Budapesti Műszaki és Gazdaságtudományi Egyetem
Mikrokontrollerek Alkalmazástechnikája (VIMM9151)

# Zene vizualizációja LED szalagon

Házi feladat

Készítette:
Izsó András (UICLAA)

Jelige:
8CE1322B

2019. 05. 23.

# Tartalomjegyzék

# 1 Feladat kiírás

A házifeladat egy Cypress CY8CKIT-059 fejlesztőkártyával megvalósított RGB LED vezérlő lesz. A feladathoz egy 5050 típusú, nem címezhető RGB LED szalagot használok. A LED szalag a zene ritmusára fog villogni, az ütésekre villan fel a szalag, majd elhalványul. A bemenet egy mikrofon és egy vonal szintű 3,5mm-es jack bemenet közül választható, gombnyomással. A LED-ek színe frekvenciafüggő lesz. Az alacsony frekvenciák a piros színt erősítik, a közepeseket a zöld, a magasakat a kék szín fogja reprezentálni, ezek között átmenettel. Az eszközben lesznek előre meghatározott színjátékok melyek közül gombnyomásra, vagy az eszközhöz soros porton csatlakozva a megfelelő parancs kiadásával válthatunk.

# 2 Részletes specifikáció

Az eszköz képes zenére villogtatni egy nem címezhető RGB LED szalagot. Két audio bemenettel rendelkezik: egy 3.5mm-es jack anya csatlakozóval (telefonról meghajtható), valamint egy mikrofonnal. A két bemenet között egy gomb megnyomásával, vagy soros porton küldött megfelelő üzenettel tudunk váltani. A bemenetek melletti LED-ek jelzik az éppen aktív bemenetet.

A bemenő jelet a szükséges erősítés után egy ADC segítségével olvassa be. A mintavételezés alapján meghatározza a minta domináns frekvenciáját és az alacsonytól a magas frekvenciákig rendre piros-zöld-kék-fehér színt rendel hozzájuk, köztük átmenetekkel. A színt a zene ritmusára, a hangerőtől függő fényerővel jeleníti meg a LED szalagon. Ütések között a fény elhalványul.

Az eszköz ezen kívül tartalmaz beépített színjátékokat:

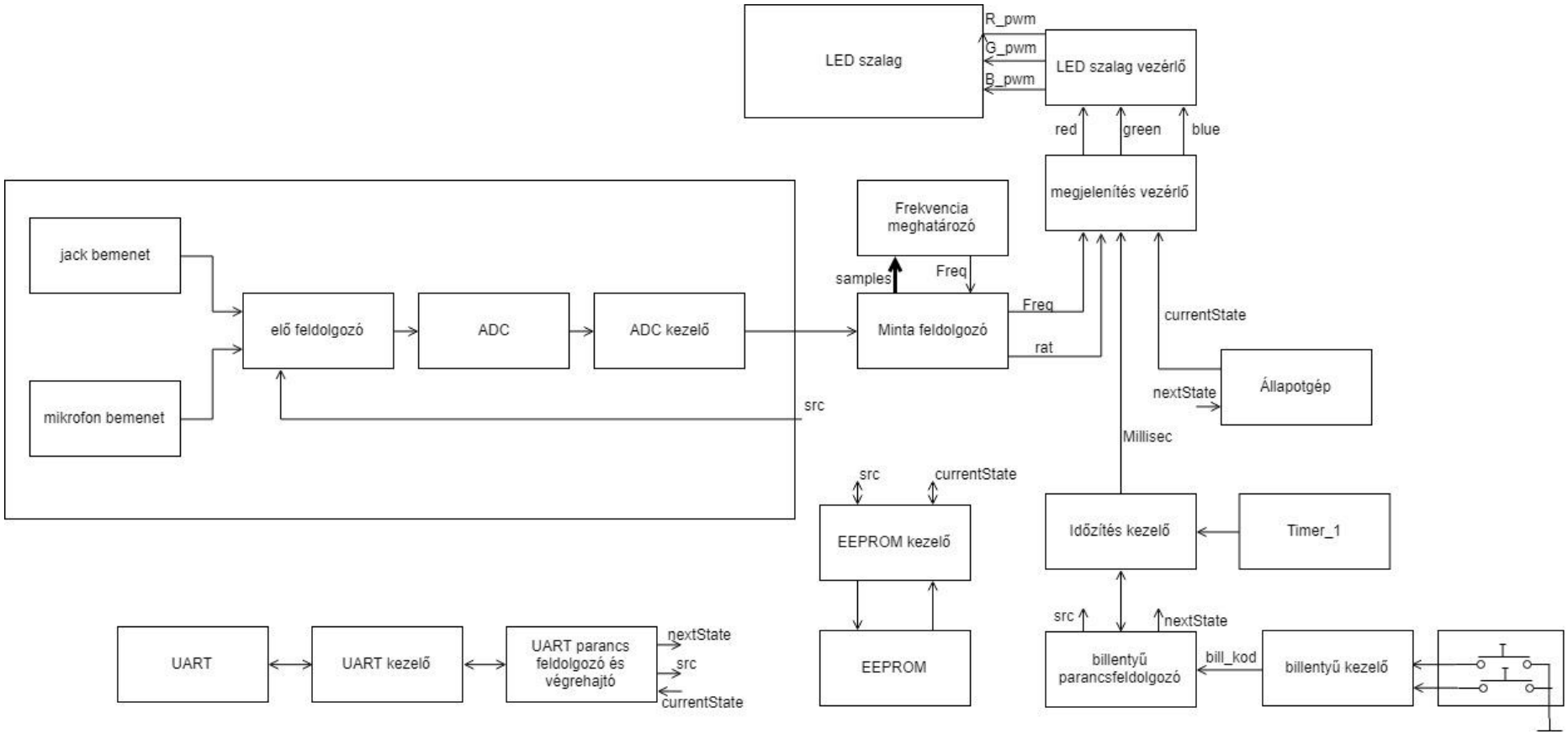
- Cycle: a színskálán megy körbe ciklikusan, folyton változó fényátmenetet generálva
- RedGreenBlue: a piros-zöld-kék színek villogtatása egymás után
- Police: a piros és kék színeket villogtatja felváltva

