Szegedi Tudományegyetem
Informatikai Tanszékcsoport

Interaktív sakktábla fejlesztése
Development of an Interactive Chessboard

Szakdolgozat

Készítette:
Lautner Tamás Zoltán
mérnök informatika szakos hallgató

Témavezető:
Dr. Mingesz Róbert
egyetemi adjunktus

Szeged
2014

Feladatkiírás

A sakk egy olyan kétszemélyes játék, melynek története több mint egy évezredre nyúlik vissza. A szakdolgozat célja egy mikrovezérlőn alapuló, "interaktív sakktábla" megtervezése, majd hardveres és szoftveres megvalósítása. A sakktáblának képesnek kell lennie a bábuk érzékelésére, valamint a sakkjáték szabályainak betartatására interaktív visszajelzés segítségével. A sakktáblát úgy kell kiképezni, hogy a jövőben személyi számítógéphez vagy egylapos PC-hez is lehessen csatlakoztatni.

Tartalmi összefoglaló

- ***A téma megnevezése:***

  Interaktív sakktábla fejlesztése.

- ***A megadott feladat megfogalmazása:***

  A feladat egy "interaktív sakktábla" megtervezése, hardveres és szoftveres kivitelezése. Az cél egy olyan eszköz megalkotása, mely megismerteti a játékosokat a sakk alapvető szabályaival, segít elkerülni a legjellemzőbb hibákat, élvezetessé teszi a szabályok elsajátításának folyamatát.

- ***A megoldási mód:***

  Specifikáció, funkcionális terv készítése. Nyomtatott áramköri elemek alkatrészeinek kiválasztása, kapcsolási rajz, NYÁK terv elkészítése, a tervek alapján a hardveres elemek elkészítése. Hardvertesztelés, hibaelhárítás.

  Vezérlő szoftver specifikációjának, funkcionális tervének meghatározása. A szoftver kivitelezése inkrementális fejlesztési mód segítségével. A szoftver tesztelése, hibakeresés, hibaelhárítás.

- ***Alkalmazott eszközök, módszerek:***

  A tervezési feladatok során az EAGLE 6.5.0, ill. 6.6.0 kapcsolási rajz és NYÁK terv szerkesztő programot használtam. A NYÁKok gyártása a Mikropan Kft-nél történt. A hardver építés során számos különböző eszközt használtam, forrasztóállomásoktól multiméterekig. A Szoftveres fejlesztés során a Silabs ConfigurationWizard 2, és a SiLabs IDE programokat használtam. A sakktábla dobozát AutoCAD 2014 program segítségével, míg a dobozra kerülő matricázás grafikáját CorelDraw x7 segítségével készítettem el. A szakdolgozatban található grafikák a Windows 8.1 SnippingTool eszköze, a sakk.chess.com online sakkjátéka, az Adobe PhotoshopCS5, és a www.lucidchart.com diagramszerkesztő segítségével készültek.

- ***Elért eredmények:***

  A feladatkiírásnak teljesen megfelelő eszközt sikerült létrehozni, célul lett kitűzve az eszköz későbbi továbbfejlesztése is.

- ***Kulcsszavak:***

  Interaktív sakktábla, hardverfejlesztés, szoftverfejlesztés, prototípusfejlesztés.

Tartalomjegyzék

1. BEVEZETÉS

A sakk egy kétszemélyes stratégiai táblajáték, sőt, sportág, mely fejleszti a logikus gondolkodást, a helyzetfelismerő képességet, és javítja a memóriát[1].

Én személy szerint sosem űztem versenyszerűen ezt a játékot, de szórakozásból barátaimmal/családommal gyakran játszom. Sajnos azonban sokszor abba a falba ütközöm, hogy egyesek azért zárkóznak el még a játék kipróbálásától is, mert nem ismerik a sakk szabályait és összefüggéseit, és túlságosan hosszúnak, unalmasnak érzik a betanulás folyamatát, vagy félnek, hogy emiatt leküzdhetetlen hátrányba kerülnének, és nem élveznék a játékot. Innen eredt az az ötlet, hogy egy interaktív, "tanító sakktábla" segíthetne ezen problémák áthidalásában.

A piacon számos 'elektromos sakkgép' megtalálható. Ezek a sakktáblák LCD kijelzővel, illetve a mezőkön nyomásérzékelők segítségével tartják a kapcsolatot a külvilággal, esetleg számítógéphez csatlakoztathatók, ahol egy sakkprogrammal együtt használhatók. Sokszor mesterséges intelligenciával is rendelkeznek, különböző kezdéseket, híres mérkőzések lépéssorozatait is tartalmazzák. Általában drágák, nem túl látványosak, így leginkább a téma iránt komolyabban érdeklődőket célozzák meg. Az ilyen táblák gyártóit az utóbbi időkben felvásárolták/bezárták, így jó minőségű eszközöket jelenleg csak másodkézből, vagy nagyon drágán lehet csak beszerezni[2].

A látványos, LEDekkel ellátott táblák általában csak a design és a látvány céljait szolgálják, ritka hogy ilyen táblákban intelligencia legyen.

Az interaktív sakktáblával célom egy, e két 'véglet' közötti eszköz megalkotása, mely látványos, felhasználóbarát, és főként a szórakozás céljait szolgálja. Megismerteti a játékosokat a sakk alapvető szabályaival, segít elkerülni a legjellemzőbb hibákat, élvezetessé teszi a szabályok megismerésének folyamatát. Mivel a tanító/segítő lehetőségek játékosonként személyre szabhatók, egy kezdő játékos is 'felveheti a kesztyűt' egy haladó ellen, sőt, a beállítások akár játék közben is módosíthatók. A játék szüneteltethető/folytatható. Ha a bábuk leborulnak, vagy helytelen lépést tett valaki, a tábla segít a helyes játékállás visszaállításában. Profik számára egy teljesen személyre szabható, önműködő, Fischer típusú sakkóra is rendelkezésre áll, így akár időre menő versenyhelyzet is létrehozható.

Már a megvalósítás végén jártam, mikor rábukkantam a Purdue ECE diákjainak projektjére[3], mely egy design szempontjából az enyémhez nagyon hasonló, bár funkcionalitásában kissé eltérő sakktáblát valósított meg. A videó alatti kommentekből ítélve egy ilyen jellegű eszközre van igény, sőt, akár kereslet is, így lehetséges, hogy az MSc szakdolgozatom keretében is ennek az eszköznek a továbbfejlesztését fogom célul kitűzni.

A sakktáblához tartozik egy, a funkcionalitását részletesen bemutató "használati utasítás", mely a függelék F1 bekezdésében található.

2. FELHASZNÁLT ESZKÖZÖK, MÓDSZEREK, FORRÁSOK

2.1 Funkcionalitás, specifikáció

A tábla funkcionalitásának, specifikációjának megalkotása során elsősorban a sakk hivatalos szabályainak betartása volt az irányadó, melyek forrásául a Wikipédia megfelelő, forrásmegjelöléssel ellátott magyar és angol szócikkeit [1], illetve a Chess.hu hivatalos szabályzatát [4] használtam. A szabályok kivonatolt formában megtalálhatók a használati utasításban, a függelék F1 fejezetében, dőlt betűvel szedett szemelvények formájában. A tábla működésének specifikációja az ebben a fejezetben leírtakkal vág egybe.

2.2 Hardvertervezés, megvalósítás

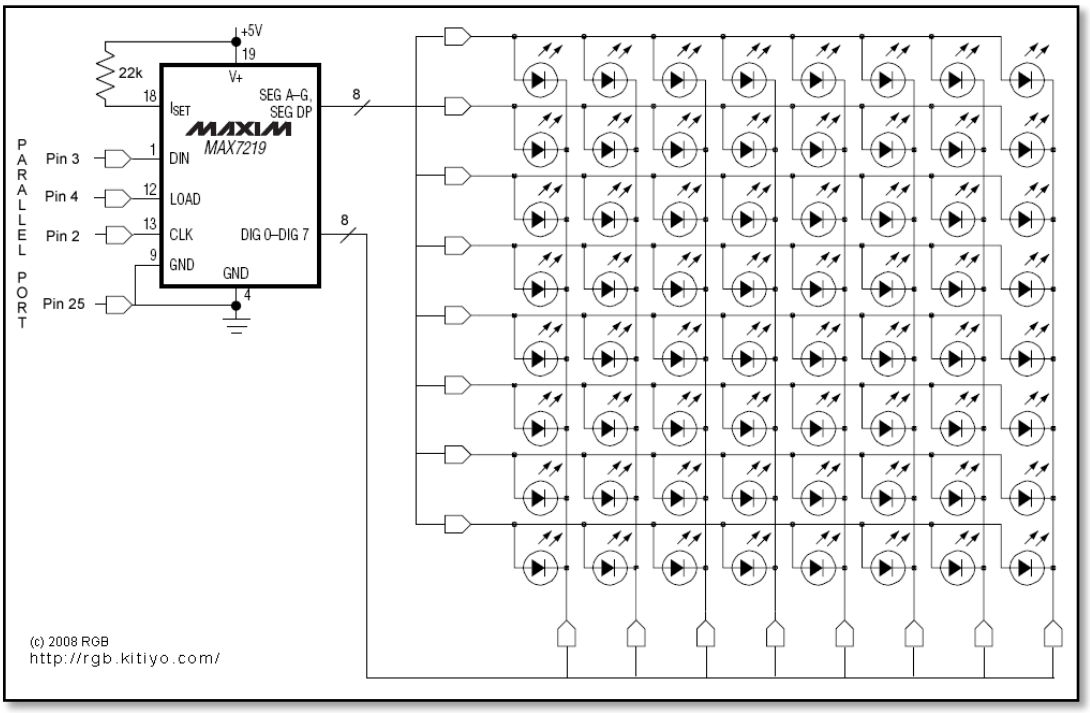
A hardveres fejlesztés tervezési fázisa során Windows 8.1 operációs rendszer alatt az Eagle 6.5.0, illetve 6.6.0 szoftvereket használtam. A tervek elkészítése során a kiválasztott alkatrészek adatlapjainak [5,6,7,8,9,10]ajánlott kapcsolásai,saját korábbi tapasztalataim, illetve a tanszéken dolgozók által tett javaslatok voltak az irányadóak, illetve internetes fórumokat/weboldalakat is használtam. Ezek közül kiemelendő a MAX7219 IC használatához referenciaként használt 1.2.a ábra, melynek forrása megtalálható a forrásmegjelölésben [11].

