-bal csatornára, így az effektezéskor az audió sáv is választható, hogy melyik(ek)re történjen az audió vizualizáció.

Választható LED szalag szakaszai:

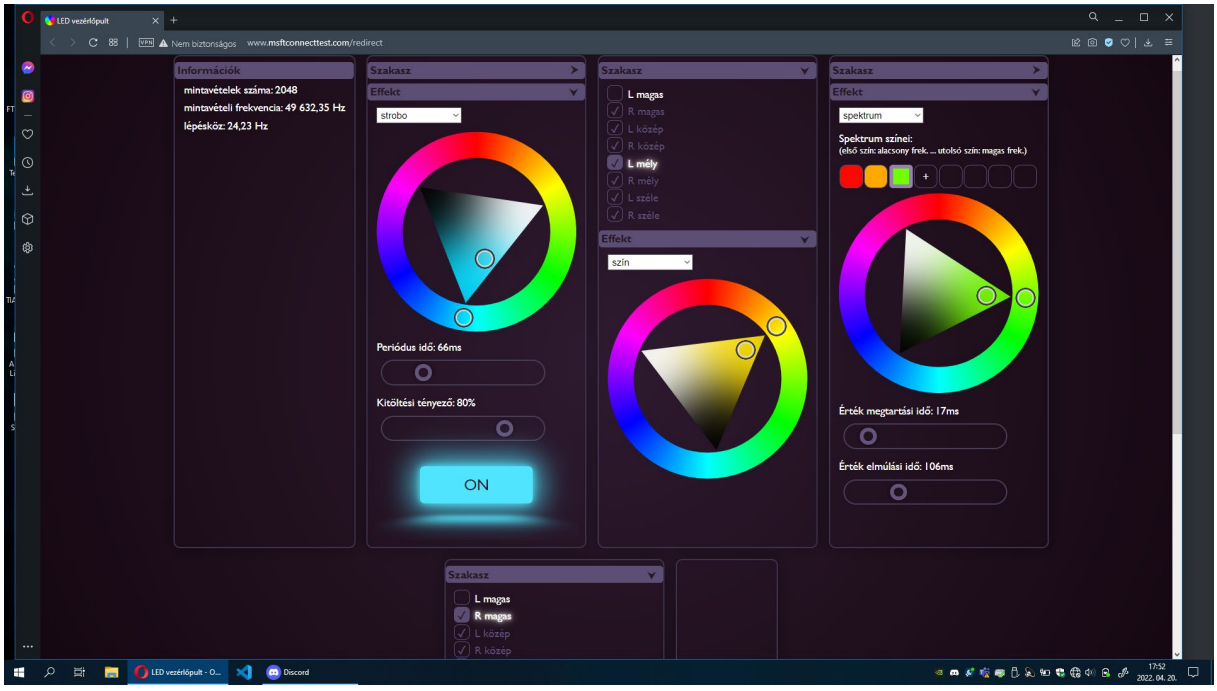
- magassugárzó körüli,
- középsugárzó körüli,
- mélynyomó körüli,
- hangfal oldalában lévő,
- vagy pedig ezek bármilyen kombinációja (akár mindegyik) LED szalag része

A kezelőfelület megnyitásakor egy teljesen üres konfiguráció esetén csak egy lehetőség van, hogy létrehozzunk egy új konfigurációt. Ez azzal kezdődik, hogy kijelöljük mely szakaszokra legyen érvényes az adott beállítás. Ha már legalább egy szakasz ki lett jelölve, utána választható az effekt típusa, majd a választott effekt alapján, annak a paramétereinek a beállításaira kapunk lehetőséget, amit az adott effekthez megjelenő specifikus kezelőszervek biztosítanak.