A beépített színjátékok között gombnyomásra, vagy a megfelelő üzenet soros porton való elküldésével lehet váltani. A soros porton keresztül tetszőleges színjátékhoz ugorhatunk.

Az eszközt soros portra csatlakoztatva nem engedi rögtön bármelyik parancsot kiadni. Először át kell váltanunk terminál üzemmódba az eszközt, csak ezután adhatjuk ki a parancsokat. Ekkor a gombok letiltódnak és akkor engedélyeződnek újra, ha kiléptünk a terminál üzemmódból.

Az eszköz újra indítás után az utoljára kiválasztott aktív bemenetet és színjátékot állítja be.

# 3 Funkcionális blokkvázlat



A jack bemenetről és a mikrofonról érkező jelek egy előfeldolgozón átesve mennek tovább az ADC-re. Az ADC beépített kezelőjén keresztül ezután a minták a Minta feldolgozóba kerülnek, ami elvégez néhány műveletet, valamint átadja az adatsort a Frekvencia meghatározó egységnek, hogy határozza meg a minták domináns frekvenciáját. Ezután egy arányszámmal együtt továbbadja a megjelenítés vezérlőnek.

A megjelenítés vezérlő a jelenlegi állapot, az időzítő flag és a mintafeldolgozótól kapott értékek alapján határozza meg a beállítandó szín RGB értékeit. Ezek alapján a LED szalag vezérlő végzi a konkrét PWM jelek kibocsájtását.

Az Időzítés kezelő a Timer_1 megszakításai alapján végzi a gomb lekérdezések valamint a megjelenítés vezérlő időzítését

A billentyű parancsfeldolgozó a billentyűzet kezelő által küldött adatok alapján határozza meg a szükséges módosításokat (jelenlegi állapot, bemenet forrása).

Az beépített EEPROM-ból az EEPROM kezelő kérdezi le induláskor a legutóbbi állapotokat, valamint ez is menti el ezeket.

Az UART-on érkező üzeneteket a beépített UART kezelő hozza kezelhető formára, és ezek alapján Az UART parancs feldolgozó és végrehajtó egység végzi el a szükséges változtatásokat, valamint ad tájékoztatást a rendszer állapotáról.

