Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

USB-MIDI interfész tervezése digitális hangszerhez

Szakdolgozat

Készítette
Páni Tamás

Konzulens
Dr. Orosz György

2016. május 28.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Páni Tamás*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. május 28.

Páni Tamás
hallgató

Kivonat

A dolgozat témája egy olyan modul tervezése és megvalósítása, amely két USB-MIDI eszköz között képes kapcsolatot létesíteni, az egymással folytatott kommunikáció lehetőségét biztosítani. A hagyományos MIDI interfész esetében ez nem jelent problémát, egy egyszerű kábelen keresztül a MIDI-eszközök összeköthetőek, azonban ma már egyre gyakrabban alkalmazzák a gyártók az USB fizikai rétegét a MIDI-kommunikációhoz is, ezzel lehetetlenné téve az eszközök közvetlen összekötését.

A feladat megoldásához teljesítendő követelmények, valamint fogalmak tisztázása után megvizsgáltam a megvalósítás lehetséges módjait. Figyelembe véve a rendelkezésemre álló időt és erőforrásokat, kiválasztottam a megfelelő konstrukciót, majd megkezdtem az ahhoz szükséges hardvereszköz tervezését.

Kétféle modult specifikáltam. Az egyik csak az USB-MIDI kommunikációt szem előtt tartva, az ahhoz szükséges feladatok ellátására alkalmas, míg a másik ennek kibővített változata, hagyományos MIDI-interfésszel, valamint több modul felfűzésének lehetőségével. A szakdolgozat keretein belül az első változat megvalósítását tűztem ki célul, mivel azonban egyrészt hardveres szempontból a bővített funkcionalitás lehetővé tétele nem vett igénybe sokkal több időt a minimálishoz szükségesnél, másrészt a hardver utólagos módosítása lényegesen nagyobb feladat, mint a szoftveré, ezért úgy döntöttem, a hardvert felkészítem a jövőbeni funkcióbővítésre, az ehhez szükséges segédáramkörök megtervezésével és megvalósításával.

A hardver elkészítése után megismerkedtem a beágyazott szoftver fejlesztőkörnyezetével, majd megkezdtem a szoftverfejlesztést. A kialakult végleges szoftververzió működésének kifejtése után leírtam az addig vezető utat is, azokat a problémákat, és megoldásukat, amelyek az első próbálkozások kudarcaitól vezettek végül a minimális funkcionalitás sikeres megvalósításáig.

Zárásként összegzem az elvégzett munka tanulságait, valamint jövőbeni terveimet a modullal, beleértve a már említett funkcióbővítést, valamint a hatékonyság növelésére irányuló fejlesztési lehetőségeket.

Abstract

The topic of the thesis is the design and implementation of a module, which is able to establish connection between two USB-MIDI devices, to provide the possibility of communicating with each other. With the use of conventional MIDI interface, this is not an issue, MIDI devices can be linked via a single cable, but USB as the physical layer of MIDI communication is now increasingly used by manufacturers, thus making direct connection of the same devices impossible.

After clarifying the needed terms and requirements to understand and solve the problem, I have examined the possible ways of implementation. Considering the time and resources available to me, I have chosen the right construction and commenced the necessary hardware design work.

I specified two types of modules. One of them is suitable for the USB-MIDI communication only, and the other is an expanded version of this with conventional MIDI interface and daisy chain capability in case of multiple modules in one system. In the thesis I aimed at the implementation of the first version, but from the hardware point of view, making the extended functionality does not take much more time than the minimal necessary. In addition, the subsequent modification of the hardware is significantly more challenging than the modification of the software, so I decided to prepare the hardware for future functional extension, which means the design and implementation of requisite auxiliary circuits.

After making the hardware I acquainted with the embedded software development environment, and then I started the development of software. After describing the operation of the developed final version software, I explained the long way I had to tour for it, the problems and their solutions, which finally led from the first attempts failures to the successful implementation of the minimum functionality.

As conclusions, I summarize the lessons learned from the work done and my future plans with the module, including the already mentioned function expansion and the opportunities of improvement to increase efficiency.

1. fejezet

Bevezető

A digitális hangképzés elterjedése hamar felvetette a hangszerek közötti kommunikáció lehetőségét. Színpadi felhasználásnál ennek az előnye több hangszer esetén a hangszerek tudásának kombinálási lehetősége. Egyetlen billentyű leütésének hatására egyidejűleg háromnégy hangszer is megszólalhat, amivel egy kisebb zenekar hangzása is elérhető. Továbbá a különböző billentyűzetek mechanizációja is nagyon eltérő lehet, a hangszerek közötti kommunikációval azonban ez a problémaforrás kiküszöbölhető, ha ugyanazt a hangszínkombinációt két különböző billentyűzetről is meg lehet szólaltatni.

Zeneszerzők számára az előny a számítógép és hangszer közötti átvitelben mutatkozik. A számítógépes zeneszerkesztő szoftverek és egy jó minőségű hangképző eszköz segítségével a zeneszerzőnek - de főleg a betanítandó zenekarnak, vagy a megrendelőnek - nem kell a képzeletére hagyatkoznia, ha egyben akarja hallani a készülő zeneművet.

A fentieknek köszönhetően, a hangszergyártók szövetségének eredményeként jött létre a MIDI, amely kifejezetten a hangszerek kommunikációjára specializált, egyszerű hardver és protokoll segítségével valósítja meg az átvitelt hangszer és hangszer, valamint számítógép és hangszer között. Előnye a rugalmas adatfolyam-kialakítás lehetősége különböző vezetékezési konfigurációkkal, hátránya a véges számú eszköz kezelése és az aránylag lassú adatátviteli sebesség.

Idővel felmerült az igény a számítógépes perifériák kommunikációs interfészeinek egységesítésére, ennek az eredménye pedig az USB lett, ami ezt a nagyfokú univerzitást azonban csak igen bonyolult protokollal tudta elérni. Ezen kívül erősen hierarchikus, definiál "host" és "device" eszközt, és ezek a szerepek az eszközök között nem felcserélhetőek. Előnye a nagy adatátviteli sebesség.

Mivel a hangszerek az USB megalkotása idején már számítógépes perifériaként is funkcionáltak, felvetődött a MIDI-adatok USB-n keresztüli átvitele, ebből pedig létrejött az USB-MIDI interfész. Eleinte a hagyományos MIDI portok mellé építették, később azonban költségcsökkentési szempontok miatt a hagyományos interfészt sok hangszernél elhagyták, ezzel megfosztva a felhasználót a hangszerek közvetlen összeköttetésének lehetőségétől.

Célom egy olyan eszköz megalkotása, mely a közvetlen kommunikációt lehetővé teszi USB-MIDI interfésszel rendelkező hangszerek között is. Jelenleg a piacon csak olyan eszköz található, amely az USB-MIDI interfészt hagyományos MIDI portokra vezeti ki (például *Kenton MIDI USB host*), a beszerzése azonban még ennek is nehézkes és drága, továbbá csak bizonyos, általános interfésszel rendelkező hangszerekkel működik együtt, amelyekből a hangszerek funkcióinak bővülésével egyre kevesebbet találni a boltokban.

Ezzel szemben az általam tervezendő eszközzel két, USB-MIDI porttal rendelkező hangszert kívántam egymással összekötni, továbbá kezelni kívántam az egyre elterjedtebb, gyártóspecifikus interfésszel rendelkező hangszereket is. Összetett rendszer esetén az úgynevezett multi-master konstrukciót is lehetővé akartam tenni a vezetékezés változtatása nélkül, ami a hagyományos MIDI képességein is túlmutat. Mindezt a jelenleg beszerezhető piaci megoldások bekerülési költségének töredékéért.

A dolgozat felépítése a következő:

- **Rendszerterv és specifikáció:**

  A tervezett eszköz szerepe, helye egy több hangszerből álló rendszerben, a szerep betöltéséhez szükséges teljesítendő követelmények meghatározása.

- **Elméleti alapok:**

  Az eszköz működésének, valamint a dolgozat későbbi fejezeteiben előforduló fogalmaknak a megértéshez szükséges ismeretek összefoglalása.

- **Megvalósítási lehetőségek:**

  A specifikált működés létrehozási módjainak sorra vétele az eszköz különböző komponensei szerint, a legkedvezőbb konstrukció kiválasztása.

- **Hardvertervezés:**

  A fizikai eszköz tervezésének menete az alkatrészválasztástól kezdve a tervezőszoftver használatán és az áramkör legyártásán keresztül a műszerdobozba való beépítésig.

- **Szoftvertervezés:**

  Az eszköz által futtatott beágyazott szoftver létrejötte, a fejlesztőkörnyezet, valamint a szoftverhez rendelkezésre álló források leírása.

- **Tesztelés:**

  A hardver építése utáni, valamint a szoftverfejlesztés közbeni tesztelési folyamat, és a közben előforduló hibák megoldása.

- **Összefoglaló, kitekintés:**

  Az elvégzett fejlesztési munka közben szerzett tapasztalatok, a jövőbeni továbbfejlesztési lehetőségek számba vétele.

2. fejezet

Rendszerterv és specifikáció

Első lépésként meg kell határozni a feladat végrehajtásához szükséges technikai követelményeket. Ez nagyban segíti a megoldáshoz vezető legsimább út minél gyorsabb meglelését.

2.1. Követelmények

A modullal szemben támasztott minimális követelmény a két USB-MIDI porttal rendelkező hangszer közötti kapcsolatteremtés. Ehhez az alábbi funkciókra van szükség:

- **Tápfeszültség előállítása**
  Lehetőleg a rendszerbe vitt újabb külső tápegység nélkül. Mivel alapvetően logikai áramkörről van szó, az USB által biztosított tápfeszültség elegendő lehet. Arra alkalmas port megléte esetén ezt akár egy hangszer is biztosítani tudja.
- **Programozható központi vezérlő**
  Megfelelő számítási kapacitással és sebességgel kell rendelkeznie két USB eszköz egyidejű kiszolgálásához, továbbá a csomagok kezeléséhez, az üzenetek értelmezéséhez.
- **Két USB host interfész**
  Mivel egyszerre csak egy eszközzel lehetséges a kommunikáció, a feladat pedig érzékeny a késleltetésekre, mindenképpen két hostra van szükség a két eszköz kiszolgálása érdekében.

A minimális követelményeken túl adódik néhány további funkció, amelyek megvalósítása megfontolandó. Ezek szoftveres implementálása ugyan túlmutat a szakdolgozat keretein, a hardvereszközt azonban érdemes felkészíteni rá a jövőbeni munka megkönnyítése érdekében. Ezek a funkciók az alábbiak:

- **MIDI ki-, és bemenet**
  A hagyományos interfész kialakításával aránylag kis fejlesztési munka befektetésével jelentősen bővíthető a modul felhasználási lehetőségeinek köre.
- **Modulok közötti kommunikációt biztosító interfész**
  Kiterjedtebb rendszerek építését megkönnyítheti egy további, nagy sebességű hozzá-

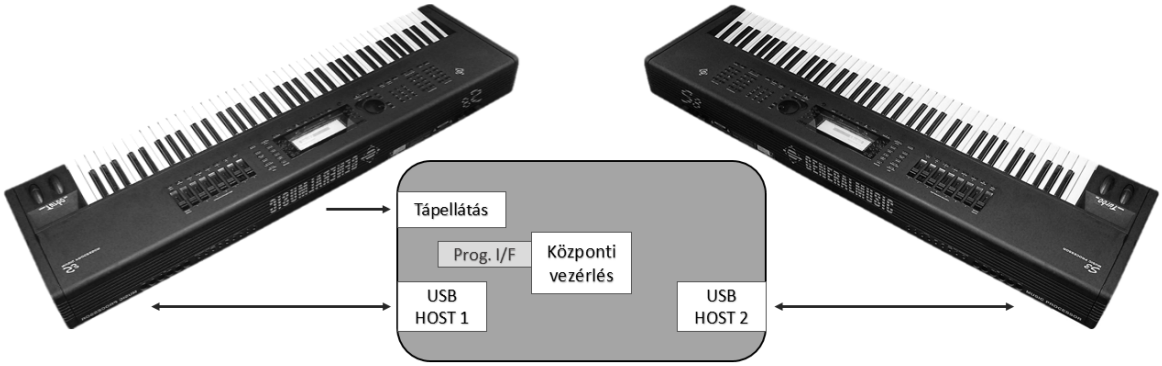
férési pont megvalósítása, amelyen keresztül több modul összeköthető egymással, így megosztva egymás között a beérkező üzeneteket.

2.2. Fizikai kialakítás, csatlakozási lehetőségek

A bevezetőben leírtakból következően két csatlakozási pontot ismerünk pontosan, ezek pedig az USB-MIDI portok a két hangszerhez, amelyeknek az egymással való kommunikációját biztosítani kívánjuk. Beágyazott vezérlőről lévén szó, feltehetőleg szükség lesz valamiféle csatolófelületre is, amelyen keresztül az eszköz programozása zajlik majd. Ezeken kívül amire még biztosan szükség lesz, az a tápfeszültség. Mivel a digitális hangszerek általában valamilyen egyenfeszültségű bemenettel rendelkeznek, ezért legegyszerűbb lehetőleg magából a hangszerből kinyerni a tápfeszültséget. Ennek pontos módját a "Megvalósítási lehetőségek" fejezetben fogom tárgyalni.

2.3. Minimális konfiguráció

Az alábbi ábrán látható ezek alapján az a legkisebb rendszer, ahol a tervezett eszköz alkalmazása indokolt. Mivel a várható esetek többségében ilyen rendszer, vagy ennek bővített változata fordul elő, ezzel egyben definiálhatjuk is azt a minimális funkcionalitást, amit egyetlen modulban érdemes megvalósítani.

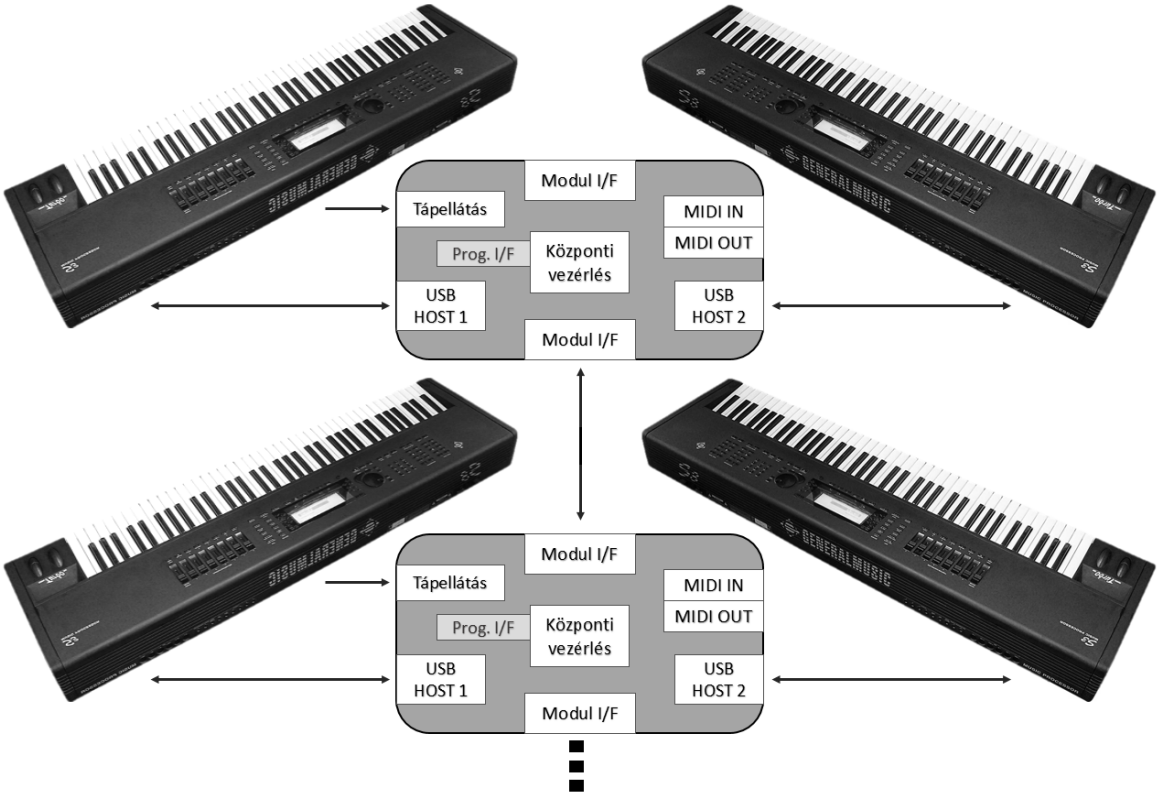


2.1. ábra. *Rendszerterv két hangszer esetén, minimális funkcionalitású modul*

2.4. Bővített konfiguráció

Már az ábrázolt konfigurációban is kényszerűen dekódolni kell az USB-MIDI kommunikáció csomagjait, így adódik a hagyományos MIDI-interfész megvalósításának lehetősége. Kettőnél több hangszer esetén a rendszer legnagyobb rugalmasságát úgy érhetjük el, ha a fenti funkcionalitást megvalósító modulból építünk be a rendszerbe a szükséges mennyiségben, és biztosítjuk a modulok közötti adatáramlást. A hagyományos MIDI interfész akár erre is alkalmas lehet, sebességbeli korlátai miatt azonban lehetőség szerint érdemes egy másik, könnyen kiépíthető interfészt alkalmazni.

A fentiek szerinti, bővített funkcionalitással rendelkező modul felhasználásával kiépíthető rendszer sematikus ábrája, kettőnél több hangszer esetén:



2.2. ábra. *Rendszerterv több hangszer esetén, bővített funkcionalitású modul*

Mint már említettem, ez a bővített funkcionalitású modul csak a hardver kiépítését tekintve képezi a dolgozat keretein belül elvégzendő munka részét, a szoftverfejlesztés csak a minimális funkcionalitású modul komponenseinek implementálását foglalja magában.

3. fejezet

Elméleti alapok

Ebben a fejezetben azokat a fogalmakat fogom tisztázni, melyek pontos ismerete elengedhetetlen a dolgozat tárgyát képező eszköz működésének, valamint az általa megoldható problémának a megértéséhez.

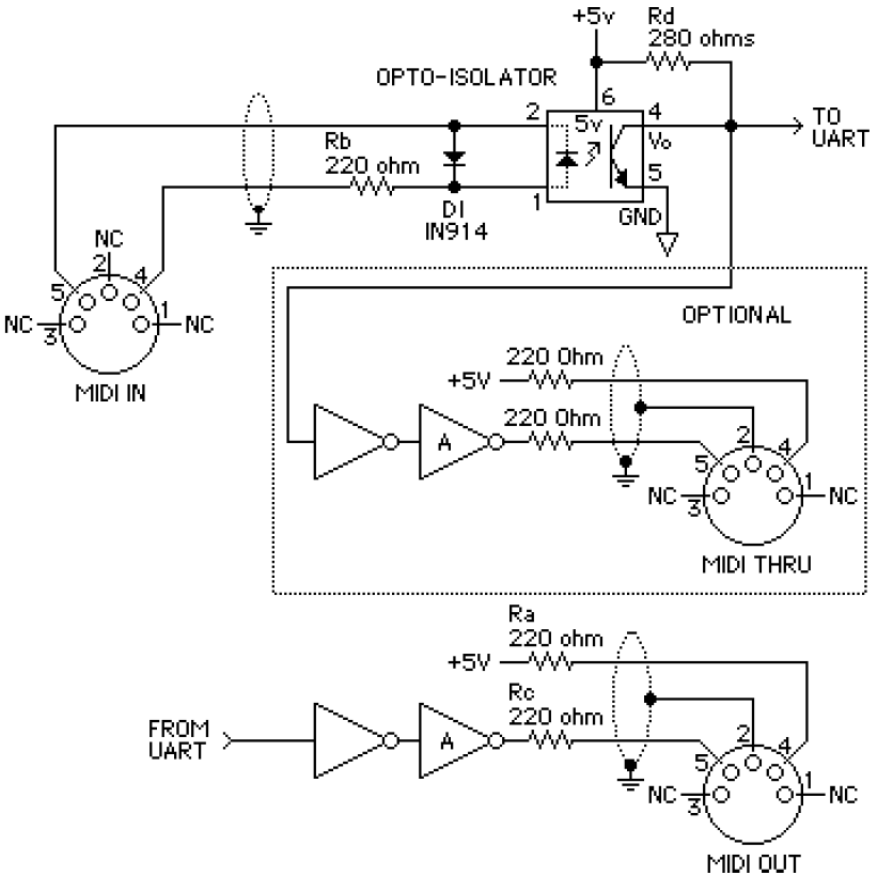
3.1. MIDI

***M**usical **I**nstrument **D**igital **I**nterface* - adatátviteli protokoll, melyet kifejezetten elektronikus hangszerek számára fejlesztettek ki 1983-ban. A specifikáció egyaránt meghatározza a szükséges hardver réteget és az interfész logikai működését. A megalkotása óta eltelt több, mint három évtized alatt a hangszerek képességei, valamint a protokoll felhasználási területeinek és módjainak száma egyre csak bővült, színpadi fényvezérlőkön keresztül a mobil eszközökig, melyek miatt mind a hardver, mind a csomag-alapú logika tekintetében különböző verziók létrejötte vált szükségessé. Mind a mai napig szerves részét képezi különösen a professzionális szórakoztatóipar technikai hátterének, még ha a felhasználónak egyre kevésbé is van szüksége az interfész ismeretére a mai eszközökbe épített magas szintű kezelőfelületeknek köszönhetően.

