**Szakdolgozat  
  
  
Teljesítményszabályozott elektromos cigaretta tervezése**

**Szoftveres oldal**

**Juhász Tamás**MGMREKVillamosmérnök BSc hallgató

Tervezésvezező:

**Szabó Norbert**

Mesteroktató

Elektrotechnikai és Elektronikai Intézet

Konzulens:   
**Dr. Tóth Lajos**

Elektrotechnikai és Elektronikai Intézet

Miskolc, 2019

[1 Bevezetés 4](#_Toc23776231)

[2 Az elektromos cigaretta 5](#_Toc23776232)

[2.1 Az elektromos cigaretta története 5](#_Toc23776233)

[2.2 Az elektromos cigaretta élettani hatásai 6](#_Toc23776234)

[2.3 A típusválasztás oka 6](#_Toc23776235)

[3 A NyÁK feltérképezése 7](#_Toc23776236)

[3.1 Vizuális vizsgálat 7](#_Toc23776237)

[3.2 Csomópontok visszarajzolása 8](#_Toc23776238)

[3.3 Kapcsolási rajz elkészítése OrCAD Capture-be 9](#_Toc23776239)

[4 Funkciók vizsgálata 13](#_Toc23776240)

[4.1 Buck/boost konverter és MOSFET vezérlők 13](#_Toc23776241)

[4.1.1 Buck/boost konverter 13](#_Toc23776242)

[4.1.2 ADP3110 MOSFET vezérlők 15](#_Toc23776243)

[4.2 INA213 árammérő erősítő 16](#_Toc23776244)

[4.3 Tápfeszültség 17](#_Toc23776245)

[4.3.1 MCP1711 feszültségstabilizátor 18](#_Toc23776246)

[4.3.2 MT3608 feszültségnövelő 19](#_Toc23776247)

[4.4 Mikrovezérlő 20](#_Toc23776248)

[4.4.1 Lábkiosztás [13]: 21](#_Toc23776249)

[4.5 Organic Light Emitting Diode – OLED 24](#_Toc23776250)

[4.6 Universal Serial Bus – USB 25](#_Toc23776251)

[5 Az ESP-IDF és a FreeRTOS 27](#_Toc23776252)

[5.1 Az ESP-IDF 27](#_Toc23776253)

[5.2 Az RTOS és a FreeRTOS 28](#_Toc23776254)

[6 A projekt áttekintése 28](#_Toc23776255)

[6.1 A hardver áttekintése 28](#_Toc23776256)

[6.2 A szoftver áttekintése 29](#_Toc23776257)

[7 A szoftver részletes ismertetése 30](#_Toc23776258)

[7.1 A GPIO lábak inicializálása és definiálása 30](#_Toc23776259)

[7.2 Az ADC lábak karakterizálása 33](#_Toc23776260)

[7.3 Bekapcsolási és kikapcsolási szekvencia 36](#_Toc23776261)

[7.3.1 A bekapcsolási szekvencia 36](#_Toc23776262)

[7.3.2 A kikapcsolási szekvencia 42](#_Toc23776263)

[7.4 Beállított teljesítmény visszajelzése, plusz és mínusz gombok 44](#_Toc23776264)

[7.4.1 A plusz és mínusz gombok működése 44](#_Toc23776265)

[7.4.2 A beállított teljesítmény visszajelzése 45](#_Toc23776266)

[7.5 A tűzgomb megnyomása utáni szekvencia, a teljesítményszabályozás 50](#_Toc23776267)

[7.5.1 A telj\_szabalyozas(); függvény leírása 51](#_Toc23776268)

[7.5.2 A függvény futtatása megszakítás hatására 54](#_Toc23776269)

[7.6 Túlhevülés védelem, USB-töltés lekezelése, engedélyező jelek kiadása, mélymerülés elleni védelem 56](#_Toc23776270)

[7.6.1 Engedélyező jelek bemutatása: 56](#_Toc23776271)

[7.6.2 enable\_outputs(); függvény leírása 56](#_Toc23776272)

[7.6.3 Tűzgombról érkező megszakítások letiltása és engedélyezése 59](#_Toc23776273)

[7.6.4 A Chargestate bemenet kiértékelése 59](#_Toc23776274)

[7.7 Az *app\_main();* függvény leírása 60](#_Toc23776275)

[8 A szoftver tesztelése 62](#_Toc23776276)

[8.1 A be-és kikapcsolás tesztelése 62](#_Toc23776277)

[8.2 A beállított teljesítmény visszajelzésének a tesztelése 63](#_Toc23776278)

[8.3 A teljesítményszabályozás tesztelése 64](#_Toc23776279)

[8.4 Az engedélyező kimenetek tesztelése, és a töltés állapot kijelzése 65](#_Toc23776280)

[9 Összefoglalás 67](#_Toc23776281)

[10 Irodalmi hivatkozások 68](#_Toc23776282)

# Bevezetés

A szakdolgozatom témája egy teljesítményszabályozott elektromos cigaretta mikrovezérlős programjának megírása, és egyedi funkciókkal való ellátása. A szakdolgozat első részében az előző féléves Komplex tervezés feladat látható, melyet csoporttársammal, Luzsinszky Bencével készítettünk el közösen. Erre azért volt szükség, ugyanis ez a háttértudás szükséges ahhoz, hogy megérthessem, hogy valójában hogyan működik egy ilyen eszköz. A továbbiakban is lesznek hivatkozások Luzsinszky Bence szakdolgozatára, ugyanis, az ő témája az, hogy ugyanezen eszköznek a hardverét megtervezze. Mivel egy a beágyazott rendszer tervezésrkor a hardver és a szoftver tervezés sem lehetséges anélkül, hogy értenénk és ismernénk a másik oldalt is.

Az alkatrészek beazonosításának első lépéseként szemmel vizsgáltuk a NyÁK-ot, illetve a kicsi áramköri elemeket és felirataikat egy digitális mikroszkóp segítségével értelmeztük. Az IC-k pontos beazonosításához egy labortápot és egy multimétert használtunk. A vizsgálatok eredményével már megkereshettük azok adatlapjait.

Miután beazonosítottuk az áramköri elemeket, azok egymással való összeköttetéseit vizsgáltuk. Ezt egy multiméter szakadásvizsgáló beállításával végeztük, amihez szükség volt a multiméter által rövidzárnak mért tekercsek, biztosítékok és söntellenállás kiforrasztására. A visszarajzoláshoz az azonosított IC-k adatlapjai alapján határozzuk meg azok lábkiosztásait.

A csomópontok feltérképezését először papíron tettük, majd az OrCAD Capture szoftver segítségével folytattuk a könnyebb átláthatóság érdekében.

A visszarajzolás közben megfigyelhettük, hogy az egyes áramköri elemek hogyan kapcsolódnak egymáshoz, valamint a mikroprocesszorhoz, így visszakövetkeztethettük azok pontos feladatait és a szoftver működésének alapelveit is.

A szoftver tervezésénél ezeket az alapelveket vettem figyelembe, és néhány extra funkcióval ellátva készítettem el a végleges programot. Ehhez egy rendelkezésemre álló ESP-32 típusú mikrovezérlőt választottam, és a szoftver fejlesztéséhez, bár Arduino IDE-ben is megvalósítható lett volna, én inkább az ESP-IDF keretrendszert választottam, ugyanis az egy valós idejű operációs rendszeren, a FreeRTOS-en, alapul, amelynek a programozását még sosem próbáltam és felkeltette az érdeklődésemet, így új tapasztalatokra tehettem szert.

# Az elektromos cigaretta

Az elektromos cigaretta egy akkumulátor által működtetett eszköz, mely egy fűtőszál hevítésével, belélegzésre alkalmas gőzzé párologtatja a benne lévő nikotinos, vagy nikotinmentes folyadékot. Fő funkciója a hagyományos cigaretta pótlása volt az, elpárologtatott folyadék nikotintartalma által.

Az elektromos cigaretták alapvetően két részből tevődnek össze: egy *mod*ból, mely az akkumulátort és az elektronikát tartalmazza, és egy *tank*ból, amely a folyadékot tárolja, valamint benne található a fűtőszál, ami a folyadékot hevíti és párologtatja.

## Az elektromos cigaretta története

**Irfan Shiekh**

A feljegyzések szerint a 16. századi Indiában egy Irfan Shiekh nevű doktor terjesztette el a Mogul irodalomban a vízipipát [1]. A vízipipát szintén az elpárologtatott nikotinos gőz elszívására alkalmazták, ezért nyugodtan nevezhetjük az elektromos cigaretta ősapjának.

**Joseph Robinson**

1927 május 3-án Joseph Robinson beadta a szabadalmi kérelmét egy ’Mechanikus bután égetésű párologtató” nevű eszközre. A készülék egy gyógyászati célú készítményt tartalmazott és párologtatott el. A szabadalmat [2] 1930-ban jóváhagyták, de Robinson sosem vitte a piacra a termékét

**Herbert A. Gilbert**

Az 1960-as években egy Herbert A. Gilbert nevű veterán keresett megoldást az égés mentes cigarettázásra. Egy fűtőszálat hevített egy akkumulátor segítségével [3], tehát az alapelgondolása megegyezett a ma is használt elektromos cigarettákkal. A megoldása az akkoriban elérhető technológiának köszönhetően igen egyszerű volt a mai elektromos cigarettákhoz képest. A találmánya sosem lett népszerű, ő sem tudta a piacra juttatni.

**Hon Lik**

Hon Lik egy kémikus az Észak-Kelet Kínai Liaoning tartományból. Elmondása szerint láncdohányos volt, de amikor az édesapja tüdőrákban meghalt, úgy döntött, hogy leszokik, de nem a hagyományos úton. Feltalált egy eszközt [4], mely egy cső volt egy lítium akkumulátorral és egy porlasztóval, mely egyben nikotinos folyadékot tárolt és porlasztott. A találmánya Kínában 2004-ben, Európában 2006-ban és az Egyesült Államokban 2007-ben került piacra.

Az elektromos cigaretták Hon Lik találmánya óta nagyon sokat fejlődtek, ám működési elvük a mai napig ugyanaz.

## Az elektromos cigaretta élettani hatásai

Az elektromos cigaretták eredetileg a hagyományos dohányzás egészségesebb alternatíváját akarták jelenteni. Mivel a gőzüknek alapvetően finom (gyümölcsös, mentolos, dohányos) íze van, valamint nikotintartalmuk miatt nyugodt érzetett biztosítanak, ezért a dohányzásról leszokók helyett a legnagyobb népszerűsége a tinédzserek körében van. Ezen okból következően komoly szabályzásokat tettek ellenne, Magyarországon a hagyományos cigarettákra vonatkozó jogszabályok vonatkoznak az elektromos cigarettákra is.

Az élettani hatásairól egyelőre keveset tudni, körülbelül tíz éve lettek elterjedtek, ez az idő nem elegendő a hosszútávú hatásaikról szóló tanulmányokhoz. Ennek ellenére a nagy népszerűségüknek (főleg a tinédzserek között) köszönhetően igen nagy erőforrásokat fektetnek a rövid- és hosszútávú élettani hatásaik kutatásába.

**Nikotin**

A nikotinnak nagyon sok élettani hatása van, mi csak a függőséggel foglalkozunk. A nikotin hatására, többek között dopamin szabadul fel az agyban, amelyet az agyunk örömélményként él meg, megerősítő hatást fejt ki, mely könnyen függőséghez vezethet [5].

**Propilén-Glikol**

A propilén-glikol az oldószer, melyben a nikotint és az aromákat oldják. A növényi glicerin után ez teszi ki a folyadék legnagyobb részét. A használók körülbelül 2%-ában allergiás reakciót okozhat.

**Aromák**

A gőz ízét adó aromák egyéb adalékanyagokat is tartalmazhatnak, melyek élettani hatásairól csak nagyon kevés információ áll rendelkezésre.

## A típusválasztás oka

A dolgozatunkhoz egy Movkin márkájú, Disguiser típusú modot választottunk. A választásban nagy szerepet játszott az ára és az elektronikájának a tudása.

****

*1. Ábra - Movkin Disguiser [6]*

# A NyÁK feltérképezése

A dolgozatunk legelső lépése a mod szétszerelése volt. Miután eltávolítottuk a rögzítőcsavarokat és leszereltük a burkolatot, közvetlen hozzáférést kaptunk a NyÁK-hoz.

A NyÁK-ról oldalcsípővel eltávolítottuk a csatlakozókat és a későbbi fotózás érdekében az OLED kijelzőt.

## Vizuális vizsgálat

Az alkatrészek beazonosításához először szemmel és nagyítóval vizsgáltuk a NyÁK-ot, így megismertük a legtöbb áramköri elem típusát. A maradék elemeken a feliratok nem voltak olvashatóak szabad szemmel, ezért egy digitális mikroszkópot használtunk segítségül. Az egyes IC-k jelölései igen nehezen olvashatónak bizonyultak, esetleg több, egymástól eltérő IC is azonosítható lett volna általuk. Emiatt őket a későbbiekben egy labortáppal feszültség alá helyeztük és egy multiméterrel figyeltük a viselkedésüket, majd ez alapján következtettük vissza a pontos típusukat, feladataikat.

A beazonosított alkatrészeket először papírra rajzoltuk, majd későbbiekben a könnyebb visszaolvashatóság érdekében lefotóztuk a NyÁK-ot és Adobe PhotoShop szoftver segítségével feliratoztuk az alkatrészeket.



*2. Ábra - A NyÁK, rajta az alkatrészek azonosítói*

Hamar egyértelművé vált, hogy az első oldalon a feszültség csökkentő-növelő, buck/boost kapcsolás található, amely a kimeneti feszültséget állítja elő, a második oldalon pedig a mikroprocesszor és az egyéb funkciókat kiszolgáló áramköri elemek láthatóak.

## Csomópontok visszarajzolása

Második lépésként az áramköri elemek egymással való összeköttetéseit vizsgáltuk. Ezt egy multiméter szakadásvizsgáló beállításával végeztük. Ehhez először szükség volt a multiméter által rövidzárnak mért tekercsek és söntellenállás kiforrasztásához.

A visszarajzoláshoz az IC-k azonosítására volt szükség. A mikroszkóp segítségével csak néhányat lehetett beazonosítani, ugyanis a rajtuk lévő feliratok gyakran olvashatatlanok, vagy félrevezetőek voltak. Mivel a jelöléseikből nem tudtuk beazonosítani a JJ, az AM, az A1T és a BCNJ feliratú IC-ket, ezért a kapcsolási rajzon betöltött szerepük alapján következtettünk vissza azok funkciójára, majd multiméterrel kimértük a lábaikat és a sejtéseink beigazolódtak.

A multiméter dióda vizsgáló funkciójával rájöttünk, hogy a JJ két közös katódos dióda, az AM egy földelt-emitteres bipoláris tranzisztor, az A1T pedig egy növekményes P típusú MOSFET.

A BCNJ feliratú IC az F30 biztosítékon keresztül az akkumulátorra van kötve, a biztosítékot a pontos mérések érdekében kiforrasztottuk, forrszemeit összeforrasztottuk, így a BCNJ közvetlenül az akkumulátorra volt kötve. Vezetékeket forrasztottunk a NyÁK B+ és B- forrszemeire és egy labortáp segítségével az akkumulátort szimuláló 3,5-4,2V feszültség alá helyeztük a kapcsolást. Ez után megfigyelhettük, hogy a BCNJ IC kimenetén 3V feszültség esik a GND1 földpotenciálhoz képest, tehát ő egy 3V kimeneti feszültségű feszültségstabilizáló. Későbbiekben a lábai bekötését megvizsgálva arra jutottunk, hogy a BCNJ megegyezik az MCP1711 nevű, ugyanilyen lábkiosztású, megegyező paraméterekkel rendelkező integrált áramkörrel, ezért annak az adatlapját használtuk a továbbiakban.

A kapcsolás központi egysége egy GigaDevice F130G8 típusú, 32 bites ARM Cortex-M3 alapú, 28 lábú mikrovezérlő. A lábai vizsgálatakor a multiméter mérőhegyeivel többször is összefésültük a rajtuk lévő forraszanyagot, amit csak mikroszkóppal láthattunk. Ebből kifolyólag többször fel kellett egy forrasztópákával hevíteni az összefésült forraszanyagot és ezáltal megszüntetni a rövidzárat.

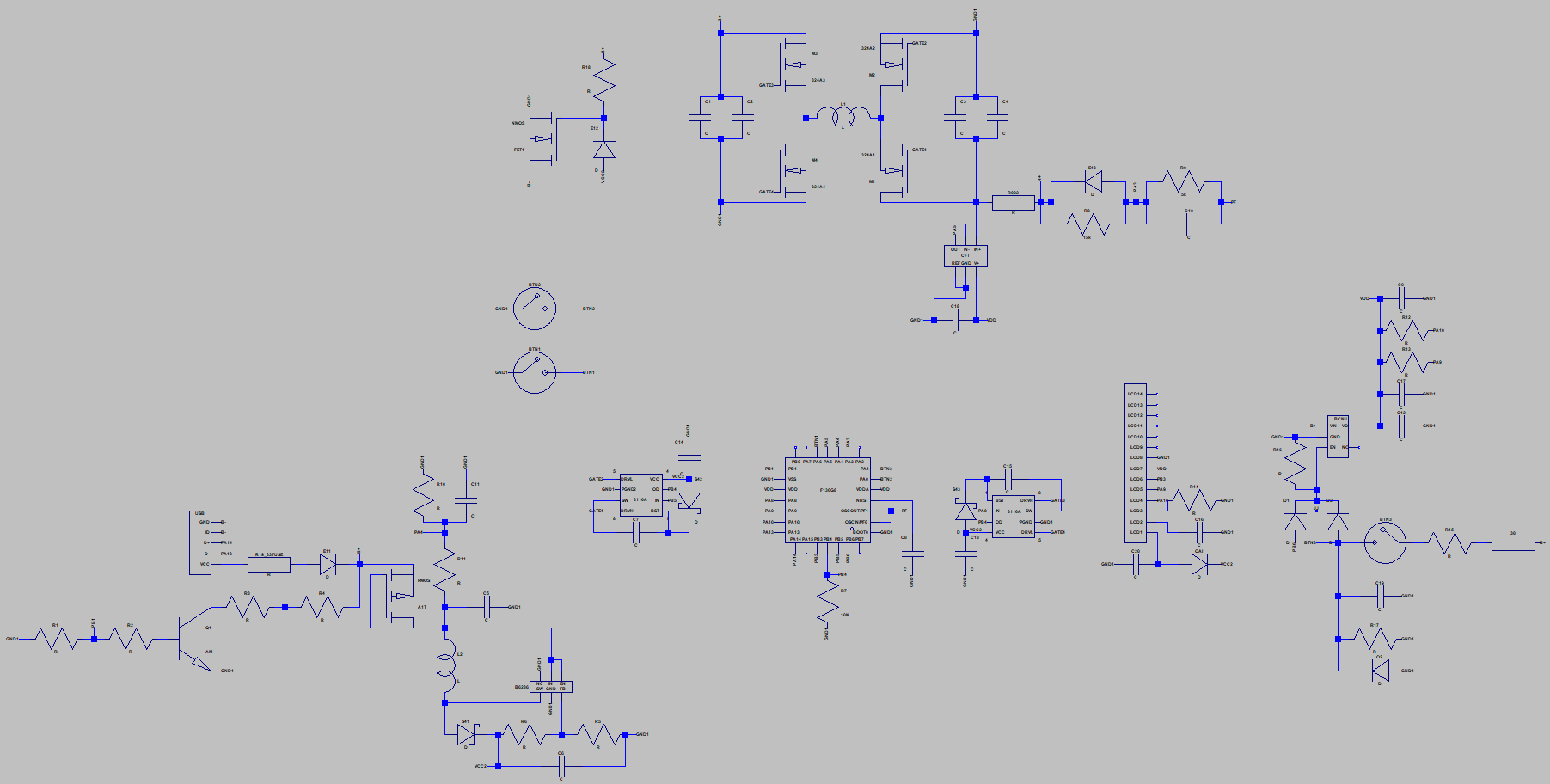
A multiméteres vizsgálat közben megállapítottuk, hogy a NyÁK hatrétegű, mivel a közbenső rétegeken adatvezetékek is futnak, valamint FR4 anyagú.

Először papíron kezdtük visszarajzolni a kapcsolást, a multiméter egyik mérőhegyét az egyes elemek egyik lábára téve végigmértük az összes többi forrszemmel, vizsgálva, hogy van-e köztük közvetlen összeköttetés. Ha közvetlen kapcsolatot találtunk két elem között, akkor papírra vetettük.

## Kapcsolási rajz elkészítése OrCAD Capture-be

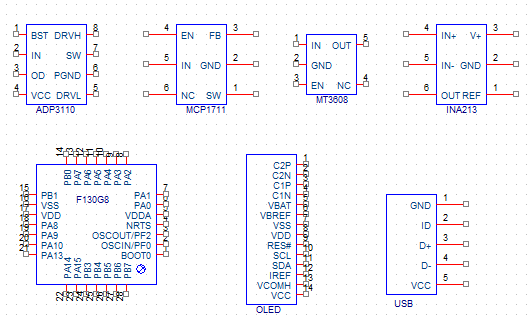
A sokadik A/4-es lap is rövidesen betelt, követhetetlenné vált, túl sok volt rajta a hibázási lehetőség, ezért úgy döntöttünk, hogy számítógépen folytatjuk.

Először az LTSpice nevű ingyenes kapcsolási rajz szerkesztővel próbálkoztunk. Megrajzoltuk a saját szimbólumainkat, összeállítottuk a kapcsolást, de a végeredmény az LTSpice egyszerűségének köszönhetően éppen középszerűre sikeredett.

  
*3. Ábra - A kapcsolás LTSpice-ban*

Végül az OrCAD Capture szoftvert használtuk, mely használatáról már voltak előzetes tapasztalataink az egyetemi tanulmányaink által.

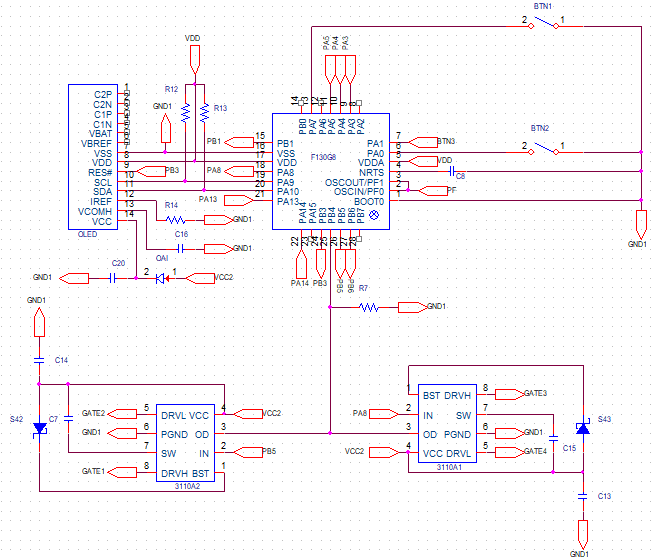
Első lépésként az integrált áramköröknek rajzoltunk szimbólumokat. Az azonosított IC-k adatlapjai alapján határoztuk meg azok lábkiosztásait.

****

*4. Ábra - Az áramköri elemek általunk rajzolt szimbólumai*

Az OrCAD Capture-ben összeállított kapcsolás 3 oldalt tesz ki.