A hardveres kivitelezés során az alkatrészeket saját anyagi forrásból a ROBTRON ELEKTRONIK TRADE KFT.[12.1], illetve az FDH Kft. [12.2] kínálatából vásároltam. A NYÁK lemezek kivitelezését ugyancsak saját forrásból a MikropanKft-vel [12.3] végeztettem.

Az alkatrészek forrasztását magam végeztem, saját eszközeim, illetve a szakmai gyakorlatom helyéül szolgáló OptIn Kft. [12.4] eszközparkjának segítségével.

A műszerdoboz tervét AutoCAD 2014 programban készítettem el, kivitelezését az általam, a Selmeczky és Társa Épületgépészeti és Kereskedelmi Kft. [12.5] kínálatából vásárolt alapanyagok felhasználásával Rabi Zsolt végezte a tanszéki CNC labor eszközparkjának használatával.

A műszerdobozra kerülő grafikákat CorelDRAW X7 programban készítettem, a matricákat saját forrásból a Perfect Profil Kft-vel[12.6] készíttettem el. A teszteléshez és a végső kialakításhoz használt sakkfigurákat a Régió Játékkereskedelmi Kft.[12.7] szegedi üzleteiben vásároltam. Az üveg sakk készletbe szerelt mágneseket a MagnetPlanet [12.8] webáruházban vásároltam.



1.2.a ábra - A LED mátrix felépítése

2.3 Szoftvertervezés, megvalósítás

A szoftveres kivitelezés során a SiLabs ConfigurationWizard 2, és a SiLabs IDE programokat használtam, a szoftver tesztelését és a hibakeresést a saját tulajdonomban lévő SiLabs USB Debug Adapterrel végeztem. A program írása során internetes fórumokat és forrásokat is felhasználtam, ezek közül kiemelendő az SPI kommunikáció bit-bang módszerrel való megvalósításának módszerét adó kód, melyhez *Randy Rasa MAX7219.C* moduljának *MAX7219_SendByte(...)* függvényét vettem alapul [13], illetve aUtil modulban a késleltetést lehetővé tevő kód megvalósítása, mely az *edaq24.c*fájlból származik, a tanszéki honlapról [14]. Az SDCC fordító használatához forgattam az SDCC felhasználói kézikönyvét [14]. Mind a hardveres, mind a szoftveres fejlesztés során segítségemre volt a "*Laboratory practicals with the C8051Fxxx microcontroller family*" című egyetemi jegyzet.

3.HARDVERFEJLESZTÉS

3.1. Specifikáció

3.1.1 Konstrukciós megfontolások

Az alapvető szerkezet már a tervezés nagyon korai szakaszában eldőlt. Szükség volt egy vezérlő panelre, lehetőleg rajta a felhasználói interfésszel, illetve egy, a sakktáblát alkotó elemre, mely a játéktér alatt a játék követését, és a visszajelzéseket kezeli.

Előbbinek az összetett elektronika miatt mindenképp egy kétrétegű NYÁKnak kell lennie, az utóbbi szerkezetének kialakítása azonban mérete, és különleges feladatai miatt hosszabb folyamat volt.

Abban biztos voltam, hogy a bábok követését szoftveresen kell megoldani, érzékelésükre pedig az olcsón beszerezhető mágneses sakk készletek miatt kézenfekvő megoldásnak tűntek a Reed csöves érintkezők. Az is gyorsan eldőlt, hogy a LEDek meghajtására MAX7219 IC-ket fogok használni, melyeknek azonban 6 kivezetéses RGB LEDekre van szükségük ahhoz, hogy a konstrukció a lehető leghatékonyabb legyen.

A költséghatékonyságot alapul véve az első ötlet az volt, hogy a LEDeket és a reléket tartó elem legyen forrpontos/forr sávos panel. Ez a terv két okból lett elvetve: egyfelől a piacon nem állnak rendelkezésre olyan furatszerelt, 6 lábú RGB LEDek, melyekre igaz lenne, hogy olcsóak, nagy fényerejűek, és nagy bevilágítási szöggel rendelkeznének, ellenben felületszerelt kivitelben a feladathoz sokkal jobban illeszkedő tulajdonságokkal bíró, kültéri kijelzők kialakítására szánt LEDekből "viszonylag" nagy a választék. Másfelől, a forrpontos/forr sávos panelek tervezése, összeszerelése nehézkes, és végső soron lehetséges, hogy költségesebb is, mint NYÁKot készíttetni.

A végső döntés végül az lett, hogy a sakktáblát alkotó LED/Reed panelt négy, teljesen azonos kialakítású részre bontom, ezek egy-egy NYÁKot fognak alkotni, melyek csatlakozók segítségével összeilleszthetők lesznek. Ez által az esetleges hibák könnyedén felderíthetők, ha véletlen egy panel megsérül, sem kell az egész táblát kukába dobni, ráadásul a csatlakozók flexibilitása miatt egy nem tökéletesen méretpontos házba is illeszthetőek lesznek. A NYÁKok további előnye, hogy ezzel a megoldással elkerülhető a félreforrasztásból eredő emberi hibatényező, és a munkaidő egy része is megspórolható.

3.1.2 LED/Reed panel általános specifikációja/feladatai

A LEDeket, illetve Reed reléket tartalmazó panelnek három fontos feladata van:

- Megfelelő érzékenységű, és érzékelési hatósugarú mezőket kell létrehozni:
  - A bábuk jelenlétét és felemelését helyesen és megbízhatóan kell érzékelni;
  - az érzékelési felületek nem fedhetik egymást, a szomszédos mezőről "belógó" bábu nem zavarhatja meg a tábla működését.
- Megfelelő fényerejű, és bevilágítottságú visszajelzés a játék felületen:
  - A jelzéseknek nagyobb intenzitású környezeti fény mellett is láthatónak kell maradniuk;
  - a jelzések nem folyhatnak össze, egyértelműnek kell lennie, hogy a jelzés mely mezőre vonatkozik.
- Moduláris kialakítás a költséghatékonyság, és a hibakeresés megkönnyítése érdekében.

3.1.3 Vezérlő panel általános specifikációja/feladatai

A vezérlő panelnek hat fontos, jól körülhatárolt feladata van:

- Reed mátrix olvasása a játék nyomon követésére;
- RGB LED mátrix vezérlése a sakkjáték közben megjelenítendő "segítő funkciók" vizualizációjához;
- Felhasználói interfész biztosítása a tábla beállításainak módosításához, és egyéb felhasználói kommunikációra;
- USB interfész biztosítása PC kommunikációra;
- "intelligencia" biztosítása a tábla működéséhez és vezérléséhez;
- megfelelő, biztonságos tápellátás biztosítása az egész berendezésnek, lehetőséget hagyva belső áramforrás használatára is.

3.2 Hardware elemek kiválasztása, főbb tulajdonságaik ismertetése

3.2.1 LED/Reed panel

3.2.1.1 Reed mátrix

A 3.1.2 bekezdés specifikációjának megvalósítására A Reed mátrix kialakítása során hagyományos, mátrixos elrendezés mellett döntöttem megbízhatósága és egyszerű tervezhetősége miatt. A nyolc sor és a nyolc oszlop a mátrix metszéspontjaiban egy-egy diódával sorosan kötött Reed relén keresztül van összekötve egymással. Egy időben mindig csak egy sorra kapcsolunk VCC-t, majd az oszlopokon megjelenő értékeket kiolvasva megtudhatjuk, az adott sorban mely relék vannak bekapcsolt állapotban, és melyek nem.

Reed relékből az elérhető legkisebb, és legolcsóbb Reed csövet választottam (AC 031520). Ez a relé alapállapotban nyitott, mágneses térbe helyezve zárt állapotú (aktív zárt), és megfelel a 3.1.2 bekezdésben megfogalmazott specifikációnak; az előzetes tesztek alapján a teszt célra megvásárolt mágneses sakkfigurákat akkor (is) érzékeli, ha azok a cső tengelyének vonalában vannak, így a relék álló helyzetben helyezhetők el a panelen. (Ez azért is nagyon fontos, mert így a játék felület, és a panel között a relék hosszának megfelelő távolság lesz, így a LEDek fénye nem pontszerű lesz, hanem egy 3cm sugarú kör).

A Reed relék diódán keresztül kell, hogy az oszlopokat és a sorokat összekössék. Erre a célra a lehető legolcsóbb furatszerelt diódát választottam (BAV 21).

3.2.1.2 LED mátrix

A LEDek kiválasztása bizonyult az egyik legnehezebb feladatnak az összes alkatrész közül. Érzékeny egyensúlyt kellett keresni az ár, a fényerő, a méret, a kibocsátott fény bevilágítási szöge között, ráadásul 6 kivezetéses (színenként szeparált anód és katód) SMD RGB LED-ekre volt szükség.