3.1.1. Fizikai réteg

Mivel a hangszerek esetében sokszor ugyanaz az eszköz kezeli a hangszer analóg hangkimenetét, valamint a digitális hangadatokat, ezért gyakori az a konfiguráció, hogy két eszköz két külön vezetéken is össze van kötve egymással. Ez a konfiguráció önmagában számos zavar forrása lehet, ami különösen a professzionális hangszerek esetében kellemetlen, mivel ott alapvető követelmény az analóg hangkimenet minél nagyobb jel-zaj aránya, valamint a kis mennyiségben mindenképpen jelen lévő zaj sztochasztikussága.

A legfontosabb elkerülendő jelenség egy ilyen esetben az úgynevezett földhurok kialakulása. Erre a legkézenfekvőbb mód digitális jelek esetében az optikai leválasztás, így ez szerepel a fizikai réteg specifikációjának központi helyén. Eszerint a MIDI-adó minden esetben egy 5 milliamper nagyságú áram kiadására képes áramgenerátorból, a MIDI-vevő pedig egy alkalmasan nagy sebességű kapcsolásra képes optocsatoló bemenetéből áll (3.1. ábra).



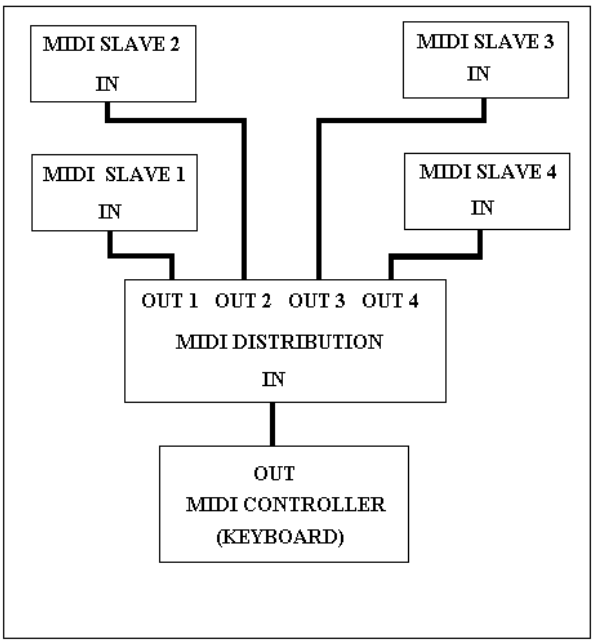
3.1. ábra. *MIDI fizikai réteg specifikáció [6]*

A kommunikáció sebessége 31.25 kbaud, protokollt tekintve gyakorlatilag mindenben megegyezik a standard UART protokollal, 1 start, 8 adat, és 1 stop bit paraméterekkel. A csatlakozó szintén szabványos, 5 pines DIN körcsatlakozó. MIDI IN, MIDI OUT és MIDI THRU végpontok megvalósítása lehetséges a specifikáció szerint. Az IN és az OUT csatlakozókról már esett szó, a THRU csatlakozó a beérkező adat közvetlen továbbítására szolgál további eszközök számára. Az optocsatoló kimeneti késleltetése terhelésfüggő, ezért a THRU csatlakozóra felfűzhető eszközök száma limitált. Ez, és a kommunikáció manapság már lassúnak számító sebessége tette mára elavulttá ezt a fizikai réteget, robusztussága miatt azonban továbbra is megtalálható még igen sok eszközön.

3.1.2. Hálózati topológia

Több MIDI-eszközből álló rendszer esetén számos módja van az eszközök közötti adatmegosztás kialakításának, a három fajta végpont felhasználása könnyedén, rugalmasan változtatható akár már kiépített rendszer esetén is. Alapvetően két topológia hozható létre: csillagpontos (3.2. ábra), vagy az úgynevezett "daisy chain" (3.3. ábra) topológia.

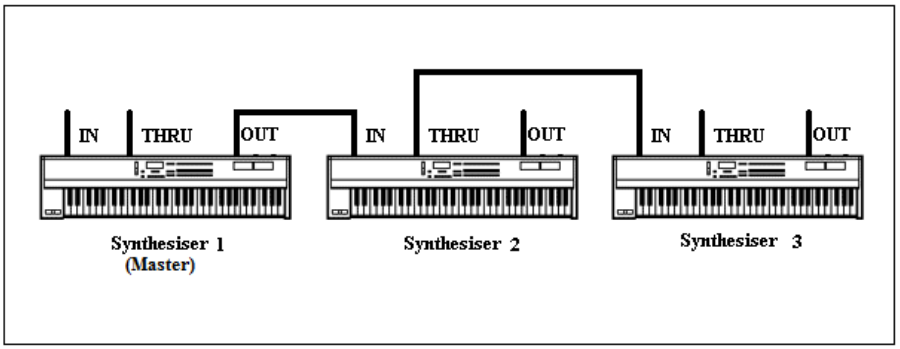
Csillagpontos rendszer kialakításához szükség van egy külső eszközre, amely biztosítja a MIDI-bemenetek működéséhez szükséges áramot. Ennek előnye, hogy THRU végponttal nem rendelkező eszközökkel is használható, valamint az eszközök számának is csak az



3.2. ábra. *Csillagpontos MIDI-topológia*

elosztó eszköz kapacitása szab határt, ellenben a már korábban említett THRU kimenet korlátaival.

A daisy chain topológia elterjedtebb, mivel nincs hozzá szükség külső eszközre, a professzionális eszközök pedig általában rendelkeznek THRU kimenettel.



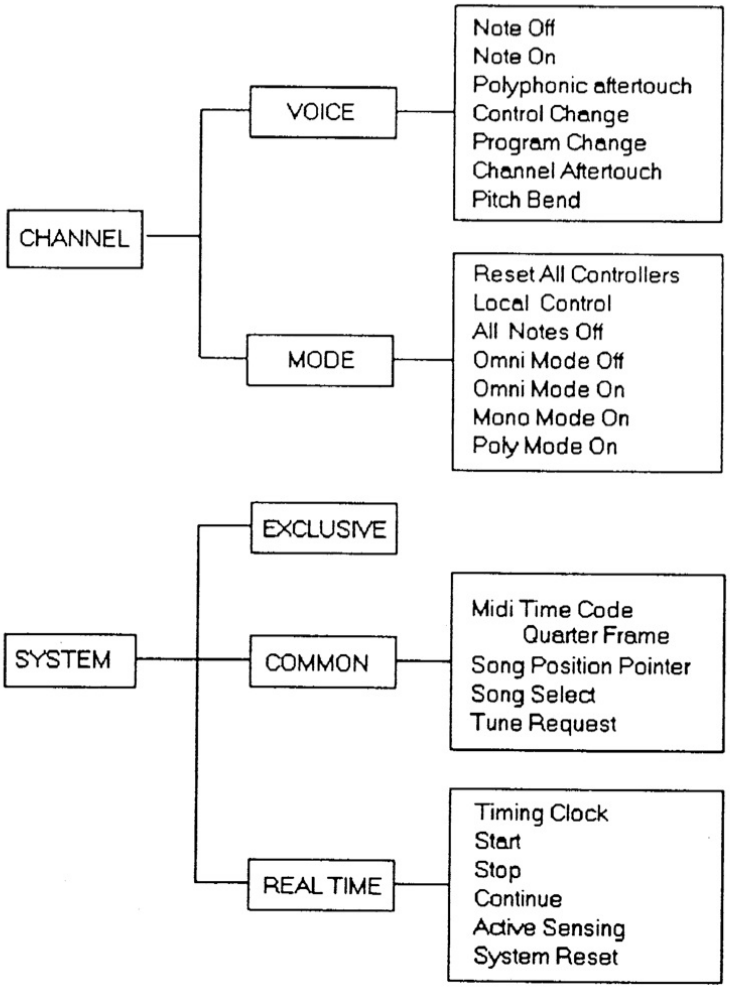
3.3. ábra. *Daisy chain MIDI-topológia*

Mindkét topológiára jellemző, hogy egyetlen eszköz vezérel, a rendszer többi eleme ennek a master eszköznek az adatait kapja meg, és azoknak megfelelően cselekszik. Jól látható, hogy a rendszer alapvetően nem számítógép-központú, a billentyűzetek, hangszerek, hangmodulok külső beavatkozás nélkül is képesek kommunikálni egymással. A MIDI csatornaszemléletének köszönhetően lehetséges az eszközöket úgy konfigurálni, hogy egy üzenetet tetszőleges számú eszköz dolgozzon fel, tehát ha a master például az 1-es számú csatornára vonatkozó üzenetet küld, azt azok az eszközök fogják feldolgozni, amelyeknek a bemenete az 1-es csatornára van állítva. Függetlenül attól, hogy van-e ilyen eszköz, netán mindegyik eszköz az 1-es csatornát figyeli, a kommunikáció működése zavartalan lesz.

3.1.3. Logikai réteg

A MIDI-kommunikáció üzenet alapú, egy üzenet általában egy, két, vagy három bájtból áll, ez alól csak az úgynevezett "system exclusive" üzenet jelent kivételt.

Az első bájt mindig az úgynevezett státusz-bájt, ez határozza meg az üzenet típusát, melyből általában következik az üzenet hossza (kivéve system exclusive), valamint a státusz-bájt után esetleg érkező bájtok jelentése. A státusz-bájtot az adatbájtok követhetik, melyeket system exclusive üzenet esetén az EOX-bájt zár le, ezzel jelezve a speciális üzenet végét. A specifikáció számos üzenettípust meghatároz (3.4. ábra), a fentebb többször említett system exclusive üzenet célja pedig, hogy megadja a lehetőséget a gyártónak a specifikáción túlnyúló funkciók implementálására.



3.4. ábra. *MIDI-üzenettípusok egyszerűsített összefoglaló táblázata [6]*

Ez egyértelműen kompatibilitási problémákhoz vezethet, ezért a kommunikáció logikai meghatározásának részét képezi az is, hogy ha egy eszköz számára nem értelmezhető, vagy nem implementált funkciót feltételező üzenetet kap, azt automatikusan figyelmen kívül hagyja, és hibajelzés, vagy a kommunikáció bármilyen szintű megakasztása nélkül várakozik a következő üzenetre.

Számunkra a legfontosabb kategória az egyes MIDI-csatornákhoz tartozó, hangkeltést befolyásoló üzenetek lesznek (az ábrán Channel/Voice), melyeket közvetíteni kell az eszközök között, de később látni fogjuk, hogy némely rendszerüzenetet is megfelelően le kell majd kezelnünk, hogy az eszközök közötti kommunikáció egyáltalán létrejöjjön.

3.2. USB

***U**niversal **S**erial **B**us* - manapság nagy népszerűségnek örvendő, sokoldalúságának köszönhetően már-már egyeduralkodó adatátviteli protokoll. Alapjait 1990-ben fektették le, akkorra merült fel az igény egy egységes, robusztus interfészre elsősorban számítógépes perifériák számára, mivel a különböző szabványok és csatlakozók addigra kialakult erdejében való tájékozódás a felhasználók számára egyre nagyobb problémát jelentett. A lehető legtöbb periféria igényeit ki akarták elégíteni a minél nagyobb szintű egységesítés érdekében, ezért az eszközök számára eszközosztályokat hoztak létre, mely osztályokon belül már az adott osztály szempontjainak megfelelően optimalizálhatták a kommunikáció pontos menetét. Meghagyták a lehetőséget a gyártóknak egyedi protokollok megalkotására is, külön eszközosztályt definiálva a gyártóspecifikus eszközök számára. Mivel univerzális interfésznek tervezték, a gyártók jóval szélesebb rétegét célozták meg, továbbá saját csatlakozókat is definiáltak, ezért a specifikáció is jóval bővebb, mint a MIDI esetén. A csatlakozók pontos méretrajzától kezdve a vezeték felépítésén és színkódolásán keresztül az átvitel állapotábrájáig, a pontos időzítésig mindent igyekeztek jól definiálni az USB-t létre hozó cégcsoport tagjai.

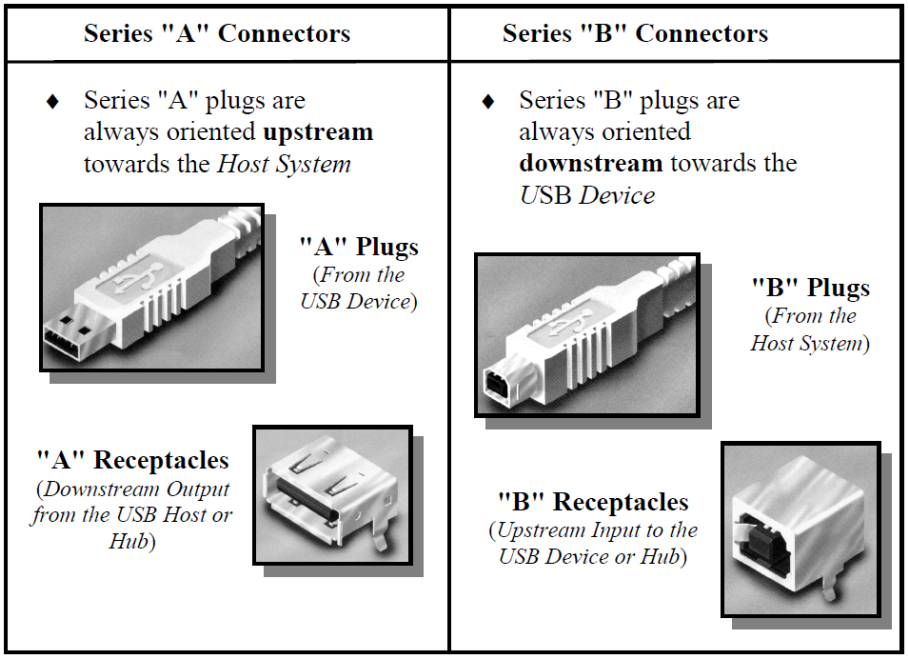
3.2.1. Fizikai réteg

Az USB esetében a nagysebességű kommunikáció zavarmentességéről differenciális jellel, jelvezetéssel gondoskodtak a tervezők, amely a vezeték esetén csavart érpárt, huzalozásnál pedig a jelek egymáshoz minél közelebbi vezetését, megegyező huzalhosszt, a jellel sorosan kapcsolt ferritet, esetleg kis induktivitású fojtótekercs-párt jelent.

A logikai réteg kifejtésénél látni fogjuk, hogy az USB egy adott eszköz számára lehetővé teszi, hogy az azt kezelő vezérlő egy fizikai eszközt akár több különböző típusú virtuális eszköznek érzékeljen. Ez azt jelenti, hogy egyetlen vezetéken egy időben több forrásból származó, különböző jellegű adatok is továbbíthatók. Ez, valamint az USB elsődleges célja (számítógépes, digitális perifériák) feleslegessé tette a külön felkészítést a két eszköz több vezetéken való összekötésekor kialakuló földhurokra, így a MIDI-vel ellentétben az optikai leválasztás nem része a specifikációnak, az optocsatolók sebességkorlátait szem előtt tartva ez nehézkes is volna.

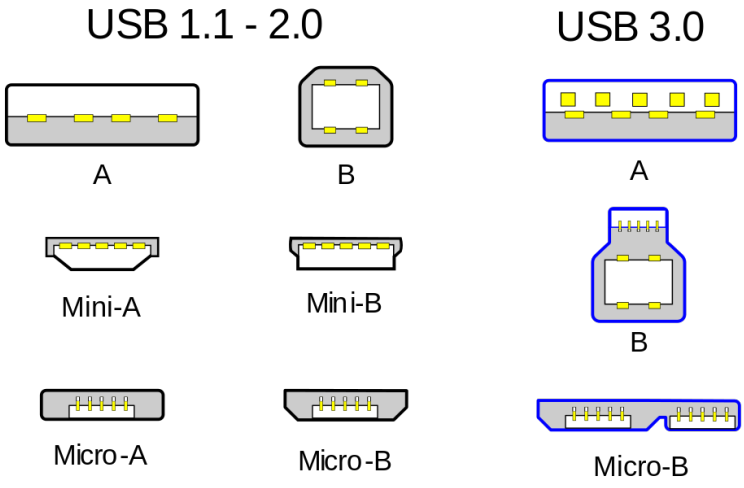
A kommunikáció sebességét tekintve is több osztály létezik. Az 1.0 verzió specifikációs hibáknak köszönhetően igen rövid életű volt, ma már csak a továbbfejlesztett, 1.1 verzió, és a későbbi, 2.0, valamint a legújabb, 3.0 verzió használatos. Az USB alapvetően hierarchi-

kus pont-pont összeköttetésen alapul, ennek megfelelően kétféle végpont létezik: USB TO HOST, valamint USB TO DEVICE. Csatlakozók tekintetében az USB számára korábbi szabványok választása helyett saját csatlakozót - illetve csatlakozókat - terveztek, melyek műszaki leírása és rajzai szintén a specifikáció részét képezik (3.5. ábra).



3.5. ábra. *USB 1.1 csatlakozók összefoglaló ábrája a specifikációban [4]*

Sebesség tekintetében ugyan az osztályok visszafelé kompatibilisek, ennek ellenére az egyre újabb osztálydefiníciók egyre több fajta csatlakozóval bővítették az USB-csatlakozók családját (3.6. ábra), a kor igényeinek megfelelően általában egyre kisebb méretben.



3.6. ábra. *USB 3.0 csatlakozók összehasonlítása a korábban specifikáltakkal (keresztmetszeti rajzok) [5]*

A változtatások nem mindig kizárólag a csatlakozók formáját érintették, az USB 3.0 esetében például az akár 5 Gb/s sebesség eléréséhez már összesen három csavart érpárra van szükség, ennek megfelelően a csatlakozók érintkezőfelületeinek száma is növekedett.

3.2.2. Hálózati topológia

Az USB alá-fölé rendelt kommunikációt valósít meg, kommunikáció mindig csak egy "host" és egy "device" eszköz között történhet. Egy host jelei egyetlen device eszközhöz juthatnak el, csak egy újabb host közbeiktatásával fűzhetőek tovább (például USB hub). Szigorúan véve tehát "hálózatról" itt nem beszélhetünk, legfeljebb az USB host portbővítéssel történő kapacitásnöveléséről. Ez fizikailag a MIDI csillagpontos elosztási módjához hasonlít, logikailag azonban a kettő nem azonos, mivel ha egy adott USB device eszköz és a host között létrejött a kapcsolat, onnantól kezdve a többi, a hub-ra csatlakoztatott device eszköz meg sem kapja a host (más device-nak címzett) jeleit, ellentétben a MIDI-vel, ahol - mint láttuk - csak a kapott jel feldolgozása függ a feladó és a címzett MIDI-csatorna egyezésétől.

3.2.3. Logikai réteg

Az USB-kommunikáció csomag alapú, melynek mérete rugalmasan változhat, a különböző eszközök igényeinek megfelelően. Egy csomag több mezőből áll össze, melyek a beérkezés sorrendjében a következők [4]:

- **Szinkronizációs mező (SYNC):**

  Minden csomag ezzel kezdődik, a két, egymással kommunikáló eszköz esetleg különböző fázisban lévő órajeleinek szinkronizálására szolgál.

- **Csomag azonosító mező (PID):**

  A SYNC mezőt követi minden esetben, a csomag típusának jelzésére szolgál, ami alapján a csomag formátuma, valamint a szükséges hibadetektálási módszer megválasztandó. 8 bit hosszúságú mező, mely tartalmazza a 4 biten ábrázolt azonosítót, valamint annak bitenként invertált változatát, elősegítendő az esetleges átviteli hibák detektálhatóságát.

- **Cím mező (ADDR):**

  Egy eszköz különböző funkcióit címzéssel lehet elérni, egy címhez egyetlen funkció tartozhat. Ennek megadására szolgál ez a mező, amely a PID-től függően egyaránt megadhatja a csomag címzettjét, vagy a feladóját.

- **Végpont mező (ENDP):**

  Egy funkció több ki-, és/vagy bemenettel rendelkezhet, ezek megkülönböztetésére szolgálnak az úgynevezett végpontok. Ez a mező a kívánt végpont megadására szolgál.

- **Keretszám mező:**

  Időzítésre érzékeny kommunikációnál van jelentősége a keretnek, ami (USB 2.0 esetén) 1 ms időintervallumot ölel fel. A keretszám mező egy 11 bites számláló mező,

amelyben a host az aktuális keret sorszámát jelzi. Lényegében időbélyegzőként szolgál arra az esetre, ha erre szükség van a kommunikáció során.