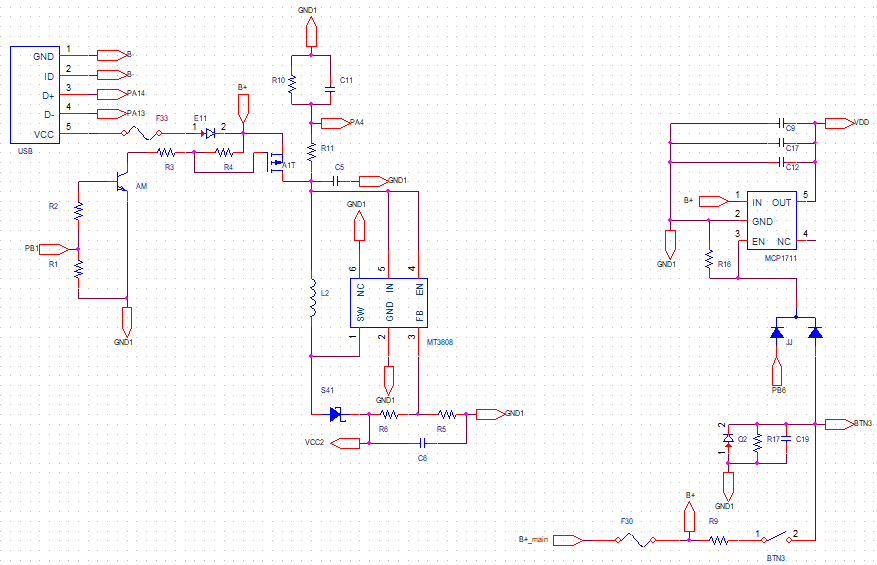
Az első oldalon a mikrovezérlő, a MOSFET vezérlők, az OLED és a le-fel funkciót szolgáló két gomb található.

*5. Ábra - Bal fent az OLED, középen a F130G8 mikrovezérlő, lent a két ADP3110 MOSFET vezérlő, valamint jobb oldalon a két gomb*

A mikrovezérlő központi szerepet lát el a kapcsolásban, többek között ő szolgáltatja az engedélyező jelet a többi integrált áramkör számára, illetve ő állítja elő a MOSFET vezérlők PWM jelét, adatot küld és fogad az USB-ről, adatot és órajelet küld az OLED-re I2C buszokon keresztül és feldolgozza a felhasználó inputjait is (gombok formájában). Ezeken kívül ő végzi el az áramkör helyes működéséhez szükséges valós idejű méréseket és számításokat, például a PA3 lába a kimeneti feszültséget méri a beépített analóg-digitális átalakítója segítségével.

A MOSFET vezérlők nyitják és zárják a buck/boost konverter MOSFET-jeit, ezzel előállítva a szükséges kimeneti feszültséget.

A második oldalon a tápfeszültségek előállításához szükséges kapcsolások, illetve az USB csatlakozó szerepelnek.



*6. Ábra - Bal oldalon az USB csatlakozó, majd a 12V-ot előállító MT3608 feszültségnövelő, jobb oldalon a 3V-ot előállító MCP1711 feszültségstabilizátor*

Az USB az akkumulátor töltésére nem alkalmas, csak a mikrovezérlő firmware-ének a frissítésére szolgál.

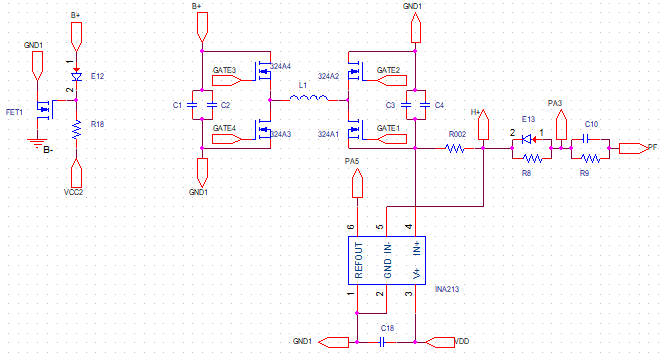
A TLV757P egy feszültségnövelő IC, ő állítja elő a VCC2 portra a 12V feszültséget.

A MCP1711 egy feszültségstabilizátor IC, ami a VDD portra állít elő 3V feszültséget.

Az AM bipoláris tranzisztort a mikrovezérlő nyitja, amennyiben nyitva van a B+ akkumulátorfeszültség port és az USB VCC lába az R4 és R3 ellenállásokon keresztül a földre van húzva, az A1T növekményes P csatornás MOSTFET nyit, és a TLV757P feszültségnövelő megkapja a tápfeszültségét. Amennyiben a mikrovezérlő a PB1 lábán nem ad ki logikai „1”-et, az AM tranzisztor zárva van, a B+ és az USB VCC lába zárják A1T MOSFET-et és a TLV757P nem kap tápellátást.

A JJ közös katódos diódák felelősek azért, hogy a BTN3 gomb és a PB6 mikrovezérlő kimenet jelei ne keveredjenek. A JJ vezeti a PB6 jelet és a BTN3 gomb jelét az MCP1711 IC engedélyezőjére. Amennyiben a BTN3 gomb egyszer megnyomódik, az MCP1711 működésbe lép, a mikrovezérlő kap tápellátást és a PB6 lábán keresztül engedélyezőjelet ad az MCP1711 IC-nek.

A harmadik oldalon a közös földpotenciálra vezető MOSFET, a buck/boost konverter, a kimeneti áramot figyelő és a kimenet található.



*7. Ábra - Bal oldalon a közös földpotenciálra vezető FET1 MOSFET, jobb oldalon a buck/boost konverter, a H+ kimenet és a kimeneti áramot figyelő INA213 IC*

A FET1 N csatornás teljesítmény MOSFET source-ja a GND1 portra van kötve, a drain-je pedig az akkumulátor negatív lábára (B-). Amennyiben a FET1 MOSFET kap elegendő tápefeszültséget a B+ pozitív akkumulátorlábról, a MOSFET nyit, így szolgáltatja a GND1 közös földpotenciált a teljes kapcsolásnak. Ez véd a fordított polaritással beüzemelt, vagy túl alacsony feszültségű (alacsony töltöttségű) akkumulátoroktól.

A buck/boost konverter állítja elő a kimeneti feszültséget a H+ portra.

Az INA213 árammérő IC az R002 söntelleálláson méri a kimenő áram nagyságát.

A PA3 port a mikrovezérlő PA3 ADC bemenetére van kötve, alapvetően az R8 ellenálláson eső feszültséget méri.

A PF port a mikrovezérlő PF0 és PF1 bemeneteire van kötve, amiket a mikrovezérlő földpotenciálra húz. Ez szolgál méréshatárváltónak a PA3-hoz, ugyanis amennyiben a PF bemenetek a földpotenciálra vannak kötve, a PA3 nem csak az R8 ellenálláson, hanem az R8 és R9 ellenállások által leosztott feszültséget méri vissza. Azért van szükség a mikrovezérlő két külön lábára, hogy nagyobb áramterhelést is kibírjanak.

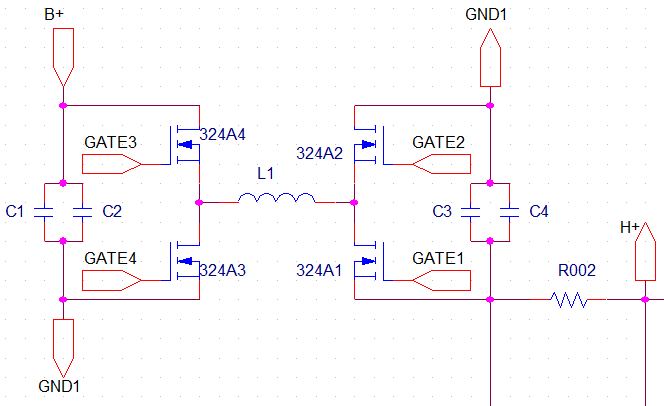
# Funkciók vizsgálata

A visszarajzolás közben megfigyeltük, hogy az egyes áramköri elemek hogyan kapcsolódnak egymáshoz. A kész kapcsolási rajzból és az áramköri elemek adatlapjaiból azok pontosabb funkcióit is megérthettük. Továbbá az áramköri elemek a mikroprocesszorhoz való csatlakozásait is nyomon követhettük, így visszakövetkeztethettük a szoftver működésének alapelveit is.

## Buck/boost konverter és MOSFET vezérlők

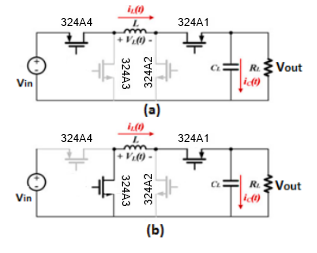
A MOSFET vezérlők nyitják és zárják a buck/boost konverter MOSFET-jeit, amik ezáltal előállítják a kimeneti feszültséget.

### Buck/boost konverter

**

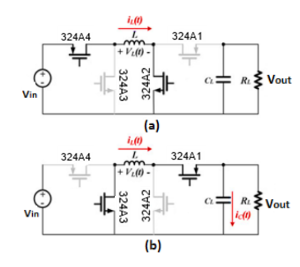
*8. Ábra - A buck/boost kapcsolás*

Buck (feszültségcsökkentő) üzemmódban feltöltéskor a 324A4 és a 324A1 MOSFET-ek nyitva vannak, a másik kettő zárva, áram folyik az L1 tekercsen a H+ kimenet felé. A sorba kötött L1 tekercs a magas feszültséget lecsökkenti a kívánt értékre. A kisülés alatt a 324A1 és a 324A3 MOSFET-ek nyitva vannak, a másik kettő zárva, az L1 tekercs ellenáll az áram változásnak és fordított feszültséget állít elő. A H+ kimenetre nem kerül tápfeszültség, csak az L1 tekercs által indukált feszültség [7].



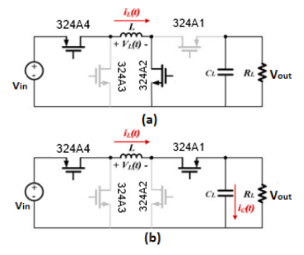
*9. Ábra – Négykapcsolós buck/boost konverter buck üzemmódban, feltöltés (a) és kisülés (b) közben [7]*

Buck/boost módban feltöltéskor 324A4 és 324A2 MOSFET-ek nyitva vannak, a másik kettő zárva Áram folyik az L1 tekercsen, egyidőben a kondenzátor feszültsége közvetlenül megy a H+ kimenetre. Kisüléskor a 324A1 és 324A3 MOSTFET-ek nyitva vannak, a másik kettő zárva. Az L1 által indukált feszültség megy a H+ kimenetre [7].



*10. Ábra – buck/boost konverter buck/boost üzemmódban, feltöltés (a) és kisülés (b) közben [7]*

Boost (feszültségnövelő) üzemmódban a 324A1 és 324A2 MOSFET-ek nyitva vannak, a másik kettő zárva. Áram folyik az L1 tekercsen keresztül a kimenetre, egyidőben a kondenzátor feszültsége is közvetlenül a kimenetre van kötve. Kisüléskor a 324A1 és 324A4 MOSFET-ek nyitva vannak, a másik kettő zárva, a bemeneti feszültség sorba kötve az L1 tekerccsel a megnövekedett kimeneti feszültséget eredményezi [7].



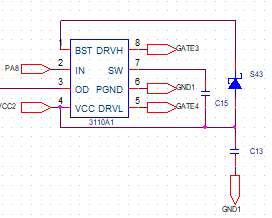
*11. Ábra – Négykapcsolós buck/boost konverter boost üzemmódban, feltöltés (a) és kisülés (b) közben [7]*

### ADP3110 MOSFET vezérlők

A 3110A jelölésű IC azonos a vele egyező paraméterekkel rendelkező ON Semiconductor által gyártott ADP3110 típusú MOSFET vezérlővel, ezért annak az adatlapját használtuk a vizsgálatokhoz.

Az ADP3110 egy dupla, nagyfeszültségű MOSFET vezérlő, két N csatornás MOSFET vezérlésére optimalizálva, amelyek a kapcsolók szinkron buck/boost átalakítóban. Esetünkben a 3110A1 MOSFET vezérlő a 324A3 és 324A4 azonosítójú MOSFET-eket, a 3110A2 MOSFET vezérlő a 324A1 és 324A2 MOSFET-eket vezérli [8].

Lábkiosztás [8]:



*11. Ábra – Az egyik ADP3110 MOSFET vezérlő*

A BST bootstrap láb a feszültségnövelő MOSFET-ek lebegő bootstrap tápja, a C7 és a C15 kondenzátoron keresztül az SW lábra kötve, a kondenzátorok tárolják a feszültségnövelő MOSFET-ek bootstrap feszültségét.

Az IN logikai PWM bemenetek a mikrovezérlő PB5 (3110A2) és PA8 (3110A1) digitális kimenet lábaira vannak kötve. Ez a láb szabályozza a vezérlő kimeneteit, amennyiben logikai „1”-et kap, a feszültségnövelő (DRVH) vezérlőt kapcsolja be, ha logikai „0”-t kap, a feszültségcsökkentő (DRVL) vezérlőt kapcsolja be.

Az OD kimeneti engedélyező láb mindkét MOSFET vezérlő esetében mikrovezérlő PB4 digitális kimenet lábára van kötve. Amennyiben logikai alacsony szintet kap, leállítja a DRVL és DRVH normális működését.

A VCC bemeneti tápellátás láb a VCC2 portra van kötve, amin az MT3608 feszültségnövelő által előállított 12V feszültség esik.

DRVL kimeneti vezérlő a feszültségcsökkentő MOSFET-eknek (324A2 és 324A3)

A PGND, teljesítményföld láb a GND1 közös földpotenciál portra van kötve.

Az SW láb megakadályozza a feszültségcsökkentő MOSFET-ek nyitását, amíg a feszültség nem esik 1V alá.

DRVH kimeneti vezérlő a feszültségnövelő MOSFET-eknek (324A1 és 324A4).

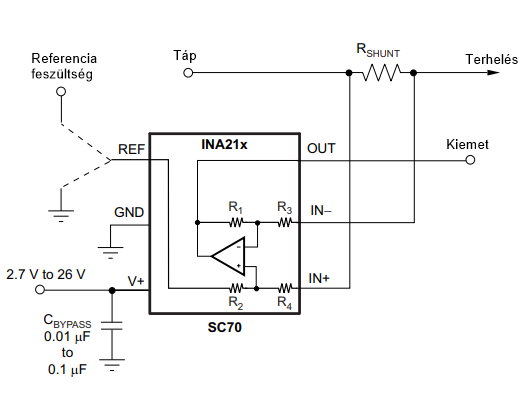
## INA213 árammérő erősítő

A CFT jelölésű IC azonos a vele egyező paraméterekkel rendelkező Texas Instruments által gyártott INA213 típusú árammérő erősítővel, ezért annak az adatlapját használtuk a vizsgálatokhoz.

Az INA213 egy feszültség kimenetű árammérő erősítő, amit gyakran használnak túláram védelemhez, precíziós áramméréshez rendszeroptimalizálásra, vagy zárt hurkú visszacsatolásos áramkörökhöz. Az erősítése 50V/V.

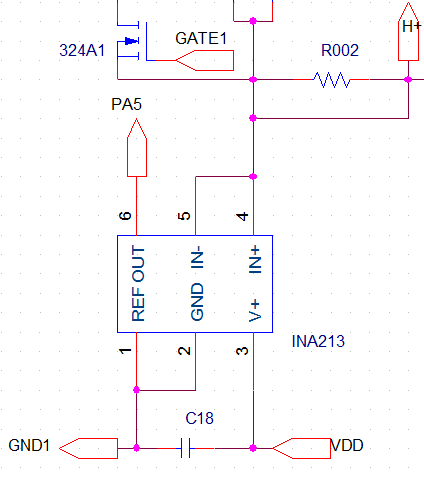
Az eszköz 2.7V – 26V tápfeszültségről üzemel, ezt esetünkben a MCP1711 low dropout feszültségstabilizátor által előállított 3V, valamint maximum 100µA tápáramot vesz fel [9].

Gyakorlatilag egy kivonó erősítő, a sönt H+ port felőli lábán eső kisebbik feszültséget vonja ki a sönt buck/boost konverter felőli lábán eső nagyobbik feszültségből. Az így kapott feszültségértéket küldi el a mikrovezérlőnek.



*12. Ábra – INA213 egyszerűsített kapcsolási rajza [9]*

Lábkiosztás [9]:

**

*13. Ábra – Az INA213 árammérő a kapcsolásban*

A GND és a REF lábai össze vannak kötve, a GND1 porton keresztül a közös földpotenciálra. A REF láb a V+ lábhoz képest referencia 0V potenciálon van.

Az IN- láb a söntellenállás kimeneti lábára van kötve.

Az IN+ láb a söntellenállás buck/boost felőli lábára van kötve.

Az NC láb a GND1 porton keresztül a közös földpotenciálra van kötve.

Az OUT kimeneti feszültség láb a mikrovezérlő PA5 ADC inputjára van kötve, az analóg feszültségértéket digitális jellé alakítja a mikrokontroller ADC-je, melyből vissza tudja olvasni a kimeneti áram értékét.

A V+ láb a VDD porton keresztül az MCP1711 feszültségstabilizátor által előállított 3V tápfeszültséget kapja.

## Tápfeszültség

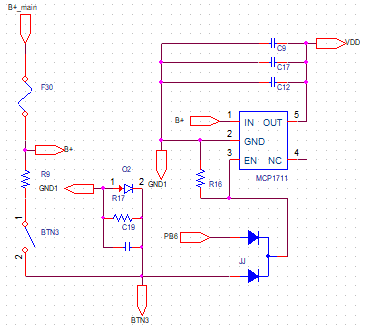
Az áramkör tápfeszültségét két egymással párhuzamosan kötött 18650-es cella szolgáltatja. Az akkumulátor pozitív lába a B+\_main porton keresztül van bekötve a kapcsolásba, az F30 biztosítékon keresztül jut el a B+ portba, ez a bemeneti tápellátása a feszültségnövelőnek, a feszültségstabilizátornak és a buck/boost konverternek.

Az IC-k tápfeszültségét két külön áramkör szolgáltatja, az MCP1711 low-dropout feszültségstabilizátor 3V-ot állít elő az akkumulátorról, a MT3608 feszültségnövelő 12V-ot állít elő az akkumulátorról.

### MCP1711 feszültségstabilizátor

A BCNJ jelzésű IC azonos a vele egyező paraméterekkel rendelkező, Microchip által gyártott MCP1711 low-dropout feszültségstabilizátorral (LDO). Ez az IC állítja elő a mikrokontroller, az OLED és az INA213 árammérő 3V tápfeszültségét (VDD port).

Az MCP1711 egy nagyon pontos, CMOS elven épült low-dropuout feszültségstabilizátor, ami 150mA áramot tud kiadni, miközben 0.6µA a nyugalmi fogyasztása. Az IC-ben beágyazott túláram és rövidzárvédelem található. A bemeneti feszültsége 1,4V-6V között változhat, ez esetünkben az akkumulátor 3,5-4,2V feszültsége. A kimeneti feszültsége 3±0,02V.



*14. Ábra – Az MCP1711feszültségstabilizátor IC a kapcsolásban*

Lábkiosztás [10]:

Az SHDN engedélyező lába a BTN3 jelzésű gombra és a mikrovezérlő PB6 lábára van rákötve, a JJ közös katódos diódákon keresztül. Amennyiben a BTN3 gomb lenyomódik az MCP1711 üzembe lép, a mikrovezérlő megkapja a 3V tápfeszültségét, az a PB6 lábon keresztül engedélyezteti a feszültségstabilizátort. Ha 0.38V-nál kevesebb feszültség esik rajta az leállítja a feszültségstabilizátort, ha több mint 0,91V esik rajta, a feszültségstabilizátor normális üzemmódba van kapcsolva. Amennyiben a JJ diódákról nem kap jelet, az R16 ellenálláson keresztül a földre van húzva.

A GND föld láb GND1 porton keresztül a közös földpotenciálon van.

A IN láb az F30 biztosítékon keresztül akkumulátor pozitív lábára van kötve. Ez a bemeneti tápellátása, amit 3V-ra stabilizál.

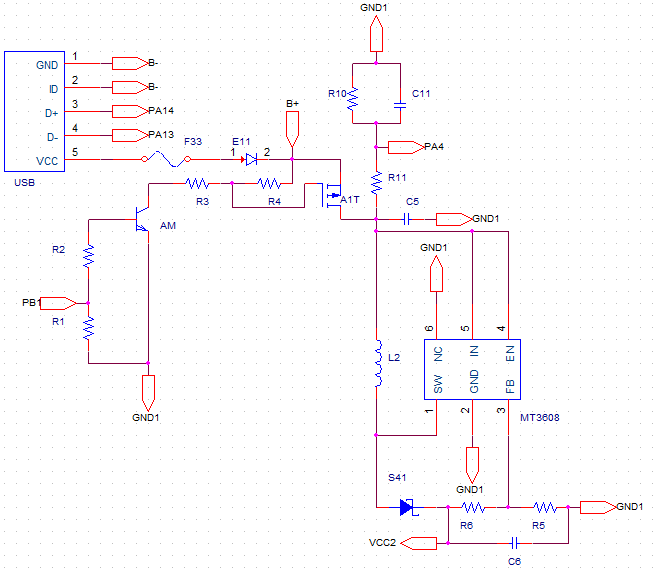
Az NC láb nincs bekötve, lebeg.

A OUT lábon a stabilizált 3V kimeneti feszültség esik a VDD tápfeszültség portra, a láb a C12 kondenzátoron keresztül a GND1, közös földpotenciál portra van kötve.

### MT3608 feszültségnövelő

A B6286 jelzésű IC azonos a vele megegyező paraméterekkel rendelkező Aerosemi által gyártott MT3608 nagy hatékonyságú feszültségnövelővel. Ez az IC állítja elő az ADP3110 MOSFET vezérlők és az OLED számára a 12V tápfeszültséget és ez a feszültség nyitja a FET1 MOSFET-et (VCC2 port).

AZ MT3608 nagy hatékonyságú feszültségnövelő 1.2MHz-en vált és lehetővé teszi a kis költségű, apró kondenzátorok és a 2mm-nél alacsonyabb tekercsek használatát. A belső intelligens indító miatt alacsony bekapcsolási áram szükséges, növelve ezzel az akkumulátor élettartamot.



*15. Ábra – Az MT3608 feszültségnövelő IC a kapcsolásban*

Lábkiosztás [11]:

Az SW láb teljesítménykapcsoló kimenet, az L2 tekercs egyik lábára és az FB láb feszültségosztójára van kötve, illetve az S41 Schottky diódán keresztül a VCC2 tápfeszültség portra juttatja az előállított 12V feszültséget.

A GND föld láb GND1 porton keresztül a közös földpotenciálra van kötve.

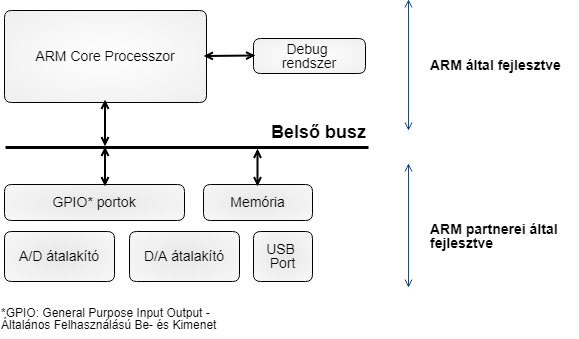
Az FB visszacsatolás láb, az R6 és R5 ellenállások közé van kötve, a feszültségosztó 0.6V-ot állít elő a lábnak.

Az EN engedélyező láb és az IN bemeneti tápellátás láb össze vannak kötve az A1T P típusú MOSFET drainjével és az L2 tekercs másik lábával. Amennyiben az A1T MOSFET zárva van, az MT3608 normális működése leáll.