A kiválasztás egyéb fő szempontjai a következők voltak:

- maximum 150-200Ft/darab
- ~20mA tápáram (A kiválasztott LED meghajtó miatt)
- <5V tápfeszültség
- A luminozitás legyen>1cd, legalább egy, de inkább 2 színnél

- A színek közötti luminozitási különbség ne legyen túl nagy, a legkisebb és a legnagyobb fényerejű szín aránya maximum két-háromszoros legyen.
- A bevilágítási szög legyen minimum 60-80 fok, de maximum 120-130 fok
    - 120 fok környéke az ideális, ekkor a LED függőleges tengelyére merőlegesen álló, tőle ~1.5cm távolságban lévő felületen egy kb. 3 cm sugarú kör az, ahol a megvilágítottság a közvetlenül a LED fölötti ponthoz képest 50%-ra esik vissza, tehát a LED ezen a felületen egy ~3cm sugarú kört világít be. Ez azért fontos, mert a terv az volt, hogy a mezők mérete 3x3 cm lesz, és a Reed relék hajlítással együtt mért magassága kb. 1.5cm körül alakul.)

A kiválasztott LED a korábban felvázolt kívánalmaknak tökéletesen megfelel.[5]

3.2.2 Control Panel elemei

3.2.2.1 Reed mátrix kezelése - Dekád számláló

A korábban vázolt kapcsoló mátrix kialakítás (3.2.2.1 bekezdés) 8 bemenet, és nyolc kimenet használatát teszi szükségessé. Azért, hogy ennél jóval kevesebb MCU pin felhasználásával megoldható legyen a feladat, a bementi oldalt egy Johnson Counteren (Dekád számláló, 74HC4017 [6]) keresztül hajtom meg, melynek csak két bemenetét kell az MCU-nak kezelnie (Reset és Step).

3.2.2.2 LED meghajtó áramkör

A LEDek meghajtására Mingesz Róbert ajánlotta a MAX7219 IC-t, mely *képes egy 8x8 LED mátrix meghajtására*, alig néhány külső alkatrész segítségével. Azon kívül, hogy 3 darab ilyen IC-vel megvalósítható a teljes kijelző panel meghajtása, emellett a modell mellett szóltak a következő tulajdonságai:

SPI buszon keresztül vezérelhető, és a három IC akár sorba is köthető, így a teljes vezérlésre elegendő csupán 3 pin az MCUn. Az IC természetesen képes a LEDek egyenkénti vezérlésére, ezen felül lehetővé teszi a fényerő globális szabályzását 16 fokozatban (a három IC így színenként fényerő beállítást/színhangolást tesz lehetővé). Az IC-nek van teszt, és shutdown üzemmódja is, előbbiben az összes LEDet felkapcsolja, utóbbiban minden LEDet kikapcsol, és alacsony fogyasztású módba áll.[7]- Ez utóbbi üzemmód lehetővé teheti az akkumulátoros üzemidő kiterjesztését is.

A meghajtók mellé egy-egy trimmer potenciométer is kerül, melyek feladata, hogy a LEDek fényerejét finom hangolni lehessen a meghajtó IC-k maximális kimenő áramának korlátozásával. Mint később kiderült, igazából nincs rájuk túl nagy szükség, a melléjük beépített korlátozó ellenállások bőven elegendőnek bizonyultak.

3.2.2.3 Felhasználói interfész

Úgy döntöttem, hogy a feladatra három gomb, és egy LCD kijelző lesz a táblára szerelve, a gombok segítségével lehetséges a menüben navigálni, míg a kijelző bármilyen üzenet megjelenítésére alkalmas lehet (sakkóra, konfigurációs menü, stb.).

A feladatra egy egyszerű, záró érintkezős SMD nyomógombot választottam (DTSM 24N/SMD). Kiválasztásánál fontos volt, hogy legyen viszonylag nagy, hogy az egyedi gomb sapkák kialakítása később egyszerű legyen.

A kiválasztott LCD kijelző az EA DOGM162W-A, mely egy2 soros, soronként 16 karakteres, (nem teljesen szabványos) SPI buszon keresztül vezérelhető LCD kijelző. Moduláris kialakítása által egyszerűen kapcsolható hozzá a projekthez illeszkedő háttérvilágítást biztosító modul. [8]Mivel a házat fehérnek terveztem, így fehér színű LED modult (EA LED55X31-W) szereztem be hozzá.

Ezen LCD kijelző nagy előnye, hogy számos konfigurációs lehetősége mellett (pl. kontraszt szoftveres konfigurációja), SPI kommunikációs interfészt használ, így vezérlése nagyon egyszerű, és viszonylag kevés vezeték/pin használatát teszi szükségessé.

3.2.2.4 USB kommunikáció

Ugyancsak tanszéki ajánlás alapján lett kiválasztva (FT 232 RL). Az MCUval UART interfészen keresztül kommunikál (2 pin), egyszerű soros összeköttetés kialakítását teszi lehetővé egy PC-vel USB interfészen keresztül, minimális számú külső elem használata mellett.[9]

3.2.2.5 Intelligencia

A sakktábla "agyául" igyekeztem olyan mikrovezérlőt választani, melynek programozását, és képességeit már többnyire ismertem, ezért rövid gondolkodás után a C8051F410-GQ mellett döntöttem. Mivel főként logikai vezérlési feladatok ellátására volt szükségem, ezért a Timerek/vegyes jelű perifériák egyáltalán nem voltak fontosak az alkatrész kiválasztásakor (bár ekkor még tervben volt a fényerő környezetei fény függvényében való szabályzása, ám ez alkatrész beszerzési gondok miatt nem valósult meg).

A tervezett sakkóra megvalósításához természetesen szükséges volt egy Timer felhasználása is, melyből az MCUban igen széles választék áll rendelkezésre.

A kiválasztott mikrovezérlő számomra fontos funkciói/tulajdonságai:

- 32kB Flash memória
    - ez a legnagyobb elérhető programmemória méret a családban.
- 2304 byte RAM
    - az előzetes becslések alapján bőven elegendő
- Beépített debug lehetőségek
    - fontos szempont, hogy már beépített eszközt is lehessen debugolni, ez a fejlesztést nagyban megkönnyíti
- 2.0 - 5.25V tápfeszültség, ráadásul független IO feszültség is megadható
    - ez mind kompatibilitási, az mind akkumulátoros tervek miatt fontos
- Beépített UART/SPI/I2C perifériák
    - Kommunikációs perifériák a többi IC felé. Mint később kiderült, sajnos kompatibilitási okokból ezeket nem sikerült kihasználni.
- 24 IO port
    - a tervek szerint éppen elegendő
- Akár 50MHz-es rendszeróra, és akár 50MIPS végrehajtási sebesség
    - Minél gyorsabb, annál jobb, főként a felhasználói élmény miatt.[10]

Nem elhanyagolható továbbá azon tény sem, hogy saját használatra rendelkeztem egy fejlesztő kittel, illetve Debug adapterrel is ezen mikrovezérlő családhoz, ráadásul a tanszéken is sokan foglalkoznak ezen MCUkra való fejlesztéssel, így a felmerülő problémákban számíthattam segítségre.

Ezen felül, a programozó interfész beépítés utáni programozásához is rendelkezik a labor cél hardverrel, mely lehetővé teszi, hogy a 10 pinen keresztül működő debug-adaptert 3 pines csatlakozó segítségével csatlakoztathassam a már beépített MCUhoz.

A kiválasztás előtt elkészítettem egy táblázatot, mely tartalmazza, hogy a különböző pinekhez milyen funkciót (bemenetet vagy kimenetet) szeretnék kapcsolni (függelék F4, de szerepel a 3.3.2.1.g ábrán is), mely által meggyőződtem, hogy az MCU elegendő pinnel rendelkezik feladatainak ellátására.

3.2.2.6 Táprendszer

Konstrukciós megfontolások, specifikáció

A táprendszer bemeneti oldalán 4.75V - 5.25V egyenáramú tápfeszültséget kap, melyet egy fali adapter biztosít. (Amennyiben később akkumulátoros üzemre is képessé szeretnénk tenni a sakktáblát, akkor az ezen a bemeneten kapott feszültséget 4V - 5.5V között kell tartani a tábla helyes működéséhez.) A tábla teljes fogyasztása max. kb. 1.5A, ebből a LED meghajtók maximális fogyasztása 1A.

A táprendszer két fő "ütemből" áll. Az elő kondicionáló szakasz biztosítja, hogy a táblára kapcsolt rendellenesen magas feszültség, vagy feszültségtüske ne okozhasson kárt (előbbi Zener, utóbbi Suppressor dióda segítségével), illetve korlátozza a tábla által felvehető maximális áramot, amennyiben rendellenes működés (pl. rövidzár) következne be (ezt egy regenerálódó biztosíték teszi meg.) Ez a szakasz közvetlenül biztosítja a LED meghajtó IC-k tápellátását. A második szakasz egy LDO regulátor segítségével megbízható, védett tápellátást biztosít a logika (MCU, USB interfész, stb.) számára, amennyiben az elő kondicionáló szakaszról jövő feszültség még mindig magasabb lenne, mint az ezen egységek által várt maximális feszültség (5.25V).