- **Adatmező:**

  0-tól 1024 bájtig terjedő méretű mező arra az esetre, ha a kommunikációban adatcserére van szükség.

- **Ciklikus redundancia-ellenőrző mező:**

  A PID mezőt leszámítva az átviteli hibák detektálásáért, valamint lehetőség szerinti javításáért ciklikus redundancia-ellenőrzés (CRC) felel, erre szolgálnak a CRC mezők. A cím-, végpont-, és keretszám mezőket 5 bites CRC mező, míg az adatmezőt 16 bites generátor polinom felhasználásával ellenőrzi a rendszer.

Négyféle végpont típus létezik, mindegyik különböző algoritmus szerinti kommunikációt igényel, annak érdekében, hogy a kommunikáció célja által meghatározott feltételek teljesíthetőek legyenek [4].

Az úgynevezett "**Bulk**" átvitel elsődleges célja a csomagok hibamentes célba juttatása. Ennek érdekében fontos szerepet kap az adatellenőrzés, valamint az algoritmus lényeges eleme a handshaking, továbbá hiba érzékelése esetén a csomag újraküldése is. Előnye a garantált átvitel, hátránya az esetleges késleltetés kiszámíthatatlansága, mivel ez a legkisebb prioritású átviteltípus, csak akkor lép életbe, ha van szabad sávszélesség a többi típusú átvitel mellett.

A "**Control**" átvitel főként a kapcsolatfelvételi szakaszra jellemző, általában a kommunikációban résztvevő eszközök konfigurálására használatos. Mivel itt is az átvitel pontossága a legfontosabb szempont, ezért az algoritmus hasonló a Bulk átviteléhez, lényegében annak egy speciális esete (előre meghatározott cím, és egyéb paraméterek, nagy mennyiségű adat átvitelére nem alkalmas, magasabb prioritás).

Az "**Interrupt**" átvitel - mint arra az elnevezés is utal - megszakítás-jellegűen működik. Az USB device jelzi a host felé, ha rendelkezésre áll kiolvasható adat, egyéb esetben a host kéréseit megfelelő handshake visszajelzés mellett figyelmen kívül hagyja. Mivel a host ennek az átviteli módnak a használata esetén rendszeresen ellenőrzi az adat rendelkezésre állását, a késleltetés számolható, és a többi üzemmódhoz képest kicsi. További előnye átviteli hiba esetén az algoritmus részét képező újraküldés a következő lekérdezésnél, hátránya az átvitt adatmennyiséghez képest nagy sávszélesség-igény az aránylag sűrű lekérdezések miatt.

"**Isochronous**" átvitel esetén a hangsúly az időérzékenységen van. A host ebben az esetben is rendszeres lekérdezéseket intéz a device felé, itt azonban ezeknek a lekérdezéseknek a gyakorisága igen nagy, már-már folyamatosnak mondható. Ennek köszönhetően a késleltetés nagyon kicsi, a hibakezelés azonban nem mutat túl a CRC-ellenőrzésen. A

hibás csomagok újraküldése az időérzékenység miatt nem lehetséges, azokat a fogadó eszköz azonnal elveti. Ez az üzemmód a hang-, és/vagy képi adatfolyam átvitele esetén lehet optimális választás, ahol néhány csomag elvesztése kevesebb problémát okoz, mint azok újraküldése.

Az USB kommunikáció mindig az enumerációval kezdődik, ami tulajdonképpen a kapcsolatfelvételt, a kommunikációban résztvevő eszközök konfigurálását jelenti. Minden device típusú eszköz rendelkezik erre a célra fenntartott azonosító adatmezőkkel, descriptorokkal, amelyeket a host az enumeráció folyamán lekérdez, és azoknak megfelelően választja meg a kommunikáció sebességét, típusát, a végpont konfigurációját, adatátvitel esetén az adatmező hosszát, és minden egyéb további paramétert.

Szintén descriptorok alapján történik az eszközosztály, valamint az azon belüli alosztály megállapítása. Az osztályba sorolás az USB egyik kulcsfontosságú eleme, segítségével a host még a kommunikáció megkezdődése előtt képes megállapítani, hogy tudja-e majd kezelni az adott eszközt, vagy sem. Erre az USB-eszközök sokszínűsége, valamint a különböző kommunikációs algoritmusok miatt van szükség. Egy eszközosztályt alkotnak például a számítógépes beviteli eszközök (egér, billentyűzet, érintőpad, stb.), az adattároló eszközök, vagy esetünkben például az audio eszközök. Minden eszközosztályhoz létezik külön specifikáció is, az annak tökéletesen megfelelő eszközök kaphatnak csak úgynevezett "class compliant" minősítést.

A bevezetőben említett USB-MIDI átalakító esetében az "általános interfész" alatt például az "Audio" osztályon belüli "MIDISTREAMING" alosztály specifikációjának megfelelő interfészt kell érteni. Ma már szinte minden digitális hangszer rendelkezik belső memóriával, amely elérésnek érdekében adattároló végpontot is implementálnak a gyártók. Így azonban az eszköz már nem felel meg a MIDISTREAMING alosztálynak, ezért átkerül a gyártóspecifikus eszközök számára fenntartott "Vendor" osztályba, amit értelemszerűen az USB-MIDI átalakító nem kezel.

3.3. USB-MIDI

A MIDI egyik változataként jelent meg az USB-MIDI, ami tulajdonképpen a MIDI-üzenetek USB-csomagokként való elküldésén alapszik. A fizikai réteg, valamint az adatátviteli algoritmus az USB-nek megfelelő, pusztán a kommunikáció tartalma maradt meg a MIDI-ből. Ennek az egyedüli előnye az USB sokkal nagyobb adatátviteli sebessége, legfőbb hátránya ezzel szemben az USB nagy fokú, rugalmatlan hierarchiája, amiből következően a hangszerek egymással képtelenek közvetlenül kommunikálni, szükség van egy USB hostra a kapcsolat létrejöttéhez. A két interfész logikai rétegének leírásából is látható, hogy az USB feleslegesen bonyolult a MIDI kommunikáció számára, ezzel a csak USB-MIDI csatlakozással rendelkező hangszereket indokolatlanul bekorlátozva a számítógépes periféria-jellegű felhasználásra.

Az USB-MIDI specifikációjának megfelelően [3] a MIDI-üzeneteket Bulk típusú átvitellel küldik és fogadják az eszközök, mivel itt az adat pontossága, valamint a csomagvesztés elkerülése a legfontosabb szempontok. Lényeges még a késleltetés is, azonban az USB nagy sebességéből adódóan a még a többi átviteli módhoz képest nagy késleltetés is bőven az emberi füllel hallható tartományon kívül esik.

Az üzenetek értelemszerűen az adatmezőben foglalnak helyet, ami általában 64 bájt hosszúságú. Eszköztől függ, hogy ha az elküldeni kívánt üzenet nem éri el a 64 bájt hosszúságot, akkor a küldő kitölti-e a maradék bájtokat "0" értékkel, vagy az átvitel közben rugalmasan kezeli az adatmező hosszát.

4. fejezet

Megvalósítási lehetőségek

Ebben a fejezetben a specifikációban meghatározott funkciók implementálásának különböző módjait fogom sorra venni, majd ezek közül a lényeges szempontok alapján kiválasztom a céloknak leginkább megfelelőt, amely végül megvalósításra fog kerülni.

4.1. Tápellátás

A tápfeszültség biztosításának konvencionális módja az úgynevezett dugasztápegység használata. Ebben az esetben az áramkörön a szükséges csatlakozót, és valamiféle feszültségszabályzó áramkört kell elhelyezni.

A feszültségszabályzás legegyszerűbb módja a lineáris szabályozó áramkör használata a hozzá tartozó pufferkondenzátorokkal. Ebben az esetben a tápegység feszültségének a lineáris szabályzók bemeneti követelményeinek megfelelően legalább néhány voltnyi mértékkel a kimeneti feszültség feletti értéket kell megválasztani, figyelembe véve, hogy a lineáris szabályzók működésükből következően a bemeneti és a kimeneti feszültség különbségét lényegében disszipáció formájában állítják elő. Érdemes tehát a lehető legalacsonyabb alkalmas bemeneti feszültséget választani.

Lineáris feszültségszabályozó helyett alkalmazhatunk úgynevezett DC/DC konverter áramkört, amely kapcsolóüzemmel éri el a kívánt kimeneti feszültséget a bemenetből. Ennek előnye a nagy (akár 90 százalékos) hatásfok, ami azonban főleg akkor jelent előnyt a lineáris szabályzókkal szemben, ha a bemeneti feszültség jelentősen nagyobb a kimenetinél. Logikai áramkörökről lévén szó, nem valószínű, hogy a nagy bemeneti feszültségre szükség lesz, a konverter hátrányai ezzel szemben a magas költség, továbbá az előállított feszültségen megjelenő nagyfrekvenciás zaj, melynek leszűrése számos kiegészítő alkatrészt igényel.

Manapság egyre népszerűbb az USB-csatlakozó használata a kommunikáció mellett a táplálás biztosítására is. A jelenség fő mozgatója a mobiltelefon-iparág, ahol a telefontöltők egységesítésére irányuló igény, valamint a mai telefonok háttértár funkciója miatt lett ideális választás az USB, azon belül is a micro-B csatlakozó.

Mivel egyre több digitális hangszer képes USB-háttértárak kezelésére erre specializált USB host segítségével, továbbá nem valószínű, hogy a tervezendő eszközben szükség lenne az USB által biztosított 5 volt feszültségnél, valamint 500 milliamper áramnál nagyobb értékekre, ezért a leendő eszköz esetén is érdemes lehet USB-csatlakozót választani a tápfeszültség számára. Amennyiben valamelyik hangszer rendelkezik szabad USB host porttal, nincs szükség dugasztáp használatára, továbbá a hangszer és az eszköz csatlakozóinak úgynevezett "shield" kontaktusát összekötve egymással kisebb az elektrosztatikus kisülés veszélye is a hangszer USB device portjának csatlakoztatásakor. Ha egyik hangszer sem rendelkezik USB host porttal, akkor a mobiltelefonok miatt ma már igen elterjedt, USB-kimenettel szerelt dugasztáp használható a tápellátás biztosítására. Hangszerek esetén USB device port gyanánt egységesen az USB 1.1 specifikációnak megfelelő USB-B csatlakozót alkalmaznak, ezért érdemes a tervezendő eszköznél is ezt használni, így csökkentve a felhasználandó kábelek diverzitását.

A fentiek alapján a legjobb konfigurációnak az USB-B csatlakozó használata tűnik, szükség esetén lineáris feszültségszabályzóval, amely 5 voltnál alacsonyabb tápfeszültségek előállítására szolgálhat.

4.2. USB host

Számos lehetőség adódik az USB host funkció megvalósítására. Alkalmas számítási sebességgel rendelkező vezérlő esetén például lehetséges akár tisztán szoftveresen kezelni az USB eszközöket, ez azonban az elméleti alapokban foglaltakból látható, hogy igen nagy szoftverfejlesztési munkát igényelne. Mivel nincs szükség az USB működésének egyedi módosítására, ezért ezt a megoldást érdemes elkerülni.

Léteznek hardveresen megvalósított USB host vezérlőáramkörök, amelyeket mikrovezérlő, vagy más központi egység perifériájaként használva jelentősen csökkenthető a szükséges szoftverfejlesztési munka mennyisége. Ennek a megoldásnak előnye, hogy a központi egység kapacitásainak megfelelően több USB host port is megvalósítható egyetlen áramköri lemezen, hátránya azonban, hogy a portok számának növekedésével a szükséges alkatrészmennyiség is jelentősen növekszik.

A fenti két megoldási lehetőség hátrányait egy olyan központi egységgel lehetne kiküszöbölni, amely integrált USB host vezérlővel rendelkezik. Számos gyártó termékei között megtalálhatóak hasonló áramkörök, így a legjobb választás egy ilyen áramkör lenne, lehetőleg két USB host porttal, hogy a specifikált minimális funkcionalitást egyetlen integrált áramkörrel meg lehessen valósítani.

4.3. Központi vezérlőegység

A feladat időzítésérzékenysége, valamint az USB aránylag nagy sebességigénye miatt felmerülhet alkalmas képességekkel rendelkező FPGA használata, amelyben adott esetben több

USB host, valamint a szükséges műveletek elvégzésére alkalmas logika is megvalósítható. Elérhetőek már készre fejlesztett HDL-komponensek, mivel azonban ezek csak az esetek kis részében alkalmazhatóak módosítás nélkül, ezért a szoftverfejlesztési munka ebben az esetben is igen nagy méreteket öltene.

További lehetőség a már említett hardveres USB host vezérlő használata. Ebben az esetben egy átlagos képességű mikrovezérlő is alkalmas lehet a portok kezelésére, valamint a csomagok értelmezésére és feldolgozására, az alkatrészigénye azonban ennek a megoldásnak továbbra is aránylag nagy.

Az optimális megoldás tehát a központi egység szempontjából is egy olyan mikrovezérlő lenne, amely perifériáit tekintve rendelkezik hardveres USB host vezérlővel. Az egyik példa erre az Atmel nevű gyártó AT90USB termékcsaládja, amelynek egyik tagja központi eleme volt az önálló laboratóriumi munkámnak. Kézenfekvőnek tűnik tehát ismét ezt az utat követni, azonban ebben az esetben csak egy USB host port állna rendelkezésre, ami kevés lenne a meghatározott minimális követelmények megvalósítására. Léteznek mikrovezérlők két USB host porttal, ezek azonban aránytalanul nagyobb felszereltségűek a projekthez szükségesnél, így bekerülési költségük is igen magas ahhoz a funkcionalitáshoz képest, amire esetünkben szükség van. A megoldás megtalálásához szemléletmódváltás vezethet.

Ez pedig abban áll, hogy olyan jó képességekkel rendelkező mikrovezérlő helyett, amely perifériaként két USB host portot is kínál, olyan két portos USB host vezérlőt kezdjünk inkább keresni, amely képes önálló működésre is. Ezzel az új szemlélettel már hamar rátalálhatunk az FTDI nevű gyártó Vinculum II termékcsaládjára, amely pontosan az általam megvalósítani kívánt feladatnak megfelelő funkcionalitással rendelkezik. Két USB portjával, amelyek egymástól függetlenül konfigurálhatóak device vagy host funkció ellátására, valamint SPI, UART és FIFO interfészeivel tökéletesen alkalmas a már említett jövőbeni bővített modulfunkcionalitás megvalósítására is, egyetlen nagy integráltságú áramkörben, a célnak megfelelő számítási kapacitással és alacsony bekerülési költséggel.

Ezen kívül nagy előnyt jelent még az eszköz egyszerű programozhatósága. Az erre szolgáló interfész lényegében egy speciális USB-UART átalakító, amely könnyedén rátervezhető az áramkörre, ezzel önállóan programozható eszközt hozva létre. A tápfeszültség bemenetként szolgáló USB-B csatlakozó adatlábai pedig alkalmasak a programozó interfész és a számítógép közötti kapcsolat biztosítására, ezzel feleslegessé téve még egy további csatlakozó beépítését a programozhatóság lehetővé tételére.

4.4. Modulok közötti kommunikáció

Bár ez csak a bővített funkcionalitású modulnak képezi részét, a hardveres vonatkozásai miatt érdemes megválasztani a modulok közötti interfészt. A cél az, hogy a MIDI THRU kimenet működéséhez hasonlóan a MIDI üzeneteket minden csatlakoztatott modul megkapja, hogy aztán dönthessen a feldolgozás miként jéről. Ez tehát lényegében broadcast

jellegű kommunikációt jelent, lehetőleg multi-master kapacitással, tehát úgy, hogy bármelyik modul küldhessen üzenetet a közös interfészre.

A fentebb megválasztott vezérlő perifériái közé tartozik egy SPI master, valamint két SPI slave interfész. Kézenfekvőnek tűnik, hogy ezek segítségével valósítsuk meg a modulok kommunikációját, ez azonban az SPI protokoll sajátosságai miatt nem egyszerű feladat. Arbitráció helyett ugyanis ez az interfész fizikai engedélyező jeleket használ, amellyel a master aktiválhatja a kívánt slave bemenetét. Ebből következően az egy master, valamint két slave interfész, amivel a vezérlő rendelkezik, összesen legfeljebb három modul egymással való kétirányú összeköttetésére alkalmas.

Az SPI-hoz hasonlóan népszerű áramkör-közi kommunikációs interfész az I^2C. Számos SPI-I^2C átalakító áramkör kapható készen, így lehetséges ennek az alkalmazása is a modulok közötti kommunikációhoz, aránylag kis alkatrészszám-növekedéssel. Előnye az SPI-hoz képest, hogy engedélyező jelek helyett az I^2C már címzést használ, ezzel lehetővé téve multi-master alkalmazás esetén az arbitrációt. A broadcast jellegű kommunikációnál már nem ennyire egyértelmű a támogatottság. Létezik egy speciális, úgynevezett "general call" cím, amelyet minden, a kommunikációban résztvevő eszköznek figyelnie kell, ez azonban csak bizonyos parancsoknál használható, az interfész dokumentációja pedig ezt a használati módot nem tisztázza egyértelműen.

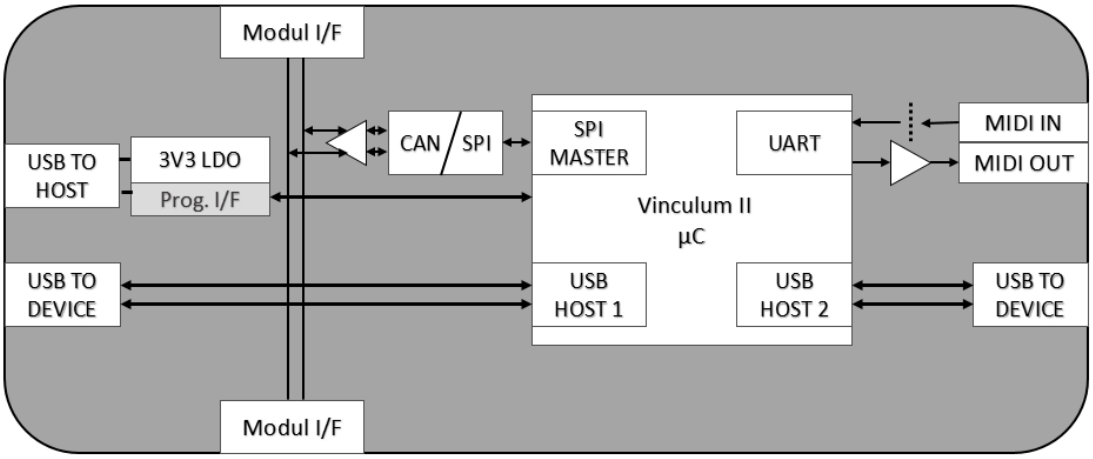
Létezik egy olyan interfész, amely ezzel szemben üzemszerűen valósítja meg a broadcast kommunikációt, ráadásul multi-master támogatással, ez pedig a CAN-busz. Robusztusságát mutatja autóipari felhasználása, ahol ma már nélkülözhetetlenné vált a járművek részegységeinek együttműködésében. A CAN protokolljának fontos részét képezi a fejlett arbitráció, valamint a hibadetektálás. Utóbbi jelentősen bonyolítja a kommunikációs algoritmust, ami számunkra hátrányt jelent. Ez a hátrány azonban kiküszöbölhető, mégpedig a Microchip gyártó egyik termékével, amely lényegében egy SPI interfésszel rendelkező CAN-busz vezérlő. Segítségével a CAN protokoll kezelése a vezérlő paraméterezésére, valamint adat küldésére és fogadására egyszerűsödik SPI interfészen keresztül, amivel a választott központi vezérlő rendelkezik. Az arbitrációt, újraküldést, hibadetektálást mind önállóan végzi a CAN-vezérlő a konfigurálása során megadott paraméterek alapján, ezzel jelentősen csökkentve a modulközi kommunikációt biztosító jövőbeni szoftverfejlesztési munkát.

A CAN-protokoll erre a célra való felhasználásának üzemszerűsége, széleskörű, minden eshetőségre kiterjedő dokumentáltsága, valamint az önálló vezérlő rendelkezésre állása miatt a CAN-busz mellett döntöttem.

4.5. Választott rendszerterv

A fenti elgondolások mentén kialakult rendszer központi eleme tehát az FTDI terméke, a Vinculum II mikrovezérlő, amely külön segédáramkör nélkül képes az USB eszközök kezelésére két USB host portjának köszönhetően. A tápellátás és a programozás egy USB-B csatlakozón keresztül lesz lehetséges, a mikrovezérlő számára szükséges 3.3 V tápfeszültséget lineáris szabályzó fogja előállítani. A dolgozat célja ennek a funkcionalitásnak a teljes körű megvalósítása.

A könnyű továbbfejleszthetőség érdekében az áramkörön helyet kapnak majd a hagyományos MIDI interfész kialakításához szükséges csatlakozók, meghajtó áramkör és optocsatoló, valamint a modulok közötti adatátvitelt lehetővé tevő csatlakozók, SPI-CAN átalakító és differenciális buszmeghajtó áramkör. Az ezekhez szükséges szoftverkomponensek megvalósítása későbbi fejlesztési munka tárgya.