Az NC láb a GND1 porton keresztül a közös földpotenciálra van kötve.

## Mikrovezérlő

A kapcsolás mikrovezérlője az ARM által fejlesztett Cortex-M3 alapú (ARMv7 architektúrán alapuló), GigaDevice által gyártott F130G8 típusszámú 32 bites általános felhasználású mikrovezérlő.



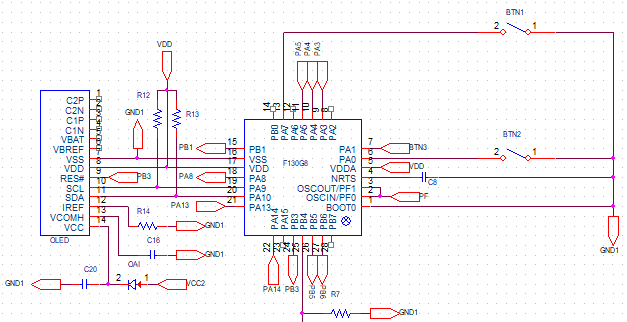
*16. Ábra - ARM processzorok és perifériáik blokkdiagramja [12]*

Ahogy a *16. Ábrán* is látható, az ARM csak a processzort fejleszti, a perifériákat esetünkben a GigaDevice fejlesztette hozzá.

A GigaDevice F130G8 jellemzői és periféria listája [13]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flash memória | 64KB |  | USART | 2 |
| SRAM | 8KB | I2C | 2 |
| Általános időzítők (32-bit) | 1 | SPI | 2 |
| Általános időzítők (16-bit) | 5 | GPIO | 23 |
| Fejlett időzítők (16-bit) | 1 | EXTI | 16 |
| Rendszer időzítő | 1 | ADC egységek | 1 |
| Watchdog | 2 | ADC csatornák (külső) | 10 |
| RTC | 1 | ADC csatornák (belső) | 3 |

### Lábkiosztás [13]:



*17. Ábra – Az OLED és az F130G8 mikrovezérlő a kapcsolásban]*

A PA0 logikai bemenet láb a BTN2 gomb állapotát figyeli, a BTN2 gomb másik lába a GND1 porton keresztül a közös földpotenciálon van. Amennyiben a gomb nyomva van a bemenet logikai „0”-t kap, amennyiben a gomb zárva van, az bemenet lebeg.

A PA1 logikai bemenet láb a BTN3 porton keresztül figyeli a BTN3 gomb állapotátt, mely az R9 ellenálláson és az F30 biztosítékon keresztül az akkumulátorra pozitív lábára van kötve. Az akkumulátor az R17 ellenálláson keresztül a GND1 közös földpotenciálra van kötve, ezzel védve a mikrovezérlőt.

A PA2 I/O láb nincs bekötve, lebeg.

A PA3 ADC bemenet láb az R8 ellenálláson méri a feszültséget, ha PF0 és PF1 lebeg. Ha PF0 és PF1 földpotenciálra van húzva a mikrovezérlő által, az R8 és R9 ellenállások által leosztott feszültséget méri. A bejövő analóg feszültségértéket a mikrovezérlő beépített analóg-digitális átalakítója (12-bit ADC) egy 0 és 4095 közötti digitális értékké alakítja. Ez alapján számolja vissza a mikrovezérlő a valós feszültségértéket.

A PA4 ADC bemenet láb az akkumulátor feszültségét méri az R10 és R11 ellenállások feszültségosztóján. A bejövő analóg feszültségértéket digitális jellé alakítja a mikrokontroller ADC-je, amelyből vissza tudja számolni az akkumulátor feszültségét.

A PA5 ADC bemenet láb az INA213 árammérő OUT lábára van kötve, az onnan kapott feszültségértéket digitális jellé alakítja az ADC, amelyből a mikrovezérlő visszaszámolja a kimeneti áramerősséget.

A PA6 I/O láb nincs bekötve, lebeg.

A PA7 logikai bemenet láb a BTN1 gomb állapotát figyeli, a BTN1 gomb másik lába a GND1 porton keresztül a közös földpotenciálon van. Amennyiben a gomb nyomva van a bemenet logikai „0”-t kap, amennyiben a gomb zárva van, a bemenet lebeg.

A PA8 logikai kimenet láb a 3110A1 MOSFET vezérlő IN logikai PWM lábára van kötve, ha logikai „1”-et ad a kimenet, a 3110A1 a feszültségnövelő vezérlőt kapcsolja be, ha logikai „0”-át ad a mikrovezérlő, a 3110A1 a feszültségcsökkentő vezérlőt kapcsolja be.

A PA9 I2C órajel busz az OLED SCL órajel lábára van kötve.

A PA10 I2C adat busz az OLED SDA adat lábára van kötve.

A PA13 USART I/O láb a microUSB D- adat lábára van kötve.

A PA14 USART I/O láb a microUSB D+ adat lábára van kötve.

A PA15 I/O láb nincs bekötve, lebeg.

A PB0 I/O láb nincs bekötve, lebeg.

A PB1 kimenet láb az R2 ellenálláson keresztül az AM bipoláris tranzisztor bázisára van kötve. Ha a tranzisztor nyit lehúzza a GND1 közös földpotenciálra a B+ akkumulátorlábat az R4 és R3 ellenállásokon keresztül, az A1T P típusú növekményes MOSFET nyit, és a B+ akkumulátorfeszültség el tud jutni a MT3068 feszültségnövelő engedélyező és bemeneti tápellátás lábaira.

A PB3 logikai kimenet láb az OLED RES# lábára van kötve. Amennyiben a mikrovezérlő logikai „0”-t ad, az OLED alaphelyzetbe áll.

A PB4 logikai kimenet láb az ADP3110 MOSFET vezérlők OD engedélyező lábaira van kötve. Amennyiben a MOSFET vezérlők engedélyező lábai logikai „0”-t kapnak, a vezérlők normális működése leáll.

A PB5 logikai kimenet láb a 3110A2 MOSFET vezérlő IN logikai PWM lábára van kötve, ha logikai „1”-et ad a kimenet, a 3110A2 a feszültségnövelő vezérlőt kapcsolja be, ha logikai „0”-át ad a mikrovezérlő, a 3110A2 a feszültségcsökkentő vezérlőt kapcsolja be. A PB6 kimenet láb a JJ közös katódos diódákon keresztül a MCP1711 feszültségstabilizátor SHDN (engedélyező) lábára van kötve.

A PB7 I/O láb nincs bekötve, lebeg.

A BOOT0 láb GND1 porton keresztül a közös földpotenciálon van.

Az OSCIN/PF0 és az OSCIN/PF1 alapvetően lebeg, ilyenkor a PA3 láb az R8 ellenálláson méri a feszültséget. Azonban nagyobb kimeneti feszültség esetén a PF0 és PF1 lábak a mikrovezérlőn keresztül földpotenciálra kerülnek, ezáltal a PA3 láb az R8 és R9 ellenállások leosztott feszültségét méri. Gyakorlatilag méréshatárváltó a kimeneti feszültség méréséhez.

Az NRTS láb GND1 porton keresztül a közös földpotenciálon van.

A VDDA és VDD bemeneti tápellátás lábak a VDD porton keresztül kapják az MCP1711 feszültségstabilizátor által előállított 3V feszültséget.

A VSS föld láb GND1 porton keresztül a közös földpotenciálon van.

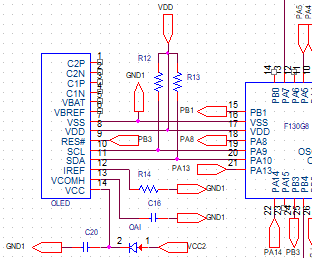
Mint a lábkiosztásból megfigyelhető a mikroprocesszor központi szerepet játszik a kapcsolásban. A hozzá kötött elemekből és az adatlapján lévő lábkiosztásból megismerhettük milyen feladatokat lát el és azokat hogyan végzi.

A mikrovezérlő főbb funkciói a kapcsolásban:

* Integrált áramkörök engedélyezése.
* Feszültségmérés a beépített analóg-digitális konverter segítségével.
* I2C buszon órajel és adat küldése.
* USART-on (magyarul univerzális szinkron adóvevő) keresztül adat küldés és fogadás az USB-ről.
* Gombok működtetése.
* PWM jel előállítása a MOSFET vezérlőknek.
* Méréshatárváltó a kimeneti feszültség mérésére.

## Organic Light Emitting Diode – OLED

Az OLED, magyarul organikus fényemittáló dióda a kapcsolás kijelzője. Ez a felület juttatja el a felhasználónak a kapcsolás által mért adatokat. A kapcsolásunkon egy SSD1306 típusú 14 pines OLED található.



*18. Ábra – Az OLED a kapcsolásban*

Lábkiosztás [15]:

C2P, C2N, C1P, C1N, VBAT és VBREF lábak a beépített DC/DC átalakítót üzemeltetik. Esetünkben a DC/DC átalakító nincs használatban, ezért a lábak lebegnek.

A VSS föld láb a GND1 porton keresztül a közös földpotenciálra van kötve.

A VDD bemeneti tápellátás láb, az OLED logikai hálózatát táplálja. A VDD porton keresztül kapja az MCP1711 feszültségstabilizátor által előállított 3V-ot.

A RES# láb a szabályozó és vezérlő resetje, a mikrovezérlő PB3 logikai kimenetére van kötve. Ha logikai „0”-t kap az OLED-et alapállapotba állítja.

Az SCL I2C órajel busz a mikrovezérlő PA9 kimenetére van kötve, az R12 ellenálláson keresztül kapja a 3V feszültséget a VDD portról.

Az SDA I2C adat busz a mikrovezérlő PA10 I/O lábára van kötve, az R13 ellenálláson keresztül kapja a 3V feszültséget a VDD portról.

Az IREF láb: áramerősség referencia a fényerősség beállításához. Az R14 ellenálláson keresztül a GND1 közös földpotenciál portra van kötve.

A VCOMH láb az OLED közös elektródájának a feszültségforrása, gyakorlatilag a kontrasztot határozza meg. A C16 kondenzátoron keresztül a GND1 közös földpotenciál portra van kötve.

A VCC bemeneti tápellátás láb az OEL panel tápellátását biztosítja. Az OAI diódán keresztül a VCC2 portra van kötve, amin az MT3608 feszültségnövelő által előállított 12V feszültség esik.

## Universal Serial Bus – USB

Az Univerzális Soros Busz egy ipari szabvány, amely soros kommunikációt és adatátvitelt tesz lehetővé egy gazdagép és egy eszköz között.

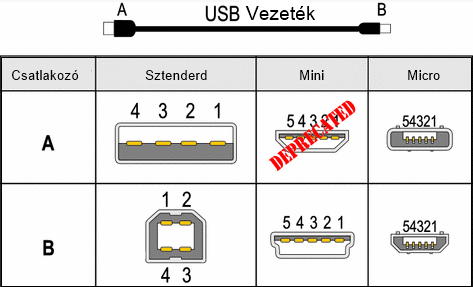
Az USB architektúra három összetevőből áll: egy gazda, az eszközök és egy vezeték. Az eredeti USB topológia egy gazdagépet és 127 eszköz csillag elrendezését tette lehetővé USB vezetékeken keresztül.

A gazdagép bármilyen számítási rendszer lehet, amely rendelkezik USB host controllerel (gazdagép szabályozó). A host controller felelős a legtöbb funkcióért, ami szabályozza a kommunikációt a soros buszon. Minden kommunikációt a buszon a gazdagép kezdeményezhet, ami mester buszként viselkedik.

Az USB eszközök a perifériák a buszra kötve. A gazdagép szempontjából az eszközök szolgaként viselkednek.

Az USB vezeték látja el a fizikai kapcsolatot az eszközök és a gazdagép között. [14]

Az USB csatlakozóknak több típusa is lehet, a mi NyÁK-unkon egy USB Micro-B csatlakozó található.

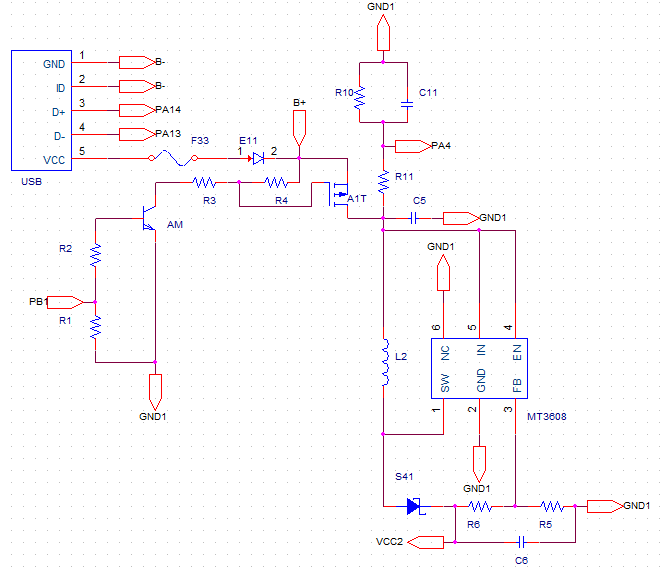


*19. Ábra – USB típusok [14]*

A sztenderd USB A csatlakozón csak 4 láb van, míg az USB Micro-B-n van egy ötödik ID láb. Ez alapján határozza meg az eszköz, hogy mester, vagy szolga eszközre csatlakozik. Amennyiben az ID láb földpotenciálon van, a csatlakozó egy mester eszközhöz tartozik, amennyiben a láb lebeg, egy szolga eszközhöz tartozik.

A kapcsolásunkon az USB az akkumulátorok töltésére nem alkalmas, csak a mikrovezérlő (és egyben a teljes kapcsolás) elindítására és annak firmware-ének a frissítésére használható.

USB Micro-B lábkiosztása:



*20. Ábra – Az USB a kapcsolásban*

1: VCC, bemeneti tápellátás láb, az F33 biztosítékon és az E11 dióda nyitóirányán keresztül a B+ tápellátás portra csatlakozik, valamint az R4 ellenálláson keresztül az A1T növekményes, P típusú MOSFET gate-jére van kötve. Amennyiben a mikrovezérlő kinyitja az AM bipoláris tranzisztort, a B+ tápellátás port és az USB VCC lába az R3 ellenálláson keresztül a GND1 közös földpotenciál portra lesz kötve, Az A1T MOSFET nyílik és a TVL757P feszültségnövelő IC megkapja a tápellátását.

2: D-, adat láb, a mikrovezérlő PA13 bemenetére kötve.

3: D+, adat láb a mikrovezérlő PA14 bemenetére kötve.

4: ID láb közvetlenül az akkumulátor negatív lábára van kötve. Földpotenciálra van kötve, tehát ez a mester eszköz.

5: GND, föld láb, közvetlenül az akkumulátor negatív lábára van kötve.

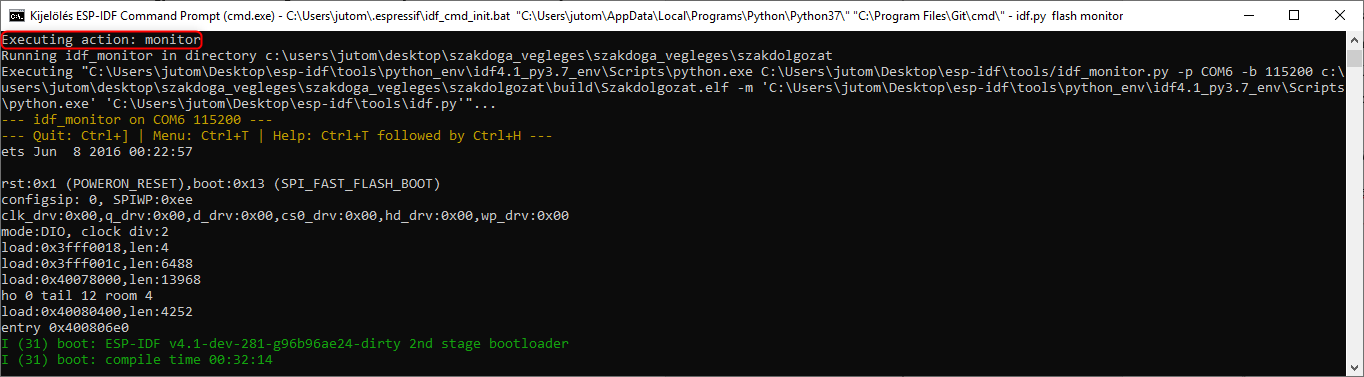
# Az ESP-IDF és a FreeRTOS

Ezen tudások birtokában, elkezdtem elkészíteni a saját elektromos cigarettánk mikrovezérlős programját. A mikrovezérlőnek egy ESP32 Devkit V1 típusú fejlesztőkártyán lévő ESP32 mikrovezérlőt választottam, azon egyszerű okból kifolyólag, hogy ez állt rendelkezésemre, és ez biztosan rendelkezik a projekt számára szükséges perifériákkal és számítási teljesítménnyel.

Következő lépésként ki kellett választanom azt a keretrendszert, amelyben a programot meg fogom írni. Gyakorlatilag két opció állt rendelkezésemre. az Arduino IDE, amely sok szempontból nagyban leegyszerűsítette volna a program tervezését, ugyanis ez a legszélesebb körben elterjedt. A másik lehetőség az Espressif által készített ESP-IDF, ami jóval kevésbé elterjedt, viszont mivel ezt, az Arduino IDE-vel ellentétben, nem használtam még, és kíváncsi voltam a FreeRTOS funkcióira is, ezért végül erre esett a választásom

## Az ESP-IDF

Az ESP-IDF egy az Espressif által gyártott keretrendszer (Espressif Integrated Development Framework), amely lehetővé teszi a FreeRTOS használatát az ESP32 mikrovezérlőkön, egyedi könyvtárakkal és függvényekkel. A használatához szükséges a hivatalos weboldalról letölthető keretrendszer telepítése a számítógépre, amely többek között tartalmaz egy fordítót, egy kommunikációs platformot és egy flashelő algoritmust, valamint ezen keresztül elérhetünk egy egyszerű grafikus felhasználói felületet is, amelyben a mikrovezérlő alapértelmezett beállításait módosíthatjuk. Ezeket a funkciókat az ESP saját terminal ablakában érhetjük el, ahol miután megnyitottuk a munkakönyvtárunkat különböző parancsok megadásával lehet kiválasztani, hogy milyen funkciókra van szükségünk. Ilyen parancsok például az **„idf.py build”**, amely lefordítja a kódunkat, valamint az **„idf.py flash monitor”**, amely USB porton keresztül felflasheli az eszközünket, majd elindítja a soros kommunikációt azzal.



*22. Ábra – ESP-IDF saját parancssora, „idf.py monitor” futtatása*

## Az RTOS és a FreeRTOS

Az RTOS (Real Time Operating System), magyarul valós idejű operációs rendszer, egy olyan operációs rendszer, amely a valós idejű feladatvégrehajtásra lett optimalizálva. Ahhoz, hogy egy operációs rendszer RTOS-nek lehessen nevezni, ahhoz muszáj determinisztikusnak lennie, és a reakció idejének megjósolhatónak kell lennie. A rendszer többi jellemzője, mint például a mérete, sebessége, ugyan fontos, azonban ezektől még nem lesz valós-idejű.[16]

A FreeRTOS egy olyan valós idejű operációs rendszer, amely direkt olyan célra lett kifejlesztve, hogy beágyazott rendszerekben lehessen használni. Éppen ezen okból kifolyólag kis méretű és viszonylag egyszerű, sok olyan funkció nem található meg benne, amelyek a komplexebb RTOS-ekben igen, azonban ezek a funkciók szinte feleslegesek is a viszonylag egyszerűbb beágyazott rendszerek esetén (tehát például nem okostelefonok tervezésére lett kitalálva). A FreeRTOS használata lehetőséget nyújt több szál kezelésére és több feladat végrehajtására. [17]

# A projekt áttekintése

A szakdolgozat feladatom elkészítéséhez használt programokat már ismerve, rövid áttekintést szeretnék adni a szoftver alapvető működéséről és funkcióiról.

## A hardver áttekintése

Először a hardver felépítését kell tisztáznom nagyvonalakban a szoftver későbbi könnyebb megértésének érdekében. A hardver nagyban hasonlít a fent már említett elektromos cigaretta felépítéséhez. A mi készülékünk egy minimalista stílust követ, így ami kívülről látható rajta az a kezelőfelület, amely három gombból áll:

* „tűzgomb” – ez a gomb szolgál arra, hogy az elektromos cigarettát ki-és bekapcsolja, valamint a bekapcsolt állapotban levő eszköznél a gomb hosszan való nyomva tartása a tankban levő fűtőszál hevítéséért felelős.
* teljesítményszabályozó gombok – ezen két gomb segítségével állítja be a felhasználó az általa kívánt teljesítmény értéket.

Az eszközön a gombokon kívül található még egy úgynevezett 510-es csatlakozó, amely egy menettel ellátott csatlakozó és ennek segítségével lehet csatlakoztatni a *mod*hoz a jelenleg a piacon kapható legtöbb *tank*ot.

A felhasználó számára a visszajelzést egy RGB LED szolgáltatja. Ennek a segítségével leolvashatja az éppen aktuális beállított teljesítményt, valamint azt, hogy töltés esetén az akkumulátor még tölt-e vagy már feltöltött.

## A szoftver áttekintése

A szoftver felépítésekor figyelembe kellett vennem a hardveres hátteret, és így kellett megalkotnom egy kompatibilis és működő programot.

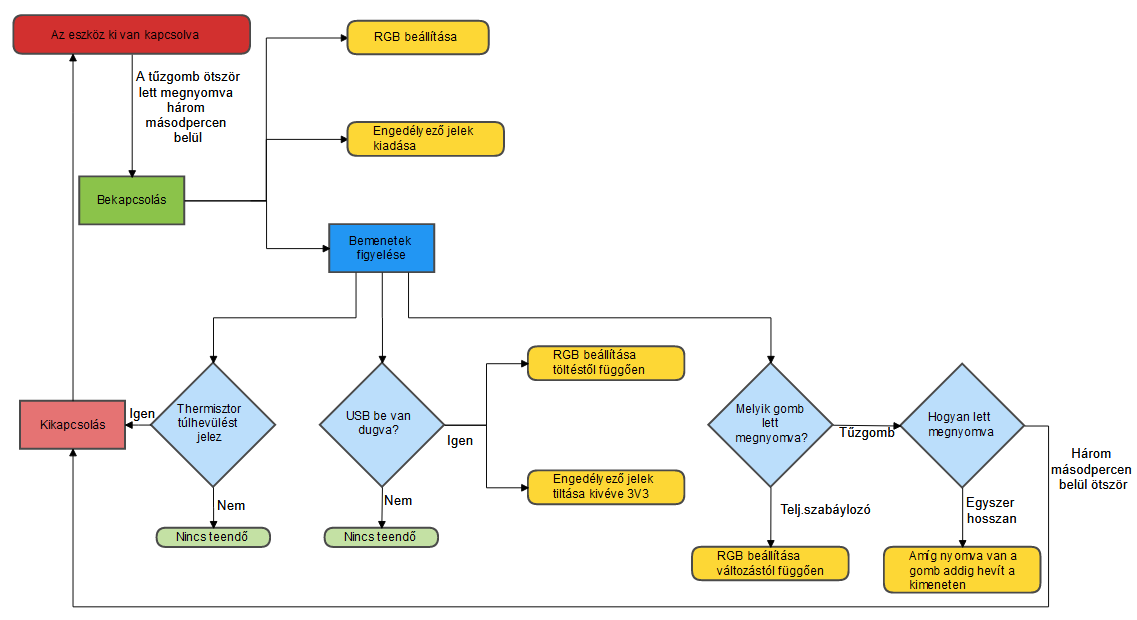
A programkód áttekintése először a bekapcsolási szekvenciával kezdődik. A bekapcsolás az a tűzgomb három másodpercen belül történő ötszöri megnyomása után történik meg. Amennyiben az eszköz sikeresen bekapcsolt, az kiadja a megfelelő kimeneteken az engedélyező jeleket a hardver többi elemének. Ez után az eszköz bekapcsolt állapotba kerül és üzemkész, az RGB LED elkezd világítani az adott színnel, majd a szoftver folyamatosan olvassa a bemeneteket.