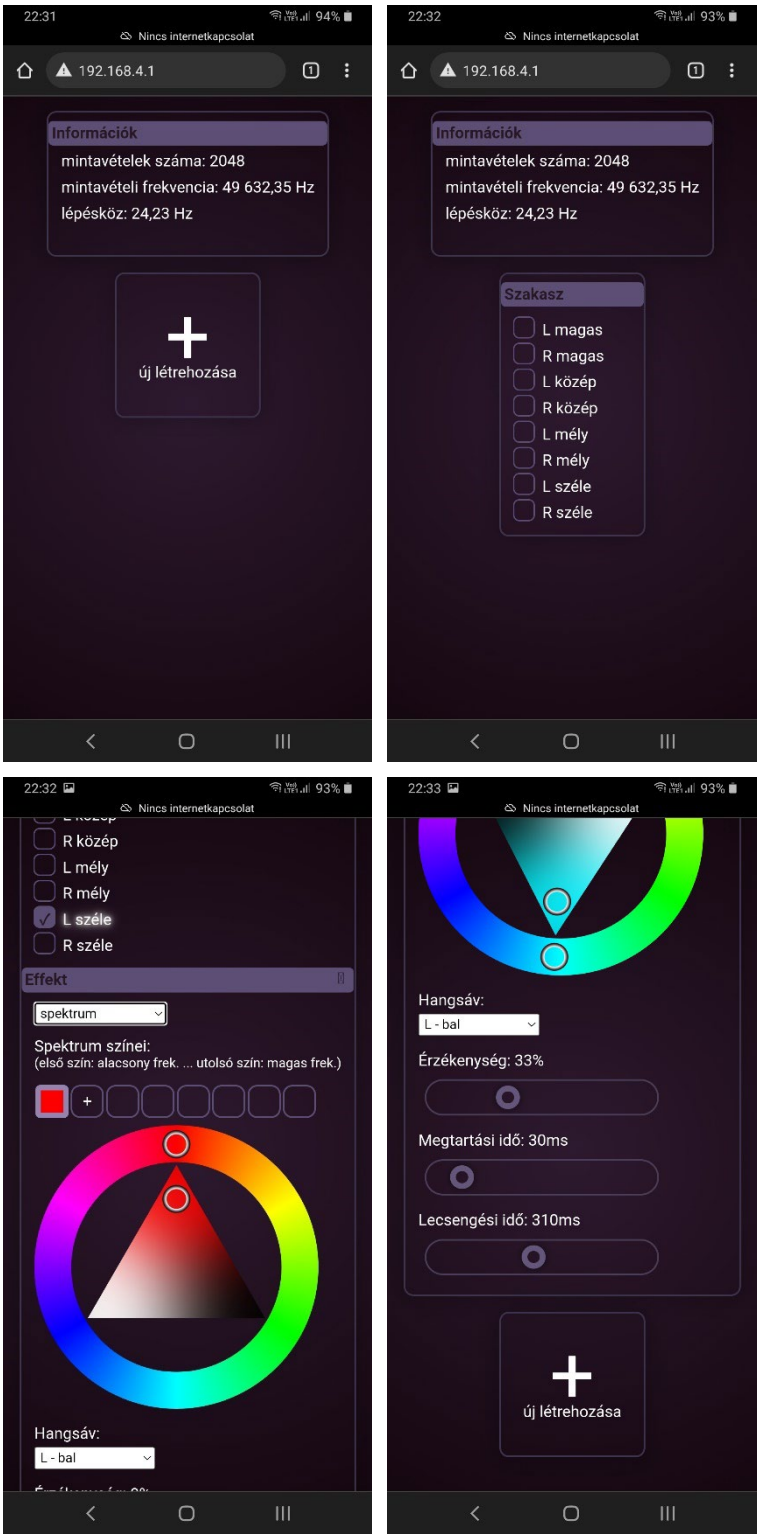
Audió vizualizáció esetén például a spektrum színének kiválasztása. Ez egy színkörből választható, mely real time módon azonnal kerül elküldésre és a beállítás érvényesítésére, nem kell semmilyen küldés, mentés, vagy bármilyen egyéb gombot megnyomni hozzá. Így azonnal meg lehet csodálni az adott színt a színkörből kiválasztva, hogyan mutat élőben. Erre úgy gondoltam azért van szükség, hogy azonnal frissüljön, mert minden egyes beállítás után kényelmetlen lenne újra és újra megnyomni egy véglegesítő gombot, így ez egy plusz könnyedén használható felhasználói élményt ad. Emellett érvel még az, hogy nem feltétlenül azonos az eszközön látott szín, mint amit az RGB LED szalag használ. Én éjszakai módban (kék fény csökkentéses üzemmód) szoktam használni általában az eszközeimet, és úgy bizony észre se veszem már a kijelzőn, hogy nem a valós színeket látom, hanem a kék fény szűrteket, így nyilván más lesz élőben. Meg a különböző eszközök használhatnak akármilyen színprofilt, ami nem biztos, hogy teljes mértékben meg fog valóban egyezni az RGB LED szalag által kiadott színekkel.