4.1. ábra. *A megvalósításra kiválasztott, részletes rendszerterv*

5. fejezet

Hardvertervezés

Ebben a fejezetben az áramkör megtervezését fogom bemutatni, beleértve a megvalósítási lehetőségeknél felvázolt megoldásokhoz szükséges alkatrészek kiválasztását, a kapcsolás logikai, valamint fizikai tervezésének és megvalósításának menetét.

5.1. Főbb komponensek kiválasztása

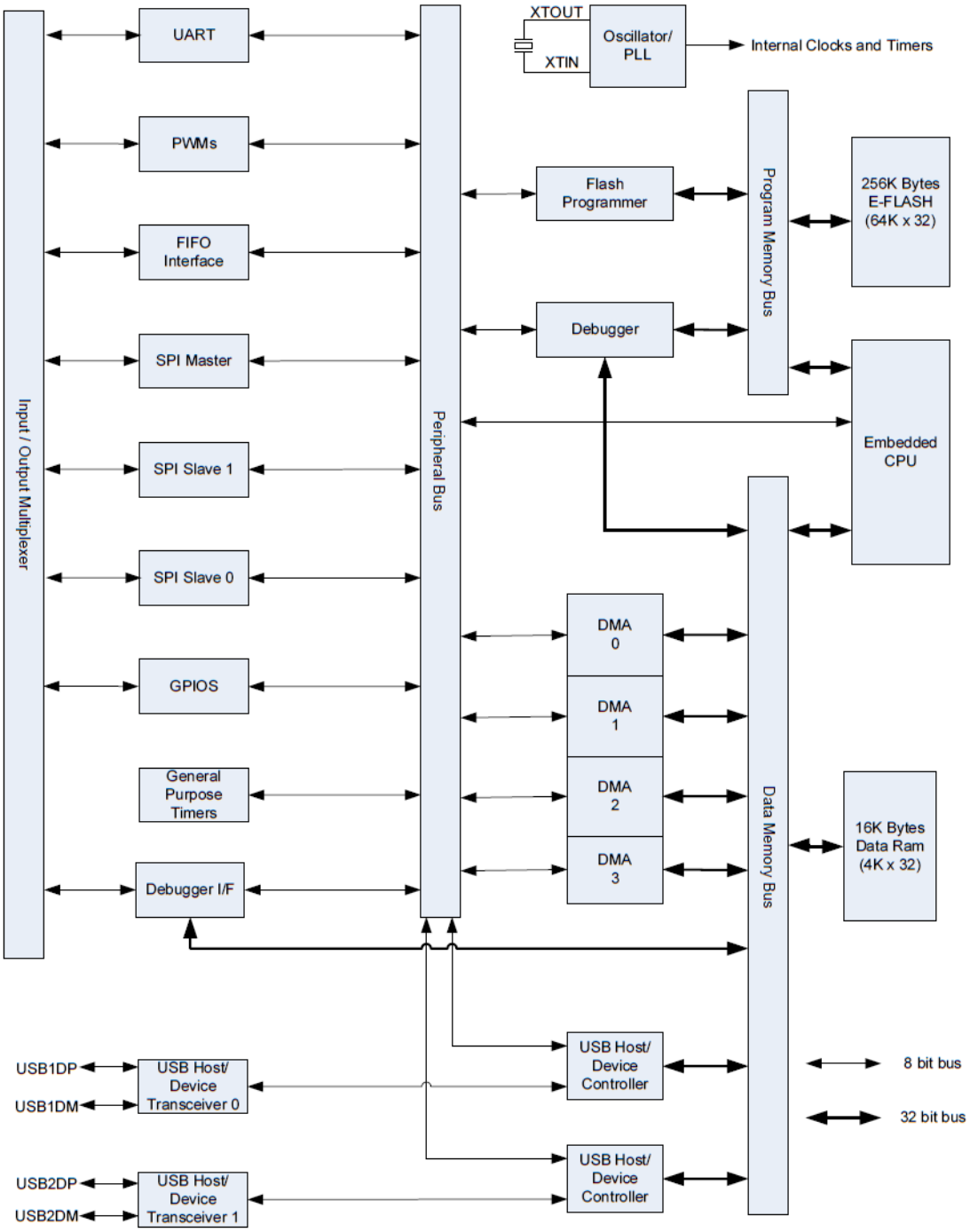
5.1.1. Központi egység

A kapcsolás központi eleme a mikrovezérlő, amely a már említett Vinculum II termékcsalád egyik tagja lesz. Figyelembe véve a bővített funkcionalitású modul perifériáit, valamint szem előtt tartva a szerelhetőséget és könnyű beszerezhetőséget, a 48 kivezetéssel rendelkező LQFP tokozású változatot találtam a kínálatból a legmegfelelőbbnek. Ennek a pontos típusszáma: VNC2-48L1B

A vezérlő fontosabb paraméterei [1]:

- **16 bites Harvard architektúra**
- **256 kB flash memória** (128k x 16 bit)
- **16 kB RAM** (4k x 32 bit)
- **UART interfész** 6 Mbaud maximális sebességgel
- **SPI interfész** (2 x slave, 1 x master)
- **FIFO interfész** (8 bit szélességű)
- **PWM kimenet** (8 csatorna)
- **28 konfigurálható I/O kivezetés** (LQFP-48 tokozás esetén)
- **12 MHz oszcillátor** az órajel előállítására külső kvarckristály alapján

A vezérlő egyszerűsített blokkdiagramja a 5.1. ábrán látható. A perifériák együttműködését DMA-vezérlők biztosítják, leszámítva az USB interfészeket, amelyek saját vezérlő segítségével férnek hozzá a processzor adatbuszához. Erre a nagy sebesség miatt van szükség, az USB vezérlők ugyanis 32 bit széles buszon keresztül kommunikálnak a processzorral a többi periféria 8 bit széles buszához képest.



5.1. ábra. *A Vinculum II mikrovezérlő blokkdiagramja [1]*

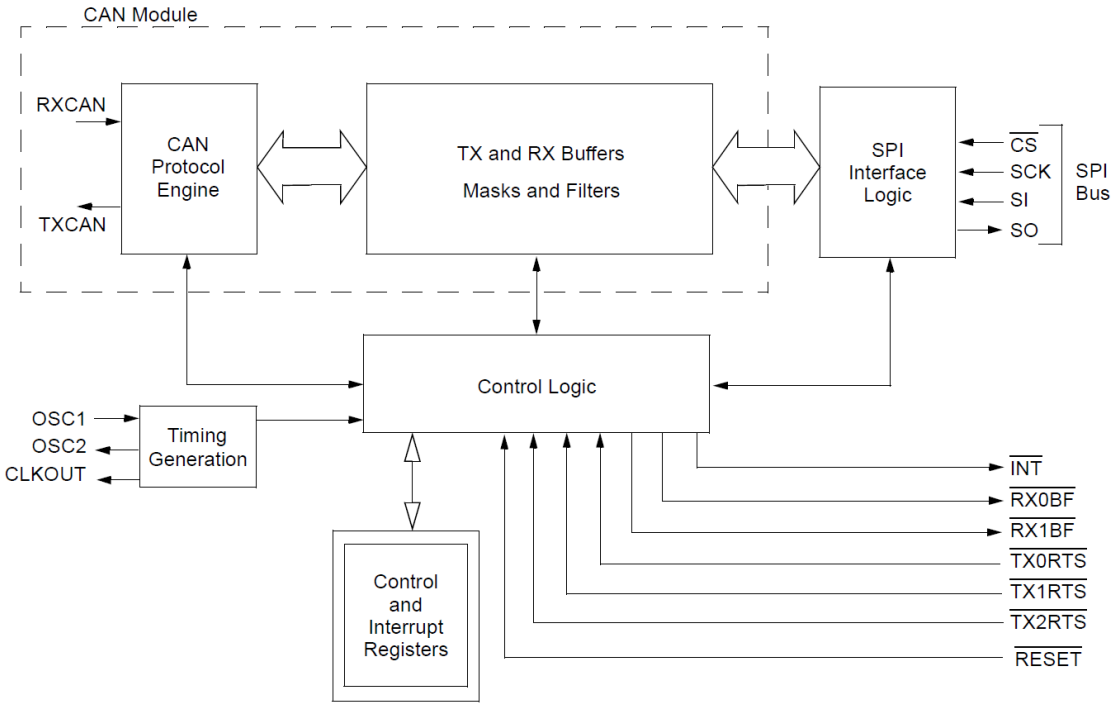
5.1.2. CAN interfész

Szó esett még a Microchip által gyártott SPI-CAN átalakítóról. Ez igényel még egy, a CAN busz fizikai kezelését, védelmét lehetővé tevő meghajtó áramkört. Az átalakítóból a TSSOP tokozásút találtam szerelhetőség és méret szempontjából is optimálisnak, típusa: MCP2515-I/ST

Főbb paraméterek [7]:

- **CAN V2.0B specifikáció megvalósítása** (1 Mb/s adatátviteli sebesség)
- **Két vételi puffertár** maszkolás és szűrés funkciókkal
- **Három adó puffertár**
- **SPI interfész** (10 MHz maximális sebesség)
- **Státuszjelző kivezetések pufferenként** (használata opcionális)

A vezérlő blokkdiagramja a 5.2. ábrán látható. Minden funkció és állapotjelző elérhető az SPI interfészen keresztül, a dedikált kivezetések használata opcionális. A központi vezérlő megszakítás-alapú működése esetén ezek felhasználása javíthatja az átvitel késleltetését.



5.2. ábra. *Az önálló CAN-vezérlő blokkdiagramja [7]*

Meghajtó áramkörnek az adatlap által ajánlott, SOIC tokozású változatot választottam, amely szintén a Microchip terméke. Kódja: MCP2551-I/SN

5.1.3. MIDI interfész

A fentieken túl a hagyományos MIDI interfész megvalósításához szükséges még az alkatrészek megválasztása. A specifikációban található ajánlásokat [6] követve a MIDI kimenet meghajtásához két, egymással sorosan kapcsolt invertáló meghajtót fogok használni. A célra egy általános inverter is megfelel, a Texas Instruments által gyártott SOIC tokozású változat kódja: CD74HC04M

Szintén a specifikáció ajánlja optocsatoló gyanánt a 6n138 konstrukciót, amelyet többek között a Fairchild Semiconductor gyárt. Az SDIP tokba szerelt verzió típusszáma: 6N138SD

5.2. Áramkörtervező szoftver

Szakmai gyakorlatom, és az azóta végzett gyakornoki munkám során a Mentor Graphics áramkörtervező szoftvercsomagját használtam, és használom jelenleg is, immár három éve, ezért egyértelmű volt, hogy ennek a hardvernek a megvalósítását is e szoftver segítségével végzem majd. A szoftvercsomag az Xpedition Enterprise VX1.2 névre hallgat, igen összetett, professzionális tervezőkörnyezetet kínál. Munkám során összesen négy szoftverkomponenst fogok használni.

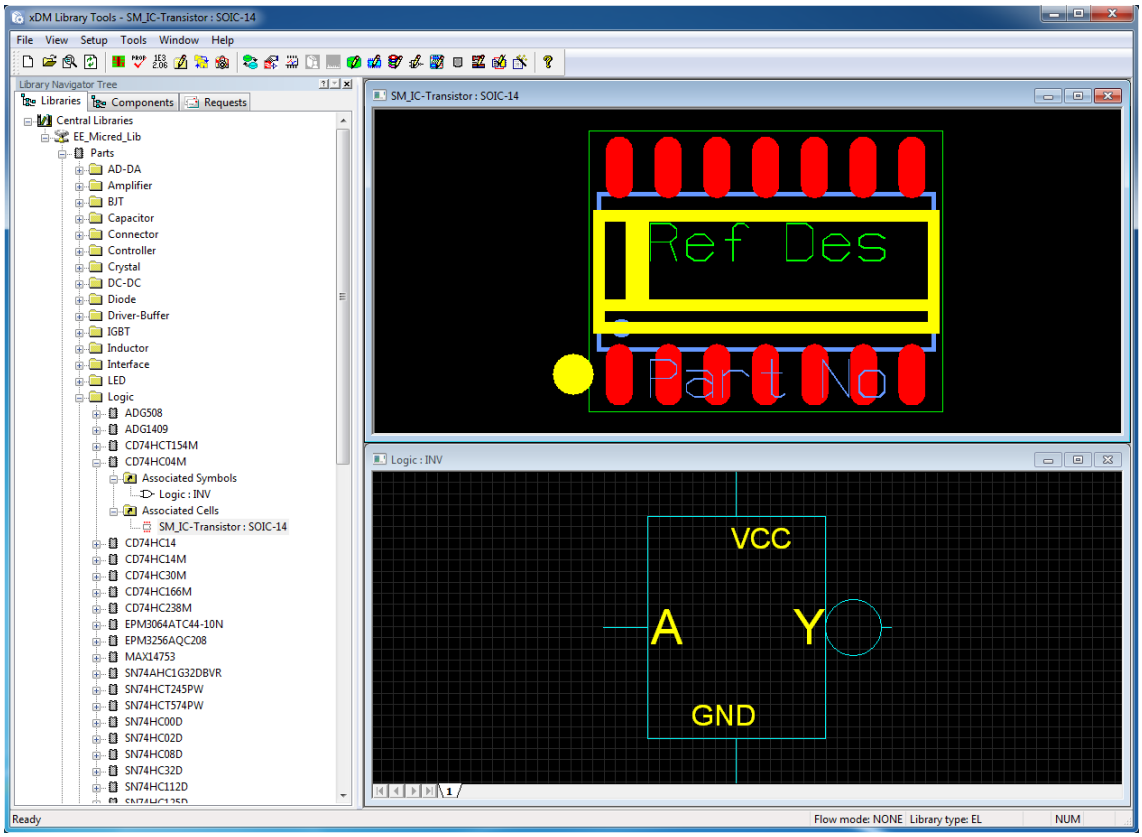
5.2.1. xDM Library Tools

Az alkatrészkönyvtár kezelésére szolgáló program, segítségével alkatrészekhez tartozó rajzjelek, valamint footprintek hozhatóak létre, módosíthatóak, csoportosíthatóak, továbbá ezek egymáshoz rendelése is ezzel a programmal végezhető (5.3. ábra).

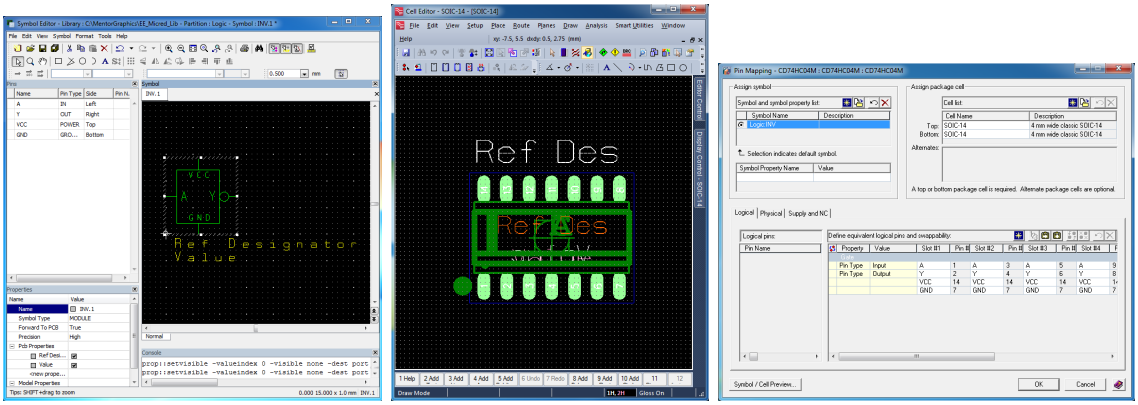
Számos alprogram által adódik ez a funkcionalitása, ezek közül a legfontosabbak a Symbol Editor (rajzjel szerkesztő, 5.4. ábrán bal oldalt), a Cell Editor (footprint szerkesztő, 5.4. ábrán középen), és a Part Editor (alkatrész szerkesztő, 5.4. ábrán jobb oldalt).

Utóbbival történik egy adott alkatrészhez tartozó rajzjel és footprint összekapcsolása. Egy alkatrészhez általában egy rajzjel, a kívánt tokozástól függően viszont akár többféle footprint is tartozhat. A lábkiosztásnak ebben az esetben egyértelműnek kell lennie minden footprint esetén. Ennek az a következménye, hogy a footprinteknek vagy azonos lábszámmal kell rendelkezniük, vagy ha nem így van, akkor a nagyobb lábszámmal rendelkező tokozás esetén a különbséget csak tápfeszültség bemenetre szolgáló, vagy úgynevezett no-connect lábak képezhetik. Ellenkező esetben új alkatrészt kell létrehozni az új tokozás számára, adott esetben ugyanazzal a rajzjellel. Egy rajzjel tehát több alkatrészhez is tartozhat.

A program lehetőséget nyújt az összetett alkatrészek rugalmas kezelésére is, például amikor ugyanazon áramkörből többet valósítanak meg a gyártók egy tokozásban. Erre jó példa a MIDI kimenethez kiválasztott invertáló áramkör, amelyből összesen hat darab foglal helyet a 14 kivezetéssel rendelkező SOIC tokozásban, a kapcsolási rajzban azonban ez nem egyetlen nagy rajzjelként fog megjelenni, hanem kisebb rajzjelekként, amelyek mindegyike egy-egy invertáló áramkört reprezentál. Így nem szükséges, hogy a fizikailag egy tokban elhelyezkedő áramkörök be- és kimenetei a kapcsolás alapvetően logikai működését ábrázoló rajzán is kényszerűen egymás mellett legyenek.



5.3. ábra. *Az xDM Library Tools kezelőfelülete*



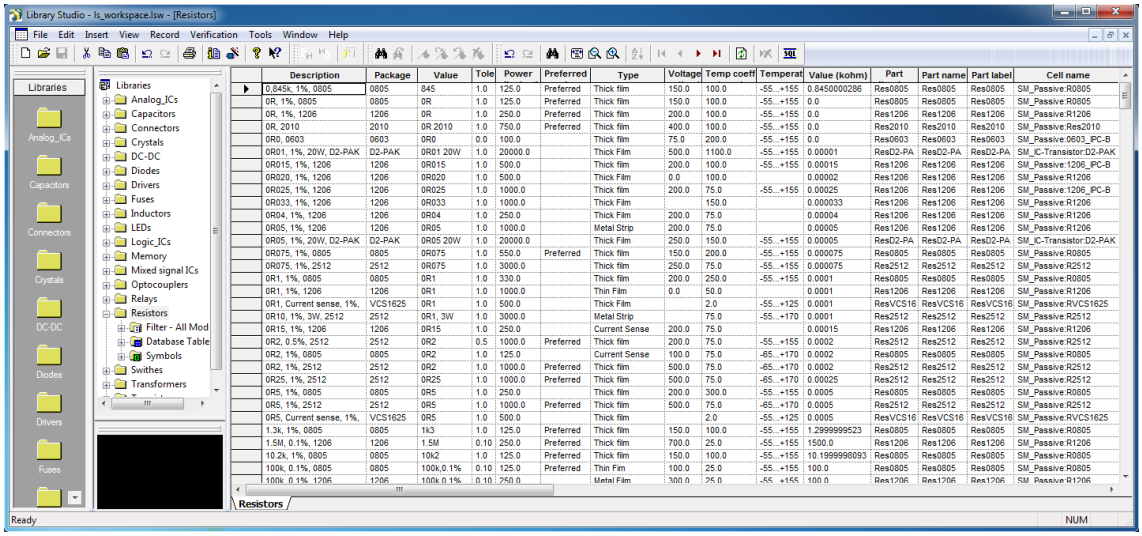
5.4. ábra. *A Library Tools fontosabb alprogramjai*

Korábbi munkáim eredményeképp az alkatrész-adatbázist csak a fentebb megválasztott, ritkábban előforduló alkatrészekkel kellett kiegészítenem, a kiegészítő alkatrészek esetén (ellenállások, kondenzátorok) olykor szerepet játszott az értékek megválasztásában az is, hogy az adott paraméterekkel rendelkező alkatrész már szerepelt-e az adatbázisban, ezzel gyorsítva a tervezést.

5.2.2. Library Studio

Ez a szoftver lényegében egy adatbázis-kezelő eszköz (5.5. ábra). Arra szolgál, hogy a rajzjelből és footprint(ek)ből álló alkatrészekhez további paramétereket társítva konkretizálni

lehessen az adott típusú, értékű alkatrészt, hogy aztán ez a kapcsolási rajz szerkesztésekor felhasználható legyen.