Amennyiben a tűzgomb kerül megnyomásra elindul a kikapcsolási szekvencia, és ezzel nagyjából egy időben az eszköz elkezdi hevíteni adott teljesítménnyel a fűtőszálat a gomb elengedéséig.

Ha a másik két gomb közül az egyik kerül megnyomásra, akkor az adott gomb funkciójának megfelelően elkezdi növelni, vagy csökkenteni a beállított teljesítmény értékét, és az RGB LED ehhez alkalmazkodván változtatja a színét.

Amikor a thermisztor bemenetén a szoftver túlhevülést (70°C és fölötte) érzékel, azonnal kikapcsolja az eszközt, és letiltja az összes engedélyező kimenetet.

Ha bedugásra kerül egy USB csatlakozó, a szoftver letiltja a tűzgomb működését, valamint az engedélyező kimenetek zömét, és az RGB LED magenta színűre vált. Amikor az akkumulátor teljesen feltöltött, az RGB LED zöld színűre változik.



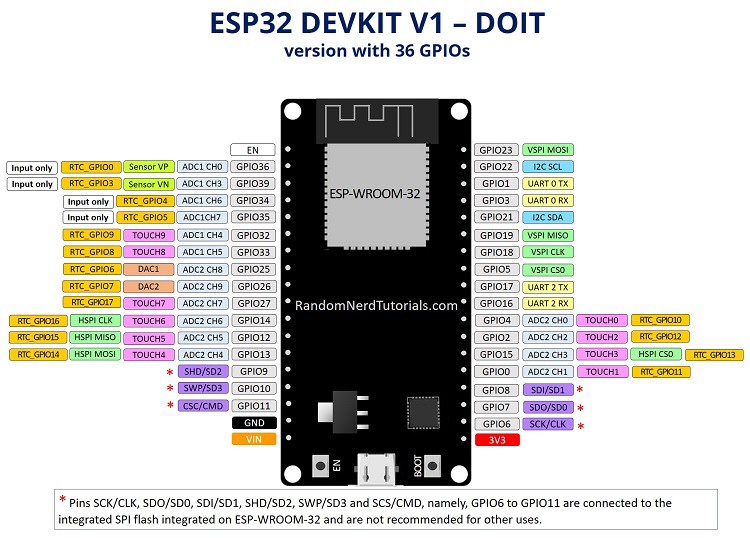
*23. Ábra – A szoftver folyamatábrája*

# A szoftver részletes ismertetése

Ebben a részben kívánom kifejteni az általam megírt szoftver részleteit. Külön ki fogok térni a szoftver minden egyes funkciójára és mélyebben bemutatni azokat.

## A GPIO lábak inicializálása és definiálása

A GPIO lábak (General Purpose Input/Output) egyedileg programozható ki-és bemenetek. Az ESP32 mikrovezérlő 40 GPIO lábbal rendelkezik, azonban nem mindegyiket ajánlatos használni ezek közül, azonban ezekre nincs is szükség, ugyanis sem a szoftver, sem a hardver megvalósítása nem kíván ennyi ki-és bemenetet. Lesznek azonban olyan lábak, amelyeket nem GPIO-ként fogunk használni, azonban ezeket jelen szekcióban nem taglalom.

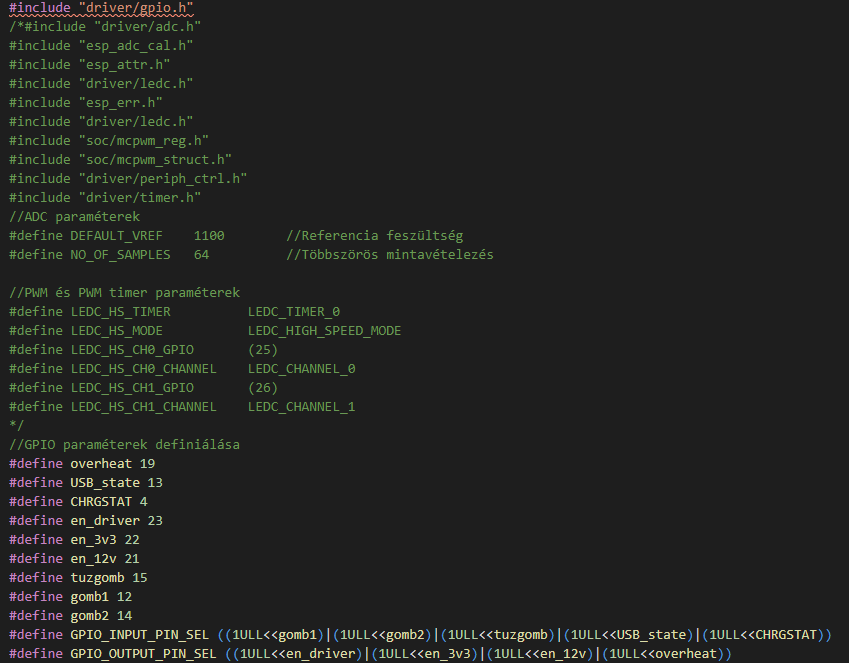


*24.Ábra – ESP32 DEVKIT V1 lábkiosztása* [17]

Ennél a programnál bemenetként a GPIO4,12,13,15 lábak lesznek használva, míg kimenetként a GPIO19,21,22,23 lábak.

Első lépésként, ahhoz, hogy a GPIO-kat megfelelően tudjam inicializálni, és a későbbiekben használni az ehhez szükséges könyvtárat kellett meghívnom a program elején. Ez a gpio.h könyvtár. Miután ez megtörtént, a kód elején definiálnom kellett az input lábakat, ezeket el kellett nevezni, majd egy úgynevezett „bitmaskot” létrehozni. Erre a bitmaskra azért lesz szükség, mert a GPIO beállításához és inicializálásához szükséges függvény ez alapján a bitmask alapján fogja tudni a GPIO regiszterben eltárolni azt, hogy mely GPIO-k kerülnek felhasználásra, és ezek milyen beállításokkal rendelkeznek. A bitmask megadásának a módja olyan, hogy egy unsigned long long ’1’-et kell elshiftelni a GPIO számának értékével. Ez lesz az az érték, amit fel tud dolgozni a függvény. A bitmaskot egy-egy definíción belül hoztam létre az összes bemenetre és kimenetre.

Az ábrákon kikommenteltem az éppen aktuális témakörhöz nem tartozó kódrészleteket, azonban a végleges kódban ezek benne vannak.



*25.Ábra – A GPIO pinek definiálása és a bitmaskok létrehozása a gpio.h meghívása*

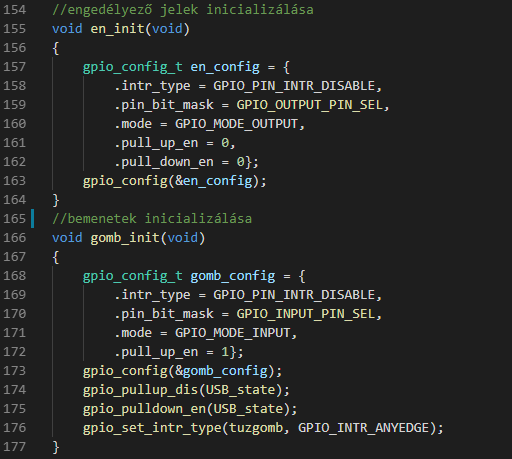
Miután megtörtént a definiálás és a bitmask létrehozás, létre kellett hoznom két függvényt, amelyek a GPIO-k inicializálását végzik, úgy a kimeneti, mint a bemeneti lábakon egyaránt. Ezekben a függvényekben létrehoztam egy-egy gpio\_config\_t típusú változót, melyeket elneveztem. A kimeneti változót „en\_config”-nak, míg a bemenetit „gomb\_config”-nak. Ezen a változón belül kellett megadnom a:

* GPIO-hoz rendelt interrupt fajtáját, ami lehet letiltott, felfutó élre, lefutó élre vagy felfutó-és lefutó élre
* GPIO lábak bitmaskját
* GPIO fajtáját, amely lehet bemenet vagy kimenet
* GPIO alapértelmezett szintjét, melyet az határoz meg, hogy a mikrovezérlőn belül milyen feszültség szintre köt rá. Ez lehet felhúzott, azaz logikai magas szinten, vagy lehúzott, azaz logikai alacsony szinten. Harmadik opcióként lehet még magas impedanciás is, azonban ebben a kódban ezt nem használtam.

A változó létrehozása után meg kell hívnom a *gpio\_config();* függvényt, amelynek meg kell adni ezt a változót, és ez majd a program lefutásakor ezekkel, az általam megadott paraméterekkel fogja bekonfigurálni a bitmaskban megadott GPIO-kat.

A *gpio\_config();* függvény azonban nem minden GPIO esetén állít be az adott láb funkciójának megfelelő beállításokat. A programot úgy terveztem, hogy ez csak a legtöbb lábra megfelelő beállításokat konfigurálja a bemeneteknél, viszont amint az a *26.Ábrán* is látszik, az USB bedugását jelző bemeneten nem lehet a belső felhúzó ellenállásra kötve, ugyanis akkor a szoftver folyamatosan úgy érzékelné, mint ha az be lenne dugva. Ezt a problémát két másik függvény meghívásával hidaltam át, az egyik a *gpio\_pullup\_dis();* amelynek meg kell adni a változtatni kívánt GPIO számát, és így az letiltja az adott láb felhúzását. A másik ilyen függvény a *gpio\_pulldown\_en(); ,* ahol szintén a GPIO számát kell megadni, és az a függvény lefutásakor a lehúzó ellenállásra csatlakoztatja azt, így az alapértelmezett értéke logikai ’0’ lesz.

Mint az a *26.Ábrán* is látszik, a bemeneteknél a megszakítások alapértelmezetten le vannak tiltva, azonban a tűzgomb megnyomásakor mindenképp szükségünk van rá, méghozzá úgy, hogy mind felfutó, mind lefutó élre történjen egy ilyen megszakítás. Ez úgy volt megoldható, hogy ennek a lábnak is külön állítottam át az interrupt beállítását, a *gpio\_set\_intr\_type();*  függvénnyel. Ennek szintén meg kellett adni a módosítani kívánt GPIO lábat, valamint második paraméternek azt, hogy milyen típusú interruptot rendeljen hozzá. Jelen esetben a *GPIO\_INTR\_ANYEDGE* beállítást alkalmaztam, így elérvén a kívánt funkciót.



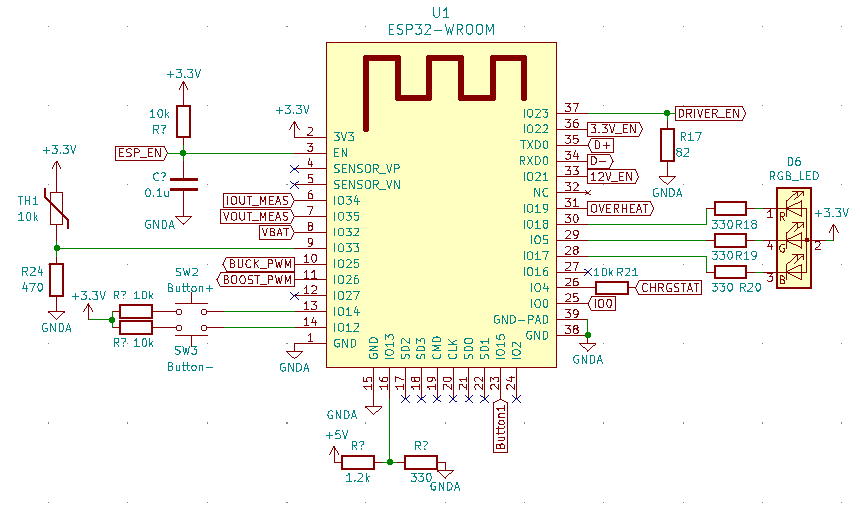
*26.Ábra – GPIO lábak inicializálása, külön kimenetek és bemenetek*

Így, hogy létre lettek hozva ezek a függvények, már bármikor meghívhatóak. Ezen függvények a fő programban lesznek meghívva, ahol egyszer lefutnak és így állítják be az általam használni kívánt lábakat.

## Az ADC lábak karakterizálása

Az ESP32-n összesen 18 analóg-digitális átalakító csatorna található. Ezek felosztása az, hogy két ADC van, az ADC1 és az ADC2. Ezek csatornákra vannak osztva. Az ADC1 -ben 8 csatorna található, míg az ADC2-ben 10, azonban az ADC2 használata a WiFi vezérlőivel egy időben nem megengedettek, így a projekthez kizárólag az ADC1 csatornáit használtuk fel, hogy a későbbiekben esetleges funkcióbővítések során ez ne okozzon problémát.

Ehhez az eszközhöz mi összesen 4 csatornát használtunk fel, melyek a mikrovezérlő 34-es, 35-ös, 32-es és 33-as GPIO lábához vannak rendelve. Az általunk felhasznált csatornák rendre a kimeneti áram, kimeneti feszültség, akkumulátor feszültség és a thermisztor mérésére szolgálnak. Ezek a csatornák mind úgy vannak bekalibrálva, hogy 0V és 1100mV között mérjenek 12bites felbontással. A hardveres tervezésnek köszönhetően, ezeken az átalakítókon nem eshet 1,1V-nál nagyobb feszültség, és mivel az ADC eddig az értékig nagyon pontos, ezért ezt egy jó megoldásnak gondolom.



*27.Ábra – Az ESP32 általunk használt lábkiosztása* [18]

Az ADC-k karakterizációját azzal kezdtem, hogy meghívtam az ADC kezeléshez szükséges két könyvtárat, ezek az *adc.h* és az *esp\_adc\_cal.h*, meghívom. Ez után definiáltam két értéket, az egyik majd az ADC kiolvasásához szükséges NO\_OF\_SAMPLES, ez a többszörös mintavételezésnél fog kelleni, ugyanis ez a szám adja meg, hogy hányszoros legyen a mintavételezés. Jelen esetben 64 ez a szám, ez elég magas ahhoz, hogy kiszűrje a hibás értékeket, azonban nem túl magas, így nem növeli meg a kiolvasás idejét. A másik definiált érték, az az alapértelmezett referenciafeszültség. Itt ezt 1100mV-ra állítottam be a fent említett indokok miatt.

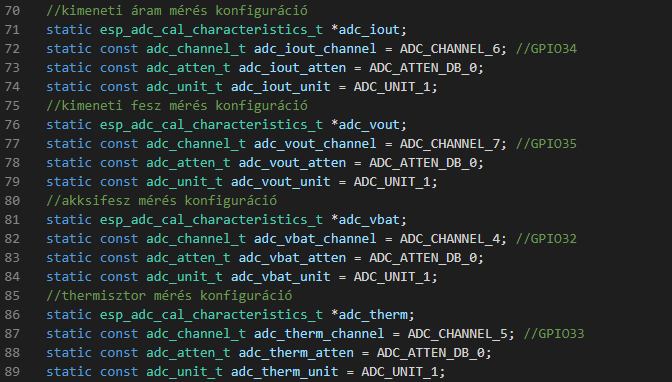


*28.Ábra – A szükséges könyvtárak meghívása és az értékek definiálása*

Ha ez megvan, akkor a karakterizációhoz szükséges, különböző típusú változókat kellett létrehoznom. A típusok a következők:

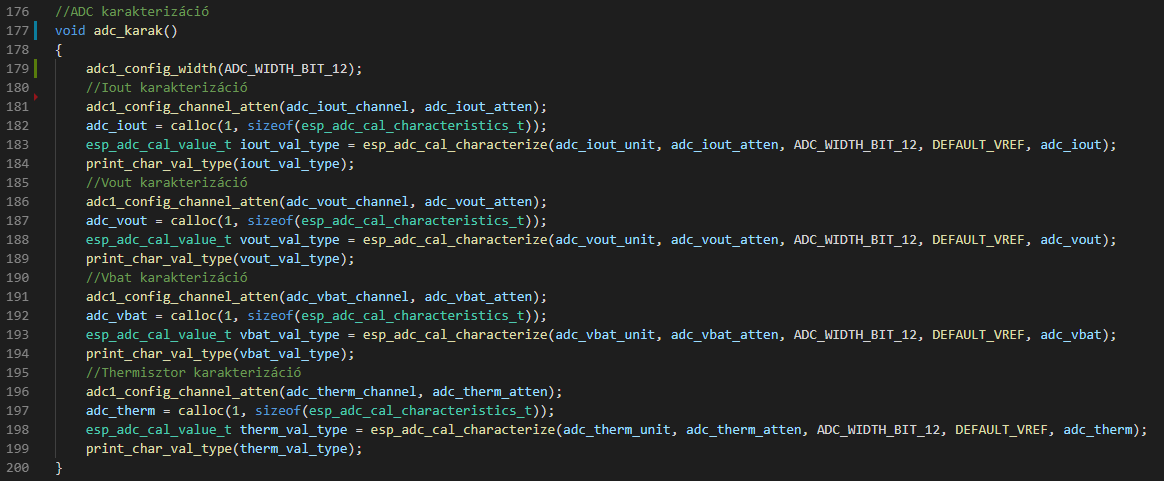
* esp\_adc\_cal\_characteristics\_t – Itt egy pointer változót kell deklarálni, amelynek később a karakterizációban lesz szerepe
* adc\_channel\_t – Az adott ADC csatornájának a definiálása. Értéke ADC\_CHANNEL\_x, ahol x-et be kell helyettesíteni az általunk válaszott csatorna sorszámára. *24.Ábrán* látszódnak a különböző csatornák kiosztásai
* adc\_atten\_t – Az adott csatorna csillapítása. Ez minden általam használt esetben ADC\_ATTEN\_DB\_0, ugyanis nincs szükség a méréshatár kiterjesztésére vagy lecsökkentésére
* adc\_unit\_t – ADC egység kiválasztása, ez szintén, minden esetben ADC\_UNIT\_1, a fentebb már említett indokok miatt az ADC1 van csak használva.

Ezen típusú változók létrehozását más-más elnevezéssel, de minden általam használt csatornára létre kell hoznom, ugyanis a továbbiakban ezek használatával lesz bekalibrálva az adott csatorna.



*28.Ábra – ADC karakterizációhoz szükséges különböző típusú változók létrehozása*

Következő lépésként létre kellet hoznom egy függvényt, amely az ADC karakterizációját ténylegesen elvégzi. Ezt a későbbiekben egyszer lefuttatva az ADC csatornák beállításra kerülnek. A függvényen belül először beállítottam az ADC1 egység felbontását az *adc1\_config\_width();* függvénnyel, aminek az *ADC\_WIDTH\_12\_BIT* értéket megadva, 12 bitesre konfiguráltam azt. Az *adc1\_config\_channel\_atten();* függvénnyel, aminek a paraméterei fentebb már létrehozott csatorna változó és az csillapítási változó, beállítom az ADC1 általam kívánt csatornáját és csillapítását. Következő lépésként egy memória részt foglalunk le, majd az *esp\_adc\_cal\_characterize*(); függvénnyel kalibráljuk az adott adc csatornát. Ennek a függvénynek meg kell adni azt, hogy melyik ADC egységet milyen csillapítással, milyen felbontással, mekkora referencia feszültséggel kalibráljon. Amint a kalibráció megtörtént, még egy függvényt meghívok, amely kizárólag hibakeresési célt szolgál. Ez a függvény a *print\_char\_val\_type();* amely lefutásakor kiíratja a soros monitorra azt, hogy az ADC karakterizálása az eFuse-ban levő értékkel történt-e. Ez a függvény az Espressif által készült, így erre nem térnék ki részletesebben. Ezt, az általam készített *adc\_karak();* függvényben szintén minden konfigurálni kívánt csatornára megismétlem, az első, *adc1\_config\_width();*, függvényen kívül, ezt ugyanis elég egyszer lefuttatni ahhoz, hogy a felbontás beállítása megtörténjen.



*29.Ábra – adc\_karak(); függvényen belüli ADC csatornák karakterizálása*

A GPIO inicializáló függvényekhez hasonlóan, ez a függvény is a fő programban lesz meghívva, ahol a mikrovezérlő indítását követően egyszer lefut, és így beállítja a szükséges ADC lábakat.

A *check\_efuse();* függvény szintén az Espressif által megírt kódrészlet, amely szintén egyszer fog lefutni, az általam készített függvényhez hasonlóan. Ez is csak hibakeresési célt szolgál. Soros monitorra írja ki azt, hogy az adott beállítások mellett, az eFuse „Two Point” beállítása és a „Vref”, azaz referenciafeszültség beállítása bele van-e égetve. Erről a soros monitoron ad vissza egy-egy rövid üzenetet.

## Bekapcsolási és kikapcsolási szekvencia

Az eszköz bekapcsolása amint az a *23.Ábrán* is látható úgy valósul meg, hogy a tűzgombot ötször kell megnyomni három másodpercen belül. Erre azért van szükség, ugyanis, ha az eszköz a felhasználó zsebében van, mindenképp valahogy el kell azt kerülni, hogy bekapcsoljon, és hogy ez után a tűzgomb véletlenül benyomódjon. Ez azért fontos, ugyanis, ha a tűzgomb túl sokáig van benyomva, az eszköz folyamatosan fűti a fűtőszálat, amely szétégetheti a vattát, ezáltal gyakorlatilag a vatta kicseréléséig használhatatlan lesz az eszköz. Ennél egyel fontosabb, a balesetek megelőzése, ugyanis a LiIon akkumulátor a túl sok ideig tartó hevítés hatására túlhevülhet, és akár fel is robbanhat. Pont az ilyen balesetek megelőzése végett van szükség az úgynevezett zsebzár funkcióra, ám a túlhevülés ellen még egy termisztorral is védekezek, azonban erre a szakdolgozatom későbbi részén is ki fogok térni ennél részletesebben.