Előkondicionáló szakasz alkatrészei

- 1.5A-nél 1s alatt kioldódó SMD kivitelű regenerálódó biztosíték - *SMD 150F-2 1.5A*
- 5V Suppressor dióda (1500W/1ms) - *SMCJ5.0A/SMD 5 V*
- 5.6V letörési feszültségű Zener dióda, max 0.33A - *SMZ5.6*

Második szakasz alkatrészei

- 5V feszültség stabilizátor IC - *LP 2985AIM5-5.0*
- *VÁLTOZÁS: A feszültség stabilizátor IC hibás működése miatt az IC helyén a végső eszközben egy egyszerű dióda kapott helyet.*

3.3 Hardware tervezés - kapcsolási rajz, NYÁK terv, ház terv készítése

3.3.1 LED/Reed panel

3.3.1.1 LED/Reed panel kapcsolási rajz (CD melléklet, függelék F3.3)

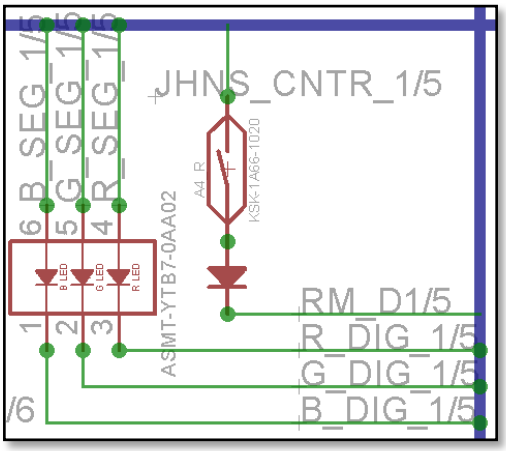
Mint az korábban, a 3.1.1 és 3.1.2 bekezdésekben már említésre került, a panel négy, teljesen azonos felépítésű NYÁK csatlakoztatásával jön létre. Az egyes elemek csak a rajtuk lévő csatlakozók típusában és elhelyezésében különböznek egymástól. Ezáltal, a tervezés ideje is nagyban lerövidült, hiszen csak a végső panel 1/4-ét kellett megtervezni.

Minden panel gyakorlatilag négy, teljesen független mátrixot tartalmaz. Egyfelől 3, független LED mátrixot az egyes színeknek, illetve a Reed relékből és a hozzájuk kapcsolódó diódákból álló "beolvasó" mátrixot.

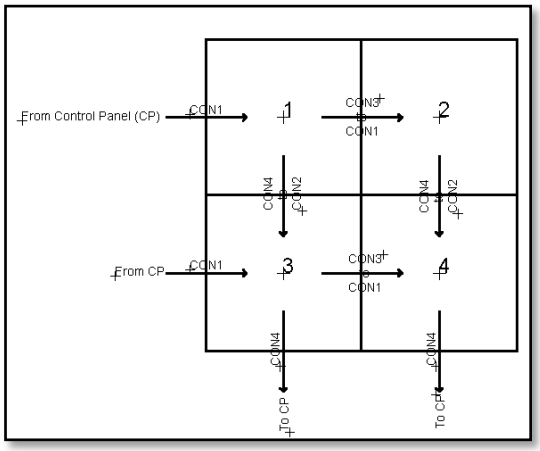
A LED mátrixok áramköri tervezésénél referenciául használtam a *http://rgb.kitiyo.com/2008/7219-app-circuit.html* oldalon található kapcsolási rajzot (1.2.a ábra). Ennek 4x4-es részét alakítottam ki, majd ebből a 4x4-es mátrixból 3-at helyeztem egymás mellé, oly módon, hogy mind a függőleges, mind a vízszintes "vezeték nyalábok" csatlakozókon keresztül továbbvihetőek legyenek mind a 4 irányba.

A Reed mátrix kialakításánál a hagyományos kapcsoló mátrix elrendezést követtem, ahogy azt a 3.2.2.1 bekezdésben már említettem, és ezt a 4x4-es kapcsoló mátrixot is a LED mátrix mellé ültettem (3.3.1.1.b ábra).

A panelek végső elrendezése (3.3.1.1.c ábra) nagyon fontos volt már a tervezésnek ebben a fázisában is, hiszen mind a vízszintes, mind a függőleges vezeték "nyalábokat" megfelelő módon kellett kezelni ahhoz, hogy később a vezérlő panelen jól azonosíthatóak legyenek, és a megfelelő csatlakozókat lehessen kialakítani.



3.3.1.1.ábra



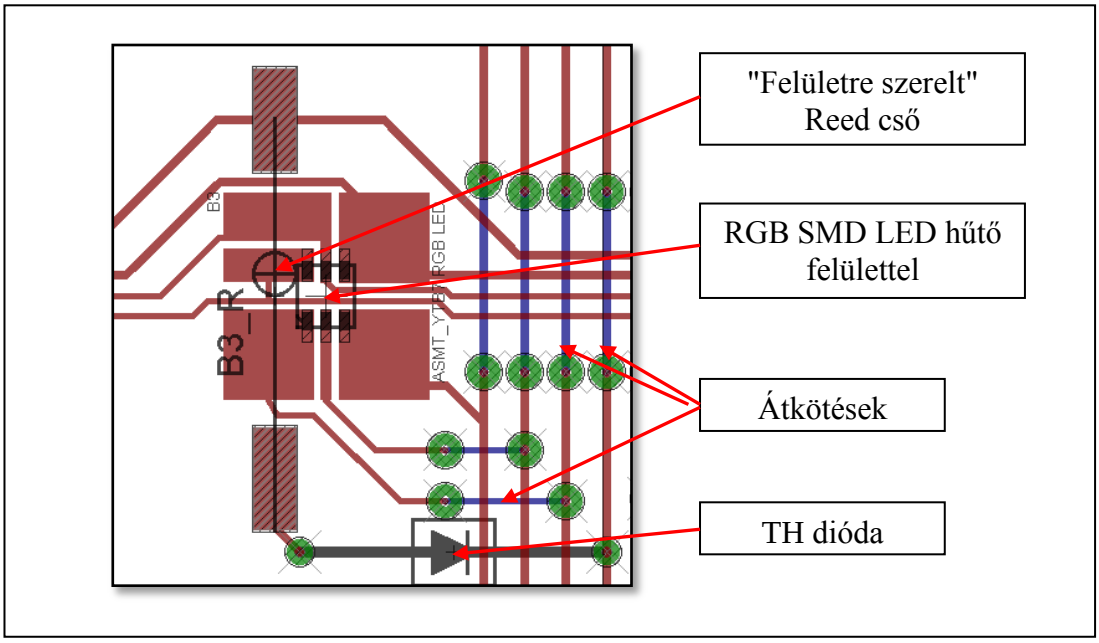
3.3.1.1.c ábra

3.3.1.2 LED/Reed panel NYÁK terv (CD melléklet, függelék F3.4)

A mátrixos elrendezés elkészítése mindenképpen szükségessé tette egymást keresztező vonalak létrehozását, mivel azonban a LED/Reed panel mérete durván 24cm x 24cm, kétrétegű NYÁK gyártása hatalmas pluszköltséget jelentett volna. Ebből kifolyólag a tervezés során szem előtt kellett tartani, hogy az egyik oldal viszonylag egyszerűen kivitelezhető legyen jól követhető, légvezetékes átkötések alkalmazásával. Ezek a vezetékek néhány helytől eltekintve mindig szigorúan vízszintes vagy függőleges irányúak, hosszuk körülbelül azonos. Ezen átkötésekhez megfelelő méretű furatokat, és körülöttük kényelmes méretű padeket hagytam, megkönnyítendő a forrasztási feladatok elvégzését.

A LEDekhez a footprint ajánlást követve viszonylag nagyméretű hűtő felületeket hoztam létre, mely biztosítja a LEDek megfelelő üzemi hőmérsékletét még nagy terhelés mellett is. A Reed mátrixhoz kiválasztott diódák TH tokozásúak, így többlet vezeték nélkül csatlakoztathatók a megfelelő függőlegesen futó vezetékhez.

A Reed relék, bár tokozásuk megengedné, az egyszerűbb szerelhetőség miatt "felületre szerelten" kerültek a panelre. (Ahhoz, hogy a kijelölt helyükön állhassanak, így csak 2 ponton kell meghajlítani a lábukat, ha furatba is kellene illeszteni a lábak végét, 4 ponton kellett volna hajlítani őket, ami a Reed csövek törékenysége miatt nagyban megnehezítette volna a feladatot.) A relék cél pozícióját az ábrán keresztet tartalmazó körök jelzik, az alkatrészek lábainak alakításakor ennek betartása volt a fő cél (3.3.1.2.a ábra).



3.3.1.2.a ábra - A mátrix egy tipikus mezője

3.3.2 Control Panel

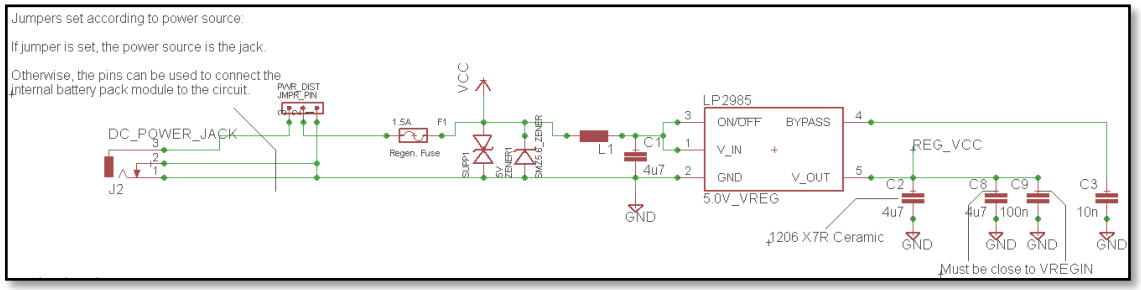
3.3.2.1 ControlPanel kapcsolási rajz (CD melléklet, függelék F3.1)

Táprendszer (3.3.2.1.a ábra)

A táprendszer kialakításakor az előkondicionáló szakasz a DC csatlakozóval kezdődik, majd egy egyszerű, tűs csatlakozón át vezet először a regenerálódó biztosítékhoz, majd a védő diódákhoz. A tűs csatlakozó célja, mint ahogy az már korábban említésre került (3.2.2.5 bekezdés), hogy belső, akár akkumulátoros tápegység is csatlakoztatható legyen. (Alapállapotban a csatlakozó jumperként van használva.)