5.5. ábra. *A Library Studio kezelőfelülete*

Vegyünk példának egy 0805 méretű SMD ellenállást. A rajzjel ebben az esetben a szokásos téglalap alak, a footprint pedig a 0805 tokozás szabványos méretű és távolságban elhelyezkedő rézfelületei. Ez a Library Tools programban már egy elkészült alkatrészként jelenik meg, azonban ebben a formájában még nem tudjuk felhasználni a tervezésnél, mivel nincs ellenállás értéke, nem tudjuk a rákapcsolható legnagyobb feszültséget és áramot, valamint azt sem, hogy melyik gyártó terméke, mi a pontos termékkódja, melyik forgalmazótól szerezhető be, stb. E paraméterek megadására szolgál a Library Studio. Ez a lépés elsőre talán feleslegesnek tűnhet, de nagyban segíti az adatbázisok átláthatóságát.

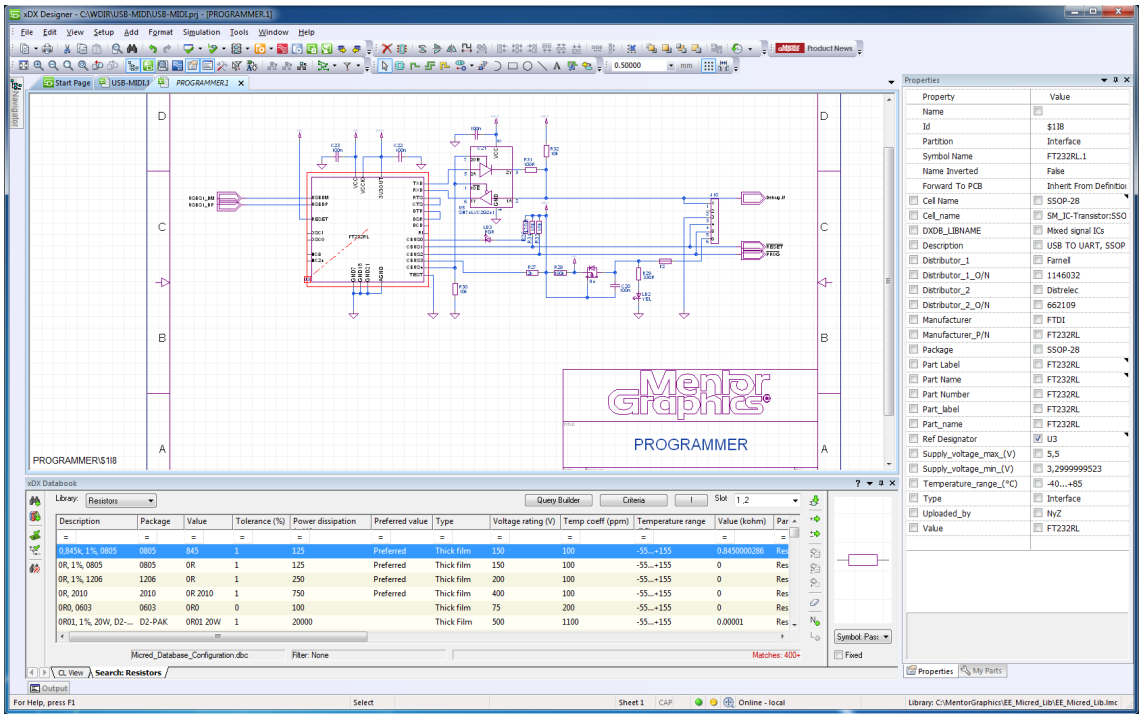
A Library Studio szabványos .mdb kiterjesztésű adatbázisban tárolja az adatokat, ezt éri el a kapcsolási rajz szerkesztő szoftver. Az elérési utat és az adatbázis struktúráját egy konfigurációs fájlban lehet továbbítani a többi szoftver számára.

5.2.3. xDX Designer

A szoftvercsomag kapcsolási rajz szerkesztő komponense (5.6. ábra). A felhasználható alkatrészeket a fentiek szerint kialakuló adatbázisból lehet kiválasztani, majd a célnak megfelelően kialakítani az összeköttetéseket.

Egy projekt feloszthatóboardokra, azon belül pedig a boardhoz tartozó kapcsolási rajz lapokra és/vagy blokkokra. Egy boardhoz egy fizikai terv társul, tehát minden kapcsolási rajz, ami a boardon belül található, egyetlen huzalozási terv részét képezi. Esetünkben a projekt csak egy boardot fog tartalmazni, aminek a kapcsolási rajza a könnyű áttekinthetőség kedvéért blokkokból fog összeállni. A blokk annyiban különbözik a laptól, hogy az alkatrészekhez hasonlóan van rajzjele, így a rajzrészleteket hierarchikusan lehet szervezni, amivel lényegében bármekkora összetettségű kapcsolási rajz gyorsan átláthatóvá tehető.

A kapcsolási rajz elkészülte után a program a projekt alapján megalkotja az egyes fizikai tervekhez tartozó alkatrész- és összeköttetés listát, majd ezeket egy .pcb kiterjesztésű



5.6. ábra. *Az xDX Designer kezelőfelülete*

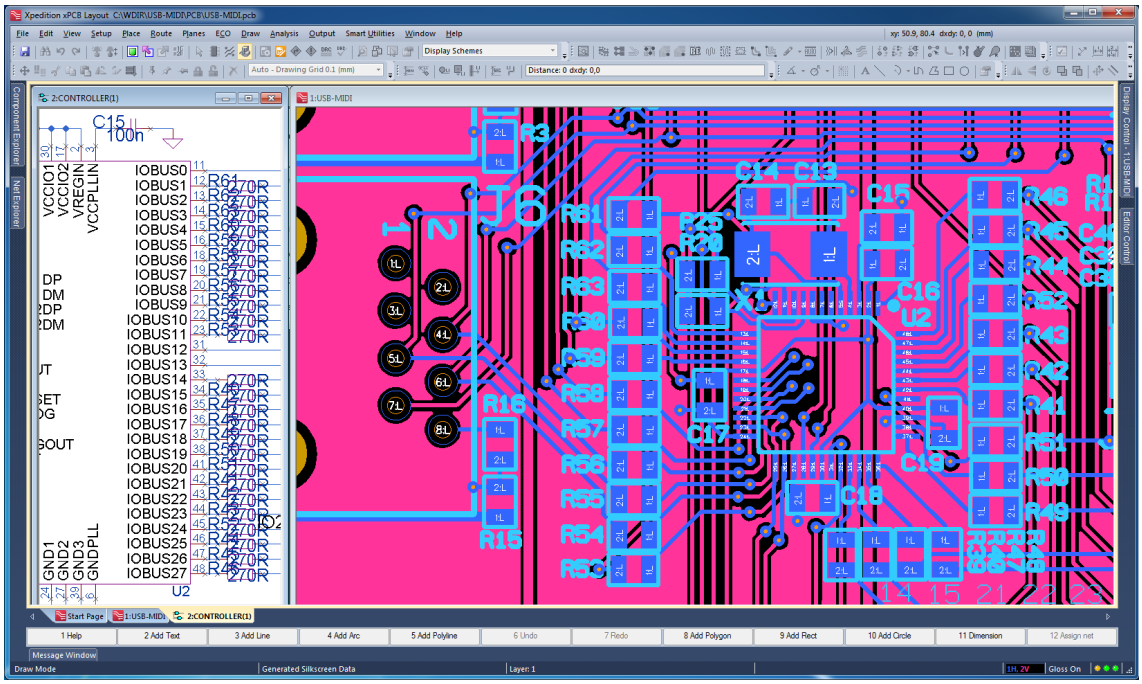
fájlban és a hozzá tartozó könyvtárszerkezetet létrehozva továbbadja a huzalozástervező komponensnek.

5.2.4. xPCB Layout

Az említett huzalozástervező program (5.7. ábra). Értelemszerűen szoros összeköttetésben áll a kapcsolási rajz szerkesztővel, mind az összeköttetések, mind az alkatrészek tekintetében beállt bármiféle változás esetén a fizikai terv frissítése szükséges.

A fizikai tervezés megkezdése mindig a kényszerezési szabályok beállításával kezdődik. Erre a Constraint Editor alprogram szolgál, ahol beállítható a minimális vezetékszélesség, a vezetékek feszültségszint szerint csoportokba szervezhetők, ami alapján a szigetelési távolságok beállíthatóak az egyes feszültségszintek között, továbbá itt lehet beállítani a differenciális jelvezetés paramétereit, és megadni, hogy mely vezetékek alkotnak differenciálpárokat. Esetünkben csak az utóbbi funkcióra volt szükség az USB adatjelei miatt, a többi paraméter alapbeállítása megfelelő volt, mivel alacsony, logikai szintű feszültségek vannak csak jelen az áramkörön.

A kényszerezés beállítása, az alkatrészek elhelyezése, és a huzalozás megtervezése után a program elkészíti a különböző rétegekhez tartozó, szabványos formátumú gerber-fájlokat, amelyek segítségével az áramkör gyárthatóvá válik, akár gyártólabor, akár házilagos módszerek segítségével.



5.7. ábra. *Az xDX Layout kezelőfelülete*

5.3. Kapcsolási rajz

A kapcsolás alapjául a mikrovezérlő dokumentációjának részét képező alkalmazási példák szolgáltak. A ki- és bemenetek védelme, a tápfeszültség szűrése, valamint az órajel előállításának módja mind megtalálható a vezérlő egyik készre szerelt kivezető áramkörrel kapható változatának kapcsolási rajzában, amelyet a gyártó nyilvánosan elérhetővé tett.

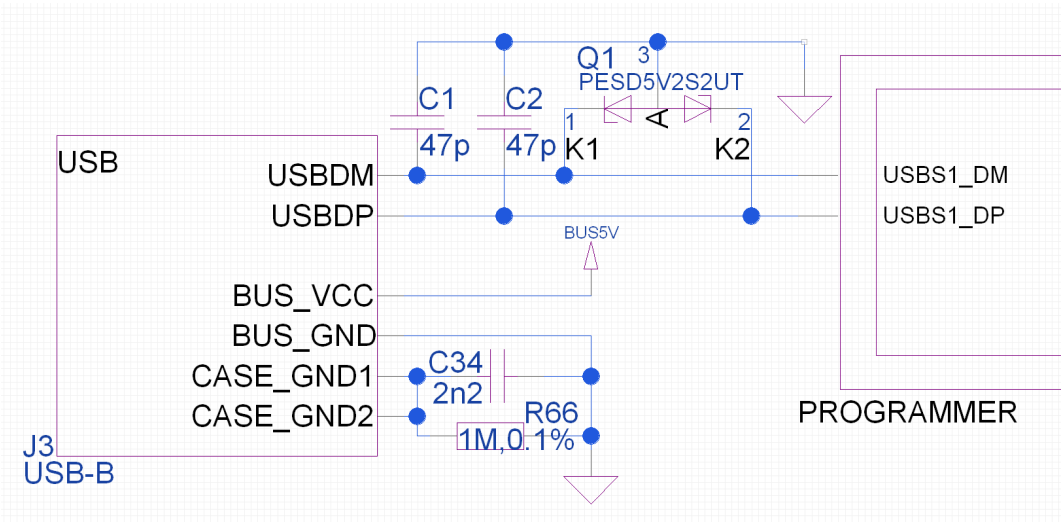
Ugyanígy jártak el a programozó és hibakereső interfész kapcsolásával, így lehetővé téve a fejlesztők számára önállóan programozható egység létrehozását, a külön kapható programozó megvásárlása, és az ahhoz tartozó csatlakozó használata helyett. Mivel ez az áramkör igen egyszerű felépítésűnek bizonyult, ezért én is a programozó interfész áramköri megvalósítása mellett döntöttem, a biztonság kedvéért meghagyva a lehetőséget a külső programozó használatához szükséges csatlakozó beszerelésére.

A kapcsolási rajzot logikailag összefüggő blokkokba szerveztem, a működés részletes kifejtését is e blokkoknak megfelelően fogom tenni.

5.3.1. Fő rajz

Ez a lap tartalmazza az egyes blokkok közötti összeköttetéseket, valamint a csatlakozókat, és a hozzájuk tartozó, áramköri védelemre szolgáló alkatelemek többségét. A legnagyobb feladat az USB-csatlakozók megfelelő áramköri védelmének kialakítása volt (5.8. ábra).

A C1 és C2 kondenzátorok az adatjelek nagyfrekvenciás zajszűrését hivatottak megvalósítani, míg a C34 kondenzátor és R66 ellenállás párhuzamos kapcsolásával a csatlakozó shield felülete és a földpotenciál közé az esetleges ESD jelenségek káros hatásai csökkent-



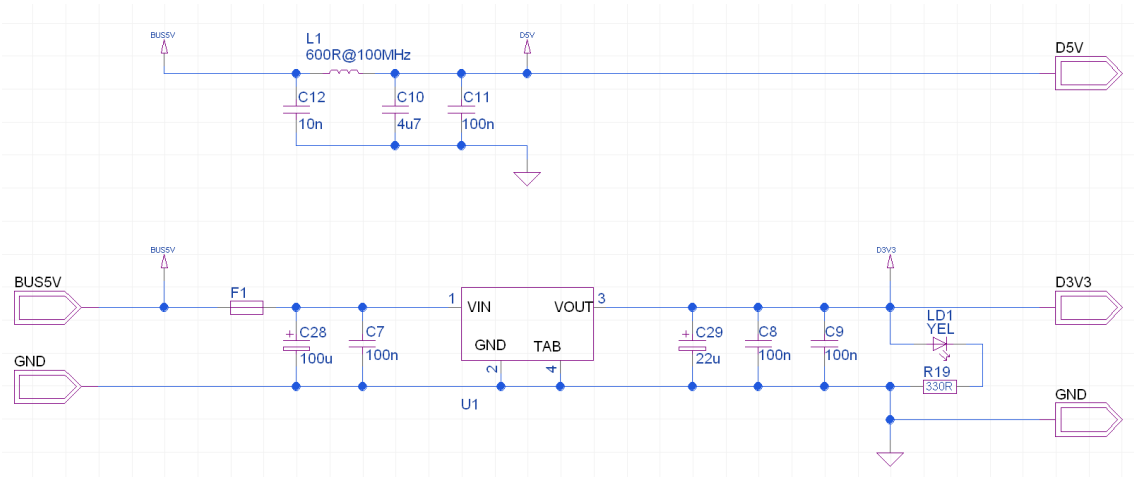
5.8. ábra. *Kapcsolási rajz főoldal részlet: USB áramköri védelem*

hetőek. A kondenzátor 2 kV feszültségig működőképes, az ellenállás pedig 1206 tokozású, hogy a lábai között ne üthessen át a sztatikus feltöltődésből eredő feszültség. A Q1 jelű eszköz két, közös anóddal rendelkező 5.2 V névleges feszültségű zener dióda, az adatjelekre kerülő feszültségcsúcsok, üzemzavarból eredő túlfeszültség elvezetése érdekében.

A többi csatlakozónál ezekhez hasonló megoldásokat alkalmaztam, valamint soros ellenállásokat, ahol szükséges volt. A CAN-busz, valamint SPI interfészek kivezetésére szabványos RJ-45 csatlakozókat választottam, ezekhez kaphatóak szerelt vezetékek csavart érpárokkal, ami a CAN differenciális jeleihez ideális.

5.3.2. POWER blokk

Az USB-B csatlakozó tápfeszültség kivezetésein beérkező 5 V buszfeszültség szűrését, valamint abból a mikrovezérlő számára szükséges 3.3 V tápfeszültség előállítását végzi (5.9. ábra).



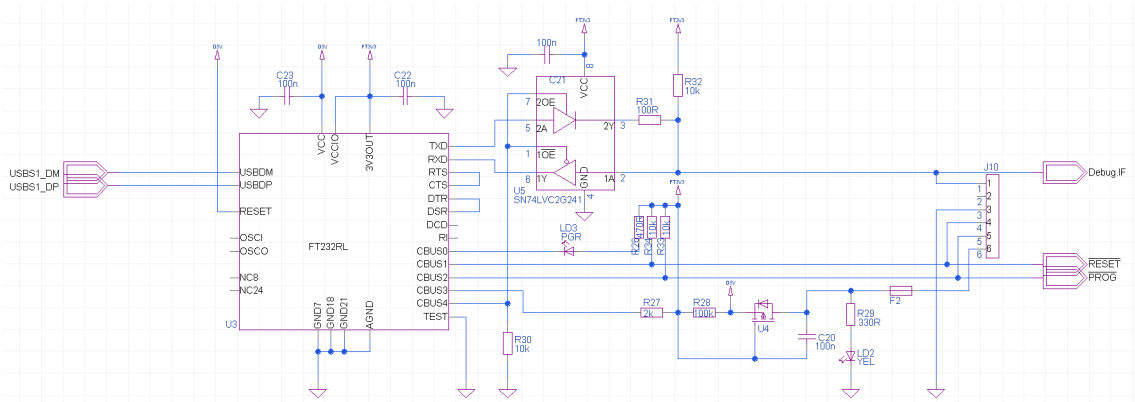
5.9. ábra. *Kapcsolási rajz részlet: POWER blokk*

A lineáris feszültségszabályzó (U1) gyártója, valamint a mikrovezérlő modul kapcsolási

rajza által ajánlott puffer- és szűrőkondenzátorokon kívül található még a bemeneten egy 500 mA áram felett működésbe lépő, hőelem-alapú biztosíték, amely a túlterhelés megszűnése esetén újra vezetőképessé válik. A kimeneten további pufferelést követően található még egy visszajelző LED is (LD1), amely a 3.3 V tápfeszültség meglétét hivatott jelezni.

5.3.3. PROGRAMMER blokk

Gyakorlatilag egy az egyben a mikrokontroller gyártója által közreadott programozó interfész kapcsolásának megvalósítása (5.10. ábra). Látható, hogy egy USB-UART átalakítón alapszik a működése, amelyet megfelelő konfigurálás, és a kétirányú kommunikáció egyetlen vonalra redukálása után lehet a mikrokontroller programozására használni.



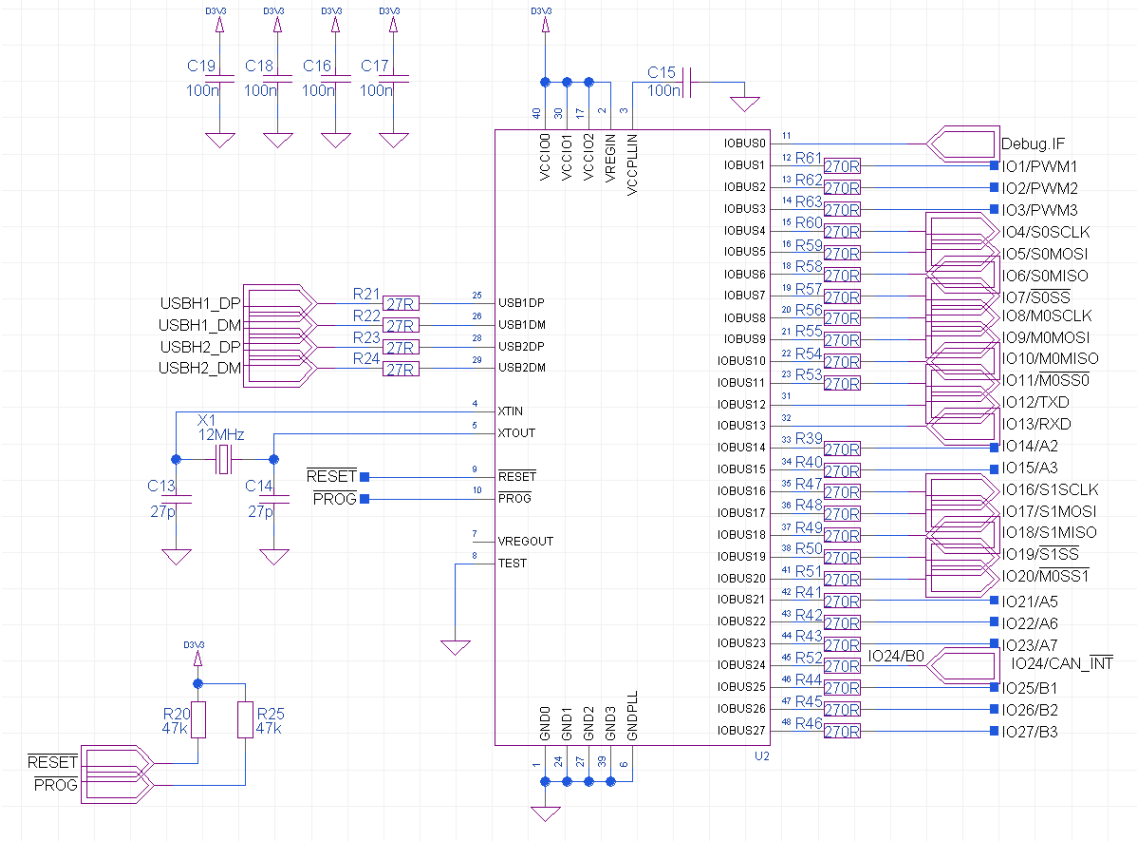
5.10. ábra. *Kapcsolási rajz részlet: PROGRAMMER blokk*

A népszerű, FT232R típusú átalakító áramkörön (U3) kívül egy kétirányú, irányonként engedélyezhető meghajtó áramkör (U5), valamint egy kapcsoló FET (U4), és a már említett programozó csatlakozó (J10) található ebben a blokkban a szükséges tápszűrő kapacitásokon, és a logikai jelek fel/lehúzására szolgáló ellenállásokon kívül. A FET arra az esetre kell, amikor a programozón keresztül történik a táplálás. Esetünkben erre nem lenne feltétlenül szükség, mivel ugyanazon az USB csatlakozón érkezik a tápfeszültség a programozó, valamint az áramkör többi eleme felé, így azonban a programozó némi módosítással önmagában is használható lesz a jövőben, adott esetben más áramkörök programozására is. A csatlakozó alapja egy egyszerű, 2.54 mm rasztertávolságú tüskesor, amelynek az egyik lábát el kell távolítani beforrasztás előtt, lehetővé téve az úgynevezett kulcsolt programozókábel használatát. Így a programozandó áramkör védve van a fordított csatlakoztatásból eredő hibák akár végzetes következményeitől.