# 4 Hardver-szoftver szétválasztás

Az audio bemeneteket egy 2 csatornás jack aljzattal (J1), valamint egy előerősítőt tartalmazó mikrofon modullal (U2) valósítottam meg. A jack aljzat és a fejlesztőkártya közé 10kΩ-os ellenállásokból (R1-3) olyan áramkört építettem, ami a kártyába beépített műveleti erősítővel összeadó áramkört valósít meg. A csatlakozót biztonsági okokból, és ha esetleg lenne, a DC komponens eltávolításért egy-egy 1µF-os kondenzátoron keresztül kötöttem be (C1-2). Az áramkörbe védődiódákat is elhelyeztem (D1-2).

A bemeneti jelek közötti váltást a kártya belső analóg multiplexerével oldottam meg. Analóg digitális konverzióhoz a beépített Delta-Sigma átalakítót használom folytonos üzemmódban, 44000Hz mintavételezéssel (41666 tényleges mintavételezéssel), Vssa és Vdda között, 16 bites felbontásban a pontosabb eredményért.

Az aktív input jelzésére egy zöld (jack, D3) és egy narancssárga (mikrofon, D4) LED-et használok a fejlesztőkártya lábaira kötve, 220Ω-os ellenállásokkal (R5-6).

A kártya erőforrásaiból használok meg egy Timer egységet, mely 1ms-onként kér interruptot, valamint 3 PWM jel előállító komponenst. Mind a 3-mat 8 bites módban használom, „kisebb” összehasonlítással, 255-ös periódussal.

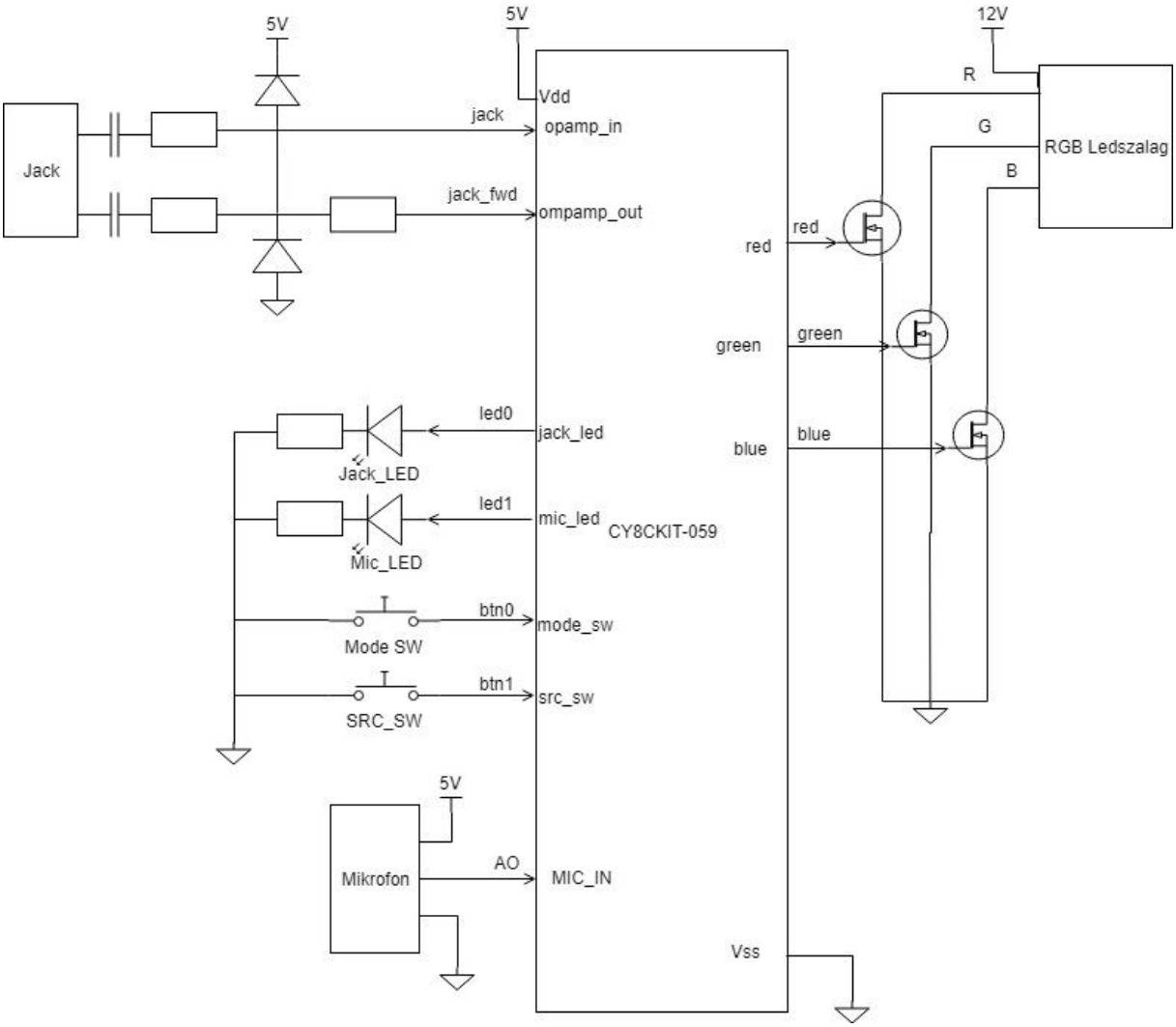
A gombokat (SW_MODE és SW_SRC) szintén a fejlesztőkártya lábaira kötve használom. A kártya beépített felhúzóellenállásait használva, szoftveres prellmentesítéssel.