A második szakasz az LDO regulátort és a hozzá kapcsolódó kondenzátorokat tartalmazza. Ezek a passzív alkatrészek természetesen az LDO adatlapja alapján lettek kiválasztva.

A táprendszer kimenetei a VCC, azaz a "nyers", LDO által nem védett táp, míg a REG_VCC a logika számára létrehozott, LDO által védett vonal.



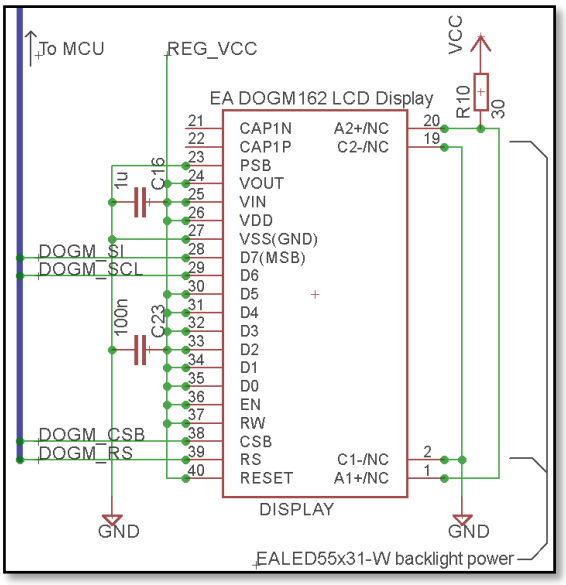
3.3.2.1.a ábra - A táprendszer kapcsolási rajza

Felhasználói interfész (3.3.2.1.b , és c ábra)

A felhasználói interfész az LCD kijelzőből, az "alá ültetett" háttérvilágításból, illetve a gombokból áll (3.2.2.2 bekezdés). Az LCD SPI-szerű buszon keresztül kapcsolódik az MCU felé.

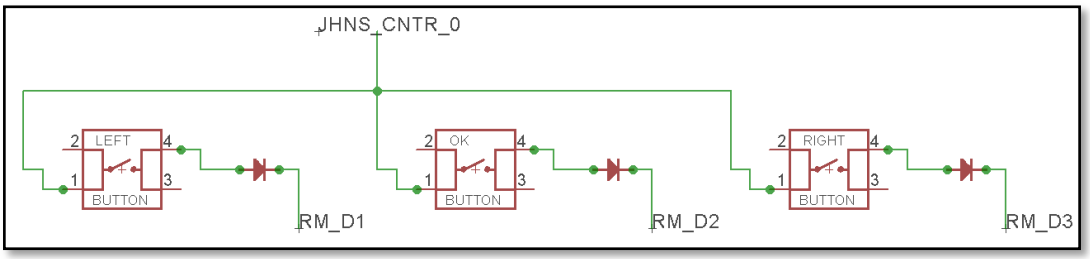
A kijelző logikája számára néhány, az adatlapban nem szereplő, de mindig célszerű hidegítő kondenzátor került felszerelésre.

Az EA LED háttérvilágítás párhuzamos tápellátással került kialakításra, az áram-korlátról egy 30 Ohmos ellenállás gondoskodik, az adatlap alapján (3.3.2.1.b ábra).



3.3.2.1.b ábra - Az LCD modul és a háttérvilágítás kapcsolási rajza

A gombok egyszerű záró érintkezők, melyek MCU pin megtakarítás végett gyakorlatilag a Reed mátrix egy extra soraként vannak megvalósítva, tehát a gombokra a dekád számláló 0. kimenete visz feszültséget, állapotuk beolvasása pedig a Reed mátrix első 3 beolvasó vonalán történik. A gombokkal sorosan itt SMD dióda gondoskodik a helyes működésről (3.3.2.1.c ábra).

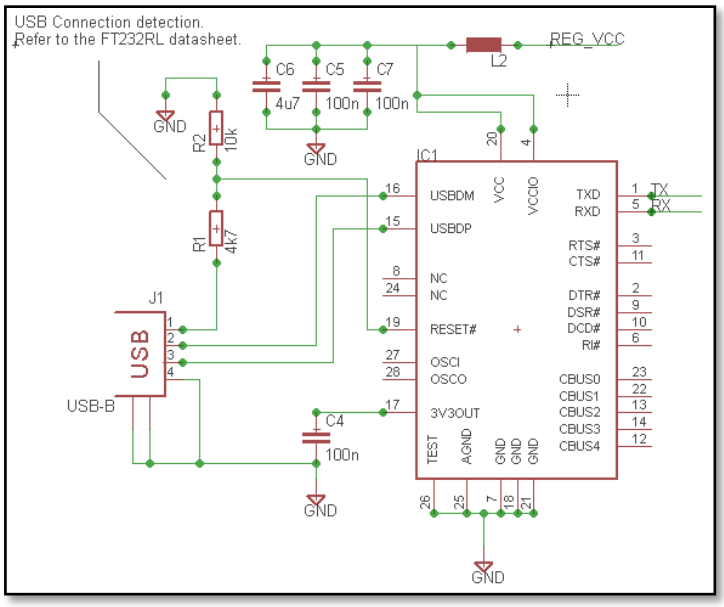


3.3.2.1.c ábra - A gombok bekötésének kapcsolási rajza

USB kapcsolat (3.3.2.1.d ábra)

Az USB kapcsolatért felelős elemek az USB-B típusú csatlakozót, az FT232RL IC-t, illetve a hozzá kapcsolódó passzív áramköri elemeket takarják (3.2.2.3 bekezdés).

A terv az IC-hez tartozó adatlap ajánlott kapcsolása alapján lett elkészítve, szem előtt tartva, hogy az IC/Panel tápellátása az USB-től független, így az USB tápvonala csak arra van használva, hogy az USB csatlakoztatásakor megszüntesse az IC Reset állapotban tartását. Az IC UART RX/TX vezetékpáron keresztül kapcsolódik az MCUhoz.



3.3.2.1.d ábra - USB kezelő elemek kapcsolási rajza

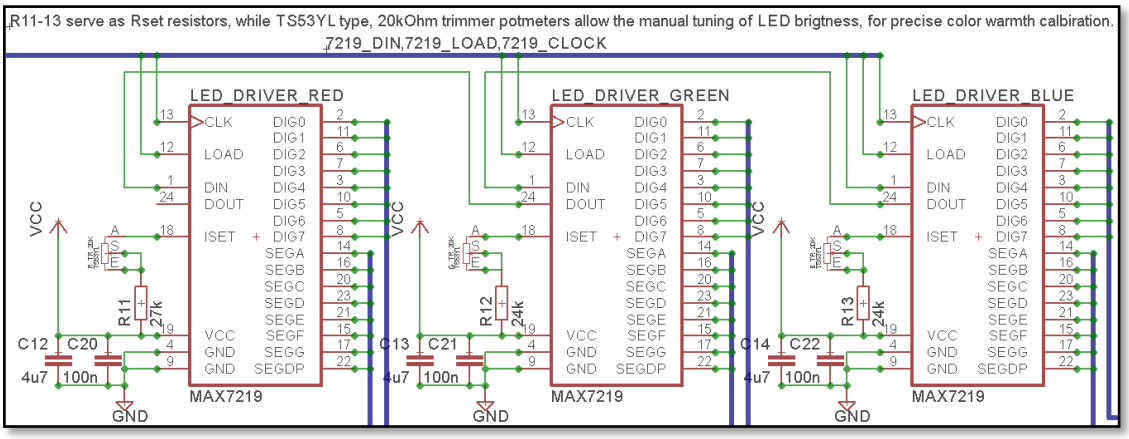
LED/Reed panel kezelő rendszer (3.3.2.1.e, f ábra)

A panelek kezelése két fő részből áll, egyfelől a LED meghajtók (3.2.2.2 bekezdés, 3.3.2.1.e ábra), másfelől a dekád számláló (3.2.2.1 bekezdés), és az MCU bemenő lábaira kapcsolt beolvasó vonalak (3.3.2.1.f ábra) felelnek a kijelző felé/felől való kommunikációért.

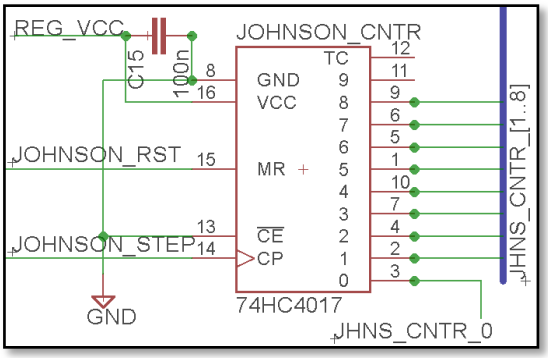
A kapcsolat 4 darab, (az eredeti tervek szerint 90 fokos, ám a ház megtervezése után egyenesre cserélt), 16 tűs szalagkábel csatlakozón keresztül jön létre Control Panel és a LED/Reed Panel között. A MAX7219 meghajtók a 3.3.1.1.a ábra szerint kapcsolódnak a LED mátrixhoz.