A színek választására szolgáló színkör megalkotásakor arra törekedtem, hogy minél kevesebb csúszka, vagy bármilyen kezelő szerv legyen szükséges ahhoz, hogy be lehessen állítani a kívánt színt. Leggyakrabban csak a színárnyalat megváltoztatása a cél, amire egy olyan elrendezés a legkézenfekvőbb, ahol lehetséges a színárnyalton folyamatosan változtatni úgy, hogy egy teljes színárnyalat körgyűrűn lehet választani körbe-körbe kijelöléssel. Viszont legyen mód a fényerősség, és telítettség változtatására is, a kiválasztott színárnyalatra mutató háromszöggel oldottam ezt meg. Ez a sarka a háromszögben a teljes telítettség, és feles fényerőt jelent. A háromszög további két sarka pedig a teljes fényerőt és teljes feketeséget valósítja meg. A sarkok között pedig ezek átmenete található. A kiválasztott szín így színárnyalat, telítettség és fényerősségen alapuló szín meghatározáson alapszik, melynek a neve HSL (Hue, Saturation, Lightness). Ezt viszont könnyedén átlehet konvertálni RGB színértékre, ami a LED szalagon lévő pixelek címzésénél használatos.

5.4. Kezelőfelület megjelenése



19. ábra: Kezelőfelület megjelenése 16:9 arányú monitoron Windows rendszeren



20. ábra: Kezelőfelület megjelenése Android rendszerű okostelefonon

Beállítás lépései a képek sorrendjével megegyezik:
1. új konfiguráció létrehozása,
2. szakasz kiválasztása,
3. effekt kiválasztása, hozzá paraméterek beállítása
4. opcionálisan akár újabb konfiguráció létrehozása