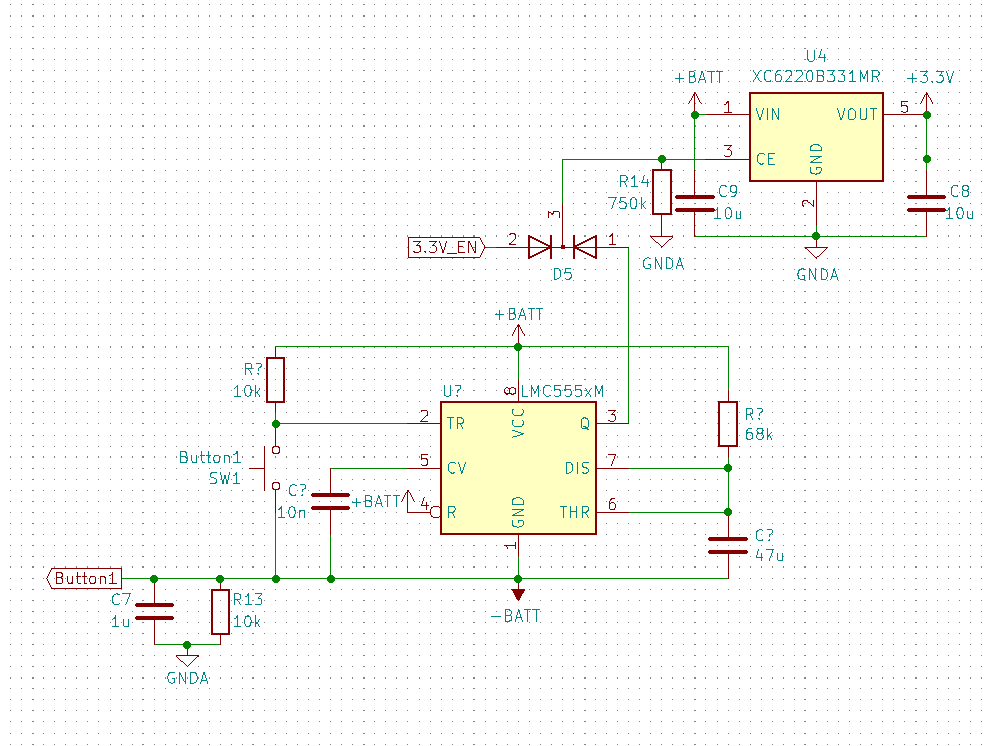
### A bekapcsolási szekvencia

Mint már említettem, az elektromos cigaretta bekapcsolása a tűzgomb három másodpercen belül, ötszörös megnyomásával jön létre. Ahhoz, hogy ez szoftveresen megvalósítható legyen, először a hardveres megvalósításra kell kitérnem.

Mivel a mikrovezérlő a tápfeszültségét egy 3,3V-ot előállító kis feszültségesésű feszültségstabilizátorról(LDO) kapja, így azt kellett megoldani, hogy a gomb megnyomásakor ez az IC megkapja az engedélyező jelét, és ezt meg is tartsa a gomb elengedése után is legalább három másodpercig. Ehhez egy LM555 IC lett használva, amely egy trigger jel hatására az (1) képletben kiszámolt értékű RC kapcsolással 3,52 másodpercig kiadja az engedélyező jelet.

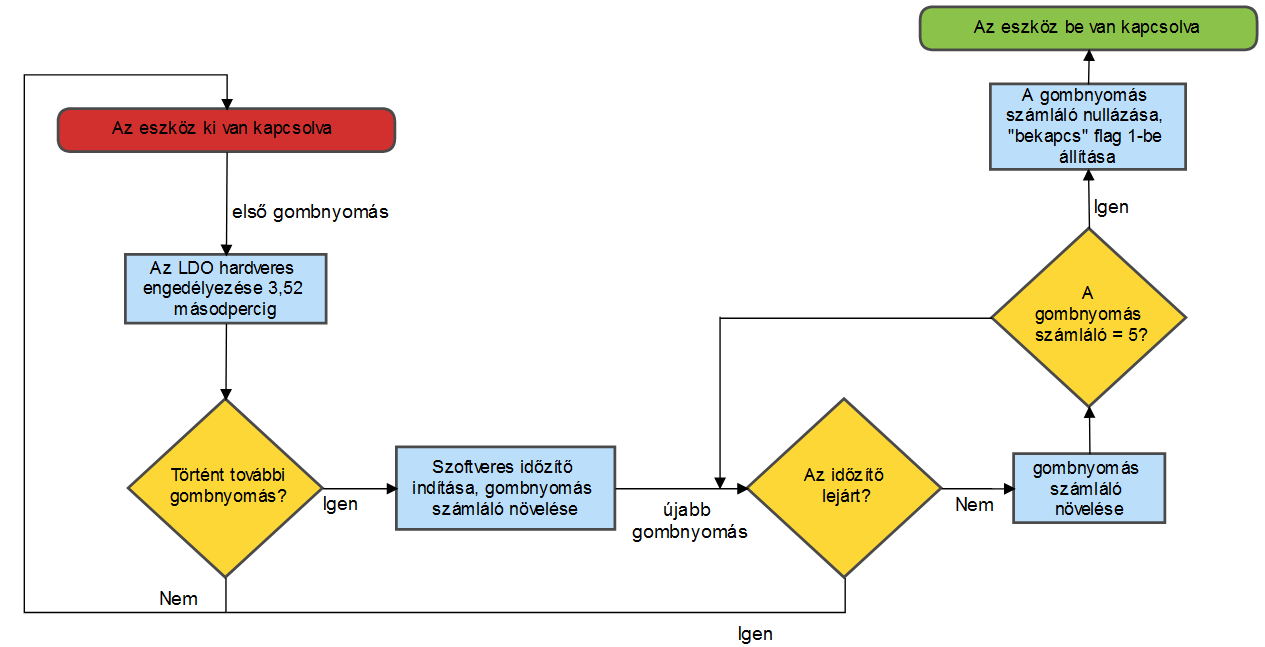
(1)

Ezen a ponton megjegyzendő, hogy ilyen hosszú impulzusra van szükség, így a kimeneti jel késleltetése jelentős, körülbelül 0,85 másodperc[18].



*30.Ábra – Az LM555 IC és a 3,3V-ot előállító LDO[18]*

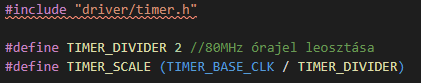
Szoftveres megvalósításnál, ha a hardveres háttér adott, azt a megoldást választottam, hogy amikor a tűzgomb megnyomásra kerül, az hardveresen táp alá helyezi a mikrovezérlőt, azaz bekapcsolja azt. Ekkor a „*bekapcs”*  flag bit alapértelmezetten 0 állapotban áll, így egyik engedélyező jel sincs a mikrovezérlő által kiküldve. Amennyiben nem történik további gombnyomás, az LM555 IC 3,52 másodperc után megszakítja a tápellátást és a mikrovezérlő leáll. Azonban, ha történik gombnyomás, az egy megszakítást küld a mikrovezérlőnek, mely megszakításra reagálván az MCU elindít egy három másodperces időzítőt. Az időzítő lejárása végéig, a szoftver figyeli a további gombnyomásokat, és amennyiben azok összege elérte az ötöt, a program kiküldi az engedélyező jeleket, többek között a 3,3V-os feszültségstabilizátornak is, így már folyamatossá téve az eszköz működését.



*31.Ábra – A bekapcsolási szekvencia*

Ahhoz, hogy a bekapcsolás megvalósítható legyen szoftveresen, először egy időzítőt kell létrehozni és inicializálni. Az ESP32 két hardveres időzítő csoportot tartalmaz, ezeken a csoportokon belül található két-két időzítő. Ezek közül a jelen felhasználásban csak egyre lesz szükség. Az időzítő inicializálásához meg kell hívni a *timer.h* könyvtárat, és definiálni kell két értéket. Az egyik a *TIMER\_DIVIDER*, amely egy 2 és 16 közötti szám kell, hogy legyen. Ez fogja meghatározni, hogy a 80MHz-es órajel hányad részére legyen leosztva. Ebben a programban én 2-nek definiáltam ezt az értéket, ugyanis 40MHz-cel könnyen el tudtam végezni a szükséges számításokat.

A másik előre definiált érték a *TIMER\_SCALE,* amely az időzítő skálaosztását adja meg. Ez egy olyan érték, amely az alap órajel és az órajelosztó hányadából jön össze.



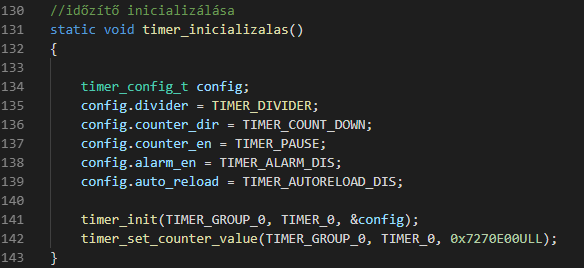
*32.Ábra – A timer.h meghívása valamint a két érték definiálása*

Következő lépésként létre kellett, hogy hozzak egy függvényt, mely az inicializálását végzi el az időzítőnek, ezt a *timer\_inicializalasa();* függvényt később egyszer fogom meghívni, és ez fogja elvégezni a szükséges beállításokat, amint az ESP32 megkapja a tápfeszültséget. Itt egy *timer\_config\_t* típusú változót hoztam létre, amelyen belül a következőket kellett megadnom:

* *divider* – Az órajelosztó értéke, jelen esetben ez az előre definiált *TIMER\_DIVIDER*
* *counter\_dir* – A számláló irányát adja meg, ebben az alkalmazásban egy ez a számláló lefele számlál, így a megadott érték *TIMER\_COUNT\_DOWN*
* *counter\_en –* A számláló inicializálás utáni viselkedését írja le, jelen esetben *TIMER\_PAUSE* értéket adtam neki, így a számláló nem indul el azonnal.
* *alarm\_en* – A számláló küldjön-e jelzést a lejárása esetén, ezt a funkciót letiltottam, így *TIMER\_ALARM\_DIS* értéket adtam neki
* *auto*\_*reload* – A számláló lejárása után automatikusan visszaálljon-e a kezdeti értékre. Ezt én a későbbiekben „manuálisan” oldottam meg, így ezt a funkciót letiltottam és a *TIMER\_AUTORELOAD\_DIS* értéket adtam meg neki.

Az általam létrehozott függvényben a változók értékeinek megadása után a *timer\_init();*  függvényt kellett meghívnom, mely elvégzi az időzítő inicializálását az előzőleg megadott értékek alapján. Ennek a függvénynek meg kell adni a haználni kívánt időzítő csoportot, az ezen belül használni kívánt időzítő sorszámát, valamint a fentebb kifejtett változó típust, az összes megadott adattal.

A *timer\_init();* függvényen kívül szükség volt még arra, hogy megadjam a számláló kezdeti értékét. Ezt a *timer\_set\_counter\_value();* függvény meghívásával tettem meg, amelynek a bemenetei szintén az időzítő csoport, az azon belüli időzítő sorsszáma, valamint a kezdeti érték. A kezdeti értéket az (2) képlettel kaptam meg, hogy így az időzítő mindig három másodpercig számoljon, majd ezt a kapott értéket váltottam át hexadecimális értékre, amely így 0x7270E00.



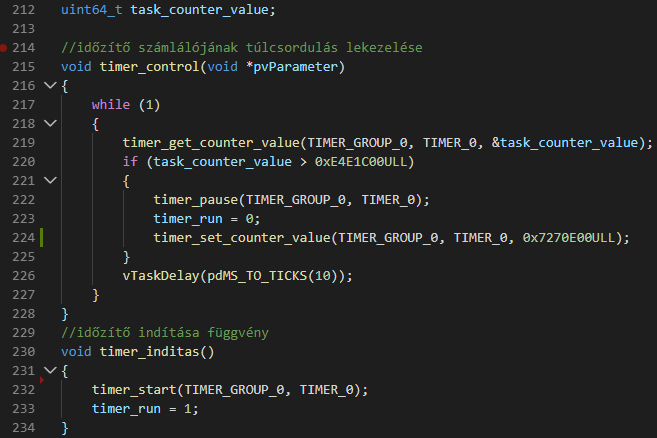
*33.Ábra – A timer\_inicializalas(); függvény*

A *timer\_inicializalas();* függvény létrehozása azonban még nem elegendő. Ahhoz, hogy a számomra megfelelő módon működjön az időzítő szükségem volt még két másik függvény létrehozására, az egyik a *timer\_inditas();*, míg a másik az időzítő számlálójának túlcsordulását lekezelő *timer\_control();* függvény.

A *timer\_inditas();* függvény, amint azt a neve is leírja, kezeskedik arról, hogy az időzítőt el tudjam indítani egy általam kívánt programrészben. A függvény a *timer\_start();* függvény meghívásával az időzítőt elindítja. Ennek a függvénynek a két bemeneti paramétere az időzítő csoport kiválasztása és az ezen belüli időzítő kiválasztása. A *timer\_inditas();*  tartalmazza még a *timer\_run* flag bit 1-be állítását. Ennek a flagnek a későbbiekben lesz haszna, ugyanis ennek a segítségével tudom leellenőrizni azt, hogy az időzítő éppen fut-e.

A *timer\_control();* függvény egy folyamatosan futó programkód, amely 10 miliszekundumonként ismétlődik. Ez felel azért, hogy ha az időzítő eléri a három másodpercet és a számláló túlcsordul, megállítsa azt. Így az időzítő 3,01 másodpercig fog maximálisan futni, azonban ez az eszköz működésében nem okoz problémát. A függvényben egy „végtelen” while ciklus fut folyamatosan. A végtelen azért van idézőjelek közt, ugyanis ez nem okoz problémát és nem fogja az eszköz működését sem lelassítani, sem elrontani, csak a FreeRTOS-nek a működéséhez ilyen „végtelen” ciklusokra van szükség, ezeket tudja az operációs rendszer futó feladatokként kezelni.

A while ciklusban először a *timer\_get\_counter\_value();* függvény fut le, amely lekéri az időzítő számlálójának az aktuális értékét, és azt beírja a *task\_counter\_value* változóba. Ennek a függvénynek is a két bemeneti paramétere az időzítő csoport és az időzítő sorszám. Következő lépésként a *timer\_control();* függvény megvizsgálja, hogy az éppen aktuális lekért érték túlcsordult-e. Amennyiben igen, akkor a *timer\_pause();* függvénnyel leállítja azt, és a *timer\_run* flag bitet 0-ba állítja, valamint az időzítőt visszaállítja az alapállapotba a *timer\_set\_counter\_value();* függvénnyel. A függvény végén található, az if-en kívül viszont a while-on belül található *vTaskDelay(pdMS\_TO\_TICKS(10));* az operációs rendszernek szükséges adat, amely megmondja, hogy a ciklust hány miliszekundumonként futtassa újra.

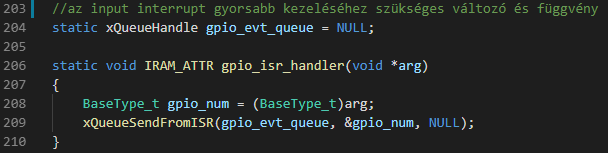


*34.Ábra – A task\_counter\_value változó létrehozása, valamint a timer\_control(); és timer\_inditas(); függvények leírása*

Ahhoz, hogy a bekapcsolási szekvenciát megértsük az időzítő részletes leírásán kívül szükség van még arra, hogy a megszakítást és annak a lekezelését is kifejtsem.

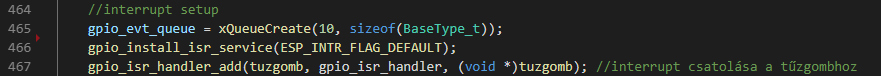
A GPIO inicializálása résznél már kitértem arra, hogy a tűzgomb beállításánál egy fel-és lefutó élre érzékeny megszakítás lett hozzárendelve az adott lábhoz. Ez az első lépése annak, hogy az operációs rendszer le tudjon kezelni egy megszakítást azonban, ez nem minden. További beállítások és függvényhívások szükségesek.

Először létre kell hozni egy változót, amely gyakorlatilag a beérkező interrupt jeleknek lesz egy tároló, ahonnan az operációs rendszer FIFO logikával fogja feldolgozni azokat, ezek után szükség lesz egy olyan függvényre, amely az interrupt kezelést a flash helyett a RAM-ból fogja meghívni, így szignifikánsan felgyorsítva a reakcióidőt ez a FreeRTOS által készített *gpio\_isr\_handler();* függvény, amely az *IRAM\_ATTR* makróval van ellátva.



*35.Ábra – A gpio\_evt\_queue változó és a gpio\_isr\_handle(); függvény*

Ezek után, az *app\_main* függvényben először létrehozzuk az interrupt tároló „sort”, amelyet hozzárendelünk a fent már létrehozott *gpio\_evt\_queue* változóhoz. Ezt az *xQueueCreate();* függvénnyel tudjuk megtenni. Ez után a GPIO-ról érkező interrupt kezelő szolgáltatást kell telepíteni, amelyet a *gpio\_install\_isr\_service();* függvény meghívásával tettem meg, ez a függvény is az *app\_main-*ben található, és ez is csak egyszer, a mikrovezérlő bekapcsolása után, kell, hogy lefusson. Ha ez megtörtént, akkor a *gpio\_isr\_handler\_add();* függvényt meghívom, amellyel hozzácsatolom a megszakításkezelő rutint az általam kiválasztott GPIO lábhoz (jelen esetben a tűzgombhoz).



*36.Ábra – Az app\_main függvényben az interrupt beállítása*

Így beállítva a megszakítás kezelő rutint, a program és az operációs rendszer már fogad megszakításokat a tűzgombról, azonban ezek még semmilyen hatással nincsenek a szoftver működésére. Ahhoz, hogy a megszakítások kiértékelése is megtörténjen, létrehoztam egy függvényt, amelynek az *interrupt\_kiertekeles();* nevet adtam. Ez a függvény felelős a bekapcsolásért, a kikapcsolásért, és azért, hogy a tűzgomb nyomvatartása közben az eszköz felfűtsön, azonban ezen funkciók közül ebben a részben csak a bekapcsolás funkcióját fogom bemutatni.

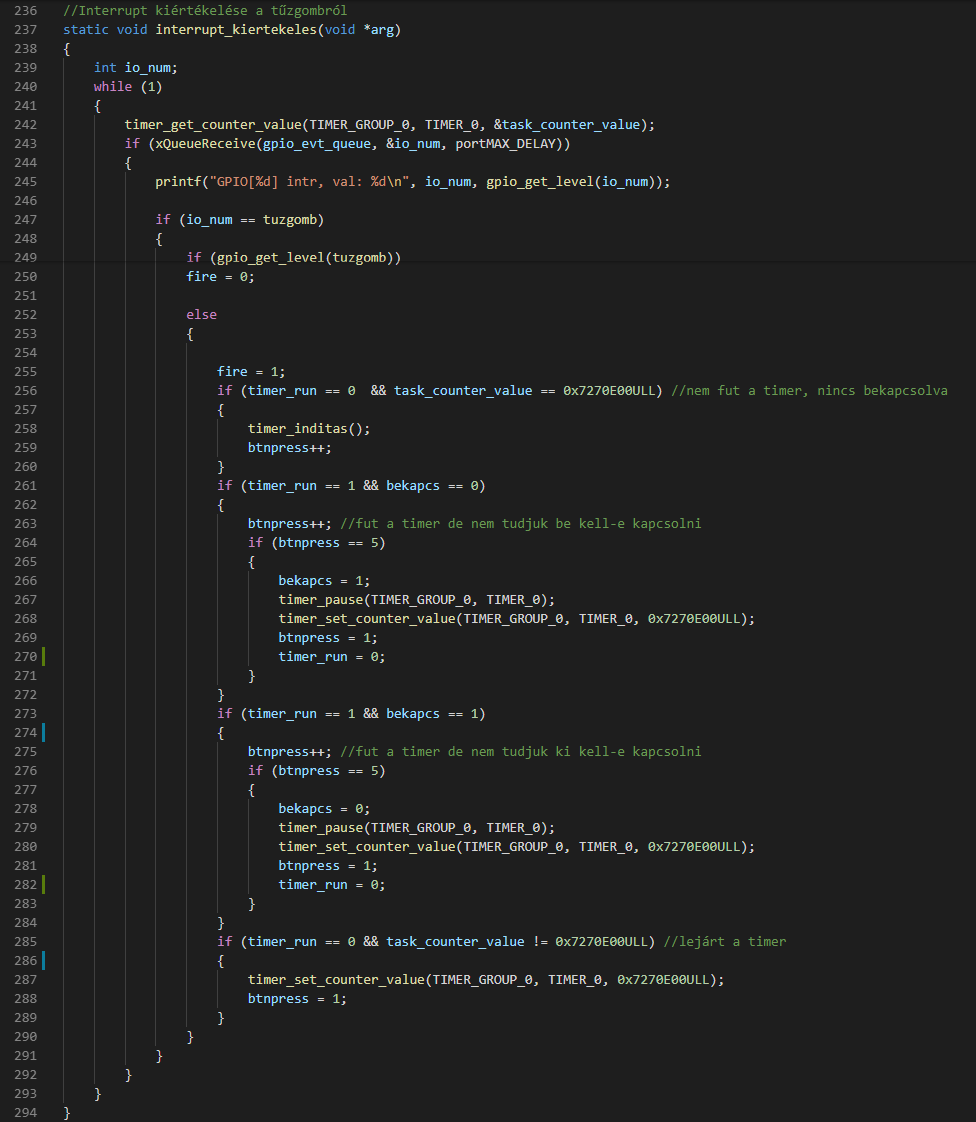
Mint már fentebb említettem, a kikapcsolt állapotban levő eszköznél az első gombnyomás ~3,5 másodpercig tápot ad a mikrovezérlőnek, aminek hatására az bekapcsol és betölti az inicializáló függvényeket, valamint elindítja azokat a feladatokat, amelyek indítása nincs feltételhez kötve. Ekkor a *bekapcs* flag bit értéke 0. A következő gombnyomást már interruptként fogja kezelni az operációs rendszer, ekkor jön képbe az *interrupt\_kiertekeles();* függvény.

Ez a függvény folyamatosan nézi, hogy áll-e a „sorban” interrupt, és amennyiben igen, megnézi, hogy melyik lábról érkezett az. Ezt az *xQueueReceive();* függvénnyel tudja megtenni. Amennyiben a tűzgombról érkezett, azonnal ellenőrzi, a tűzgomb aktuális értékét. 0 esetén a gomb meg van nyomva, 1 esetén el van engedve. Fontos megjegyezni, hogy mint már fentebb említettem a megszakítás az mind felfutó, mind lefutó élre történik.

Amennyiben a program azt látja, hogy a tűzgomb meg van nyomva, leellenőrzi, hogy az időzítő fut-e. Ha az időzítő nem fut, azaz a *timer\_run* flag 0-ban van megnézi, hogy a számláló értéke egyenlő-e a kezdeti értékkel(azaz még nem járt le és csordult túl), akkor a *timer\_inditasa();* függvénnyel elindítja azt, majd egyel megnöveli a *btnpress* számláló értékét. Ha az időzítő fut, azaz a *timer\_run* flag 1-ben van, és a *bekapcs* flag 0-ban, akkor növeli egyel a *btnpress* nevű számlálót, azaz a gombnyomások számát, és leellenőrzi, hogy az új, megnövelt érték elérte-e az 5-öt. Amennyiben nem érte el, várja az újabb interruptot és ha jön megismétli ezt a programrészt. Ha elérte az 5 gombnyomást, a *bekapcs* flaget 1-be állítja, majd megállítja az időzítőt a *timer\_pause();*  függvénnyel, és visszaállítja az eredeti értékére a *timer\_set\_counter\_value();* függvénnyel, valamint a *btnpress* számlálót visszaállítja 1-re és a *timer\_run* flaget 0-ba rakja. Ha a program bármelyik interrupt esetén azt érzékeli, hogy az időzítő nem fut, és az értéke nem a kezdeti érték, akkor az azt jelenti, hogy az öt gombnyomás nem 3 másodpercen belül történt meg, ezért a program visszaállítja az időzítőt a kezdeti állapotba a *timer\_set\_counter\_value();* függvénnyel, valamint a *btnpress* számláló értékét 1-re.

### A kikapcsolási szekvencia

A kikapcsolási szekvencia nagyban megegyezik a kikapcsolással, azzal az eltéréssel, hogy itt a *bekapcs* flag már 1 állapotban van. Az érkező megszakítások kiértékelése az *interrupt\_kiertekeles();* függvényben annyiban fog csak változni, hogy az az állapot, amikor a függvény azt ellenőrzi, hogy az időzítő fut-e azaz a *timer\_run* flag értékét, és a *bekapcs* flag értékét, akkor abba az esetbe lép be, hogy az időzítő fut és a *bekapcs* flag 1-ben van. Ez esetben a bekapcsolási szekvenciához hasonlóan itt is emeli egyel a *btnpress* számláló értékét, és vizsgálja, hogy elérte-e az 5-öt. Amennyiben elérte itt a *bekapcs* flaget 0-ba állítja. Ezen az egy részleten kívül a bekapcsolás és a kikapcsolás között nincs eltérés, sem logikában, sem programkódban.