A LED driverek mellett megtalálhatók az adatlapban meghatározott passzív elemek - a hidegítő kondenzátorok, illetve a trimmer potenciométerrel kiegészített, maximális kimeneti áramot meghatározó ellenállások. Ezen kívül a három meghajtó vezérlő regiszterei sorba vannak kapcsolva a DIN/DOUT vonalakon keresztül, így a három meghajtó vezérléséhez elegendő 3 pin az MCUn.

A dekád számlálóhoz ugyancsak tartozik hidegítő kondenzátor, a számláló kimenetei a csatlakozók megfelelő tűire vannak kapcsolva. A Reed relé mátrix felől az MCUhoz érkező adatvonalak, és a hozzájuk tartozó lehúzó ellenállások az MCU környéki kapcsolási rajzon találhatók (3.3.2.1.g ábra).



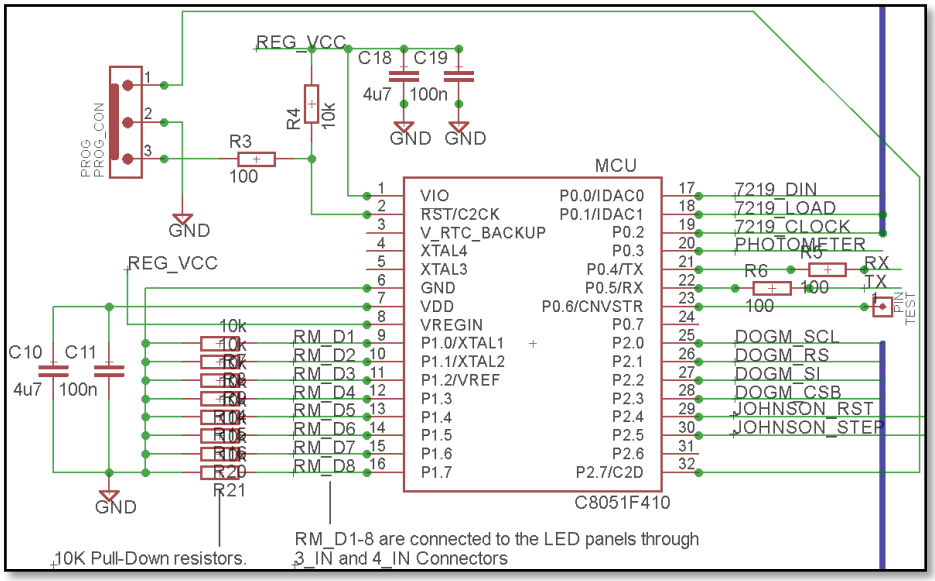
3.3.2.1.e ábra - A LED meghajtók kapcsolási rajza



3.3.2.1.f ábra - A dekád számláló kapcsolási rajza

MCU és közvetlen környezete (3.3.2.1.g ábra)

Az MCU (3.2.2.4 bekezdés) környezetében az adatlapban szereplő passzív elemek (hidegítő kondenzátorok), a három tűs programozó csatlakozó, egy test pin, illetve az adatvonalak lehúzó ellenállásai kaptak helyet.

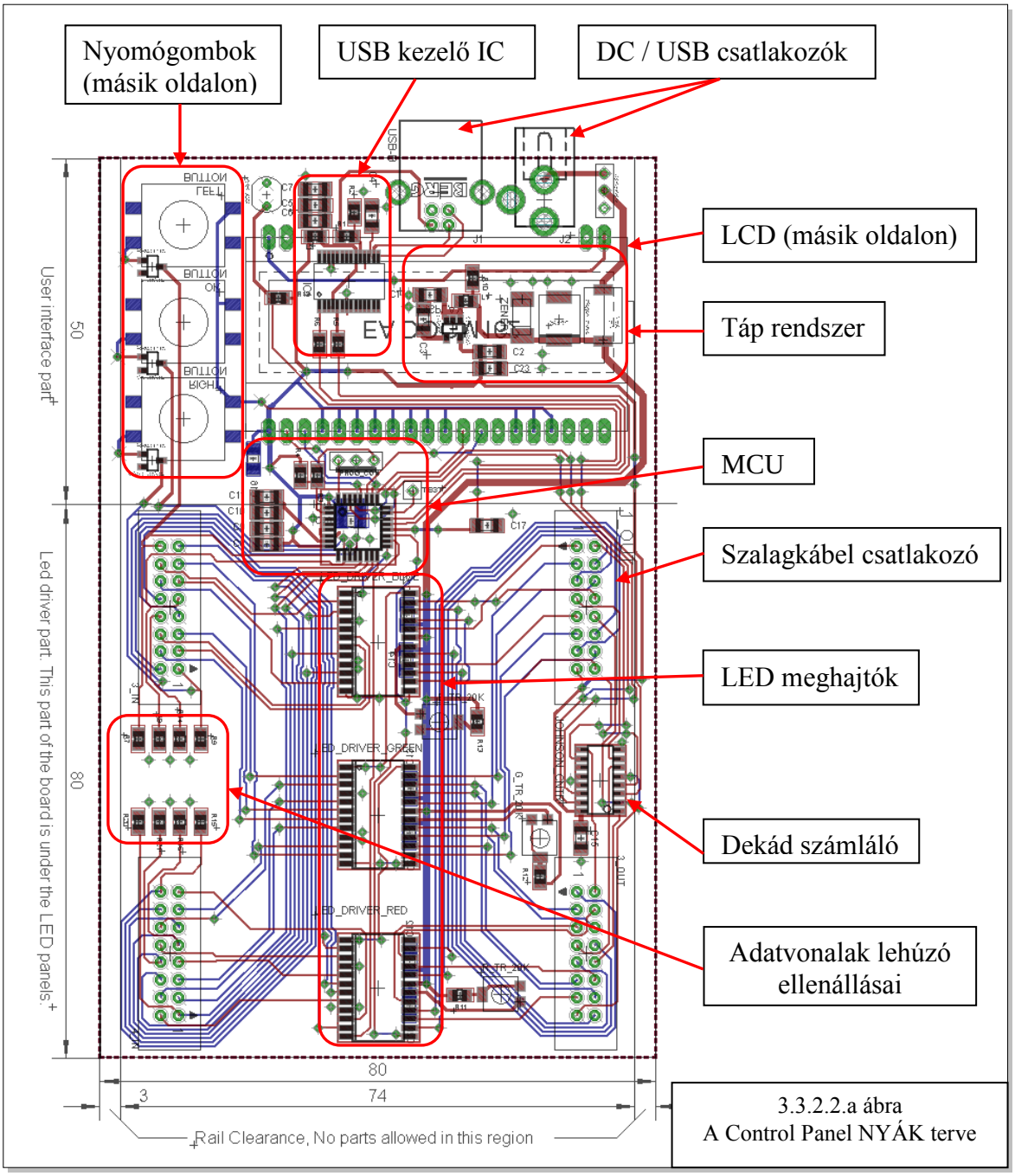


3.3.2.1.g ábra - Az MCU közvetlen környezetének kapcsolási rajza

3.3.2.2 Control Panel NYÁK terve (3.3.2.2.a ábra, CD melléklet, függelék F3.2)

A NYÁK terven látható, hogy a panel két fő részből áll. A felhasználói interfész oldalon kapott helyet az LCD kijelző, a gombok, a DC, USB csatlakozók, és a táprendszer. A LED meghajtó oldal fő elemei természetesen maguk a meghajtók, és a dekád számláló. Az MCU a két rész határán helyezkedik el, így a vezérlő vonalak hossza minimalizálható volt.

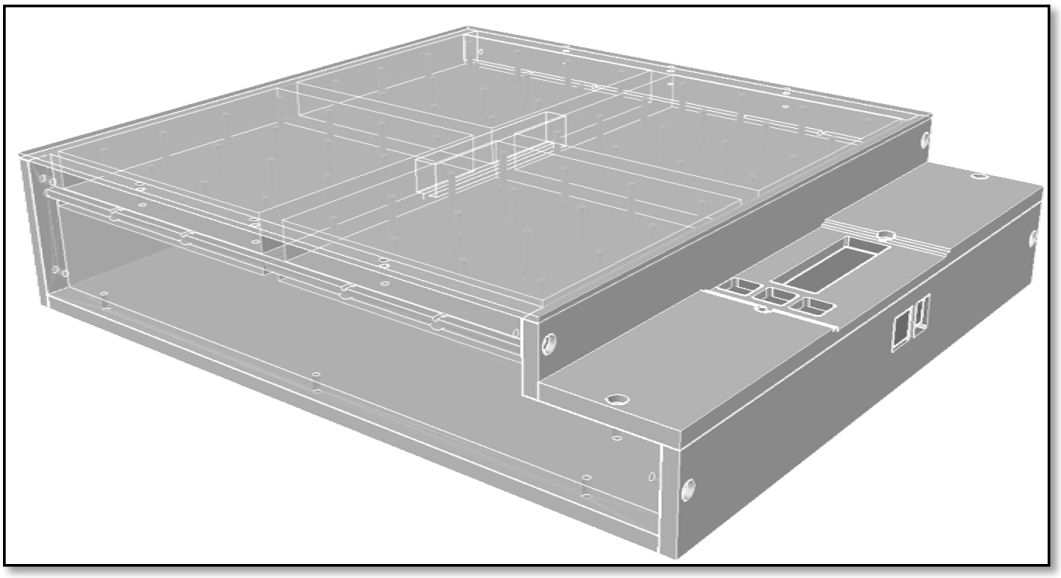
A LED meghajtók felépítését követve a kimeneti szalagkábel csatlakozók (anód oldal, illetve dekád számláló kimenetei) a panel (alulról nézve) jobb oldalán, míg a bejövő oldali szalagkábel csatlakozók (katód, és adat vonalak) a panel bal oldalán kerültek elhelyezésre.