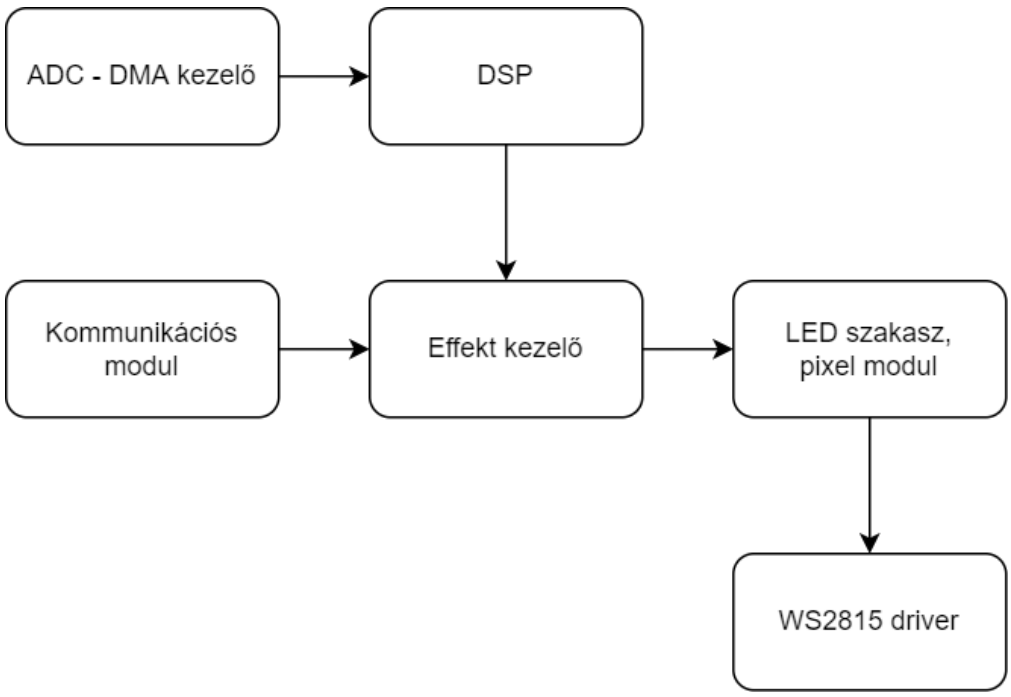
6. Szoftveres háttér

A projektben a megfelelő fizikai eszközök és áramkörök mellett hatalmas szerepet játszanak a különböző eszközökön futó programsorok. Eddig megszerzett tudásomat a projekt során rendkívül jól tudtam kamatoztatni. Akár csak az alkalmazott nyelveket tekintjük, szerepel a sima C nyelv vagy az objektum orientált nyelvek, a C++, JavaScript, továbbá a leíró nyelvek is, melyek a HTML és CSS. Tehát az ARM, ESP32, és a kezelőfelületet betöltő eszköz ezen nyelvek egyikén íródott szoftvert futtat.

6.1. ARM mikrokontroller szoftver működése

STM32-es környezet bekonfigurálható architektúrájában történt az ARM mikrokontrollerre a szoftver implementálása C nyelven az STM32CubeIDE fejlesztői környezetben. [6] [8] [9]

Indításkor a következő modulok inicializálódnak: ADC-DMA kezelő, DSP, effekt kezelő, LED szakasz, pixel modul, WS2815 driver és egy kommunikációs modul. Ezek a modulok később szekvenciálisan egymás után kerülnek meghívásra egy szálon futó magról.



21. ábra: ARM szoftver blokkvázlata

Az ADC-DMA kezelő modul feladata a DMA elindítása és beállítása arra a buffer tömbbe, melybe érkezhetnek a mérési eredmények. A DMA egyetlen megszakítást hív meg, ahol a forgó negyed tömbök közül az éppen aktuális tömb frissül. Lekérdezhető, hogy melyik időpillanatban

frissült a buffer, továbbá a helyreállított időrendi sorrendben vett mintákat szintén képes ez a modul átadni.

DSP (Digital Signal Processing) modulban történik a spektrum analízis. Inicializáláskor az optimalizáláshoz szükséges értékek előre kiszámolásra kerülnek és egy buffer tömbbe tárolódnak le a futás közbeni használatra. Az FFT algoritmust abban az esetben futtatja le a modul, ha az utoljára elkészült spektrum analízis óta újabb mérési eredményeket szállított le az ADC DMA-ja. Ebből a modulból lekérdezhető a spektrum analízis eredményeként szolgáló értékeket tartalmazó tömb. Amiből jobb, és bal csatornára is elkészül egy-egy spektrumkép, melyek külön-külön csatornánként kérdezhetőek le.

Kommunikációs modulban az ESP32 és ARM közötti SPI-on keresztül folyó adatok feldolgozása történik. Viszont ezen kívül még a debugolásra használt UART üzenetek küldése is ebben a modulban kezelődik le. Ezek olyan szoros összeköttetésben vannak, hogy minden SPI-on érkezett adat elküldésre kerül UART-on is. Ezzel monitorozhatóvá vált az adatfolyam, melynek hibáira így könnyen fényt lehetett deríteni. Ebből a modulból pedig a beérkező adat csomagok értelmezése során az effekt kezelő modul a csomagnak megfelelő része kerül meghívásra és a csomagban érkezett adatok átadásával. Hatékonyság növelése érdekében itt az ESP32 már olyan sorrendben küldi át az adatokat, melyeket lényegében ahogy megérkeztek egy az egybe az adott effekthez tartozó memóriaterületre kerülhet átmásolásra.

Az effekt kezelő modulban kerülnek meghívásra az éppen aktuális szakaszon lévő effekthez tartozó algoritmusok. Amelyek pedig az előző állapotok alapján, a beállított konfiguráció alapján és az aktuális eredmények lekérdezésével számítják ki az effekthez tartozó szakaszon lévő pixelek aktuálisan megjelenő színkódjait. Az effekt kezelő modult pedig a LED szakaszok és pixeleket lekezelő modul hívja meg.