5.3.4. CONTROLLER blokk

Ennek a blokknak is az FTDI fejlesztőmoduljának elérhetővé tett kapcsolási rajza képezi az alapját, a mikrokontrollert, és perifériáit tartalmazza (5.11. ábra).

Az órajel forrása egy 12 MHz frekvenciájú jelet generálú kvarckristály (X1), valamint az oszcillátor működéséhez szükséges, kis értékű kondenzátorok (C13, C14). A szokásos tápszűrésen, és a programozásra szolgáló kivezetések felhúzó ellenállásain túl 270 ohm értékű



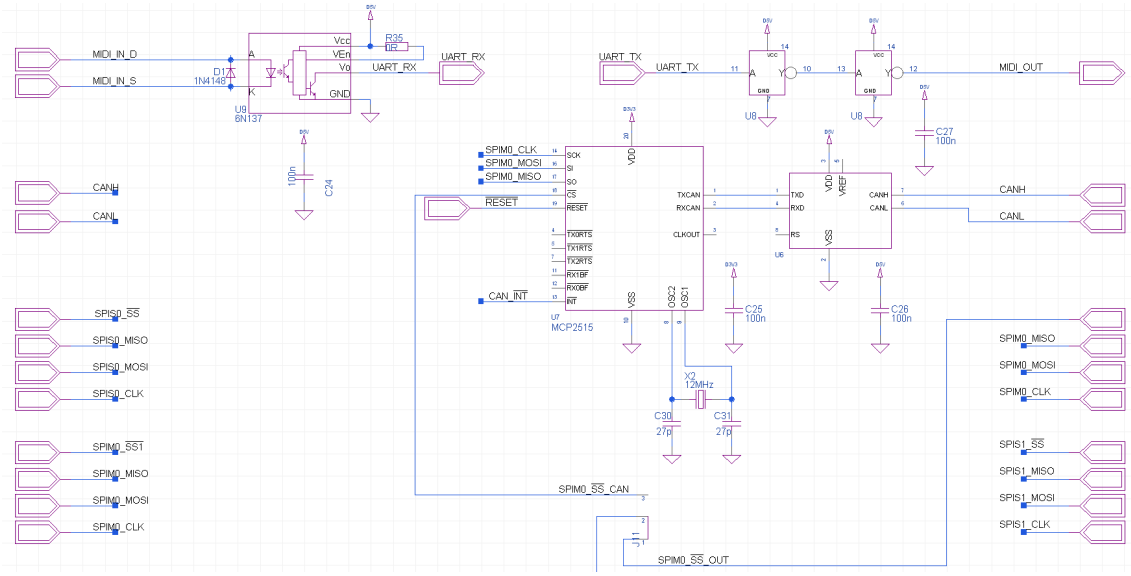
5.11. ábra. *Kapcsolási rajz részlet: CONTROLLER blokk*

soros ellenállások találhatóak a ki- és bemenetek nagy részén, elkerülendő a mikrokontroller lábainak túlterhelését, legyen szó akár a bemenetek különböző feszültségszintekre húzásáról, akár LED-ek meghajtásáról. A fentieken túl az USB adatvonalakra előírt 27 ohm értékű ellenállások (R21-R24) láthatóak még a rajz bal oldalán.

5.3.5. DRIVER blokk

Amiről még nem esett szó, azok a perifériák kezeléséért felelős áramkörök, ezeket foglalja magában ez a blokk (5.12. ábra).

A CAN-busz vezérlő és meghajtó áramkörök (U7, U6) láthatóak az ábra közepén. Előbbi bemenetére a mikrovezérlő SPI master interfészét kellett illeszteni, kimenetére pedig a meghajtó áramkör bemeneteit, figyelve arra, hogy itt nem az UART-nál megszokott konvenció volt érvényes, miszerint az adó Tx kimenetét a vevő Rx bemenetére kellene illeszteni, hanem egyenes összeköttetéssel Tx-Tx, Rx-Rx párokat kell alkotnia a ki- és bemeneteknek (a CAN meghajtó áramkörnél a Tx bemenetet, az Rx pedig kimenetet jelent). Ezen kívül a CAN vezérlő számára itt is 12 MHz órajel előállítása volt szükséges. Felmerült annak a lehetősége, hogy a mikrovezérlő és a CAN vezérlő ugyanazon órajellel működjön, ami lehetséges lett volna a CAN vezérlő "CLKOUT" kimenetének felhasználásával, ez a megoldás azonban logikailag a CAN vezérlő alá rendelte volna a mikrovezérlőt, az órajelek szinkronjának pedig ebben az esetben nincs különösebb jelentősége, ezért maradtam végül a két órajel egymástól független előállításánál.



5.12. ábra. *Kapcsolási rajz részlet: DRIVER blokk*

Ezen kívül a MIDI ki- és bemenetek megvalósítása látható még a rajzon, ez teljes mértékig a specifikációban foglaltakat követi. Látható, hogy a két inverter szimbólum ugyanazt az alkatrész-azonosítót kapta (U8), ez a már említett összetett alkatrész megvalósítása miatt van így. Említést érdemel még a bemenet optocsatolója mellett található ellenállás (R35), amely kompatibilitási okokból került az áramkörbe. Az ajánlott 6n138 optocsatoló beszerezhetősége kétséges volt, azonban az ahhoz hasonló 6n137 típus is alkalmasnak bizonyult a feladatra, azzal a különbséggel, hogy utóbbi rendelkezik engedélyező kivezetéssel, amelyet a tápfeszültségre kell húzni, hogy működjön a kimenet. Erre szolgál ez az ellenállás.

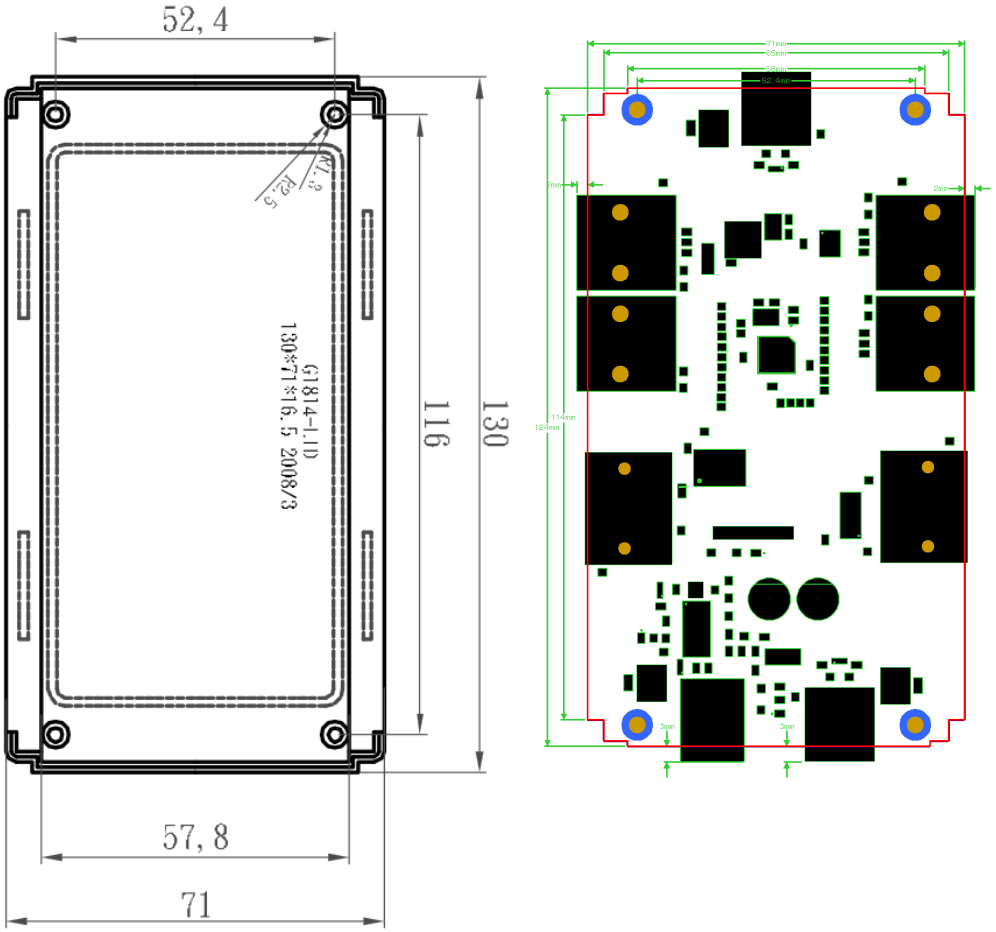
5.4. Áramköri terv

A logikai felépítés megalkotása után meg lehet kezdeni a fizikai tervezést. Ennek főbb lépései az alkatrészek elhelyezése, a huzalozás kialakítása, majd a beültetési ábra igazítása.

5.4.1. Alkatrészek elhelyezése

Első lépésként ki kell alakítani az áramkör befoglaló alakját, minimálisan szükséges méreteit, hogy lehetőség nyíljon alkalmas műszerdoboz választására. Aránylag nagy méreteik miatt a csatlakozókkal érdemes kezdeni, melyek elhelyezkedésével és irányával a logikai felépítést törekedtem követni (5.13. ábra).

Ennek megfelelően az áramkört felülről nézve a táp- és programozó USB csatlakozó, valamint az egyik USB host port alulra, a másik USB host felülre, az egyéb interfészek esetén pedig a bemenetek a bal, a kimenetek pedig a jobb oldalra kerültek. A mikrovezérlő, valamint további nagyobb alkatrészek elhelyezése után elegendő információ állt rendelkezésre, hogy kiválasszam a megfelelő dobozt.



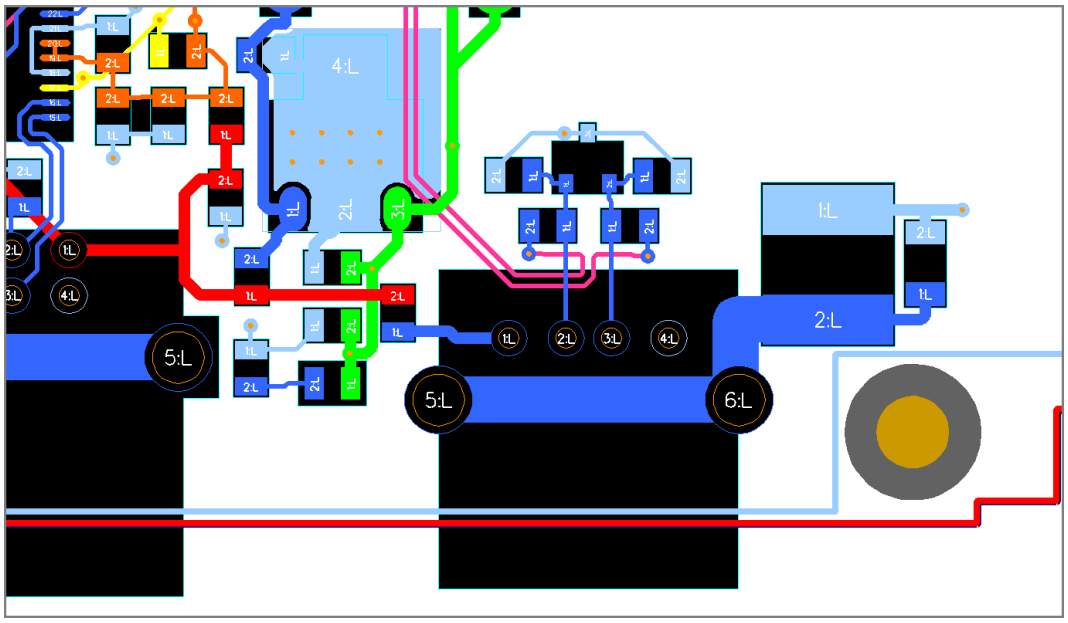
5.13. ábra. *Áramkörtervezés: műszerdoboz méretrajz (bal), alkatrész-elhelyezési terv (jobb)*

Az alkatrészek elhelyezésénél további szempont még a zavarvédelem, amit az egymással kapcsolatban álló alkatrészek távolsága erősen befolyásolhat, különös tekintettel a kommunikációs szálakra a mikrovezérlő és a perifériák között. Az USB esetében különösen fontos a differenciális jelvezetés miatt az adott porthoz tartozó adatvonal-védelem szimmetrikus, és minimális huzalhosszúságú kialakítása, de például a programozó áramköri blokk, valamint a tápfeszültség előállításáért felelős blokk alkatrészeit is igyekeztem egymáshoz minél közelebb, csoportosítva elhelyezni. Ugyanígy a CAN-busz kezeléséért felelős áramköri elemek is a hozzájuk tartozó RJ-45 csatlakozó magasságában kaptak helyet, ahogy a MIDI ki- és bemenetekhez tartozó alkatrészek is a MIDI csatlakozókhoz kerültek, értelemszerűen. A fentiek elvégzése után gondoskodtam a kisebb alkatrészek, kondenzátorok, ellenállások megfelelő elhelyezéséről is.

5.4.2. Huzalozás

Miután végleges helyükre kerültek a komponensek, valamint a korábban említetteknek megfelelően megtörtént a kényszerezési szabályok beállítása, beleértve az USB és CAN buszok differenciális párjainak kijelölését, kezdetét veheti a huzalozás. A vezetékek megkülönböz-

tetését elősegítendő a program lehetőséget nyújt a vezetékek funkció szerinti csoportosítására, és színekkel való megjelölésére (5.14.). Ez különösen a tápfeszültségek továbbítására szolgáló vezetékeknél hasznos.



5.14. ábra. *Áramkörtervezés: USB differenciális jelvezetés, áramkörvédelem, tápfeszültségek megkülönböztetése színekkel*

Az alkatrészek aránylag kis száma miatt két rétegen végeztem a huzalozást, ennél több réteg nem volt indokolt, ami előnyös az esetleges hibajavítások, és a bekerülési költség alacsonyan tartása szempontjából is. Mivel az áramkör legyártását az egyetem keretein belül működő UniPCB Kft. gyártólaborjában terveztem elkészíttetni, ezért az általuk megadott gyártási paramétereket vettem alapul mind a huzalozás kialakításánál, mind a kényszerezésnél.

5.4.3. Beültetési ábra

A dokumentáció alapján ugyan be lehet majd azonosítani az egyes alkatrészeket, azonban ezt a folyamatot mind a beültetésnél, mind az esetleges későbbi hibakeresésnél jelentősen meggyorsíthatja egy megfelelően kialakított beültetési ábra (angol nevén silkscreen). Ennek érdekében érdemes energiát fektetni a rajzolat minél egyértelműbb megvalósításába, beleértve az alkatrészek azonosítójának pozícióit, valamint az irányérzékeny komponensek (diódák, integrált áramkörök) irányítását jelölő grafikák elhelyezését.

5.5. Gyártás

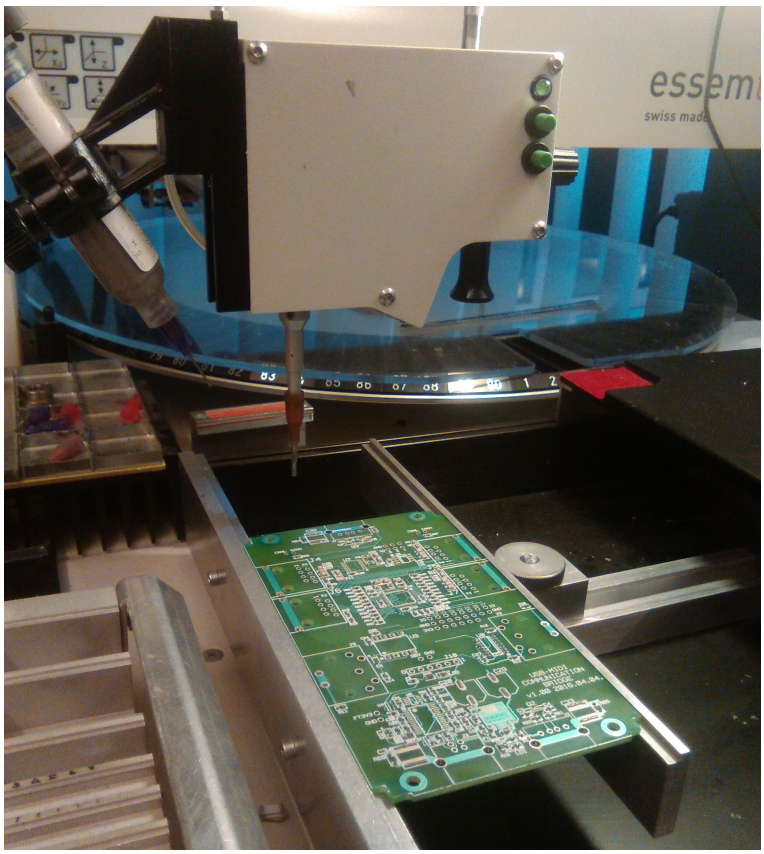
Az áramköri terv elkészülte, valamint a program által kiexportált gerberfájlok ellenőrzése után a gyártás következik.

5.5.1. Nyomtatott huzalozású lemez

A nyomtatott huzalozás elkészítését a házilagos gyártási módszerekben szerzett tapasztalataim, valamint idő hiányában jobbnak láttam a már említett UniPCB Kft. laborjában elkészíttetni. Ebben az esetben fontos tényező a gyártási határidő, ami jelentősen befolyásolja a bekerülési költséget.

5.5.2. Pasztázás

Gyakornoki munkám révén hozzáféréssel rendelkezem egy kézi pick-and-place géphez, ami az alkatrészek ültetése mellett pasztázásra is alkalmas. Van ugyanebben a laborban egy pasztázó keret is, ehhez azonban speciálisan az áramköri lemezhez gyártott pasztamaszkra lenne szükség, aminek a költsége az alkatrészek számához képest aránytalanul magas, ezért a kézi pasztázás mellett döntöttem az említett gép segítségével (5.15. ábra).



5.15. ábra. *Áramkörgyártás: Kézi pick-and-place gép, és az áramkör a paszta felhordása után*

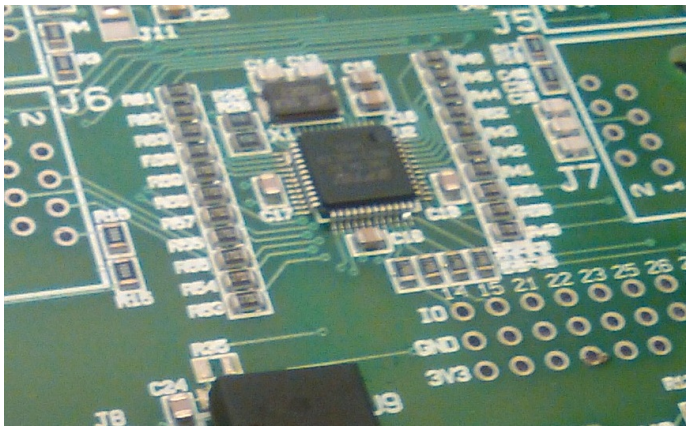
5.5.3. Beültetés

A forraszpaszta felvitele után következett az alkatrészek beültetése, szintén a kézi pick-and-place géppel. Az ültetés szabályainak megfelelően a legalacsonyabb alkatrészekkel kezdtem a munkát, hogy az ültetőgép mozgatása során a lehető legkisebb legyen a valószínűsége az ültetőfej, és egy már beültetett alkatrész ütközésének.

5.5.4. Reflow

A pasztázás és beültetés után a következő lépés a felhordott paszta megolvasztása reflow eljárás segítségével, amihez szintén rendelkezésemre állt egy speciális kemence. Az előre beállított hőprofilok között volt olyan, ami megfelelt a felhasznált alkatrészek forrasztási követelményeinek, így nem volt szükséges új hőprofil felvételére.

Az újraolvasztási program lefutása után következett az áramkör vizsgálata forrasztási hibák, az eljárás folyamán általában keletkező forraszgöbök után kutatva (5.16. ábra).