*37. Ábra – A be-és kikapcsolási szekvencia, és az interrupt\_kiertekeles(); függvény*

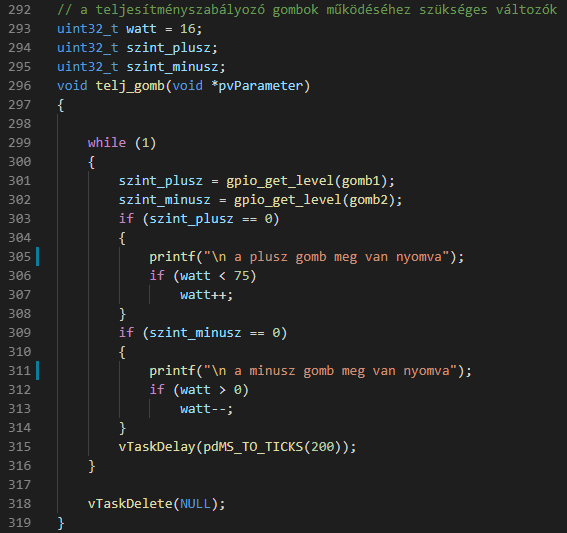
## Beállított teljesítmény visszajelzése, plusz és mínusz gombok

Az általunk tervezett elektromos cigaretta teljesítményszabályozása wattonként állítható a felhasználó által. Ezt a beállítást a felhasználó az eszközön található plusz és a mínusz gombok megnyomásával állíthatja, és az éppen aktuális beállított teljesítményt egy RGB LED jelzi vissza számára. A beállított teljesítmény minden bekapcsolás után 16 watt, ami egy olyan érték, amely elég univerzális, bármilyen ellenálláshuzalt is használjon a felhasználó.

### A plusz és mínusz gombok működése

Ahhoz, hogy a gombok működését megértsük, vissza kell tekinteni a szakdolgozat 8.1. pontjára, ahol a GPIO lábak inicializálása történt. A gombok működéséhez létre kellett hoznom három változót, valamint egy függvényt, amely majd egy feladatként fog 200 miliszekundumonként lefutni az *app\_main* függvényben, a FreeRTOS alatt. A három létrehozott változó a:

* *watt* – ez az aktuálisan beállított teljesítmény érték, a kezdeti értéke mindig 16
* *szint\_plusz –* ebbe a változóba kerül beírásra a plusz gomb aktuális értéke, amely mivel a belső felhúzó ellenállásra lett kötve (lsd.: *26.Ábra*), így amikor meg van nyomva 0 értéket ad vissza
* *szint\_minusz –* ez a változó ugyan arra szolgál, mint a *szint\_plusz*, azzal a különbséggel, hogy itt a mínusz gomb aktuális értéke kerül bele

**

*38.Ábra – A teljesítményszabályozó gombok változói és a telj\_gomb(); függvény*

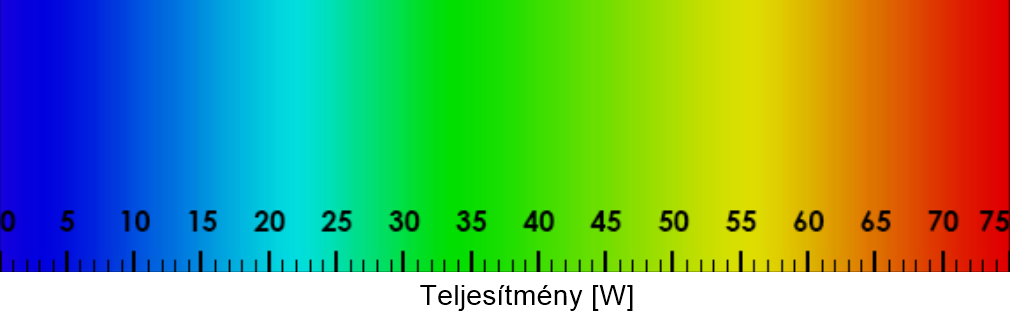
Az általam létrehozott függvény, a *telj\_gomb();*  függvény, amelyben egy while cikluson belül először a *gpio\_get\_level();* függvények kétszeri meghívásával folyamatosan kiolvassuk a két gomb éppen aktuális értékét, és azt beleírjuk az adott változóba, majd a függvény vizsgálja a változók értékét, és ha azt érzékeli, hogy a plusz gomb került megnyomásra, akkor megvizsgálja, hogy a *watt* változó aktuális értéke kisebb-e mint 75, azaz az általunk megszabott maximális teljesítmény. Ha kisebb, akkor megnöveli eggyel az értékét azonban, ha már elérte a 75 wattot, akkor nem csinál semmit a függvény.

Amennyiben a mínusz gomb került megnyomásra, megnézi, hogy a beállított teljesítmény nagyobb-e mint 0, és ha igen, akkor egyel csökkenti annak az értékét.

A függvényben található még két *printf();* függvény is, amelyek hibakeresési célt szolgálnak, a gombok megnyomása esetén a soros monitorra kiírják, hogy melyik gomb van megnyomva.

### A beállított teljesítmény visszajelzése

A beállított teljesítményt egy RGB LED adja vissza a felhasználónak. Ez a LED 0 watt beállított teljesítmény esetén kéken, és 75 watt esetén pirosan világít, a köztes értékeknél meg egy átmeneti színben kék és piros között, amint azt a *39.Ábra* is mutatja.



*39.Ábra – Az RGB LED színe különböző teljesítményeknél[18]*

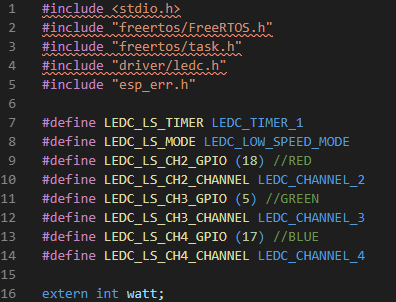
Ahhoz, hogy ezt elérjem, az RGB LED lábain impulzusszélesség modulált jelet (továbbiakban: PWM) kellett kiadnom, ezzel tudom szabályozni az egyes különálló LED-ek intenzitását. Ahhoz, hogy ezt a jelet ki tudjam küldeni, először inicializálnom kellett a PWM beállítást az adott GPIO lábakra. Ez, mivel egy viszonylag hosszú programrész, ezért egy külön C fájlban írtam le, ez az úgynevezett *rgb\_control.c.* Itt először meghívtam a szükséges könyvtárakat, többek közt a *ledc.h* könyvtárat, amely tartalmazza a pwm jel generálásához szükséges függvényeket és paramétereket. Ez után definiálnom kellett különböző értékeket, majd a *main.c*-ből át kellett hoznom különböző változókat, és csak ez után térhettem rá az *rgb\_control();* függvény leírására.

Az ESP 32 mikrovezérlőben 16 úgynevezett „led control” csatorna található, amelyeknek a fele nagy, míg a másik fele kis sebességű módban működik. Ezen módok között annyi a különbség, hogy amíg a nagysebességű módnál a kitöltési tényező változtatása lehet automatikus, és hardver implementált, addig az alacsony sebességű módban a kitöltési tényező változtatásának pillanata a szoftvertől függ.[19]

Jelen alkalmazásban én az alacsony sebességű módot választottam, mivel ennek a függvénynek a leírása nem igényli a nagysebességű mód előnyeit.

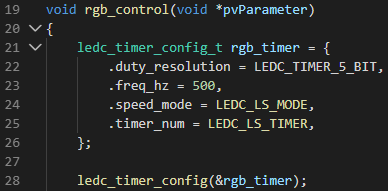
A *ledc.h* könyvtár meghívása után a következő értékeket definiáltam:

* *LEDC\_LS\_TIMER* – A pwm időzítőjének a kiválasztása, itt a *LEDC\_TIMER\_1*-re esett a választásom, ugyanis a 0 sorszámú időzítő már felhasználásra került egy másik programrészben
* *LEDC\_LS\_MODE* – Az időzítő alacsony, vagy magas sebességű módjának kiválasztása, jelen esetben *LEDC\_LOW\_SPEED\_MODE*.
* *LEDC\_LS\_CHx\_GPIO* – Az „x” sorszámú pwm csatorna GPIO-hoz rendelése, itt a 18-as, 5-ös és 17-es számú GPIOk-hoz lettek rendelve
* *LEDC\_LS\_CHx\_CHANNEL* – Az „x” sorszámú csatorna definiálása



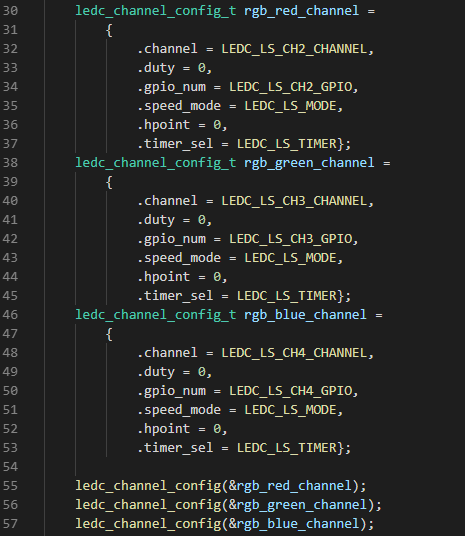
*40.Ábra – A könyvtárak meghívása, az értékek definiálása és a külső változó létrehozása*

Miután megtörténtek a definiálások, létrehoztam az *rgb\_control();* függvényt, amelyben első lépésként beállítottam a pwm generátor időzítőjét. Ehhez létrehoztam egy *ledc\_timer\_config\_t* típusú változót *rgb\_timer* néven, majd megadtam neki a kitöltési tényező felbontását, a pwm jel frekvenciáját, az időzítő sebességét, valamint az időzítő sorszámát. Ezután a *ledc\_timer\_config();* függvénynek megadtam bemeneti paramétereként az előbb létrehozott változót, amely így bekonfigurálja az időzítőt. A pwm jel frekvenciája jelen esetben 500Hz.



*41.Ábra – Az rgb\_control(); függvény létrehozása, és az időzítő beállítása*

Amikor megtörtént az időzítő beállítása, akkor következő lépésként a csatornákat kell egyesével bekonfigurálni, ehhez létrehoztam három *ledc\_channel\_config\_t* típusú változót, amelyeket elneveztem *rgb\_red\_channel, rgb\_green\_channel*  és *rgb\_blue\_channel*-nek. Ezekben a változókban adtam meg azt, hogy ezek melyik csatornán vannak, mi a kezdeti kitöltési tényezőjük, hányas GPIO lábra csatlakozzanak, milyen legyen a sebességük, a hpoint értékük, és azt, hogy melyik időzítőt használják. Ez után háromszor meghívtam a *ledc\_channel\_config();* függvényt, ahol bemeneti paraméternek rendre megadtam az előbb létrehozott három változót, ezzel beállítva a piros, kék és zöld csatornákat.

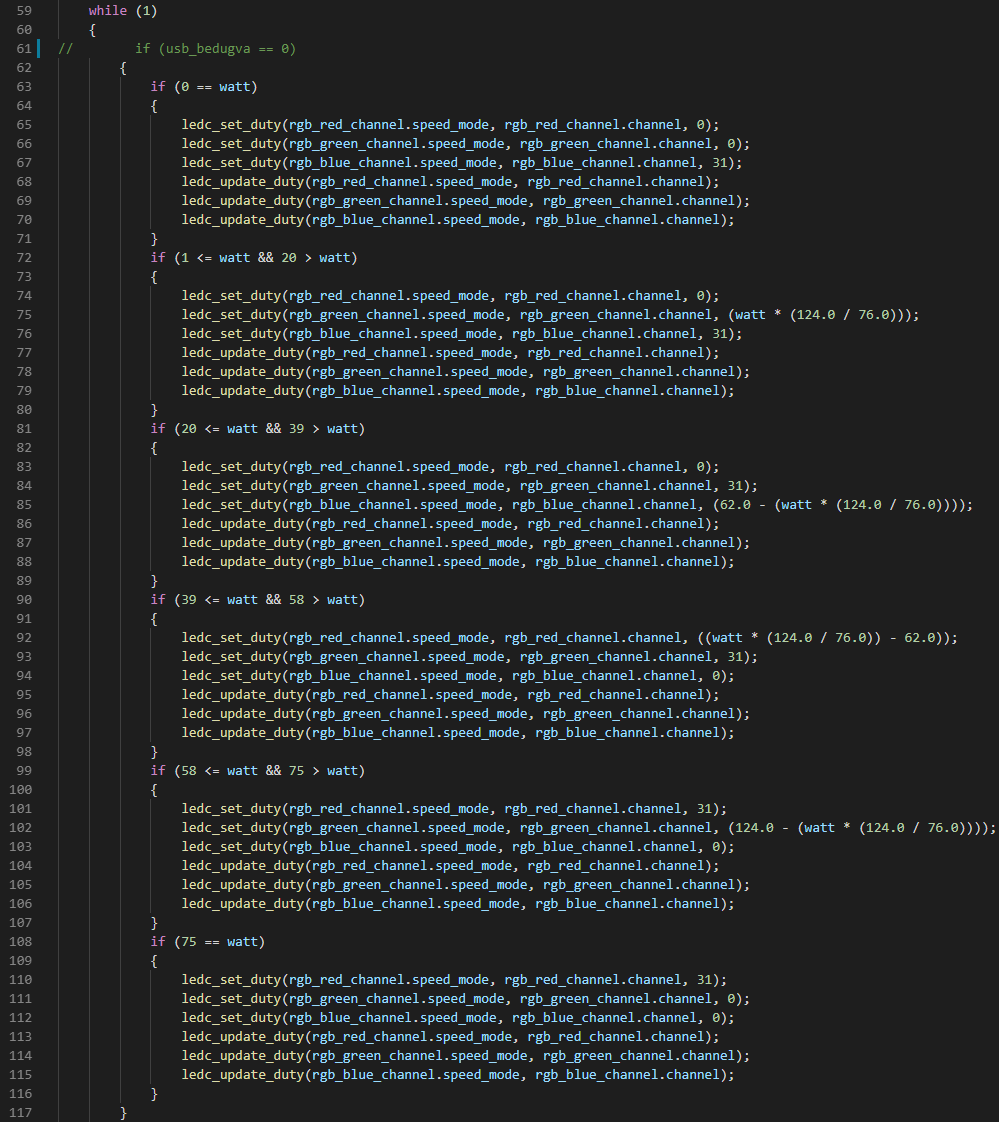


*42.Ábra – A pwm csatornák változóinak létrehozása és beállítása*

Következőleg létrehoztam egy *while* ciklust, amelyen belül különböző teljesítmény értékek esetén, a szoftver kiszámolja, hogy milyen kitöltési tényezővel kell a három csatornát vezérelni ahhoz, hogy megfelelő színnel világítson a LED. Ehhez több képletet és több *if* függvényt is alkalmaztam. Ha a beállított teljesítmény:

* 0 – akkor csak az kék LED világít, 100%-os kitöltési tényezővel, ami jelen esetben 31-et jelent, ugyanis minden csatorna felbontása 5bit
* 1 és 19 watt közötti – A kék led továbbra is 100%-on világít azonban a zöld LED fényereje folyamatosan nő, és 19 wattnál eléri a maximumát lásd: (3) képlet, azaz a LED cián színű lesz.
* 20 és 38 watt közötti – A zöld led marad 100%-on, azonban a kék LED folyamatosan halványul a (4) képlet alapján, így 38 wattnál már kizárólag a zöld LED fog égni.
* 39 és 57 watt közötti – A zöld LED továbbra is 100%-os kitöltéssel operál, azonban mellette a piros LED elkezd erősödni az (5) képlet alapján, majd 57 wattnál eléri a maximumát, ekkor a LED sárga színű lesz.
* 58 és 74 watt közötti – A piros LED 100%-on ég, miközben a zöld folyamatosan halványul a (6) képlet alapján, 74 wattnál már alig látható a piros szín
* 75 – Kizárólag a piros LED ég 100%-os kitöltési tényezővel.

Az *if* függvényeken belül hat függvény hívása történik meg, minden kimenetre kettő. Az egyik a *ledc\_set\_duty();*, amelynek meg kell adni, hogy melyik csatornának szeretnénk a kitöltési tényezőjét beállítani, valamint a csatorna sebességét, és a beállítani kívánt kitöltési tényezőt 0 és 31 között. A fenti képletek rendre ezeknek a függvényeknek a kitöltési tényező részébe kerültek beírásra. Amikor a *ledc\_set\_duty();* függvénnyel megváltoztatjuk a kitöltési tényezőt, ez még nem fog megjelenni a kimeneten, csak a mikorvezérlő elmenti az adatokat. Erre szolgál a *ledc\_update\_duty();* függvény, amely meghívásakor megváltoztatja a kimeneti pwm jel kitöltését az előzőleg beállított értékre.

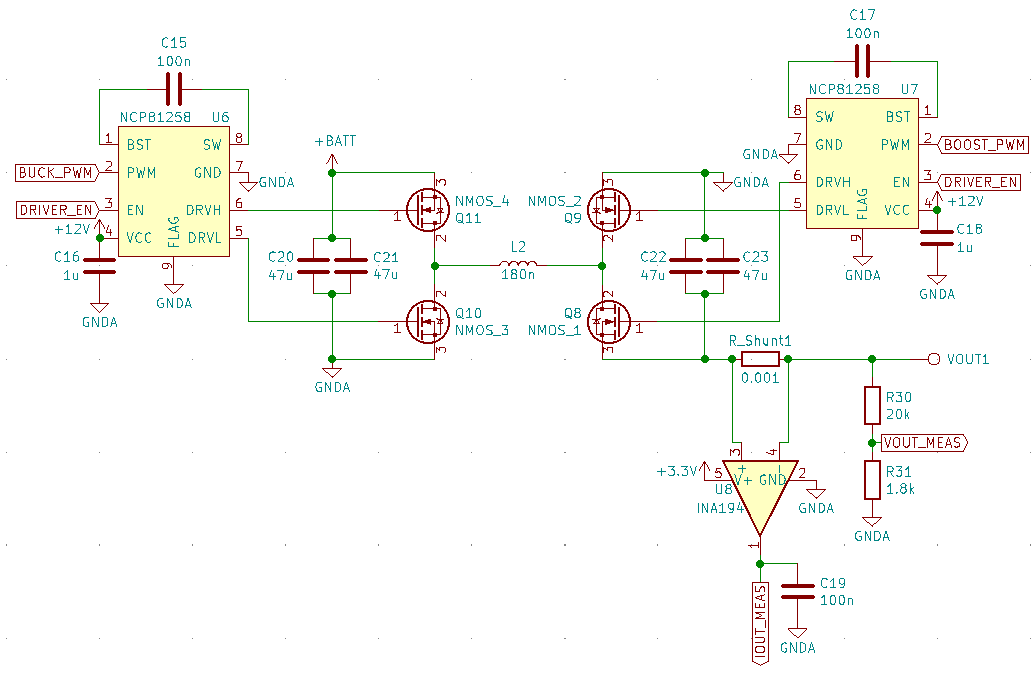


*43.Ábra – A while cikluson belüli kitöltési tényező állítások a (3)(4)(5)(6) képletek használatával*

A függvény további részei egy későbbi részben lesznek kifejtve. Ezt 100 miliszekundumonkénti futtatásra állítottam be, így a képen nem látszik, de a while ciklus végén a *vTaskDelay(pdMS\_TO\_TICKS(100));* függvény még meghívódik.

## A tűzgomb megnyomása utáni szekvencia, a teljesítményszabályozás

Az eszköz fő funkciója a tankban levő fűtőszál felhevítése. Ehhez egy szinkron feszültség csökkentő – növelő (továbbiakban buck-boost) kapcsolást alkalmaztunk, amelynek segítségével egy állandónak tekinthető, bár változó bemeneti feszültségből képesek vagyunk egy folyamatosan változó kimeneti feszültség létrehozására. A szinkron üzemű buck-boost konverterek a hagyományosakkal ellentétben, kettő helyett négy MOSFET-et tartalmaznak, és ez nagyban megnöveli azoknak a hatékonyságát.



*44.Ábra – Az általunk használt szinkron buck-boost konverter, méretezve, és a MOSFET vezérlők, valamint az árammérő IC*[18]

A kapcsolásban szereplő NMOS FET-ek ki-és bekapcsolásával lehet elérni azt, hogy a bemeneti feszültség egy meghatározott értékre nőjön, vagy csökkenjen. Mivel ezeknek a MOSFETeknek a kapcsolási feszültsége 12V, ezért szükség volt még két MOSFET vezérlő IC-re is, amelyek előállítják ezt a feszültséget, valamint mivel van egy magas és egy alacsony kimenetük, ezért biztosítják azt, hogy az ellenütemben vezérelni kívánt MOSFET-ek esetén ne legyen arra lehetőség, hogy azok egyszerre nyitva, valamint egyszerre zárva vannak. A vezérlő IC-k bemenete egy-egy pwm jel, valamint egy-egy engedélyező jel.

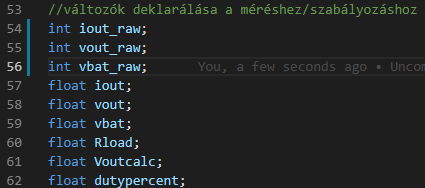
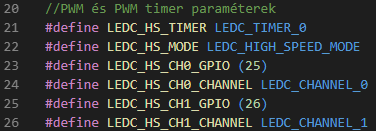
Az árammérő IC egy söntön keresztül méri az azon átfolyó áramot, és azt egy maximum 1100mV-os feszültséggé alakítja, melyet így egy ADC lábon az ESP32 vissza tud mérni, a kimeneti és a bemeneti feszültség egy-egy feszültségosztó kapcsolással van leosztva úgy, hogy a maximális mikrovezérlő lábán eső feszültség ne haladja meg az 1100mV-ot.[18]

A fenti információk birtokában készítettem el a szoftvert, pontosabban azt a függvényt, amely a teljesítményszabályozást megvalósítja. Ez az úgynevezett *telj\_szabalyozas();* függvény.

### A *telj\_szabalyozas();* függvény leírása

A függvény működéséhez először a, már 8.4 és 8.2 részben kifejtett, pwm generátor és ADC lábakat kellett inicializálnom, valamint a szükséges értékeket definiálnom és a függvényhez szükséges változókat létrehoznom.

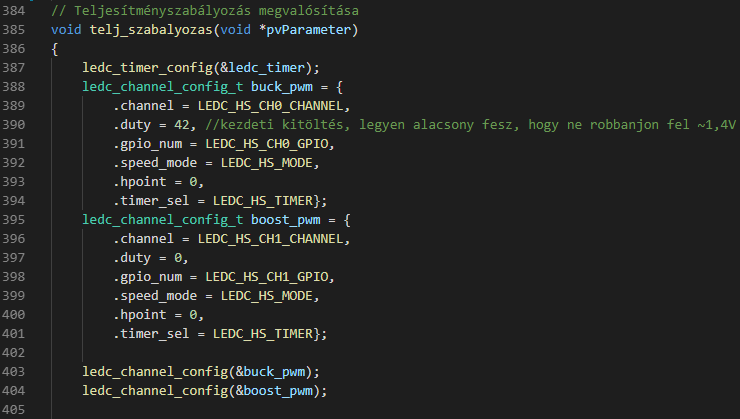
A pwm generátorhoz használt időzítő jelen esetben a 0 sorszámú, és ez magas sebességű módba lett állítva, ugyanis itt fontos a gyors reakcióidő. A pwm jel kiadására szolgáló két csatorna itt a 0 és 1-es, amelyek rendre a 25-ös és a 26-os GPIO lábhoz vannak csatlakoztatva. A kapcsolás méretezéséből adódóan, a kapcsolási frekvenciát 500kHz-re kellett állítanom, azonban így a kitöltési tényező felbontása maximálisan 7 bit lehet, azonban ez elegendő a wattonkénti állításhoz.