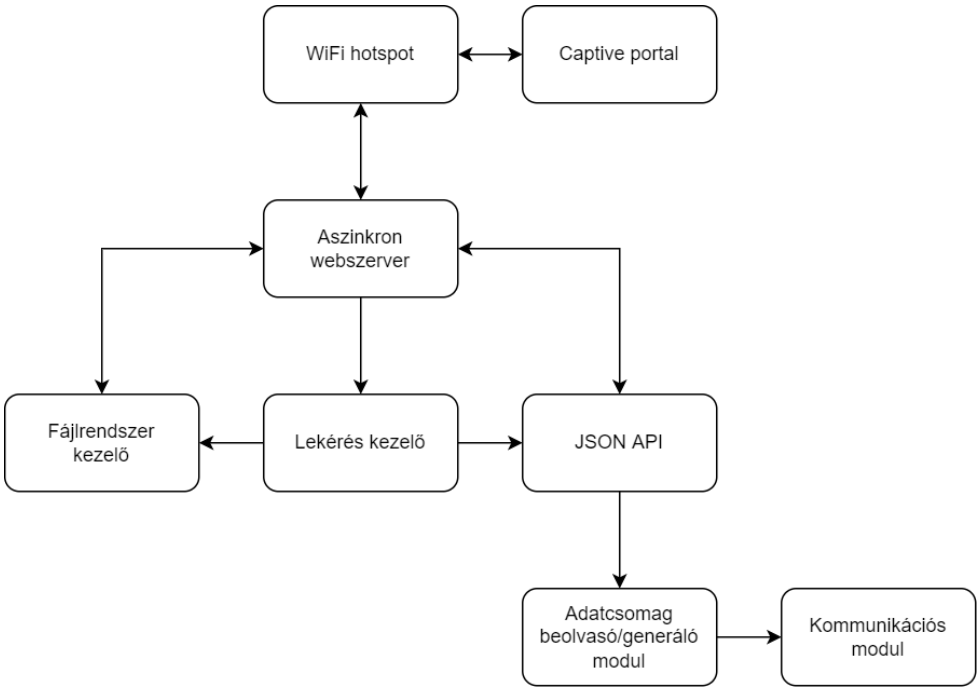
Minden egyes szakaszhoz tartozik egy struktúra, melyben a tulajdonságai tárolódnak le a szakasznak. Továbbá lekérdezhetőek a hozzá tartozó pixelek is. Amint egy szakasz megkap egy konfigurációt, onnantól kezdve a beállított effektnek megfelelően ez a modul gondoskodik arról, hogy a hozzá tartozó effekt kezelő algoritmust hívja meg. Ez a modul, ha történt a pixelek értékeiben változtatás, a hozzá tartozó LED szalagra kérni fogja a WS2815 driverétől a pixelek tényleges frissítését a LED szalagon.

A WS2815 drivere pedig a dokumentáció alapján lett megírva. Tehát ez alapján fogja végrehajtani a LED szalag megcímzését. Ez a modul generálja le a pixelek RGB értékeiből a PWM jel kitöltési tényező értékeit. Minden egyes frissítéskor egy DMA van elindítva, mely

végrehajtsa lényegében az adatfolyam kiküldését azzal, hogy minden PWM periódus letelte után automatikusan átírja a kitöltési tényezőt a soron következő értékre. Ez a modul kezeli le tehát mindkét hangfalban lévő LED szalag állapotait, és a szigorú időzítéseknek megfelelő vezérléséről külön-külön elszeparálva gondoskodik DMA segítségével.

6.2. ESP32 mikrokontroller szoftver működése

Arduino platformon történt az EPS32 szoftverének implementálása Microsoft Visual Studio Code fejlesztői környezetben Platform IO kiegészítőt használva. [7] [10] Ennek a platformnak pedig a nyelve már C++. Szoftverben az alap Arduino könyvtárán kívül ESP32 mikrokontrollerre írt Aszinkron webszerver és JSON formátumot támogató könyvtárak lettek felhasználva. Az ESP32 alap könyvtárában pedig a vezeték nélküli WiFi hálózat kezeléséhez szükséges könyvtárak már beépítve rendelkezésre álltak.