5.16. ábra. *Áramkörgyártás: Reflow utáni ellenőrzés*

A reflow után keletkezett hibák kijavítása, valamint az óngöbök eltávolítása után a furatszerelt alkatrészek kézi beforrasztásával az áramkör elkészült.

5.6. Beépítés

Az utolsó lépés a kiválasztott műszerdoboz előkészítése az áramkör fogadására. A kör alakú MIDI csatlakozók számára lyukfúró fejjel ellátott állványos fúrót használtam, míg a téglalap alakú csatlakozók helyét előfúrás után négyzet keresztmetszettel rendelkező tűreszelő segítségével alakítottam ki (5.17. ábra).

5.17. ábra. *Áramkörgyártás: A műszerdoboz mechanikai kialakítása*

5.18. ábra. *Áramkörgyártás: Az elkészült modul*

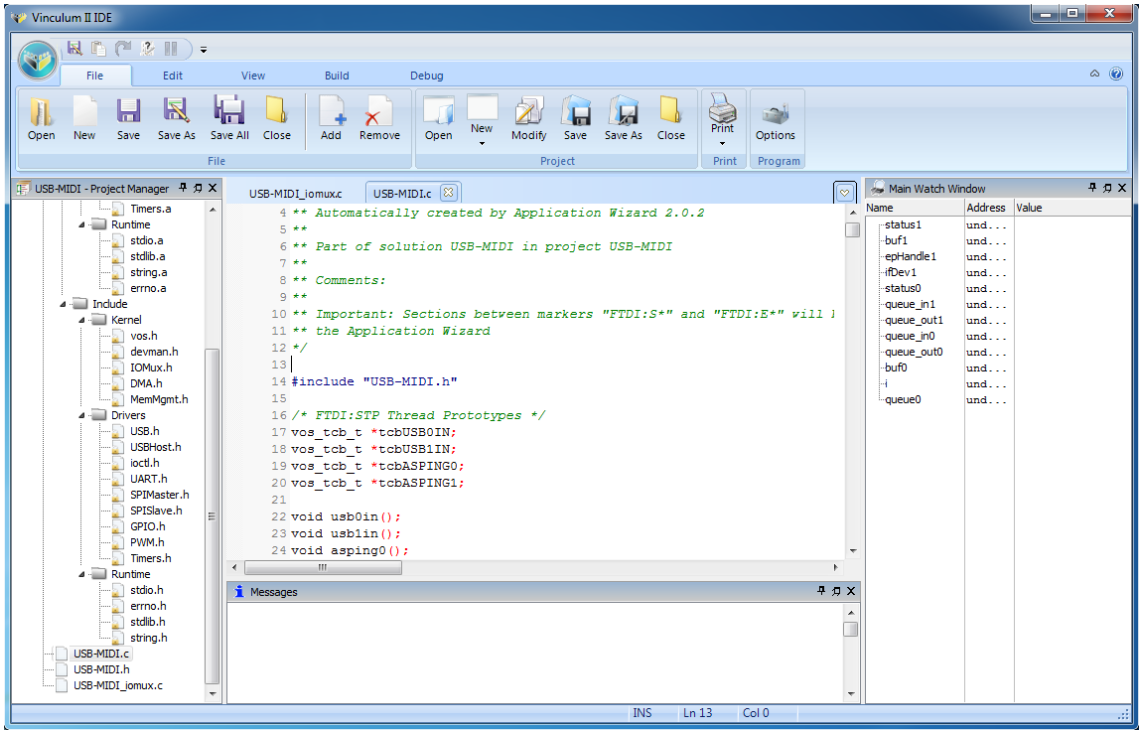
6. fejezet

Szoftvertervezés

A következőkben a mikrovezérlőn futó beágyazott szoftver megalkotásának menetét fogom ismertetni, a fejlesztőkörnyezet bemutatásától kezdve a szoftver folyamatábrájáig.

6.1. Fejlesztőkörnyezet

A vezérlő gyártója készített egy, külön a Vinculum II sorozatú termékekhez használható fejlesztőkörnyezetet, amely C++ kódnyelven íródott szoftverek kezelésére képes [2]. Mivel ez az egyetlen program, ami alkalmas a vezérlő programozására, ezért ezt kell használnom, aránylag kezdetleges kezelőfelülete ellenére.



6.1. ábra. *Szoftvertervezés: Vinculum II IDE kezelőfelülete*

A felület kezdetlegessége sajnos nem elsősorban a kinézetében nyilvánul meg, sokkal inkább a funkciók működésében, illetve azok hibáiban. Ezek kitapasztalása után már aránylag

jól használható a program, számos olyan funkcióval rendelkezik, ami jelentősen gyorsítja a fejlesztési munkát.

Ez a program is alprogramokból áll össze, melyek főként a vezérlő konfigurálására szolgálnak. Egy új projekt létrehozása a céleszköz kiválasztásával kezdődik, amely lehet valamelyik, az FTDI által gyártott fejlesztőkártya, vagy a mikrovezérlő egyedi környezetben. Utóbbi esetben a tokozás típusát is meg kell adni.

Ez után ki kell választani a használni kívánt perifériákat is. Nagy segítséget jelent a fejlesztésben, hogy a gyártó előre megírt drivereket épített be a fejlesztőkörnyezetbe, amelyek jól dokumentáltak, és a vezérlő által kínált minden funkciót előre megírt függvények segítségével lehet elérni. Az ebben a lépésben kiválasztott perifériákhoz tartozó drivereket tölti be a program a projektbe.

Rendelkezik a mikrovezérlő egy multiplexerrel, amely a fizikai kivezetések, valamint a vezérlőn belüli funkcionális egységek ki- és bemenetei közötti kapcsolatot hivatott kialakítani. Ennek a multiplexernek a segítségével lehet konfigurálni a vezérlő bekötését, tehát hogy melyik kivezetésre kerüljenek például az UART, vagy az SPI interfészekhez tartozó jelek. Nem minden láb konfigurálható, a tápfeszültségekhez, valamint az USB-hez tartozó lábak például rögzítettek, továbbá a multiplexer is négyes csoportokban képes csak kezelni a kivezetéseket, tehát bármelyik jel nem kerülhet bármelyik lábra. Erre tehát oda kellett figyelni a huzalozásnál, de a konfiguráció megválaszthatósága nagy könnyebbséget jelentett.

A következő lépés a processzormag beállítása, ami az órajelfrekvencia kiválasztását, valamint a számítási egység méretének megadását foglalja magában. Az órajelen kívül ezeket a beállításokat csak rendellenes működés esetén érdemes módosítani, akkor is csak alapos körültekintéssel.

Végül a programszálak (threadek) felvételével zárul a mikrovezérlő konfigurálása. A nagy fokú integráltság, valamint az időzítéskritikus funkciók miatt az FTDI kifejlesztett a Vinculum sorozatú processzorok számára egy valósidejű operációs rendszert, a VOS-t [2], amelyben található egy feladatidőzítő modul. Így lehetségessé válik a több szálon futó alkalmazások fejlesztése, és erre a funkcióra az USB jellege miatt általában szükség is van. Ezeket a szálakat lehet ebben a lépésben beállítani, beleértve az egyes szálak prioritását, a futtatandó függvény nevét, valamint a lefoglalandó memóriaterületet, amire a függvénynek szüksége van a futtatáshoz.

A fent leírt konfigurációs lépések bármelyike megismételhető a szoftverfejlesztés során, ez azonban hajlamos a már említett működési hibák miatt váratlan állapotokat előidézni. A multiplexer például a konfigurációs eszköz első megnyitásakor az alapbeállítást mutatja, tehát ha nem figyelünk rá, az egyedi beállításokat felülírva a gyári lábkiosztással ruházza fel a vezérlőt. Ezen kívül újrakonfigurálás után az esetleg módosított, de addig nem mentett fájlokon végzett munka is bármiféle figyelmeztetés nélkül elvész.

A szoftver fejlesztésében nagy segítséget nyújtanak a gyári példakódok, számos metódust vettem át ezekből munkám során. A példakódokon kívül az FTDI által közzétett nagy mennyiségű, részletes dokumentáció is említésre méltó.

6.2. Futási környezet

A hardvermodulok alacsony szintű kezelését tehát a VOS végzi, a szoftverben általában az erre szolgáló konfigurációs függvények lefutását kell csak biztosítani, valamint a VOS kéréseit kezelni, szintén a célfüggvényeken keresztül. Ez biztosítja például, hogy az USB kommunikáció során az időzítések nem tolódnak el adott esetben a processzor felhasználó általi leterhelése miatt. A VOS magas prioritással kezeli a modulok igényeit, a felhasználó által írt függvények futtatását csak a hardveres igények kiszolgálása után végzi.

A program fő függvényét is a fejlesztőkörnyezet generálja, ezt csak abban az esetben érdemes bővíteni, ha olyan globális erőforrást kívánunk használni a threadekben, amelyeket a konfiguráció során nem adtunk meg. Erre jó példaként szolgálnak a szemaforok, és az úgynevezett mutexek (6.1. kódrészlet).

```
1  vos_semaphore_t initDev0;
   vos_semaphore_t initDev1;
3
   // USB device handle to the next interface
5  usbhost_device_handle_ex ifDev0 = 0;
   vos_mutex_t ifDev0m;
7
   // USB device handle to the next interface
9  usbhost_device_handle_ex ifDev1 = 0;
   vos_mutex_t ifDev1m;
11
   #define QUEUE_SIZE 64                  // FIFO queue size
13 uint8 queue0[QUEUE_SIZE];               // FIFO queue to buffer MIDI messages
   uint8 queue_in0 = 0, queue_out0 = 0;   // indices of head and tail
15 vos_mutex_t mQueue0;                    // mutex to lock the queue access

17 uint8 queue1[QUEUE_SIZE];               // FIFO queue to buffer MIDI messages
   uint8 queue_in1 = 0, queue_out1 = 0;   // indices of head and tail
19 vos_mutex_t mQueue1;                    // mutex to lock the queue access

21 [...]

23 vos_init_semaphore(&initDev0, 0);
   vos_init_semaphore(&initDev1, 0);
25
   vos_init_mutex(&ifDev0m, 0);
27 vos_init_mutex(&ifDev1m, 0);

29 vos_init_mutex(&mQueue0, 0);
   vos_init_mutex(&mQueue1, 0);
```

6.1. kódrészlet. *Szemaforok és mutexek inicializálása*

Látható, hogy a VOS kezelésére szolgáló függvények mindig "vos_" előtaggal rendelkeznek, az előre definiált állandók pedig ehhez hasonlóan "VOS_" előtagot kapnak majd.

A szemaforoknak és mutexeknek egyébként fontos szerepük van a szoftver működésében, ezek segítségével lehet ugyanis a programszálakat szinkronizálni, valamint az erőforrásokat megosztani a szálak között. A szemaforok csak jelzésre szolgálnak, ha az egyik szál beállít egy olyan szemafort, amire egy másik szál várakozott, azzal lényegében jelzi az egyik szál a másiknak, hogy nem szükséges tovább várakoznia, folytathatja a tevékenységét.

Ezzel szemben a mutex segítségével egy lefoglalható erőforrás (változó, tömb, egyéb

struktúra) hozható létre, amelyen egyszerre csak egy programszál végezhet műveleteket. Ebben az esetben törekedni kell a kód megírásakor, hogy a lehető legrövidebb ideig foglalja az adott szál az erőforrást, mivel a hosszú ideig tartó foglalással akadályozhatja a többi szál futását.

A VOS számára automatikusan generált programrészekkel a fenti inicializáláson túl további teendőm nem volt, következett a threadekhez tartozó függvények megírása. Összesen négy szálon fut a program, ezek azonban lényegében páronként megegyeznek, két függvény két-két példánya dolgozik különböző paraméterekkel, így alakul ki a négy programszál.

A program működése a megosztott erőforrásokon alapul, amelyek a fent inicializált, 64 bájt méretű FIFO tömbök. Egy tömbhöz két függvény tartozik, címszavakban összefoglalva az egyik feltölti a tömböt az egyik USB-n keresztül beérkezett MIDI-üzenetekkel, a másik pedig kiolvassa a tömb tartalmát, és kiküldi azt a másik USB-n. Így valósul meg az USB hostok közötti adatátvitel az egyik irányba. Ahhoz, hogy ez a folyamat mindkét irányba működhessen, a tömb és a hozzá tartozó függvények duplikálására van szükség, a ki- és bemenetként szolgáló USB hostok megcserélése mellett.

6.2.1. usb0to1 (/usb1to0) thread

Ez a függvény felel a bejövő üzeneteket tároló tömb figyeléséért, új adat megjelenése esetén annak elküldéséért a megfelelő USB eszköz számára. Ezen kívül szintén itt történik az enumeráció is, valamint a kapcsolatfelvétel, amennyiben az eszköz a MIDI-eszközosztályba, vagy a gyártóspecifikus eszközök osztályába tartozik (6.2. kódrészlet).

```
  status0 = 0;
2 connectstate0 = PORT_STATE_DISCONNECTED;

4 // wait for device to connect
  do
6 {
    vos_gpio_write_pin(GPIO_A_2, 1);        // USB0 LED blinking
8   vos_delay_msecs(49);
    vos_gpio_write_pin(GPIO_A_2, 0);
10  vos_delay_msecs(1);
    hc_iocb0.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
12  hc_iocb0.get = &connectstate0;
    vos_dev_ioctl(hUSBHOST_1, &hc_iocb0);
14 } while (connectstate0 != PORT_STATE_ENUMERATED);

16 // set up USB host structures as per MIDI class codes
  hc_iocb_class0.dev_class = USB_CLASS_AUDIO;
18 hc_iocb_class0.dev_subclass = USB_SUBCLASS_AUDIO_MIDISTREAMING;
  hc_iocb_class0.dev_protocol = USB_PROTOCOL_ANY;
20 hc_iocb0.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
  hc_iocb0.handle.dif = NULL;
22 hc_iocb0.set = &hc_iocb_class0;
  hc_iocb0.get = &ifDev0;
24 vos_dev_ioctl(hUSBHOST_1, &hc_iocb0);     // get MIDI controller
```

6.2. kódrészlet. *USB enumeráció és kapcsolatfelvétel*

Amennyiben a kapcsolatfelvétel sikeresnek bizonyul, a program jelzi egy szemafor segítségével az eszköztől érkező adatok kiolvasására hivatott függvénynek, hogy a kapcsolat létrejött. Ezután továbblép, és megkeresi az USB eszköz Bulk OUT végpontját, amelyen

keresztül a korábban ismertetett, Bulk átviteli algoritmus szerinti kommunikációval lehetséges az adatok kiküldése az eszköz számára. Ez után következhet a FIFO állapotának ellenőrzése, majd pedig az eredmény függvényében az esetleg beérkezett MIDI üzenetek küldése (6.3. kódrészlet).

```
memset(buf0, 0, 64);
xfer0.buf = buf0;

[...]

do                                      // try to access FIFO
{
  mutex_state = vos_trylock_mutex(&mQueue1);
} while (mutex_state != VOS_MUTEX_UNLOCKED);

if(queue_in1 != 0)                      // check if something in the queue
{
  queue_out1 = 0;
  vos_gpio_write_pin(GPIO_A_2, 1);      // USB0 LED transfer sign
  for(i=0; i<64; i+=1)
  {
    if(queue_out1 == queue_in1) i=64;   // end of new data
    else
    {
      buf0[i] = queue1[queue_out1];
      queue_out1 = (queue_out1+1)%QUEUE_SIZE;
    }
  }
  queue_in1 = 0;                        // reset FIFO
  queue_out1 = 0;
  vos_unlock_mutex(&mQueue1);           // release FIFO
  status0 = vos_dev_write(hUSBHOST_1, (unsigned char *)&xfer0, sizeof(
    usbhost_xfer_t), NULL);
```

6.3. kódrészlet. *FIFO-ba beérkezett adat kiküldése*

Mivel ezek a lépések egymásra épülnek, ezért ha a fentiek közül egy művelet meghiúsul, a program egy lépéssel hátrébb kerül, és addig próbálkozik a szükséges művelet végrehajtásával, amíg sikerrel nem jár. Ha a sorban eggyel korábbi művelet is meghiúsul, szintén visszalépés történik, adott esetben egészen az USB eszköz enumerálásának fázisáig, amelybe a program rögtön az indulása után kerül.

6.2.2. as_ping0 (/as_ping1) thread

Az USB bemenetet figyelő szál, amely folyamatosan próbál olvasási műveletet végrehajtani a csatlakoztatott eszköz Bulk IN végpontján. Mivel az erre szolgáló függvény blokkoló eljárást hajt végre, ezért amennyiben nem áll rendelkezésre kiolvasható adat, a függvény, és egyben a programszál futása megakad az adat legközelebbi rendelkezésre állásáig. Első sorban ezért van szükség a thread jellegű programszerkezetre, hogy a többi, ettől független függvény futása biztosított legyen akkor is, ha épp nem érkezik be adat.

Mivel az USB eszköz enumerációja a kimenet kezelését végző szálon történik, ezért ennek a függvénynek az első lépése a várakozás az eszközzel való sikeres kapcsolatfelvételt jelző szemaforra, amit a másik szálon futó függvény állít be. Miután ez megtörtént, a következő lépés a hozzáférés megszerzése az USB eszköz Bulk IN végpontjához. Egy eszközt egyszerre csak egy függvény kezelhet, ezért a hozzáférést mutex védi (6.4. kódrészlet).

```
  vos_wait_semaphore(&initDev0);            // wait for device initialisation

  do                                        // try to access device
  {
    mutex_state = vos_trylock_mutex(&ifDev0m);
  } while (mutex_state != VOS_MUTEX_UNLOCKED);

  hc_iocb0.handle.dif = ifDev0;

  epHandle0 = NULL;
  hc_iocb0.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;
  hc_iocb0.get = &epHandle0;
  vos_dev_ioctl(hUSBHOST_1, &hc_iocb0);     // get IN endpoint
  vos_unlock_mutex(&ifDev0m);               // release device
```

6.4. kódrészlet. *Szemafor és mutex kezelése az USB végpont kiválasztásánál*

Amennyiben sikerül megszerezni a hozzáférést a kívánt végponthoz, megkísérelhető az üzenetek kiolvasása. Mint már említettem, ez egy blokkoló hatású függvény, csak abban az esetben fut le, ha az olvasás sikeres. Ezért nem szükséges ezt külön is ellenőrizni, a függvény utáni kódsorok végrehajtására garantáltan csak abban az esetben kerül sor, ha adat érkezett. Ezek a kódsorok pedig a kiolvasott adatokat tároló pufferben az érvényes MIDI üzenetek keresését, találat esetén pedig azok FIFO-ba való beírását végzik.

```
memset(buf0, 0, 64);
xfer0.buf = buf0;

status0 = vos_dev_read(hUSBHOST_1, (unsigned char *)&xfer0, sizeof(usbhost_xfer_t),
     NULL);
vos_gpio_write_pin(GPIO_A_2, 1);        // USB0 LED transfer sign

[...]

for(i=0; i<64; i+=4)                    // scan for MIDI messages
{
  cin0=buf0[i]&0x0f;                    // get code index number
  if(buf0[i]==0) i=64;                  // empty data block

  if((cin0==0x03)||(cin0==0x04)||(cin0==0x07)||(cin0==0x08)||(cin0==0x09)||(cin0==0
    x0a)||(cin0==0x0b)||(cin0==0x0e))    // 3-byte-message code index numbers
  {
    vos_lock_mutex(&mQueue0);           // access FIFO
    queue0[queue_in0] = buf0[i];
    queue_in0 = (queue_in0+1)%QUEUE_SIZE;
    queue0[queue_in0] = buf0[i+1];
    queue_in0 = (queue_in0+1)%QUEUE_SIZE;
    queue0[queue_in0] = buf0[i+2];
    queue_in0 = (queue_in0+1)%QUEUE_SIZE;
    queue0[queue_in0] = buf0[i+3];
    queue_in0 = (queue_in0+1)%QUEUE_SIZE;
    vos_unlock_mutex(&mQueue0);         // release FIFO
  }
}
```