A LED szalag vezérlését 3db IRLZ44N FET (Q1-3) segítségével végzem, melyek gate-je kapcsolódik a fejlesztőkártya lábaihoz. LED szalagnak egy 3m-es 5050 típusú LED szalagot használok (U3).

Nem felejtő memóriának a kártya beépített EEPROM egységét használom. Sorosport kommunikációra szintén a beépített UART komponenst használom, az alapértelmezett lábakat használva.

Fejlesztőkártyának egy Cypress PSoC CY8CKIT-059 típusú eszközt használtam (U1).

# 5 Hardver rendszerterv




A jack bemenet felől 2 csatornán jön a jel. Mindkettő átmegy egy kondenzátoron, az esetleges DC komponens eltávolítása végett. A közös ponttól az egyik vezeték a műveleti erősítő bemenetére megy, a másik pedig egy ellenálláson keresztül a kimenetére, így valósul meg az összeadó áramkör.

A két input led egy-egy ellenálláson keresztül csatlakozik a földre, így a mikrokontroller lábairól vezérelhető. A gombok a föld és a mikrokontroller felhúzó ellenállása közé vannak kötve.

A mikrofonnak szüksége van tápra, földre, és így egy analóg vezetéken keresztül lehet levenni róla a jelet.

A LED szalagot 3 FET-en keresztül vezérlem. Ezen kívül szüksége van egy 12V-os tápra.

# 6 Szoftver rendszerterv

## *6.1 Fő program*

A program fő metódusa (`int main(void)`) először egy inicializáló függvényt hív (`void init()`) amely beállítja a szükséges változókat valamint felparaméterezi és elindítja a PSoC beépített komponenseit.

A fő programban egy állapotgép futtatása és a soros port kezelése történik. Először az ADC kikapcsolása történik, amennyiben nincs rászükség (ezt minden ciklusban ellenőrizni kel, mivel lehetséges, hogy állapotátmenet történt). Ezután a következő állapotba váltás történik meg (`void updateState()`). Ennek során frissítem a jelenlegi állapotot tároló változót (`LightStates currentState`) az előre eltárolt következő állapot alapján (`LightStates nextState`), majd elmentem az új állapotot az EEPROM-ba. Az állapotgép minden színjátékhoz tartalmaz egy állapotot. Ezek nyilvántartására szolgál a LightStates enumeráció, mely minden állapothoz rendel egy értéket és még egy állapotszám érték is elérhető. Ezzel a forráskód is olvashatóbb lesz, és elkerülhetjük, hogy nem létező, vagy ugyanolyan értékeket használjunk.

Az állapotfrissítés után meghívom az állapotnak megfelelő do<FényjátékNeve> függvényt, mely elvégzi a megfelelő beállításokat a LED szalagon. Ezek paraméter illetve visszatérési érték nélküliek.

Az állapotgép kezelése után a program minden ciklusban Ellenőrzi, hogy érkezett-e üzenet a soros porton, és ha igen akkor annak megfelelően állítja be a program változóit.

## *6.2 Interruptok*

A fő program futását kettő interrupt szakíthatja meg. Az egyik az időzítést kezeli (`CY_ISR(TimerIT)`), a másik az ADC megszakításait (`CY_ISR(AdcIT)`). Paraméterként a kezelendő interrupt programban használt nevét kell átadni.