22. ábra: ESP32 szoftver blokkvázlata

Inicializáláskor létrejön a WiFi hotspot, és elindul a kliensektől érkező kéréseket váró webszerver modulja. Továbbá az SPI kommunikáció felépül, és az ARM felé egyből adat kerül elküldésre. Melyre az ESP32 választ vár, mely csak egy olyan egyszerű csomag, ahol az ARM elküldi, hogy mekkora méretűre van beállítva a mérési eredményeket tartalmazó buffer. Ez az adat arra lesz jó, hogy majd a kliensnek tovább küldve az kitudja számolni a frekvencia lépésköz értékét a paraméterek állításánál.

A WiFi hotspotra egy Captive Portal, avagy bejelentkező felület van állítva. Ez azt a célt szolgálja, hogy amint csatlakozott egy új eszköz a hálózatra akkor megkapja egyből felugró ablakként a rendszer kezelőfelületét. Ez miatt rögtön a WiFi-re csatlakozáskor elérhetővé válik a kezelőfelület, még csak semmi egyéb címre se kell elnavigálni így.

A már a hálózaton lévő eszköz küld tehát HTTP kéréseket, akkor azt a webszer modulja fogja feldolgozni, és az abban regisztrált eseménykezelő meg fog hívódni. Ekkor a cím URL alapján eldönthető, hogy melyik további modulba fog átmenni a vezérlés. Ha éppen a weboldal fájljait kéri le a kliens az URL alapján, akkor a fájlrendszerből kiolvasásra, majd elküldésre kerülnek ezek a fájlok az ezt elvégző modulnak hála. Ha pedig az URL a backend API-ra mutat, ahová a kliens adatok küldése esetén csatlakozik, akkor pedig a POST kéréseket kiolvasni képes JSON API modul fog aktiválódni.

Az értelmezett adatot pedig feldolgozza, majd tovább küldi az ESP32 az ARM részére. Ami úgy történik meg, hogy megfelelő formátumú adatcsomag létrejön, majd átkonvertálásra kerülnek bele a webszerveren keresztül a klienstől jött adatok. Ez a modul az adatkonverzió után tehát a kommunikációs modult hívja meg, melyben nem történik más, mint az SPI kommunikáció megvalósítása.

6.3. Kliens oldali frontend szoftver működése

Ez a szoftver már platformfüggetlen, csupán böngésző szükséges a futtatására. Ennek fejlesztése szintén Microsoft Visual Studio Code programban történt. Egy Live Server nevű kiegészítőnek hála kódírás közben végig tesztelhető, kipróbálható tudott lenni az addig lefejlesztett weboldal. Maga a weboldal leírása pedig HTML nyelven van. Hozzá csatolható formázási stílus lap pedig CSS nyelvet használja. [5] Továbbá a weboldal esemény vezérelt, ahol is az eseményekre beavatkozó kódra pedig JavaScript nyelvet lehet használni.

Az esemény vezérelt működés alatt pedig azt kell érteni, hogy a weboldalnak nincs folyton futó programsora, hanem csak a regisztrált eseményeknél lévő kódok fognak lefutni abban az időben, amikor az esemény meghívódik.

A fő moduljának a weboldalnak az új konfiguráció létrehozása gomb megnyomásakor történő reagáláson kívül pár alap funkciónak a megvalósítása is megtörtént. Az egyik ilyen a fetch API-t használó rész, mellyel megtudtam könnyíteni például az egységes URL-re történő küldését a POST kéréseknek. Továbbá ebben a modulban történik még a weboldal betöltését követő inicializálás, ahol a mintavételek számának a lekérdezése történik meg egyből.

Következő modulba lépés akkor történik meg, amint a felhasználó megnyomta az új konfiguráció létrehozása gombot. Mivel ez a modul felelős azért, hogy összegyűjtse az eddig felhasznált, és elérhetővé tegye a még nem felhasznált LED szalag szakaszok kiválasztását.

Amint egy új szakaszt, vagy pedig meglévő konfigurációnál másik effektet választunk ki, abban ez esetben az aktuálisan kiválasztott szakaszokra az adott effekt elküldése kerül sorra. Tehát a felhasználó bármit módosít, egyből reagál rá a rendszer, és valós időben láthatja az átállított beállítások következményét. Értelemszerűen, ha elvesszük a kijelölést egy szakasztól, akkor ez a modul gondoskodik arról, hogy a többi konfigurációnál újra elérhetővé tegye azt a szakaszt, és egy alaphelyzetbe állítást kérő csomagot is elküldjön a vezérlésnek, hogy érvénybe lépjen az effekt törlése a szakaszról.