*45.Ábra – A pwm jelhez szükséges definíciók és a változók létrehozása a mérésekhez és számításokhoz*

Létrehoztam egy *ledc\_timer\_config\_t* típusú változót *ledc\_timer* néven, amelyben megadtam a 8.4 részben már kifejtett értékeket, valamint a frekvenciát beállítottam 500kHz-re és a kitöltés felbontását 7 bitre. Ez után a *telj\_szabalyozas();* függvényben meghívtam a *ledc\_timer\_config();* függvényt, amivel beállítottam a pwm időzítőjét ezekre a beállításokra. Miután ez megtörtént szintén létre kellett hogy hozzak két *ledc\_channel\_config\_t* típusú változót, az egyik a *buck\_pwm* nevet kapta, és ez fogja vezérelni a *44.Ábrán*ak a bal oldalán látható MOSFET vezérlő IC-t, és így a feszültségcsökkentő kapcsoláshoz használt két MOSFET-et. A másik változó a *boost\_pwm* nevet kapta, amely a másik vezérlőt és két MOSFET-et vezérli majd.

A *buck\_pwm* csatornát úgy állítottam be, hogy az alapértelmezett kitöltése az legyen 42 értékű, azaz ~33%-os kitöltésű, míg a *boost\_pwm* 0 értéket kap, így amikor meghívódik ez a függvény és a *ledc\_channel\_config();* függvények beállítják a két pwm csatornát, a buck-boost konverter egy körülbelül 1,4V feszültséget fog előállítani a kimeneten. Erre azért van szükség, mivel így lesz egy kezdeti visszamérhető feszültség és áramerősség érték, amelynek segítségével ki tudjuk számolni a fűtőszál ellenállását. Ezek a függvény meghívásakor csak egyszer fognak lefutni, ezzel beállítva a kívánt értékeket.

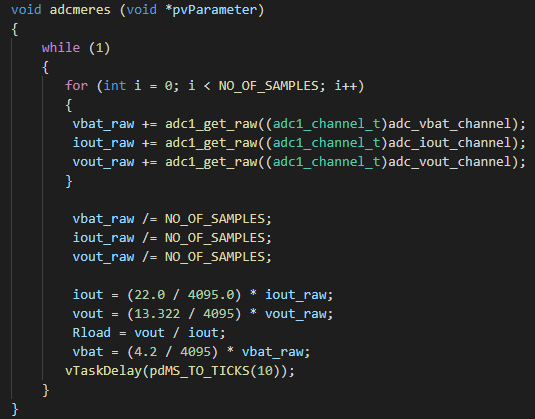


*46.Ábra – A telj\_szabalyozas(); függvény egyszer lefutó részei, a pwm generátor konfigurálása*

Az ADC lábakon történő mérésekhez létrehoztam egy ciklikusan lefutó függvényt, *adcmeres();* néven, ugyanis a program további részeiben is szükség lesz ezekre a mért értékekre, és így tudtam elérni azt, hogy ezek mindig az aktuálisak legyenek és ne csak akkor amikor a tűzgomb lenyomásra kerül. Itt először a program a 8.2 részben már bemutatott többszörös mintavételezési technikával megméri a három ADC lábon a kimeneti áramot, a kimeneti feszültséget, valamint az akkumulátor feszültségét és ezeket a mért értékeket beleírja az ezeknek létrehozott változókba, rendre az *iout\_raw, vout\_raw* és *vbat\_raw-ba,* majd ezeknek az értékét elosztja a vett minták számával, így visszakapva egy 0 és 4095 közötti nyers integer típusú értéket. Ha ez megtörtént, a program ezen nyers adatokat felhasználva rendre visszaszámolja a tényleges áram és feszültség értékeket, amelyet egy egyszerű arányossággal számol ki, és a kimenet egy Amper vagy egy Volt érték lesz, ezeket rendre beleírja az *iout, vout,* és *vbat* lebegőpontos(float) típusú változókba. Ezek után, a kimeneti áram és feszültség értékekből a program egy egyszerű Ohm-törvénnyel visszaszámolja a fűtőszál ellenállás értékét Ohm-ban, és ezt is beleírja az erre létrehozott lebegőpontos típusú *Rload* nevű változóba.

Az ezt követő lépés az, hogy ki kell számolni a beállított teljesítmény eléréséhez szükséges feszültség értékét úgy, hogy az éppen aktuális ellenállását veszi figyelembe a fűtőszálnak. Ehhez a (7) képletet használjuk fel, amely a teljesítményszámítás képletének az átalakított verziója.

Az így kapott feszültség értéket a *Voutcalc* változóba mentem el, amely szintén lebegőpontos típusú.



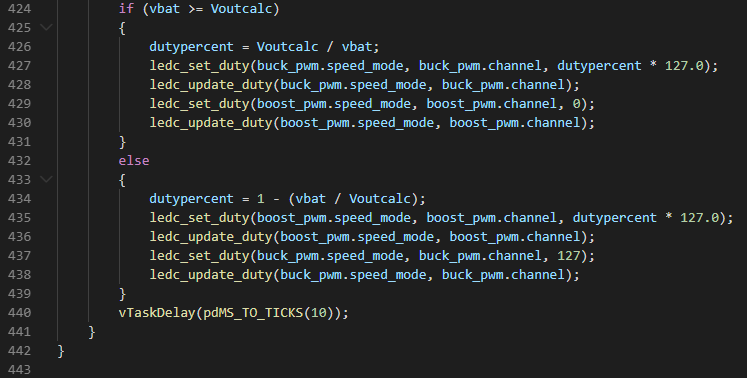
*47.Ábra – Az ADC lábakon való mérések, és a teljesítményszabályozáshoz szükséges számítások elvégzése az adcmeres(); függvényben*

Ez után a *telj\_szabalyozas();* függvény megvizsgálja, hogy az elérni kívánt feszültség érték kisebb, nagyobb vagy egyenlő az akkumulátor feszültségével, vagyis hogy a bemeneti feszültség növelésére, csökkentésére vagy szinten tartására van-e szükségünk. Amennyiben csökkenteni szeretnénk, akkor a *buck\_pwm* csatorna kitöltési tényezőjének a változtatására van szükség, míg a *boost\_pwm* csatorna folyamatosan 0 jelet kell, hogy kiadjon magából. Azt, hogy a kitöltési tényező hány százalékos kell legyen, azt a (8) képlet fogja meghatározni.

Ez a képlet egy 0 és 1 közötti számot fog visszaadni, amelyet fel kell szorozni a maximális kitöltési tényező értékével. Mivel 7 bites a felbontása a kitöltési tényező változtatásának, ezért a kitöltés értéke 0 és 127 között változhat, így a kiszámított D értéket 127-tel kellett megszoroznom.

Ha a bemeneti feszültséget növelni kívánjuk, akkor a *boost\_pwm* csatorna kitöltési tényezőjét kell változtatni, és a *buck\_pwm* csatornát folyamatos 1-re állítani, azaz 127-es kitöltésre. A *boost\_pwm* kitöltésének változtatását a (9) képlet írja le, és az azzal visszakapott 0 és 1 közötti értéket, melyet a *dutypercent* nevű változóba íratok bele, fel kell szorozni 127-tel, így visszakapva a szabályozáshoz szükséges értéket.

Ezeket a 8.4-es részben kifejtett *ledc\_set\_duty();* és *ledc\_update\_duty();* függvények kombinációjával állítom be az éppen változtatni kívánt csatornának.



*48.Ábra – A függvény megvizsgálja, hogy növelni vagy csökkenteni kell a feszültséget és a megfelelő képlet alkalmazásával beállítja a kívánt kitöltési tényezőt.*

A *while* ciklus végén ismét megtalálható a *vTaskDelay(pdMS\_TO\_TICKS(10));* függvény, amely a FreeRTOS számára jelzi, hogy amíg a függvény feladatként létezik, 10 miliszekundumonként ismételje meg a futtatást.

### A függvény futtatása megszakítás hatására

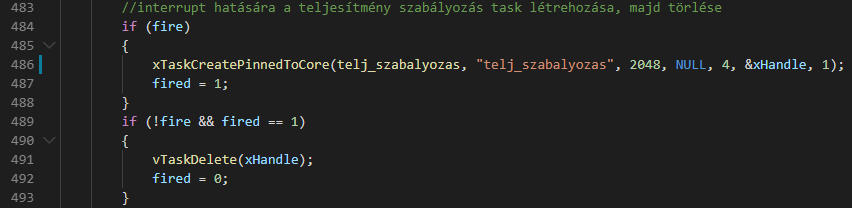
A 8.3 részben kifejtettem, hogy a tűzgombhoz kapcsolt interrupt milyen folyamatokat indít el a gombnyomás hatására, azonban arra a részre nem tértem, hogy maga a hevítés hogyan valósul meg a megszakítás hatására.

Amikor az *interrupt\_kiertekelese();* függvény kap egy megszakítást a tűzgombról, akkor megnézi a gomb pillanatnyi értékét. Ha ez az érték „megnyomott”, azaz 0, akkor egy flag bitet, amelyet *fire*-nek neveztem el, 1-be billent. Ha azonban a gomb értéke „nem megnyomott”, azaz 1, akkor a *fire* flag bit értéke 1-re áll be. Ez a *37.Ábrán* látható.

A *fire* bit az *app\_main();*, azaz a fő függvény *while* ciklusán belül kerül feldolgozásra. A függvény megvizsgálja, hogy a *fire* bit éppen aktuális értékét.

Amennyiben azt látja, hogy 1, az *xTaskCreatePinnedToCore();* függvénnyel, létrehozza a *telj\_szabalyozas* elnevezésű feladatot, valamint a *fired* bitet 1-be állítja. Ez a bit arra szolgál, hogy visszajelezze azt, hogy a feladat létre lett-e hozva, vagy nem. Az így létrehozott feladat egy korábban létrehozott *TaskHandle\_t* típusú *xHandle* nevű „kezelő” változóval van ellátva, és amíg a feladat törlésre nem kerül, addig ciklikusan, 10miliszekundumonként futtatja a *telj\_szabalyozas();* függvény *while* ciklusban levő részét, miután egyszer lefuttatta a *while*-on kívüli részt.

Amennyiben az *app\_main();* függvény azt látja, hogy a *fire* bit 0, és a *fired* bit 1, azaz a feladat már létre lett hozva, azonban érkezett egy megszakítás, amely a *fire*-t 0-ba állította, a program a *vTaskDelete();* függvénnyel törli a *telj\_szabalyozas* nevű feladatot és a *fired* bitet visszaállítja 0 állapotba. A *vTaskDelete();* függvénynek a bemenete az *xHandle* változó.



*49.Ábra – Megszakítás hatására a teljesítmény szabályozás feladat létrehozása, majd törlése*

## Túlhevülés védelem, USB-töltés lekezelése, engedélyező jelek kiadása, mélymerülés elleni védelem

Az engedélyező jelek küldése vagy megszakítása több tényezőtől függ. Először a különböző engedélyező jeleket szeretném bemutatni, majd azt, hogy ezek hogy függnek az eszköz bemeneteire érkező különböző értékektől.

### Engedélyező jelek bemutatása:

Az eszköz 4 GPIO lába, mint azt már a 8.1 részben kifejtettem, engedélyező kimenetekként vannak definiálva. Az ESP32 általunk használt lábkiosztása a *27.Ábrán* látható. A kiküldeni kívánt engedélyező jelek a következők:

* *en\_3v3 –* GPIO 22 – A 3,3V-ot előállító feszültségstabilizátor engedélyező jele, ez amennyiben 1, akkor kap tápfeszültséget a mikrovezérlő
* *en\_12v –* GPIO 21 – A 12V-ot előállító feszültségnövelő IC engedélyező jele, amennyiben ez 1-et kap, az IC 12V-ot állít elő a kimenetén. Ez például a MOSFET vezérlő ICk tápfeszültség ellátását szolgálja
* *en\_driver –* GPIO 23 – A MOSFET vezérlő ICk engedélyező jele, ha 1, akkor az ICk üzemelnek.
* *overheat –* GPIO 19 – Az akkumulátor töltő IC-nek szükséges. Amennyiben ez 1, akkor az IC megszakítja az akkumulátor töltését.

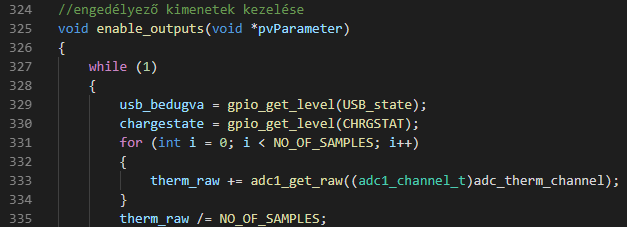
### *enable\_outputs();* függvény leírása

Az *enable\_outputs();* függvényben történik az engedélyező jelek különböző bemenetke esetén történő letiltása vagy engedélyezése. A függvény egy *while* ciklust futtat 10miliszekundumonként, és ezen a cikluson belül ellenőrzi a bemeneteket és állítja át a kimeneteket.

Először az *usb\_bedugva* változóba beleírja a 13-as GPIO láb értékét, amely kizárólag akkor lesz 1 állapotú, amikor az eszköz egy bedugott USB kábelen keresztül megkapja az 5V feszültséget.

Következő lépésként a *chargestate* változóba írja bele a 4-es GPIO-n olvasott értéket. Ezt az akkumulátor töltő IC állítja elő, amennyiben 1, az akkumulátor tölt, és amikor 0, akkor az akkumulátor teljesen feltöltött.

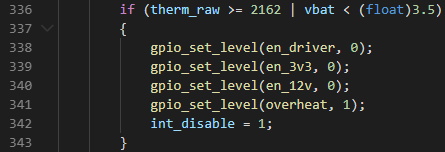
Ez után következő lépésként a már fent többször említett módon, többszörös mintavételezéssel a 33-as ADC lábon olvasott adatot írja bele a függvény a *therm\_raw* változóba, ez a láb egy thermisztorra és egy ellenállás közé van kötve, amely az eszköz hőmérsékletének változásának hatására más-és más mértékben osztja le a 3,3V-os tápfeszültséget.



*50.Ábra – A GPIO lábak és az ADC láb kiolvasása, és a kiolvasott értékek változókba írása*

Az első vizsgált esetben a függvény megnézi, hogy az eszköz túlhevült-e, vagy hogy az akkumulátor feszültsége nem esett-e le 3,5V alá. Az akkumulátor feszültsége a *telj\_szabalyozas();* függvényben kerül megmérésre. A thermisztor adatlapja szerint 80°C-on az ellenállása 1,67kΩ, a feszültségosztóban szereplő másik ellenállás 470Ω, így ezen a hőmérsékleten a leosztott feszültség a (10) képlet [18] alapján 0,725V, ami a visszaolvasott nyers értékre átváltva 2162-t jelent a 12bites felbontású ADC lábon.[18]

Amennyiben a thermisztoron olvasott érték eléri vagy meghaladja a 2162-t, valamint azt látja, hogy az akkumulátor feszültsége 3,5V alá csökkent a *gpio\_set\_level();* függvény minden GPIO lábra történő meghívásával az összes kimenetet 0-ba állítja, az *overheat*(GPIO19) láb kivételével, amit 1-re állít. Ezen kívül egy *int\_disable* nevű flag bit értékét 1-re állítja. Ez a változó lesz majd felelős a tűzgombról érkező megszakítások letiltásáért.



*51.Ábra – A túlhevülés és mélymerülés elleni védelem*

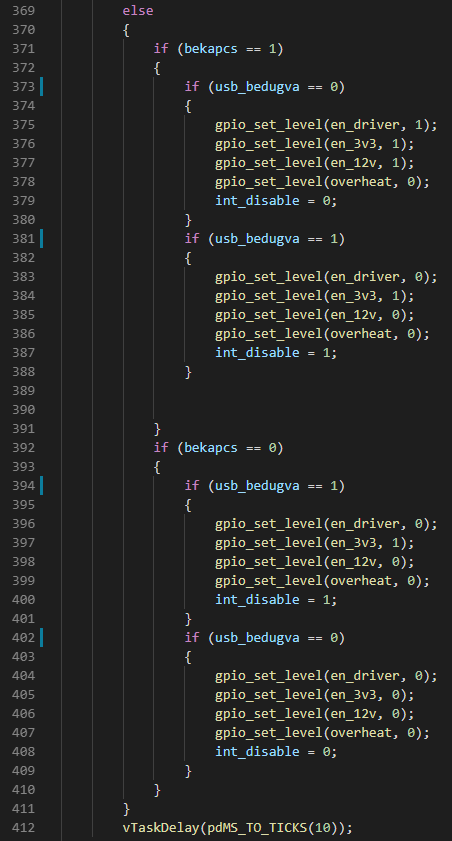
Ha az eszköz nincs túlhevülve, és az akkumulátor feszültsége is 3,5V fölött van, akkor a függvény megvizsgálja a 8.3 részben már bemutatott *bekapcs* bit állapotát.

Ha a *bekapcs* bit 1, és az *usb\_bedugva* bit 0, akkor a program az összes engedélyező kimenetre 1 értéket ad, az overheat kivételével, amely ekkor 0 lesz, valamint az *int\_disable* flaget 0-ba billenti.

Amikor az *usb\_bedugva* bit 1, akkor a mikrovezérlő kizárólag a 3,3V-os feszültségstabilizátor engdélyező jelét adja ki, az összes többi 0-ba vált és az *int\_disable* bit 1-be áll, azaz a program letiltja a tűzgombról érkező megszakításokat.

Amikor a *bekapcs* bit 0, viszont az *usb\_bedugva* bit 1, akkor szintén kizárólag az *en\_3v3* elnevezésű GPIO láb küld 1-es jelet, valamint a megszakítás ebben az esetben is le van tiltva a tűzgombról.

Az *usb\_bedugva* bit 0 és a *bekapcs* bit továbbra is 0, akkor egyetlen engedélyező jel sincs kiküldve, viszont a tűzgombról érkező megszakítások engedélyezve vannak.

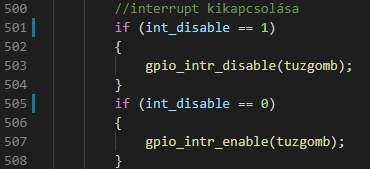


*52.Ábra – Az engedélyező jelek kiadása a bekapcs és az usb\_bedugva bitek függvényében*

### Tűzgombról érkező megszakítások letiltása és engedélyezése

Az *app\_main();* függvény *while* ciklusában vizsgálom, hogy az *int\_disable* bit milyen értéken van. Amennyiben az értéke 1, akkor a *gpio\_intr\_disable();* függvény meghívásával a tűzgombról érkező megszakításokat letiltom, így a felhasználónak nem lesz lehetősége például töltés közben fűteni a fűtőszálat, valamint ki-és bekapcsolni az eszközt. Amennyiben az *int\_disable* bit értéke 0, akkor nincs szükség erre az óvintézkedésre, így a *gpio\_intr\_enable();* függvénnyel engedélyeztetem a megszakításokat.

Mind a *gpio\_intr\_disable();,* mind a *gpio\_intr\_enable();* függvénynek a bemeneti paramétere a változtatni kívánt GPIO száma, jelen esetben ez a definiált *tuzgomb* nevű állandó.



*53.Ábra – A tűzgomb megszakításának letiltása és engedélyezése*

### A Chargestate bemenet kiértékelése

Az *rgb\_control.c* fájlban. az *rgb\_control();* függvényben megvizsgáltatom, hogy az USB be van-e dugva. Amennyiben nincs, a függvény a 8.4 részben leírtaknak megfelelően működik.

Azonban, ha be van dugva az USB, a függvény megvizsgálja, hogy a *chargestate* külső változó értéke mennyi. Amennyiben 1, azaz az akkumulátor tölt, akkor az RGB LED-pwm csatornáinak a beállítása úgy történik meg, hogy a piros és a kék csatornán 100%-os kitöltésű pwm jel jelenik meg, azaz a LED magenta színnel fog világítani.

Abban az esetben, ha a *chargestate* változó értéke 0, tehát az akkumulátor feltöltött, az RGB LED kizárólag a zöld csatornán fog 100%-os kitöltésű jelet kapni, míg a többi csatorna 0%-os lesz, így a LED zöld színnel fog világítani.

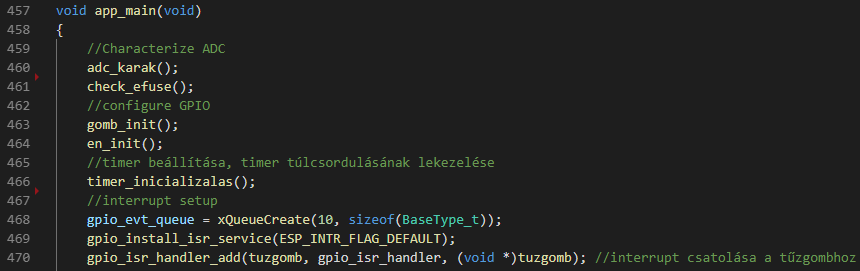
## Az *app\_main();* függvény leírása

A mikrovezérlő kizárólag az *app\_main();-*t és az abban meghívott függvényeket futtatja, így mondhatni, hogy ez a teljes szoftvernek a lelke ebben a függvényben kell meghívni az inicializáló és az egyszer lefutó függvényeket, itt kell létrehozni a FreeRTOS által ciklikusan futtatott feladatokat, valamint az itt létrehozott „végtelen” *while* ciklus lesz az, amely külön feladat létrehozás nélkül fut le újra és újra.

Az *app\_main();* függvényben először meghívtam az inicializáló függvényeket, úgy mint:

* *adc\_karak(); -* Az adc lábak karakterizációja, és inicializálása. Lásd:8.2. rész
* *check\_efuse(); -* Hibakereséshez használt függvény, az ADC beállításait íratja ki. Lásd 8.2. rész
* *gomb\_init(); -* A bemeneti GPIO-k inicializálása. Lásd 8.1. rész
* *en\_init(); -* A kimeneti GPIO-k inicializálása. Lásd 8.1. rész
* *timer\_inicializalas(); -* Az időzítő inicializálása. Lásd 8-3. rész

Itt állítottam be és hoztam létre a GPIO-ról történő megszakításokhoz szükséges beállításokat, valamint változókat. Lásd 8.3. rész



*54.Ábra – Az inicializáló függvények és a megszakítás beállításai*

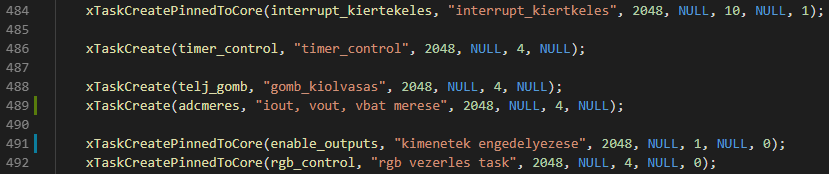
Ha ezek a beállítások megtörténtek, itt hoztam létre a FreeRTOS számára a feladatokat, melyek majd ciklikusan és folyamatosan futni fognak, amint a mikrovezérlő elindul. Ezeket a függvényeket még létrehozásukkor egy *\*pvParameter* pointerrel kell ellátni.