6.5. kódrészlet. *Kiolvasás után a MIDI üzenetek eltárolása*

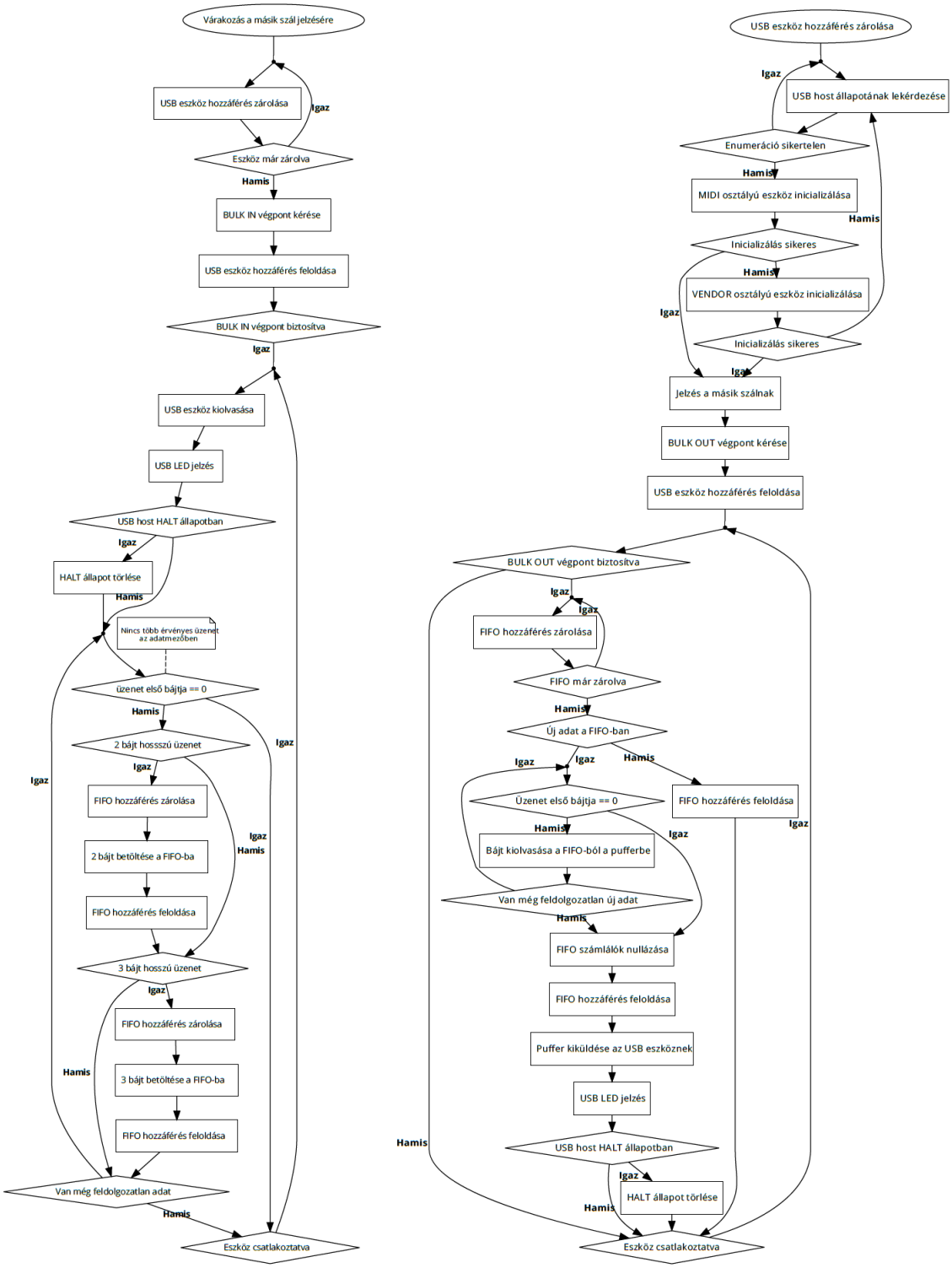
Esetünkben, mivel elsődleges célom a billentyűleütések továbbítása, érvényes MIDI üzeneteknek a három bájt hosszúságú üzeneteket tekintettem. Mint azt a dolgozat elején ismertettem, az üzenetek hosszára az első, üzenettípus jelzésére szolgáló bájt alapján lehet következtetni. Mivel ezek az USB-MIDI specifikációjából ismertek voltak, ezért lehetőségem nyílt a kívánt, három bájtos üzenetekre szorítkozni a kommunikációnál. A MIDI, valamint

USB-MIDI specifikációk egyaránt lehetővé teszik az üzenetek maszkolását az eszközök működésének gyorsítása érdekében, ezért ez az eljárás szabályos a részemről.

A függvény egyszerűségénél fogva jelen esetben nem volt szükség fázisokat létrehozni, ciklikusan ismétlődik mindaddig, amíg él a kapcsolat egy kompatibilis USB eszközzel.

6.3. Folyamatábrák

A következő oldalon találhatóak a két thread logikai működését szemléltető folyamatábrák (6.2. ábra). Jól látható a különböző műveletek sikertelensége esetén a visszatérés a program korábbi fázisaiba, valamint a programszálak közötti erőforrás-megosztás működése, a szemafor- és mutex kezelés fentebb említett menete.



6.2. ábra. *A két programszál folyamatábrája: as_ping (bal), és usbAtoB (jobb)*

7. fejezet

Tesztelés

Az előző fejezetekben leírt hardverműködés, valamint szoftverfelépítés értelemszerűen hosszú folyamat eredménye, melynek az egyik legfontosabb eleme a tesztelés, valamint az ennek folyományaként kiderülő hibák javítása. Az alábbiakban sorra veszem, hogy milyen problémákkal szembesültem a hardver élesztését, valamint az első szoftververzió lefuttatását követően.

7.1. Kiküszöbölt hardverhibák

Tervezési hiba a hardver eddig használt funkcióit tekintve nem fordult elő, ebben nagy segítséget jelentett a munkám révén szerzett tapasztalatom a hardvertervezésben. Természetesen, mivel a szoftverkomponensek a dolgozat elején leírtaknak megfelelően nem használják az összes hardverfunkciót, ezért nem jelenthető ki biztosan, hogy a hardver teljes mértékig hibamentes.

Amik azonban előfordultak, azok a gyártási, valamint alkatrész-beszerzési fázisból eredő hibák voltak.

7.1.1. Érintkezési hibák

Az SMD reflow folyamat utáni ellenőrzésnél az összeforradt alkatrészlábak, valamint óngöbök az áramkör tüzetesebb átnézése által felfedezhetők, ennél azonban könnyebben megbúvó hibaforrás a nem elégséges folyasztószer, és/vagy ónmennyiség az egyes forrasztásoknál.

Ebből eredt egy hiba a mikrovezérlő GPIO kivezetéseinél, amely egészen a szoftverfejlesztési szakaszig felfedezetlen maradt, ezzel okozva némi fejtörést. A forrasztási hiba ugyanis épp azt a kivezetést érintette, amely az egyik USB host állapotát volt hivatott LED segítségével jelezni a külvilág felé. Miután látszólag egyik logikai szint sem volt jelen a LED lábán, valamint a beépített debug interfészen keresztül meggyőződtem arról, hogy az adott láb kimenetként való konfigurálásáért felelős programsor lefut, ekkor merült fel bennem a gyanú a forrasztási hibára, amely végül jogosnak bizonyult. Az adott alkatrészláb újraforrasztása után a hiba megszűnt.

7.1.2. Programozó interfész

Mint azt láthattuk a hardvertervezést leíró fejezetben, a programozó- és debug interfész fontos eleme egy kétirányú meghajtó áramkör, ami az USB-UART átalakító, valamint a mikrovezérlő közötti kétirányú adatfolyamot biztosítja a vezérlő egyetlen kivezetésének felhasználásával. A gyártó által megadott kapcsolási rajzban szereplő alkatrész szállítása késett a többiéhez képest, ezért megkíséreltem egy hasonló meghajtó áramkörrel helyettesíteni azt. Az alkatrész, amit erre használtam, felépítését tekintve alkalmas lett volna a célra, annyiban különbözött a gyártó által ajánlottól, hogy a két irányhoz használható meghajtó áramkörök kimenetei nem voltak külön kivezetve, hanem már eleve összekötötték őket egymással, továbbá ez a meghajtó szimmetrikus jelet állított elő a kimenetén.

A helyettesítés működött, de nem elég megbízhatóan. A fejlesztőkörnyezet néhány próbálkozásonként ismerte csak fel a programozót, ezért végül nem kockáztattam a mikrovezérlő programozását ebben a konfigurációban. Inkább megvártam az ajánlott alkatrész beérkezését, amivel már stabilan működött a programozás.

A sikertelenség lehetséges oka a helyettesítő meghajtó sebessége, ami egy nagyságrenddel alacsonyabb, mint az ajánlott alkatrészé. A programozó interfész a specifikáció szerint 1 Mbps adatátviteli sebességet igényel, ez feszegeti a lassabb meghajtó áramkör határait.

7.2. Kiküszöbölt szoftverhibák

Mivel ezen a platformon most fejlesztettem először beágyazott szoftvert, ezért ezekből a hibákból jóval több adódott, mint hardveres eredetűekből, mire kitapasztaltam a VOS, valamint a mikrovezérlő általános működését.

7.2.1. Thread konfiguráció

A feladatütemező használata beágyazott környezetben nem volt teljesen új a számomra, itt azonban a threadek között megosztott erőforrások bonyolították a programszálak működését. Számos esetben kellett változtatni a szálak prioritását a kívánt működés elérése érdekében. Erre jó példa a végleges, fentebb részletezett programszálak esete, ahol eleinte az adatok beolvasásáért felelős függvény kapott magasabb prioritást, rövid lefutása miatt azonban ekkor annyival gyakrabban kapta meg a hozzáférést az USB eszközhöz, hogy a kiküldést lebonyolító szál nagyon változó késleltetésekkel futott csak le, ha egyáltalán lefutott. Ez értelemszerűen a hangszereken a hangok megszólalását erősen befolyásolta. A megoldás a prioritások megváltoztatása volt, az addig instabil időzítésű küldések stabilizálódtak.

7.2.2. Enumeráció

MIDI-osztályba tartozó eszközöknél az enumeráció nem okoz különösebb problémát, mivel az osztály-, valamint alosztály azonosító kódok mind részei az osztályspecifikációnak, az enumeráció pedig ezek alapján megy végbe. Ennek azonban csak az egyik hangszer

egyik üzemmódja felelt meg, a másik üzemmód, valamint a másik hangszer is gyártóspecifikus kezelést igényelt. Ennek módjáról még az önálló laboratóriumi munkám során a hangszer és számítógép közötti kommunikáció vizsgálatának eredményeiből tájékozódtam. A gyártóspecifikus eszközök osztályának azonosítója ismert volt számomra, mivel szintén részét képezi az USB specifikációjának, az alosztály kódját viszont ebben az esetben már a gyártó adja meg. Ehhez volt szükségem arra a kommunikációs naplóra, amit korábban előállítottam egy úgynevezett USB sniffer program segítségével. Miután beállítottam a helyes alosztály azonosítót, a kommunikáció lehetségessé vált a gyártóspecifikus hangszerek esetén is.

7.2.3. Buffer kezelés

Amikor először sikerült létrehozni a kapcsolatot az egyik hangszerrel, a program működése látszólag az első üzenet beérkezéséig tartott, utána a modul nem volt hajlandó se fogadni, se küldeni az üzeneteket. A debuggernek hála megtaláltam a jelenség okát, ami az USB host érzékenysége volt az üzenetek hosszára. Ismert volt számomra, hogy az USB-MIDI eszközök esetében a maximális csomagméret általában 64 bájt, ami több MIDI-üzenetet is tartalmazhat, amennyiben azok egy időben érkeztek. Ennek megfelelően az USB host puffertárát 64 bájt méretűre állítottam, amivel viszont eleinte nem voltam tisztában, az az, hogy ilyenkor az USB host mindenképpen 64 bájt hosszúságú csomagokat vár. Ha ennél rövidebb csomag érkezik, ezt a host "buffer underrun"-hibaként értékeli, és HALT üzemmódra vált, ami lényegében annyit jelent, hogy felfüggeszti minden tevékenységét. Ebből az üzemmódból ráadásul csak manuálisan lehet kivenni.

```
1  status0 = vos_dev_read(hUSBHOST_1, (unsigned char *)&xfer0, sizeof(usbhost_xfer_t
     ), NULL);
   vos_gpio_write_pin(GPIO_A_2, 1);          // USB0 LED transfer sign
3  if (status0 == USBHOST_EP_HALTED)
   {
5    hc_iocb0.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_CLEAR_HOST_HALT;
     hc_iocb0.handle.ep = epHandle0;
7    vos_dev_ioctl(hUSBHOST_1, &hc_iocb0);   // clear halt state on endpoint
   }
```

7.1. kódrészlet. *USB HALT állapot megszüntetése*

Az általam korábban vizsgált hangszernél ez nem okozott gondot, mivel az fix, 64 bájtos csomagméretet használ. Ha az elküldendő üzenet rövidebb, a maradék bátjokat 0x00 értékkel tölti fel. Ez azonban nem volt igaz a másik hangszerre, ami csak olyan méretű csomagot küldött, amire épp szükség volt. Két eljárást dolgoztam ki a jelenség elkerülésére, melyeket párhuzamosan alkalmaztam. Az egyik, hogy az USB puffertárát előre feltöltöttem 0x00 értékekkel, ekkor a fogadott bájtok a puffer elején megjelentek, de a puffer további tartalma is érvényes volt a függvény számára, így nem állt be a HALT állapot. Ennek biztosítására a másik "megoldás" az volt, hogy a kiolvasó függvény visszatérési értékét rögtön az olvasás után megvizsgáltam, és ha az értéke megfelelt a HALT állapot hibakódjának, automatikusan kivettem a hostot ebből az állapotból. Ez az eljárás problematikus lehet bi-

zonyos szempontból, de a hagyományos MIDI koncepciójának lényegében megfelel, amely tisztázza, hogy érvénytelen adatok esetén sem szakadhat meg a kommunikáció folyamata. Az érvényes üzenetek fogadását mindenképpen biztosítani kell, az érvényteleneket pedig figyelmen kívül kell hagyni.

7.2.4. Active Sensing

Az egyik hangszerrel már hibátlanul kommunikált mindkét irányba a modul, a másik hangszert azonban ugyanazzal a metódussal nem tudtam kezelni. Jobban megvizsgálva a működést, felfedeztem, hogy ez a hangszer akkor is küld egy bizonyos üzenetet, amikor nem történik billentyűleütés. Ez az üzenet ismerős volt számomra, úgyhogy ismét elővettem az önálló laboratóriumi munkámból származó naplófájlt. Ott is megállapítottuk, hogy ez az üzenet bizonyos, nagyjából állandó időközönként megjelent a kommunikációban, de hogy pontosan mire szolgált, azt akkor nem tudtuk végül megállapítani.

Mivel ezúttal jobban belemélyedtem a MIDI specifikációjába, olvastam benne az active sensing technológiáról. Ennek lényege, hogy ha valamelyik eszköz a kommunikáció során fogad egy bizonyos, erre szolgáló üzenetet, akkor ha ez nem ismétlődik meg 300 ms időintervallumon belül, úgy értékeli, hogy a kapcsolat bontásra került, így a továbbiakban nem küld, vagy fogad üzeneteket, amíg ezt a speciális üzenetet ismét meg nem kapja. Az üzenet tartalmát ellenőrizve nyilvánvalóvá vált, hogy ez okozza a problémát.

Hogy ez az USB-MIDI esetében hogy működik pontosan, az elsőre nem volt számomra nyilvánvaló. Első próbálkozásként beállítottam a modult, hogy küldje el az active sensing üzenetet a hangszereknek 200 ms idő elteltével. Mindkét hangszer gond nélkül fogadta ezeket, azonban a működésüket nem befolyásolta, az egyik működött, a másik nem. Következő próbálkozásként úgy módosítottam a programot, hogy a kiolvasott active sensing üzenetet küldje vissza az adott hangszernek. Ezzel a módszerrel már sikerült működésre bírnom a másik hangszert is, ugyanis - mint kiderült - USB-MIDI esetén az üzenetet küldő eszköz is megszakítja a kommunikációt, ha az üzenetét a fogadó eszköz nem igazolja vissza handshake formájában, ami szabályos kiolvasás esetén végbemegy. Az active sensing üzenet visszaküldésére tehát nincs szükség, csupán a kiolvasására. Emiatt alakítottam át a programszálak struktúráját a végleges, fentebb ismertetett formájúra.

A technológia kellemetlen hozadéka, hogy a debug interfész használatakor, amennyiben a program futása egy breakpointnál megáll, értelemszerűen az active sensing üzenet sem kerül időben kiolvasásra. Ez is okozott néhányszor problémát a szoftverfejlesztés során.

8. fejezet

Összefoglaló, kitekintés

A projekt elérte elsődleges célját, az adatátvitel megvalósult két, USB-MIDI interfésszel rendelkező hangszer között. A lehetőségek feltérképezésére, a hardver tervezésére és megvalósítására, valamint a működőképes szoftver kifejlődéséhez vezető rögös út bejárására a fél éves időtartam épphogy elegendőnek bizonyult. Önálló laboratóriumi munkám, valamint gyakorlati tapasztalataim ellenére sem voltam egészen meggyőződve a félév elején arról, hogy a félév végén mindezt elmondhatom majd, mivel ennek a feladatnak a megvalósítására nem állt előttem semmilyen példa. Találtam ugyan projekteket, amelyek a feladat egyes részeit megvalósították, de olyan rendszert, amiben két USB-MIDI eszköz, PC segítsége nélkül kommunikált egymással, nem láttam még az általam fejlesztetten kívül, éppen ez volt a fejlesztésre az elsődleges motivációm.

A jövőben elsősorban a bővített funkcionalitás megvalósítására fogok koncentrálni, a hardver után a szoftverkomponensek tekintetében is. A mikrovezérlő adottságai ehhez még kérdéses, hogy elegendőek lesznek-e, tekintettel arra, hogy pusztán az USB-MIDI működésében is adódtak késleltetésbeli problémák, feltehetően a kód optimalizálásának hiánya miatt. Ha már minden komponens és interfész megfelelően működik, következhet a szoftver működésének gyorsítása, stabilizálása, további modulok építése, a több modulból álló rendszer tesztelése.

Amit tehát a dolgozatban bemutattam, egy nagyobb, valószínűleg években mérhető időtartamú fejlesztési munka kezdete, amely fél év alatt elérkezett az első néhány fontos állomásra, ezzel bizonyítva a koncepció működőképességét.

Köszönetnyilvánítás

Fél éves munkámban többen is segítségemre voltak, akiknek a közreműködése nélkül nem biztos, hogy időben elértem volna az általam kitűzött célokat.

Ezért szeretném megköszönni természetesen a konzulensem, Orosz György munkáját, aki azon túl, hogy vállalta a saját témám konzultálását, lelkiismeretesen végezte ezt a feladatot, és az egyéb, adminisztrációval kapcsolatos egyetemi és tanszéki ügyeimben is segítségemre volt.

Hálával tartozom még a Mentor Graphics magyarországi leányvállalata, a MicReD, azon belül is közvetlen felettesem, Katona Balázs segítségének, amit az áramkör és az alkatrészek beszerzésében nyújtott, továbbá engedélyezte, hogy munkaidőm lejárta után (!) használhassam a tervezőszoftvert, valamint a laboratóriumi eszközöket a szakdolgozatom elkészítéséhez.

Irodalomjegyzék

[1] FTDI. *Vinculum-II Embedded Dual USB Host Controller IC Datasheet.* Future Technology Devices International, 2011.
URL: http://www.farnell.com/datasheets/1913743.pdf.

[2] FTDI. *Vinculum-II Embedded Dual USB Host Controller IC User Guide.* Future Technology Devices International, 2011.
URL: http://www.ftdichip.com/Support/Documents/AppNotes/AN_151_Vinculum-II_User_Guide.pdf.

[3] USB IF. *Universal Serial Bus Device Class Definition for MIDI Devices.* USB Implementers Forum, 1996.
URL: http://www.usb.org/developers/docs/devclass_docs/midi10.pdf.

[4] USB IF. *Universal Serial Bus Revision 2.0 Specification.* USB Implementers Forum, 2000.
URL: http://www.usb.org/developers/docs/usb_31_052016.zip.

[5] USB IF. *Universal Serial Bus Revision 3.1 Specification.* USB Implementers Forum, 2014.
URL: http://www.usb.org/developers/docs/usb_31_052016.zip.

[6] MIDI MA. *MIDI 1.0 Detailed Specification.* The MIDI Manufacturers Association, 1995.
URL: http://oktopus.hu/uploaded/Tudastar/MIDI%201.0%20Detailed%20Specification.pdf.

[7] Microchip. *MCP2515 Stand-Alone CAN Controller with SPI Interface Datasheet.* Microchip Technology Inc., 2012.
URL: http://www.farnell.com/datasheets/1669372.pdf.

Függelék

F.1. CD-melléklet

A mellékelt CD tartalma:

- Jelen dolgozat digitális változata (Szakdolgozat_Pani_Tamas_ZFEED6.pdf)
- A hardver teljes kapcsolási rajza (SCH_USB-MIDI.pdf)
- A hardver gyártásához szükséges szabványos gerber fájlok (USB-MIDI_gerbers.zip)
- Az általam írt függvényeket tartalmazó forrásfájl (USB-MIDI.c)
- A Vinculum II IDE projekt a teljes szoftverrel (USB-MIDI_FW.zip)
- A működést demonstráló videó (USB-MIDI_demo.avi)
- A működést demonstráló videó online megtekinthető változatának URL-je (USB-MIDI_onlinedemo.txt)

F.2. Online tárhely

A fent felsorolt mellékletek megtalálhatók egy online tárhelyen is, melynek címe:

`https://drive.google.com/folderview?id=0B45IbEA8OcHNUklaNWhzenJ5anM`