### 6.2.1 Timer interrupt

A timer interrupt milliszekundumonként hívódik meg. Az interrupt kezelő függvény elsőként beállítja a fényjátékok időzítéséhez használt flag-et (`extern bool milsec`), majd ellenőrzi a gombok állapotát. A függvény folyamatosan nyilvántartja az utolsó 5 ciklusban olvasott értékeket, és amennyiben bármelyik is igaz volt, úgy átállítja a gomb állapotát jelző változót (`bool sw_modeStatus, bool sw_srcStatus`). Ekkor a megfelelő változó állításával kezeli a gombnyomást (`nextState, InputChannel srcChannel`). Elengedés detektálás hasonló módon történik, viszont itt az utolsó 5 olvasásnak hamisnak kell lennie, és csak a gomb állapotát jelző változót kell átállítani.

A bemenet váltását a setInput metódus kezeli (`void setInput(InputChannel channel)`). Ennek paraméterként a kívánt bemenet értékét kell átadni. A függvény átállítja a kapcsolódó változót (`srcChannel`) valamint elmenti az EEPROM-ba.

A bemeneti csatornák könnyebb kezelésére ugyancsak bevezettem egy enumerációt.

### 6.2.2 ADC interrupt

Az ADC intteruptjait kezelő függvényben a minták gyűjtése történik. Amennyiben még nem értük el a kelló mintaszámot (`WINDOW_SIZE`), akkor a megfelelő bemeneti buffer (`int16 audioBuff[2][WINDOW_SIZE], uint8 buffIndex`) megfelelő pozíciójába (`uint16 elementIndex`) írjuk a beolvasott értéket, és frissítjük a pozíció számlálót. Itt azért nem enumerációt használtam, mert végig 1 értéről van szó, nem vehet fel más értéket (pl.: a jelenlegi állapot 4 féle is lehet).

## *6.3 LED szalag kezelő, színjáték függvények*

A színek egyszerűbb kezeléséhez bevezettem egy `RGBColor` nevű struktúrát, amely 3db uint8 típusú adattagból áll, rendre red, green, blue elnevezésekkel. Ebből a struktúrából a `currentColor` tárolja a jelenleg kijátszott színt (a halványítás miatt szükséges).

### 6.3.1 setColor

A szín egyszerű beállítását a setColor metódus teszi lehetővé (`void setColor(uint8 red, uint8 green, uint8 blue)`). Paramétereiben kell megadni a piros, zöld, kék értékeket, 8 bittel, ahogy RGB színeknél általában szokták. Itt nem a struktúrát várom paraméternek, mivel legtöbb esetben csak feleslegesen kéne létrehozni belőle egy példányt, hogy azt át tudjam adni a függvénynek. A függvény frissíti az aktuális színt (`currentColor`), valamint beállítja a PWM komponensekre a megfelelő összehasonlítási értéket.

### 6.3.2 doCycle

A függvény minden meghíváskor először ellenőrzi, hogy eltelt-e egy millisec az előző óta. Ezután egy 10-es előosztást még elvégez (`static int counter`), tehát 10ms-onként frissíti a fény beállítást. Ehhez először megnöveli a piros értéket 0-ról 255-re (fokozatosan), majd mellé a zöldet is, aztán a pirosat elhalványítja és a kéket erősíti fel, stb. Ehhez egy számlálót tart karban (`static int color`), amit túlcsordulás elkerülése végett mindig visszaállít, ha végigért a teljes színskálán.

### 6.3.3 doRedGreenBlue

Ez a függvény is csak milliszekundumonként hajt végre bármit is. Egy számlálót tart karban (`static long time`), aminek a 3000-red részének a 3-mal vett modulusa alapján határozza meg, hogy most éppen piros, zöld, vagy kék színt kell kijátszania.

### 6.3.4 doPolice

Hasonlóan működik az előző színjátékhoz, viszont itt 650ms a periódus, és csak a piros és kék színek váltakoznak.

### 6.3.5 getDominantFreq

A domináns frekvencia meghatározására használt függvény (`int getDominantFreq(int buffIndex)`). Paraméterként annak a buffer-nak a sorszámát várja, ami tartalmazza az analizálni kívánt mintákat. Autokorrelációs algoritmust használ, és végül a MAX_FREQ és MIN_FREQ között határozza meg a domináns frekvenciát. Azért elegendő 2000Hz-ig vizsgálni a frekvencia tartományt, mert most csak az alap frekvenciák érdekelnek, a felharmónikusok nem, és az átlagos zenében nem szokott ilyen magas hang előfordulni.