A feladatok létrehozását két módon oldottam meg, az egyik, az *xTaskCreate();* függvény meghívásával. Ennek a függvénynek meg kell adni rendre azt, hogy:

1. melyik függvényből csináljon feladatot
2. mi legyen a feladat elnevezése,
3. mekkora legyen a feladatnak létrehozott veremtár mérete
4. egy mutató, amely annak azt paramétert tartalmazza, hogy a feladat létre lett hozva
5. a feladat prioritását
6. azt a handlert, amellyel referálni lehet a későbbiekben a létrehozott feladatra

A legtöbb feladat létrehozásánál a 4. és 6. paraméterre nincs szükség, így azokat NULL értékkel helyettesítettem.

A másik módja a feladat létrehozásának az *xTaskCreatePinnedToCore();* függvény meghívásával történik. Ennek a függvénynek megegyeznek a bemeneti paraméterei az előzőjével, azzal az egy különbséggel, hogy itt meg kell adni egy 7. pontot is, amellyel azt tudjuk megadni a függvénynek, hogy ezt a feladatot melyik processzormagon futtassa.



*55.Ábra – Feladatok létrehozása*

Ezek után már csak az *app\_main();* függvényben levő *while* ciklus maradt, amely a feladatokhoz hasonlóan folyamatosan fut. Itt egy pár hibakeresésre szolgáló kiíratáson kívül csak olyan dolgok vannak, amelyeknek a működését már taglaltam korábban a 8.5. és a 8.6. részben.



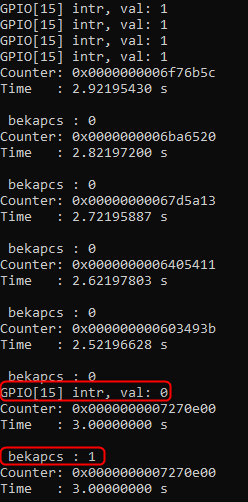
*56.Ábra – Az app\_main();-ben levő while ciklus*

# A szoftver tesztelése

A szoftver teszteléséhez a rendelkezésemre álló ESP32-DevkitV1 fejlesztőkártyát használtam, amelynek a lábait egy próbapanelre kötöttem, és különböző rendelkezésemre álló komponensekkel igyekeztem minél realisztikusabban leszimulálni a felhasználás tényleges körülményeit. Azonban, mivel ez a szoftver egy beágyazott rendszer programkódja, így ez egy specifikusan erre a célra tervezett hardver nélkül a tesztelés problémás és pontatlan eredményeket adhat, viszont egy áttekintést és egy alapvető visszajelzés kinyerhető az általam létrehozott szimulált környezettel is.

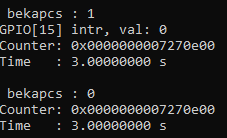
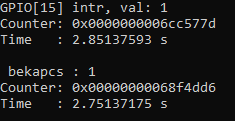
## A be-és kikapcsolás tesztelése

Ez a legnehezebben tesztelhető része a szoftvernek, ugyanis a rendelkezésemre álló eszközök segítségével nem tudtam a gombnyomásokat a végső hardverhez hasonlóan reprodukálni. A gomb megnyomásakor nagyon sok mechanikus zaj keletkezik, amelyeket a szoftver mind külön gombnyomásnak érzékel, ezeket igyekeztem kiszűrni egy alul áteresztő szűrő segítségével, azonban ezzel sem jártam sikerrel, így pontos teszteredményeket nem tudok bemutatni, azonban azt, hogy a tűzgombról érkező megszakítás hatására elindul az időzítő, valamint azt, hogy ha megtörténik az öt gombnyomás, akkor a *bekapcs* bit átbillen, ez látható a soros monitorra kiírt értékek alapján.



*57.Ábra – A soros monitorra kiíródik az időzítő éppen aktuális állapota, valamint a bekapcs bit értéke és a tűzgombról érkező megszakítások*

Az *57.Ábrán* látható, hogy először négy megszakítás érkezik egyszerre nagyon rövid időn belül a gomb pergésének köszönhetően, amely elindítja a számlálót, majd az ötödik megszakítás érkezése után, a *bekapcs* bit 1-be áll, és az időzítő megáll és visszaáll az eredeti értékére.

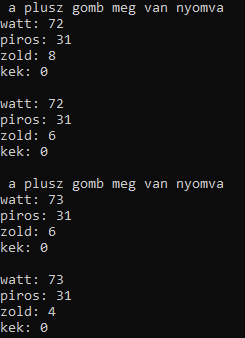
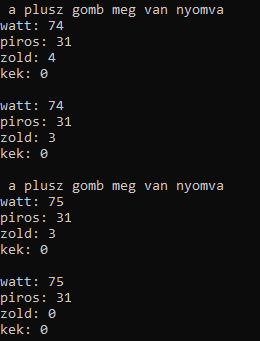


*58.Ábra – A soros monitoron látszódik a kikapcsolás*

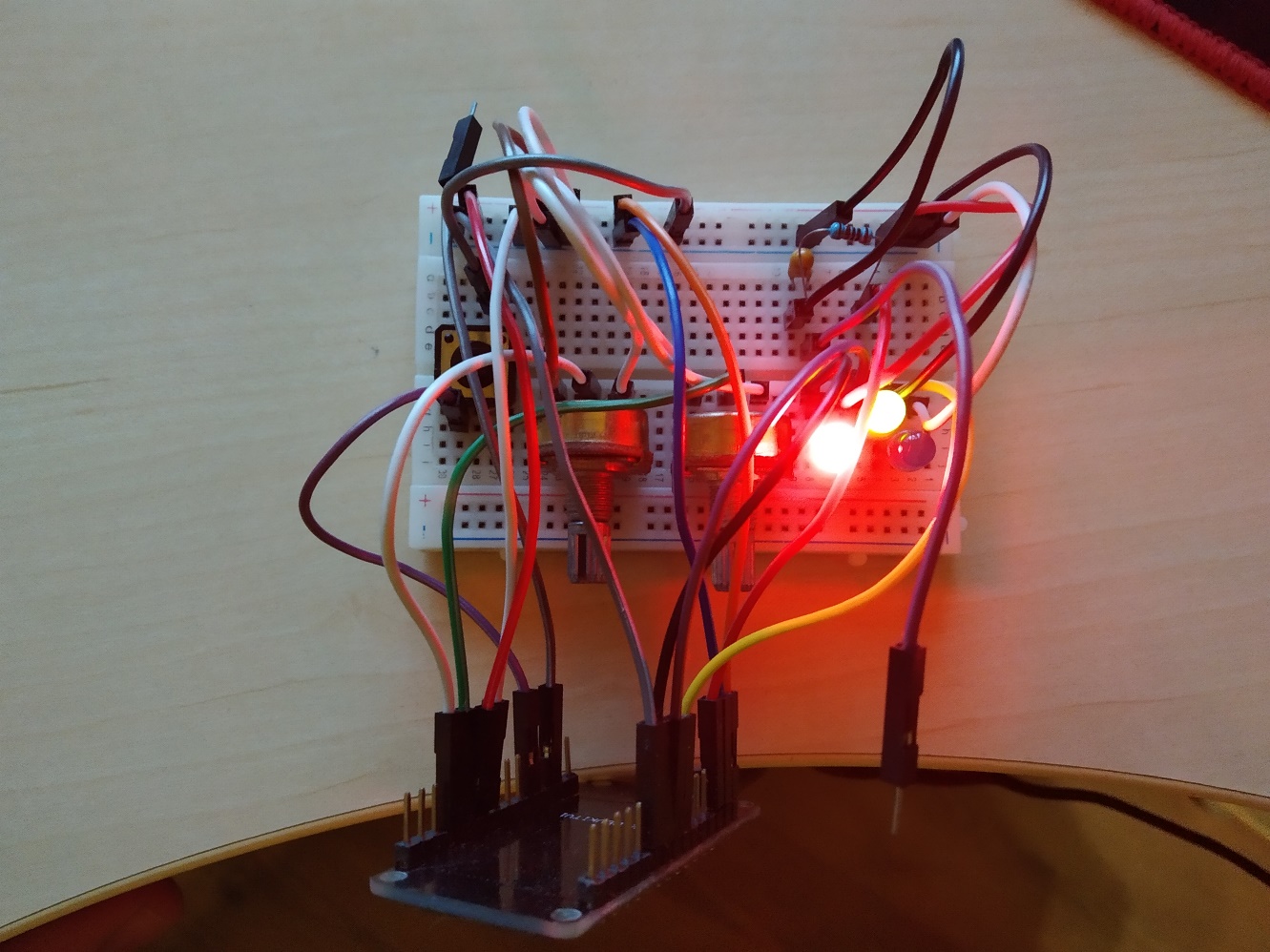
Az *58.Ábrán* látszódik, hogy a kikapcsolási szekvencia is ugyanúgy működik, mint a bekapcsolás. A bekapcsolt eszközre beérkezik a megszakítás, amely elindítja a számlálót (bal oldal), majd az 5. érkező megszakítás után a számláló leáll, és visszaáll az eredeti értékére, valamint a *bekapcs* bit 0-ba áll be.

## A beállított teljesítmény visszajelzésének a tesztelése

Ehhez a tesztlépéshez összeállítottam egy, a végső hardverhez hasonló, kapcsolást, ahol egy piros egy zöld és egy kék LED-et használtam, valamint egy gombot, amelyet attól függően, hogy növelni vagy csökkenteni szeretném a beállított teljesítményt, a GPIO12 vagy GPIO14 lábra kötöttem, és ezek mellett a *printf();* függvénnyel a soros monitorra is kiírattam azt, hogy épp melyik gomb van megnyomva, valamint az aktuálisan beállított teljesítményt és azt, hogy így milyen intenzitással világítanak a LED-ek.

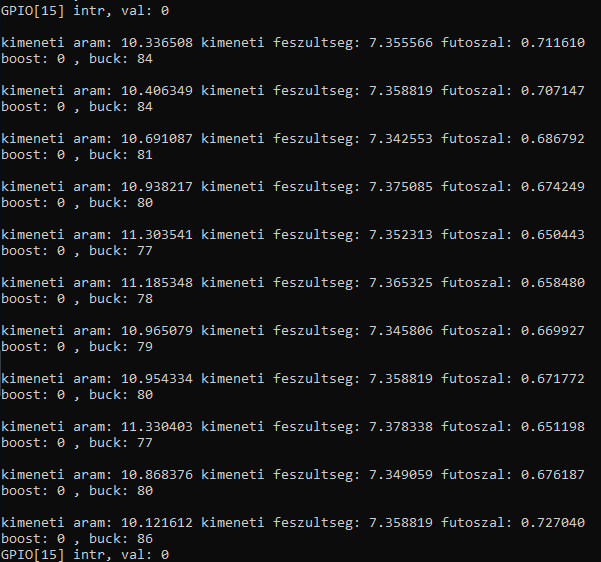
*59.Ábra – A soros monitoron látszódik a gombnyomás hatására megnő a teljesítmény, és a szoftver beállítja a kitöltési tényezőt.*



*60.Ábra – Látszódik, hogy 72W beállított teljesítménynél a kék LED nem ég, míg a zöld nagyon halványan és a piros maximálisan*

## A teljesítményszabályozás tesztelése

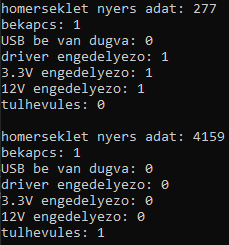
A *60.Ábrán* látható két potenciométert rendre a GPIO34 és a GPIO35 lábra kötöttem, melyek a kimeneten folyó áram és eső feszültség visszamérésére szolgálnak. A GPIO32-es láb folyamatos tápfeszültségen van, így az mindig 4,2V körüli értéket fog érzékelni. A *printf();* függvény meghívásával ennél a tesztlépésnél is soros monitorra írattam ki az éppen aktuális kimeneti áram és feszültség értékeket, a fűtőszál ellenállását, valamint a GPIO25-és 26 kimeneteken levő PWM jel kitöltési tényezőit. A két potenciométer eltekerésével látható a kimeneteken a mért értékek változása, valamint a kimeneti PWM jelek kitöltési tényezőinek a változásai, valamint az, hogy a teljesítményszabályozás csak akkor fut, amikor a tűzgomb meg van nyomva.



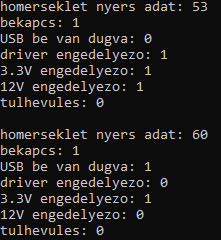
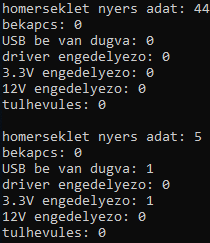
*61.Ábra – A soros monitoron a változó ellenállás hatására változó kitöltési tényezők 16Watt beállított teljesítmény esetén*

## Az engedélyező kimenetek tesztelése, és a töltés állapot kijelzése

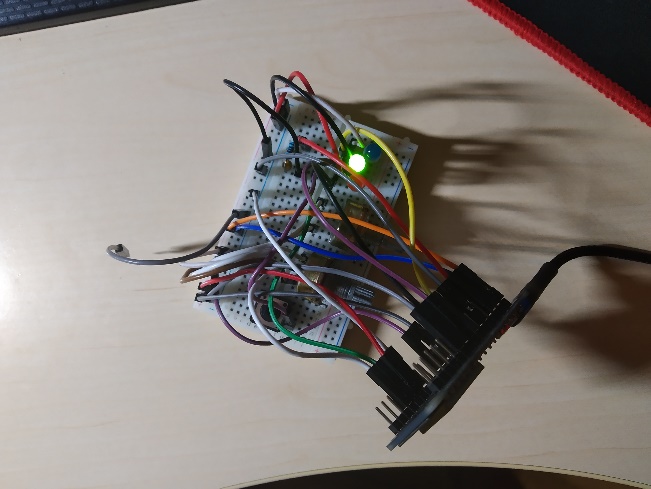
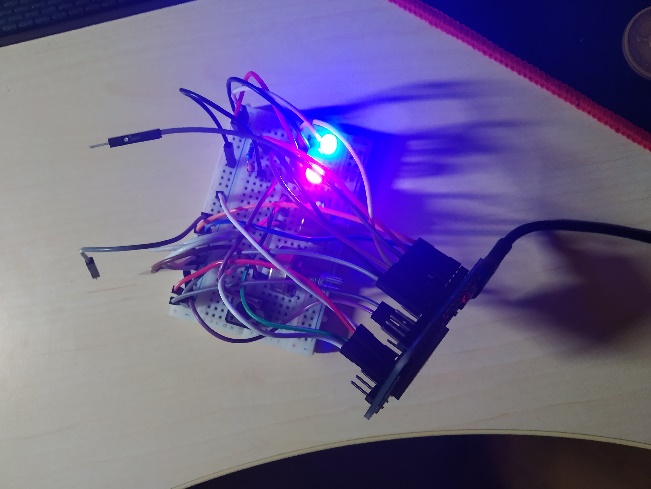
Az engedélyező kimeneteket úgy vizsgáltam, hogy a soros monitorra kiírattam az értékeit, majd először a thermisztor lábát földre, majd tápfeszültségre húztam. A *62.Ábra* is mutatja, hogy amint a thermisztor túl magas hőmérsékletet érzékel, a kimeneti engedélyező jeleket azonnal letiltja, valamint kiküldi a túlhevülés jelet. A többi funkciót amelyet a 7.6 részben leírtam, szintén leteszteltem, ezeket a *63. és 64.* *ábrákon* lehet látni.



*62.Ábra – A thermisztor túlhevülés esetén kikapcsolja az eszközt*

** **

*63.Ábra – Az USB bedugása és kihúzása be és kikapcsolt állapotban*

**

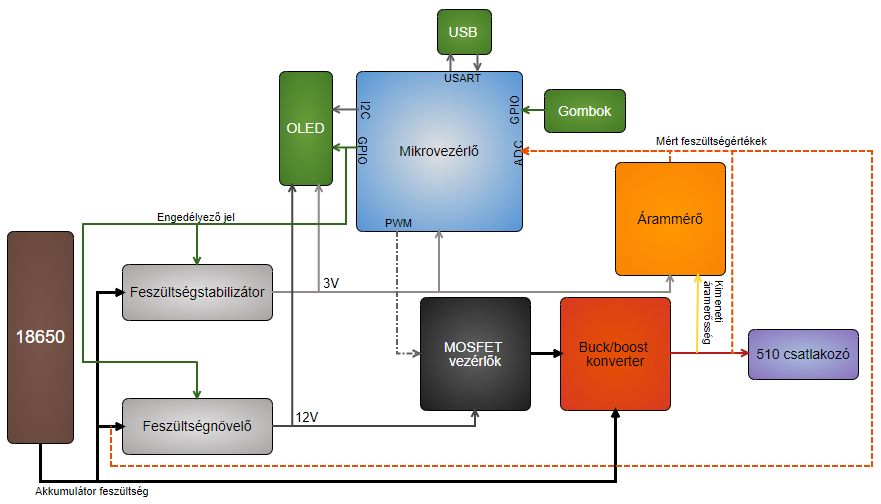
*64.Ábra – USB-n keresztül tölt az akkumulátor, valamint feltöltött az akkumulátor*

# Összefoglalás

Az elektromos cigaretták nikotintartalmú, belégzésre alkalmas gőzt párologtatnak egy fűtőszál hevítésével. A dolgozatunkban a fűtőszál hevítéséért felelős elektronika működési elveit tanulmányoztuk.

Ehhez elsősorban egy NyÁK-ra volt szükség, melyet egy a már tulajdonunkban lévő mod-ból, az elektromos cigaretták akkumulátorainak és elektronikájának tárolásáért felelős egységből szereltünk ki.

Beazonosítottuk az alkatrészek pontos típusait és végigmértük egy multiméterrel az egyes alkatrészek összeköttetéseit. Ezek alapján meg tudtuk határozni az áramköri elemek feladatait.



*21. Ábra – Elektromos cigaretta blokkdiagramja*

A tűz gomb nyomására a mikrovezérlő PWM jelet ad a MOSFET vezérlőknek, azok a PWM jel függvényében váltják a buck/boost konverter MOSFET-jeit, amik a kimeneti feszültséget állítják elő. A kimeneten a fogyasztó egy ellenálláshuzal, amely a rajta folyó áram hatására felhevül, emiatt nő az ellenállása. Ezt az ellenállást a mikrovezérlő a visszamért kimeneti feszültség és áram hányadosából határozza meg és ez alapján adja a PWM jelet a MOSFET-eknek, hogy a fogyasztó teljesítménye állandó maradjon.

Végeredményként egy teljes képet kaptunk a NyÁK-unk áramköréről, mely tudás segítségével lehetőséget látunk egy saját, egyedi funkciókkal rendelkező elektromos cigaretta tervezésére.

# Irodalmi hivatkozások

[1] [The Wealth of India](https://books.google.com/books?id=Hm1XAAAAMAAJ&q=hookah+isndia&dq=hookah+india). Council of Scientific & Industrial Research. 1976. „The smoking of hookah and hubble-bubble started in India during the reign of the great Moghul emperor, Akbar”

[2] US 1,775,947, Patented Sept. 16, 1930, J. Robinson, of New York, N.Y., Assignor of one-fourth to Roy M. Wolvin, of New York, N.Y., „ELECTRIC VAPORIZER”, Filed May 3, 1927, Serial No. 188,559

[3] US 3,200,819, Patented Aug. 17, 1965, Herbert A. Gilbert, „SMOKELESS NON-TOBACCO CIGARETTE”, Filed Apr. 17, 1963, Serial No. 273,624

[4] US 8,393,331 B2, Patented Mar. 12, 2013, Lik Hon, „ELECTRONIC ATOMIZATION CIGARETTE”, Filed Nov. 11, 2010

[5] vaper.hu: „A nikotin átka és áldása”, [Online] Letöltve: <http://www.vaper.hu/info/nikotin.html> (Utolsó hozzáférés dátuma: 2019. 04. 07.)

[6] ecigsadvice.co.uk: „Movkin Disguiser 150W Box Mod Review”, [Online] Letöltve: <https://www.ecigsadvice.co.uk/> (Utolsó hozzáférés dátuma: 2019. 04. 07.)

[7] Krishnaveni S.R. : „Analysis of Four Switch Positive Buck Boost Converter Based On Mode Selection Circuit for Portable Battery Applications”, IEEE Sponsored 2nd International Conference onInnovations in Information Embedded and Communication Systems, 2015. Január, [Online] Letöltve: <https://www.researchgate.net/publication/283803869_Analysis_of_four_switch_positive_buck_boost_converter_based_on_mode_selection_circuit_for_portable_battery_applications>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[8] ON Semiconductor®:„Dual Bootstrapped, 12 V MOSFET Driver with Output Disable, ADP3110”, [Online] Letöltve: <https://www.onsemi.com/pub/Collateral/ADP3110-D.PDF>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[9] Texas Instruments: „INA21x Voltage Output, Low- or High-Side Measurement, Bidirectional, Zero-Drift Series, Current-Shunt Monitors”, 2008. Május, [Online] Letöltve: <http://www.ti.com/lit/ds/symlink/ina213.pdf>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[10] Microchip: „MCP1711 150 mA Ultra-Low Quiescent Current, Capacitorless LDO Reguator)”, [Online] Letöltve: <http://ww1.microchip.com/downloads/en/devicedoc/20005415d.pdf>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[11] AEROSEMI „MT3608 High Efficiency 1.2MHz 2A Step Up Converter” , [Online] Letöltve: <https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[12] Ata Elahi, Trevor Arjeski: „ARM Cortex-M3 Processor and MBED NXP LPC1768”, ARM Assembly Language with Hardware Experiments, 2014. December, pp 83-95, [Online] Letöltve: <https://link.springer.com/chapter/10.1007/978-3-319-11704-1_6>, (Utolsó hozzáférés dátuma: 2019. 04. 07.)

[13] GigaDevice Semiconductor Inc.: „GD32F130xx ARM® Cortex®-M3 32-bit MCU”, [Online] Letöltve: <https://datasheet.lcsc.com/szlcsc/GigaDevice-Semicon-Beijing-GD32F130K6U6_C81552.pdf>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[14] Manuel Jiménez, Rogelio Palomera, Isidoro Couvertier „Principles of Serial Communication”, Introduction to Embedded Systems, 2013. December, [Online] Letöltve: <https://link.springer.com/chapter/10.1007/978-1-4614-3143-5_9>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[15] DENSITRON® DISPLAYS: „OLED DISPLAY MODULE Product Specification”, [Online] Letöltve: <http://www.farnell.com/datasheets/609753.pdf>, (Utolsó hozzáférés dátuma: 2019.04.07.)

[16] chibios.org „RTOS Concepts”, [Online], Letöltve: <http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:rtos_concepts>, (Utolsó hozzáférés dátuma: 2019.10.21.)

[16] freertos.org „What is FreeRTOS”, [Online], Letöltve: <https://www.freertos.org/about-RTOS.html>, (Utolsó hozzáférés dátuma: 2019.10.22.)

[17] randomnerdtutorials.com „ESP32 Pinout Reference: Which GPIO pins should you use?” [Online], Letöltve: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>, (Utolsó hozzáférés dátuma: 2019.10.22.)

[18] Luzsinszky Bence: „Teljesítményszabályozott elektromos cigaretta tervezése – Hardveres oldal”, még nem elkészült dokumentum

[19] Espressif Systems: ESP32 Series Datasheet, [Online], Letöltve: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>, (Utolsó hozzáférés dátuma: 2019.10.26.)