3.3.2.2.a ábra
A Control Panel NYÁK terve

3.3.3 Ház/Műszerdoboz megtervezése (Terv a CD mellékleten)

A terv egy viszonylag egyszerű házat takar, melynek két fő része van: egyfelől a felhasználói interfész, másfelől maga a játék felület. Előbbi egy, a tábla egyik oldalán megtalálható, alacsonyabban lévő "lépcsőfok", így a Control Panel a könnyebb csatlakoztatás érdekében a LED/Reed panelek alá nyúlhat, míg utóbbi egy egyszerű, 24x24cm-es sakktábla, alatta a LED/Reed panelekkel (3.3.3.a ábra).

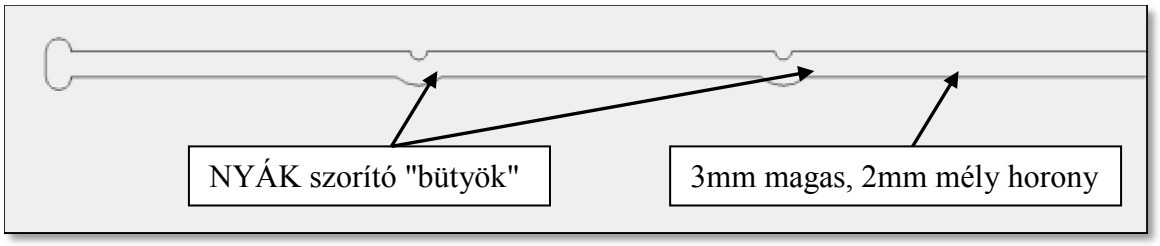


3.3.3.a ábra - Ház/Műszerdoboz AutoCAD terve

A tábla anyaga az oldalfalak, a középső tartóelem, illetve a Control Panelt tartó elemek esetén 10mm vastag víztiszta plexi, míg a többi felületen 6mm vastag habosított PVC. A játéktér 2mm vastag víztiszta plexi, melyen a sötét/világos mezőket egy, a plexi lap alsó oldalára felvitt öntapadó matrica alakítja ki (hasonlóképp, a felhasználói interfész grafikái is ilyen matrica segítségével kerülnek a helyükre).

Az igazi probléma a NYÁKok rögzítésének módja, melynek koncepciója Szépe Tamás segítségével született meg. A ház falában, körben egy horony fut, melybe a NYÁKok szélén kialakított, alkatrészeket nem tartalmazó sávok illeszkednek. A hornyok 3mm magasságúak, és változó mélységűek, a NYÁKok 2mm vastag lemezeit kis "pöckök"(3.3.3.a ábra) szorítják a hornyokba. Azért, hogy a LED/Reed panelek összeillesztés után teljesen vízszintesen álljanak (a csatlakozók nem elég merevek ehhez), egy, a játéktér közepén álló tartó elem támasztja meg őket.

Az elemeket M3 típusú, süllyesztett fejű csavarok rögzítik egymáshoz.



3.3.3.a ábra - A Control Panelt tartó egyik elem oldalnézeti képe (furatok nélkül)

3.4 Hardver megvalósítása

3.4.1 Eltérések a terv és a megvalósítás között

Alapvetően nincs túl sok eltérés a terv és a megvalósított hardver között, csupán néhány kisebb hibára/eltérésre derült fény.

1. A táprendszer első szakaszán a DC táp csatlakozó hibás láb kiosztással szerepelt az Eagle könyvtárban, így a terv szerint nem a középső tűről van levéve a tápfeszültség. Ezt a NYÁK és a csatlakozó "megbuherálásával" sikerült áthidalni.
2. A táprendszer második szakasza valamilyen, máig ismeretlen oknál fogva nem volt hajlandó működni, az LDO kimenetén nagyon alacsony (<3.5V) feszültség jelent meg, mely sem az IC, sem a kondenzátorok cseréjével nem oldódott meg. Lehetséges, hogy az LDO bemenete és elvárt kimenete közti kis különbség volt a hiba oka, lehet hogy a túl magas forrasztási hőmérséklet, ám a megoldásra nem sikerült rájönni. A probléma úgy lett áthidalva, hogy az LDO-t egy egyszerű diódával helyettesítettem, mely 0.6V-ot vesz le a bejövő feszültségből, így biztonságos szinten tartja a REG_VCC vonalat.
3. Volt egy kisebb probléma a Control Panel és a LED/Reed panelek közti szalagkábelekkel is. Az eredeti terv szerint itt mindenhol 90 fokos csatlakozók lettek volna beépítve, hogy a tábla a lehető legvékonyabb lehessen, azonban a ház tervezése során kiderült, hogy ezeket egyenes csatlakozókra kell cserélni. Sajnos ekkor már a NYÁKok gyártás alatt voltak, így bár a csatlakozók lenyomata azonos, a tűkre vezetendő jelek helytelenek lettek (az egyik oldalon hosszában tükrözni kellett volna a pineket). A megoldás az lett, hogy a szalagkábeleken páronként meg kellett cserélni a vezetékeket, hogy minden a helyére kerüljön.
4. Apró hiba volt még, hogy a Control Panelen a furat átmérők meghatározásánál a csatlakozóknál és a tüskéknél figyelmen kívül hagytam a furatgalván vastagságát, azonban a tűk smirglizése percek alatt megoldotta a gondot.

4 Szoftverfejlesztés

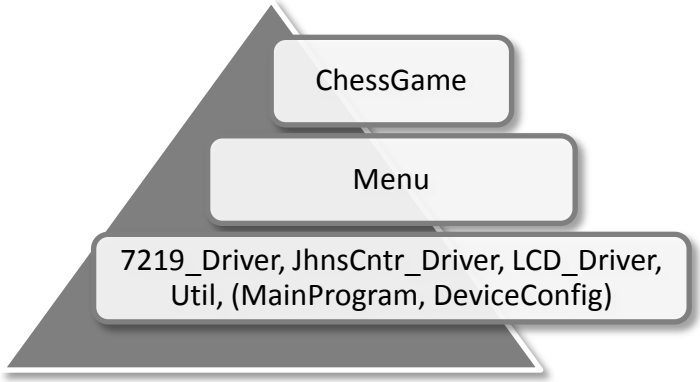
A szoftver specifikációja gyakorlatilag egybevág az (F1) mellékletben olvasható, megvalósított funkciókkal, és ott teljes egészében kifejtésre került. A következőkben modulról modulra végigveszem az egyes szoftver részeket, feladataikat, és kiemelek néhány olyan függvényt, melyek megvalósítása kihívás volt, vagy egyéb okból érdekes lehet.

A szoftver fejlesztése során az inkrementális fejlesztési modellt követtem, és bár a fejlesztés során keletkeztek tervek, az egyes modulok az újabb és újabb funkciók hozzáadásával folyamatosan változtak, fejlődtek, ebből kifolyólag a fejezet nem lesz "Tervezés" és "Megvalósítás" szakaszokra bontva.

4.1 Koncepció, felépítés

A program írásakor igyekeztem betartani a C programozás írott és íratlan szabályait. Az egyes sorok tabulátorral igazítottak, a függvények, illetve azok működése kommentekkel van ellátva. Minden állomány (a MainProgramon kívül) header és source fájlokból áll, előbbiek tartalmazzák a "publikus", más programrészek által is használható függvények prototótípusait, a használható makrókat, és kommentben használatuk módját, míg a source fájlok tartalmazzák a "publikus" és "nem publikus" függvények implementációját. E fájlok elején találhatóak a modulra nézve globális változók, feladatukkal, és használatukkal kommentezve, alattuk a saját, belső függvények, míg a fájlok végén, külön jelzés után az egyéb állományok által is használható függvények implementáció állnak.

A program modulok funkcionalitásukat tekintve két kategóriába sorolhatók: driver/meghajtó modulok, illetve felhasználói modulok, azaz a Menüt,és a Sakk játékot kezelő modulok (4.1.a ábra).



4.1.a ábra - A szoftver felépítése

4.2 Driver modulok

4.2.1 DeviceConfig, és MainProgram

Ezek a modulok kötelező elemei minden mikrovezérlő programnak, alapvető feladataik a mikrovezérlő konfigurációja, és a felhasználói program indítása. A DeviceConfig.cfájl gyakorlatilag a *SilabsConfigurationWizard 2* által generált fájl. Főbb konfigurációs beállítások a VDD monitor engedélyezése, a kimeneti vonalak push-pull módba kapcsolása, a rendszer órajel 25MHz-re emelése, a sakkóra működtetéséhez használt Timer0 konfigurációja és engedélyezése, a megszakítások globális engedélyezése, és a WatchDogTimer kikapcsolása (a végső program nem használ egyéb perifériát, mert mind a MAX7219, mind az EA DOGM LCD modullal való kommunikáció - előre nem észrevett inkompatibilitási okokból - bit-bang módszerrel lett megvalósítva).

A gond röviden abból fakadt, hogy az eredeti terv szerint a MAX7219-ek felé bit-banggel lett volna megoldva a kommunikáció, az LCD felé pedig az SPI buszon. Később azonban kiderült, hogy az LCD nem teljesen SPI kompatibilis, mivel az RS lábon keresztül is inputot vár, történetesen az RS láb állapotától függően van parancs vagy adat módban. Az RS láb implicit állapot beállítását nem sikerült az SPI perifériára kényszeríteni. (A meglévő pin kiosztás miatt pedig az SPI interfészt nem lehetett a MAX7219 felé vezető vonalakra kapcsolni.)