### 6.3.6 audioSetColor

Kiszámolja a kijelzendő színt az átadott frekvencia és arány (ratio) alapján (`void audioSetColor(int freq, float rat)`). Ehhez részekre osztja a frekvencia tartományt, és a doCycle metódushoz hasonló módon határozza meg a színt, csak itt a frekvenciát véve alapul, nem az időt. Ezt még halványítja az átadott arány alapján, ami a hangerőből számítódik.

Az algoritmus, ha túl alacsony a frekvencia a maximális beállítottat adja vissza, ezért kell az if szerkezet elejére a plusz feltétel.

### 6.3.7 getVol

A paraméterben átadott indexű bufferre négyzetes közepet számol ki (`int getVol(int buffIndex)`). Ez megfelelő aránynak bizonyult a hangerőfüggés implementálásához.

### 6.3.8 doAudio

Ez a függvény csak akkor végez érdemi feladatot, ha megtelt az egyik input buffer. Ekkor átállítja a használatban lévő buffer indexét, elmenti, hogy melyikben vannak az adatok és nullázza az elem számlálót.

Ezután kiszámolja az adott minta négyzetes középértékét. Ha a hangerő nem ér el egy megfelelő szintet 0-nak tekintjük, és nem vesszük figyelembe, így magas marad az átlag hangerő, és látványosabb lesz a színjáték. Ha elég magas volt a hangerő „beleátlagolja” az eddigi eredménybe. Számtani közép helyett az eddigi eredmény és a mostani érték átlagát veszi, így elérhető, hogy a régebbi adatok ne befolyásolják a jelenlegi vizualizációt.

Ezután átállítjuk az eddigi maximum hangerőt, ha a mostani érték meghaladja, valamint elmentjük, hogy mikor volt az utolsó ilyen eset. Erre azért van szükség, mert ha egy bizonyos ideig nincs új max hangerő, a zene elhalkult és utána kell állítani a vizualizációt, hogy látványos maradjon.

Ezután kiugrásokat keresünk a jelben. Ehhez összehasonlítjuk az előző hangerővel a mostanit, és ha elég nagy a változás, akkor belevesszük a kiugrások közé (hasonlóan, mint a hangerő átlagolásánál).

Ezután kiszámoljuk a frekvenciát, és ha a hangerő nagyobb, mint az átlagos kiugrások, akkor kijelezzük (ez jelenti azt, hogy ütés volt a zenében), egyébként pedig halványítjuk a már kijelzett színt.

A függvény végén frissítjük az előző hangerőt és ha szükséges lejjebb vesszük a max hangerőt.

# 7 Felhasználói leírás és kezelési útmutató

Az eszköz képes a beépített színjátékok lejátszására, valamint a bemenetére csatlakoztatott 3,5mm-es audio jack bemeneten, vagy a mikrofonjáról érkező jel függvényében a LED szalagot zenére villogtatni.

Az eszközt a táphoz csatlakoztatva automatikusan elindul. A 3 szín felvillanását követően visszavált abba az állapotba, ahol hagytuk.

A kezelőfelületen 2 gomb található. A bal oldalit megnyomva válthatunk a beépített színjátékok között. A jobb oldali gomb az audio jack és a mikrofon között tudunk váltani, hogy melyik alapján történjen a villogtatás. A megfelelő bemenet mellett világító LED jelzi az aktív bemenetet.