Két kezelőszerv külön-külön modulokba került elhelyezésre. Egyik a szín kiválasztó kör, mely sok belső számítást igényel. Ezért vált szükségessé külön modulba elszeparálása a kezelőszervtől. Továbbá különböző értékek tól-ig történő állítására létrehoztam egy tologatható csúszkát, mely számok pötyögése helyett egyszerűbb és gyorsabb beállításra ad lehetőséget. Tehát a másik modulban ez a csúszka foglal helyet.

A kezelőszervek módosításakor szintén egy esemény hívódik meg, melyben kiszámításra kerül az adott kezelőszerv új értéke. Mivel módosult az effekt egy paramétere, ezért ez a konfiguráció egységében található többi paramétere is begyűjtésre majd küldésre is kerül az ESP32 részére.

A CSS formázással pedig a különböző méretű kijelzőkhöz alkalmazkodás JavaScript kód futtatása nélkül is elérhetőek. Így ahol lehetett CSS formázást alkalmaztam a megjelenő blokkokra. Viszont egyes kezelő szervek módosításakor olyan JavaScript kód fut le, mely felülírja a CSS formázás egyes részeit. Továbbá a barátságosabb felület eléréséhez bizonyos animációk játszódnak le egyes esetekben.

Weblap fejlesztése közben minden egységet itt fokozottan részletesen teszteltem le, számtalan megoldásban próbáltam ki, módosítottam a megjelenített felületet szinte minden újabb verzióban. Egyszerre kellett Windows, és okostelefonos rendszeren is kipróbálni mindent, hogy valóban működik-e minden funkció. Erre a Windowsra elérhető Google Chrome böngésző fejlesztői eszközét tudtam használni, és ez által nagyon sok problémára elég hamar fény derült, és kijavításra került.

7. Összefoglalás

7.1. Elkészült projekt

A projektem fő célját, hogy egy látványos audió vizualizációt megvalósítsak ESP32 segítségével WEB-es technológiával kiegészítve, ezzel testreszabhatóvá téve ezt az egészet, sikerült is véghez vinnem. Ehhez az eredményhez vezető rögös úton rengeteg tapasztalatot tudtam gyűjteni, és új dolgokat felfedezni, kipróbálni.

Az ESP32 lehetőségeit, használatát ezen projekt alatt sajátítottam el teljes mértékben. Az ARM és a hozzá tartozó STM32 környezetében rejlő lehetőségek közül ismertem meg újabbakat az eddigi tudásomhoz képest, és alkalmaztam a projekt során. Természetesen foglalkozni is kellett a mikrokontrollereket kiegészítő áramkörök megtervezésével, kivitelezésével, és tesztelésével is. Továbbá a címezhető LED szalagok felépítését és vezérlését is el kellett sajátítani. A WEB-es technológia beépítése a projektbe pedig a legtöbb utánajárással, próbálgatással, majd szintén teszteléssel is járt.

Projekt során olyan problémákkal kellett megküzdenem, mint az ARM mikrokontrolleren futó kódban lehető legtöbb optimalizálás implementálása a minél gyorsabb és pontosan spektrumkép készítés érdekében. Erre a DMA egység használata, felkonfigurálása megfelelően. Magát pedig a spektrum analízist lebonyolító algoritmus implementálása C nyelvre valós idejű eredményekkel szolgáló módon. Ez mellett pedig az effektek lekezelése is. Két mikrokontroller közötti kommunikáció megvalósítása SPI-on keresztül.

Végül, de nem utolsó sorban pedig a rendkívül reszponzív weboldalként elérhető kezelőfelület megalkotása. Amiben megtalálható például egy saját fejlesztésű szín kiválasztó kezelő szerv is, mely a többi kezelő szervvel együtt dinamikusan jön létre, változik meg, egy másik weblap újra betöltése nélkül. Ebben pedig nagy szerepet játszó ESP32, mint backend webszerver API mely ezt szolgáltatni képes a kliens számára. Tehát ennek a megvalósítása igazán nagy kihívást jelentett.