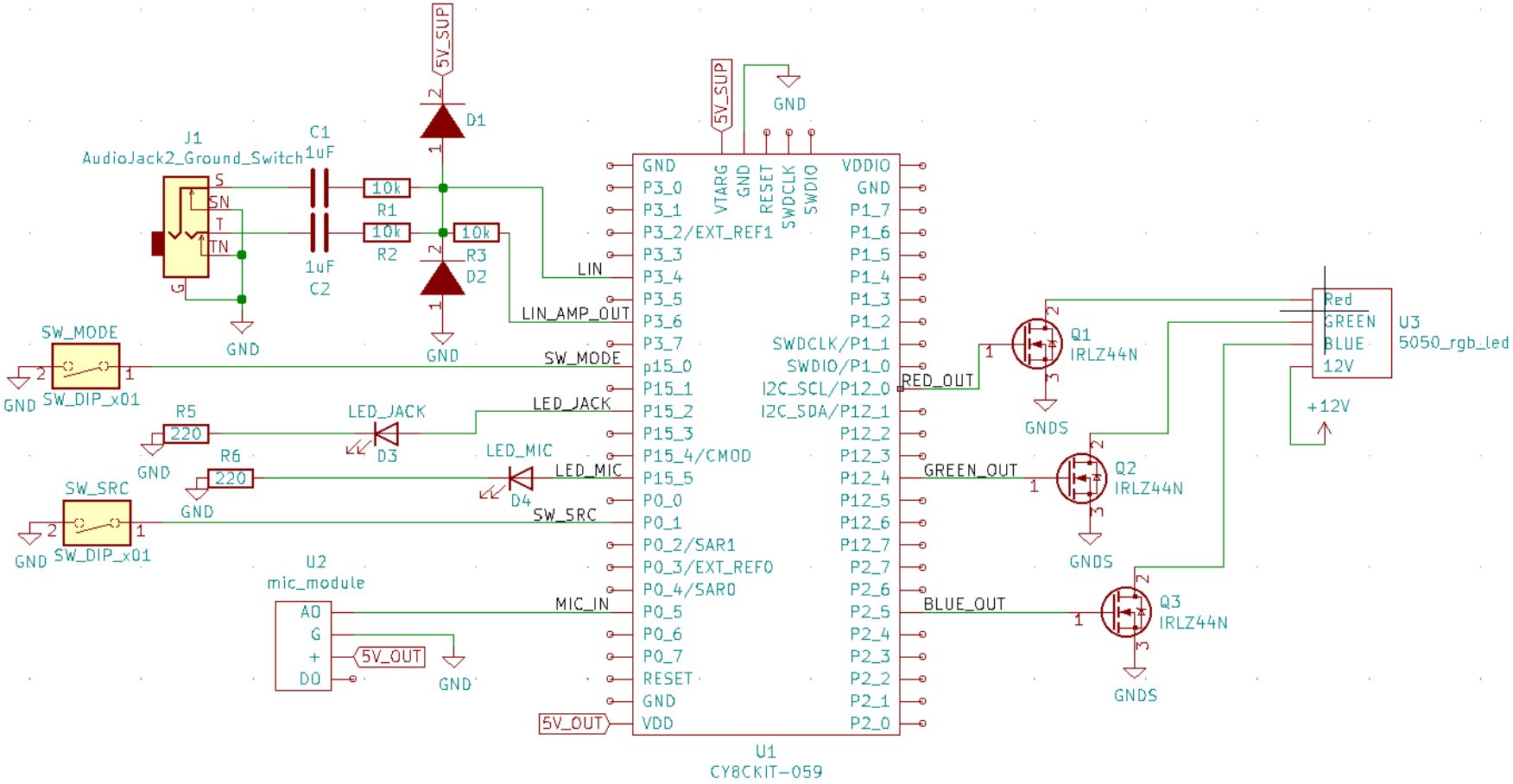
Az eszközhöz soros porton csatlakozva az alábbi parancsok adhatók ki:

| **Begépelt parancs** | **Akció** |
|---|---|
| T | Terminálon keresztüli vezérlés bekapcsolása. Engedélyezi parancsok begépelését és letiltja a gombokat |
| E | Terminálon keresztüli vezérlés kikapcsolása. Engedélyezi gombokat és letiltja a parancsok begépelését |
| C | A szín átmenetes színjátékra vált |
| R | A piros-zöld-kék színt villogtató színjátékra vált |
| P | A piros-kék színt villogtató színjátékra vált |
| A | Az audio bemenet alapján történő villogtatásra vált |
| N | A következő fényjátékra vált |
| J | Az audio jack-et állítja be aktív bemenetként |
| M | A mikrofont állítja be aktív bemenetként |
| S | Átváltja az aktív bemenetet |

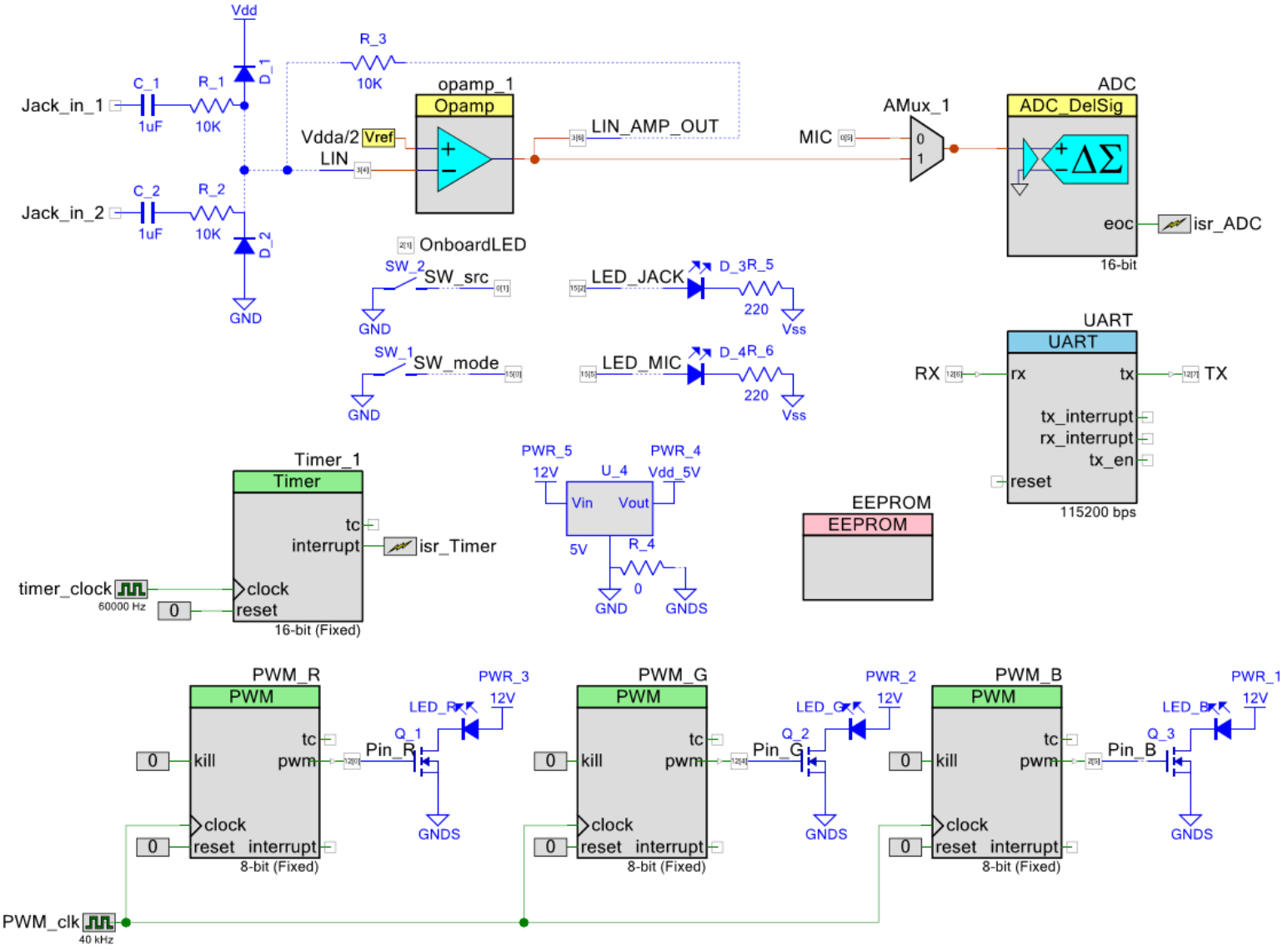
# 8 Mellékletek

## *8.1 Kapcsolási rajzok*

### 8.1.1 Külső kapcsolás



### 8.1.2 PSoC-n megvaolósított belső kapcsolás



## *8.2 Alkatrészjegyzék*

| Megnevezés | Mennyiség |
|---|---|
| 5050 LED szalag | 3m |
| Cypress CY8CKIT-059 PSoC fejlesztőkártya | 1 |
| Dióda | 2 |
| Ellenállás, 10k | 3 |
| Ellenállás, 220 | 2 |
| IRLZ44N FET | 3 |
| Jack anya aljzat | 1 |
| Kondenzártor, 1 | 2 |
| LED, narancssárga | 1 |
| LED, zöld | 1 |
| Mikrofon modul | 1 |
| Nyomógomb | 2 |

## *8.3 Program forrásainak listája*

### 8.3.1 Fejléc fájlok

- cyapicallbacks.h
- ledlight.h

### 8.3.2 Forrás fájlok

- ledlight.c
- main.c