7.2. Fejlesztési lehetőségek

A projekt során rengeteg ötlet felmerült, amivel még ki lehetne egészíteni a jelenleg elkészült verziót. Egyik ilyen hasznos dolog, amivel ki lehetne egészíteni, hogy a beállított konfiguráció ne vesszen el, mivel jelenleg akár, ha újratöltjük a weboldalt akkor már elveszett. A mikrokontroller pedig az utoljára beállított konfigurációt addig jeleníti meg, amíg újra nem indul. Szóval a konfigurációt elmenteni a memóriába, illetve a jelenlegi konfigurációt a kezelőfelület betöltődésekor meg is jeleníteni egyből.

Kiválasztható effektek tárházát a végtelenségig lehetne bővíteni, tehát relatív egész keveset valósítottam csak még meg. Már vannak olyan effektek, amit tud a rendszer, csak még nem készítettem el hozzá a kezelő szerveket, így a kezelőfelületen nem érhető el, nem lehet így rákonfigurálni egyik szakaszra sem. Továbbá a meglévő olyan effekteket, amik nem reagálnak a zenére, azokat továbbfejleszteni olyanra, hogy valamilyen módon reagáljanak a hangokra színek, fényerő, vagy futófény sebességének változtatásával.

A rendszer legnagyobb gyenge pontja pedig az, hogy csak breadboardon van megépítve, így csomó zajt összeszed, meg kontaktus hibák is fellépnek néha, ha 1-2 pin lötyögős. Egy nyomtatott áramköri lapot gondos tervezéssel csinálni hozzá, az bizony ezeket a sok kellemetlenséggel járó hibákat kitudná küszöbölni egy csapásra.

8. Irodalomjegyzék

8.1. Felhasznált elektronikus jegyzetek

[1] Dr. Iváncsyné Csepesz Erzsébet:

ELEKTRONIKA I., *Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Automatika Intézet, 2015* (Egyetemi Moodle rendszerből letöltve: 2021.09.)

[2] Zalotay Péter:

Digitális technika I., *BMF Kandó Kálmán Villamosmérnöki Kar Elektronikus jegyzet* (Egyetemi Moodle rendszerből letöltve: 2022.01.)

[3] Sarcevic Péter:

Mérésadatgyűjtés, jelfeldolgozás, Szegedi Tudományegyetem Mérnöki Kar (http://eta.bibl.u-szeged.hu/2139/1/Sarcevic_P_Meresadatgyujtes_Jelfeldolgozas.pdf linkről letöltve: 2022.03.)

8.2. Tájékozódásra használt internetes források

[4] Alwayslearn.com, DFT and FFT TUTORIAL

(https://www.alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_BasicIdea.html linket használva: 2022.02.)

[5] W3schools.com, HTML, CSS, JavaScript, stb… oldalai

(https://www.w3schools.com linket használva 2021.10 – 2022.04 között)

[6] STMicroelectronics fórum oldalán böngészés

(https://community.st.com/s/ linket használva 2021.08 – 2022.04 között)

[7] PlatformIO ESP32 dokumentációja

(https://docs.platformio.org/en/latest/platforms/espressif32.html linket használva 2021.10 – 2022.04 között)

8.3. Felhasznált eszközökhöz tartozó adatlapok

[8] NUCLEO-F756ZG board-hoz tartozó User Manual, *UM1974 Rev 8 August 2020*

(https://www.st.com/en/evaluation-tools/nucleo-f756zg.html#documentation linkről letöltve: 2021.08)

[9] STM32F756ZG MCU-hoz tartozó Reference Manual, *RM0385 Rev 8 June 2018*

(https://www.st.com/en/evaluation-tools/nucleo-f756zg.html#documentation linkről letöltve: 2021.08)

[10] Nodemcu-32s board adatlapja
(https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf linkről letöltve: 2021.10)

[11] BTF-Lighting WS2815B adatlapja
(https://m.media-amazon.com/images/I/81PhbduinSL.pdf linkről letöltve: 2021.10)