A MainProgram tartalmaz egy *_sdcc_external_startup (void)* függvényt, mely a processzor indulása után közvetlenül fut le, és kikapcsolja a WatchDogot. Ez azért szükséges, mert a sok __xdata területre definiált statikus változó foglalása olyan sok időt vesz igénybe a program elején, hogy a WatchDog előbb reseteli az MCUt, minthogy a program elérne a főprogramban az InitDevice()-ig.

A MainProgram ezen felül meghívja a perifériák inicializációs függvényeit, melyek alapállapotba konfigurálják az eszközöket, majd meghívja a StartMenu() függvényt, mely már felhasználói program, és helyes működés mellett sosem szabad, hogy visszatérjen.

4.2.2 JhnsCntr_Driver - a Reed mátrix kezelő modul

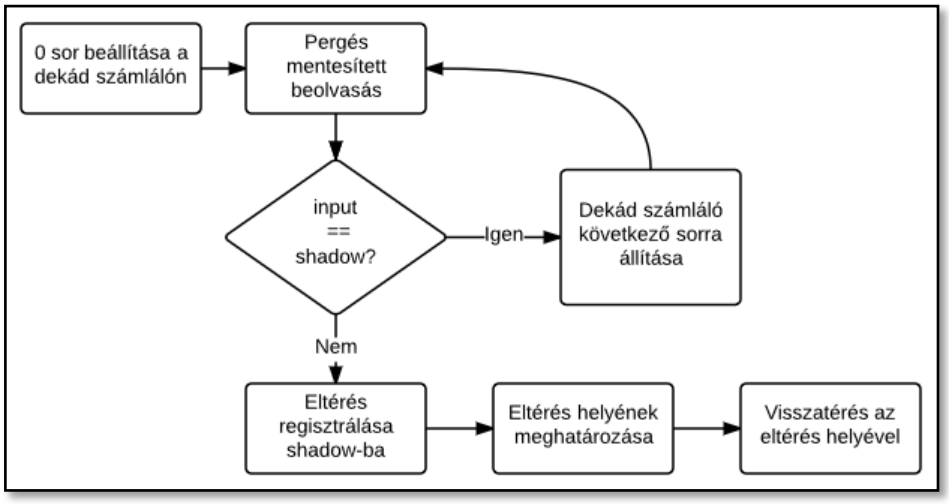
A JhnsCntr_Driver.cfájl felelős a Reed mátrix beolvasásáért, illetve a mátrixban keletkező változások detektálásáért. A JHNS_rst és step vonalakon keresztül közvetlenül irányítja a dekád számlálót, és P1 - JHNS_RM_in porton keresztül olvassa be az aktuális sor (játék szempontjából egyébként oszlop, vagy a gombok) állapotát.

Ezen felül a modul karban tart egy rm_shadow nevű, 8x8 bitmátrixot - a gyakorlatban 8 byte méretű tömböt - melyet változások detektálására referenciaként használ.

4.2.2.1 JHNS_isFieldChanged() és JHNS_getColState()

Az~isFieldChanged() függvény feladata, hogy meghívása esetén jelezze, ha történt, és ha igen, hol történt változás a beolvasó mátrixban. A függvényt csak akkor alkalmazom, amikor valamilyen jól meghatározott felhasználói inputot várok, például egy báb felemelését, elhelyezését, vagy egy gomb megnyomását.

A függvény működését a 4.2.2.1.a ábra szemlélteti:



4.2.2.1.a ábra - A JHNS_isFieldChanged függvény működése

A mellékelt forráskódban látható, hogy a függvény kiterjedt része foglalkozik azzal, hogy az eredményül kapott koordináták a char típusú visszatérési értékbe kényelmesen felhasználható módon formázva kerüljenek bele.

Ahhoz, hogy a függvény az elvárt módon működjön, szükséges, hogy a shadow megfelelően fel legyen töltve adatokkal (ez alól csak a gombok érzékelése kivétel, azok mindig a "0" állapothoz vannak hasonlítva). Erre dedikált függvény nincs implementálva, mivel a feladatot a ~getColState(...) függvény, működése közbeni járulékos haszonként végzi el, a főprogram működése pedig biztosítja, hogy ez a függvény mindig meg legyen hívva, mielőtt az ~isFieldChanged() használva lenne.

A ~getColState(...) függvény egyszerűen végiglépteti a dekád számlálót minden soron, frissíti a shadow mátrixot a beolvasott értékekkel, majd visszaadja azon sorát, melyet a paraméterben kértek tőle. Fontos megjegyezni, hogy a függvény hatékonysági okokból nem végez semmiféle biztonsági ellenőrzést a paraméterül kapott col indexen, és nem

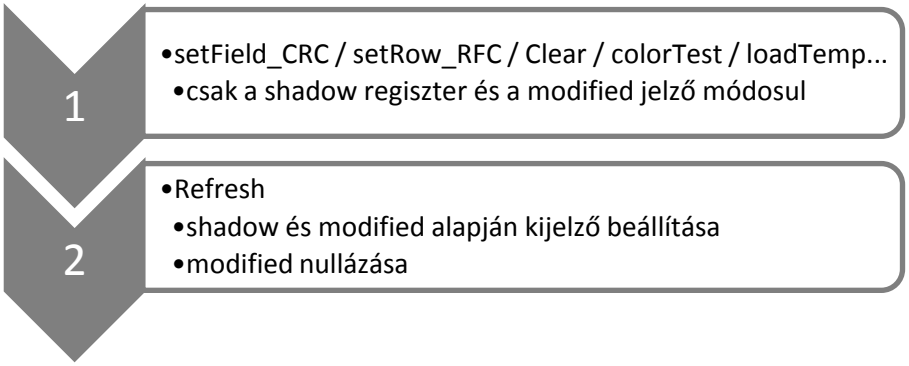
formázza a visszaadott értéket sem (pl. nem alakítja át a gombok sorindexét, mint az ~isFieldChanged() függvény).

4.2.3 7219_Driver

A MAX7219 LED meghajtók vezérléséért felelős modul az egyik legnagyobb, és legösszetettebb driver modul. Lehetővé teszi a három LED meghajtó IC együttes, kényelmes vezérlését, kezdve a konfigurációs beállítások kezelésétől, az egyes mezők, vagy teljes sorok színének beállításáig.

A modul bit-bang eljárással kommunikál a LED meghajtókkal, melynek implementációjához *Randy Rasa MAX7219.C moduljának MAX7219_SendByte(...)* függvényét vettem alapul (források között feltüntetve, [14]). Ezután elkészítettem a *MAX7219_Write(...)* függvényt, mely *Randy Rasa* azonos nevű függvényének [14] 3 LED meghajtó együttes vezérlésére alkalmas változata. A többi függvény már teljes egészében az IC adatlapja alapján, saját munkám eredményeként jött létre.

A modul a kimeneti állapot követése érdekében két dinamikus adathalmazt tart fenn. Az egyik egy 3x8x8 bitmátrix, gyakorlatban 3x8 char mátrix, mely itt is shadow néven szerepel. Ez a mátrix gyakorlatilag egy átmeneti tároló, mely tartalmazza a kijelző aktuális állapotát (8x8 led, 3 szín). A másik a modified változó, mely tartalmazza, mely sorok esetén van feltehetőleg eltérés az aktuálisan kijelzett, és a shadow mátrixban tárolt adatok között. Ha a ~setField_CRC(...), vagy a ~setRow_RFC(...) függvényekkel módosítjuk a kijelző valamely mezőjét vagy komplett sorát, a módosítás csak a shadow mátrixban kerül eltárolásra, és a modified változó regisztrálja, mely sor változott meg. Ha azt szeretnénk, hogy a módosított kép a kijelzőn is megjelenjen, a ~Refresh() függvényt kell meghívni, mely a shadow mátrix módosított tartalmát kiviszi a meghajtókra (4.2.3.a ábra).



4.2.3.a ábra - A 7219_Driver használatának lépései

Ezen kívül a modul támogatja az aktuális kép eltárolását és későbbi visszaírását a kijelzőre. Ennek a funkciónak akkor van szerepe, ha például valamilyen hibás konfiguráció jelenik meg a táblán. Ekkor az aktuálisan kijelzett képet eltárolhatjuk (~saveTemp()), és kijelezhetjük a hibás mezőket. Miután a konfigurációt a felhasználó helyreállította, a kép is visszaállítható (~loadTemp()).

A modul további hasznos funkciói a shadow mátrix teljes törlését (~Clear()), színes képpel való tesztelését (~colorTest()) teszik lehetővé, de bármely konfigurációs regiszter is elérhető. Van amelyik globálisan, és meghajtó specifikusan is rendelkezésre áll (pl. fényerő szabályzás), és van olyan is, mely csak globálisan, az összes meghajtót egyszerre szabályozva érhető el (pl. teszt üzemmódba, vagy kikapcsolt állapotba kapcsolás).

Végül, de nem utolsó sorban a modul felelős azokért az "animációkért", melyek a játék közben megjelennek. A ~fadeOut() és a ~fadeIn() függvények lehetővé teszik, hogy a kijelző fényerejét rövid idő lefolyása alatt fokozatosan változtassuk a maximális és a minimális fényerő között, ezáltal finom átmenetek, vagy akár villogó animációk is létrehozhatók.

4.2.4 LCD_Driver

Az LCD driver feladata az LCD kijelző kezelése, főleg a kurzor megfelelő helyre mozgatása, és a szövegek megjelenítése (ezen felül periféria inicializációt hajt még végre).

A driver a 7219_Driverhez hasonlóan bit-bang módszerrel kommunikál, kiegészítve a megfelelő késleltetésekkel. A bit-bang módszer az RS vonal megléte miatt szükséges, mely nem szabványos SPI adatvonal. Feladata, hogy megkülönböztesse, mikor akarunk vezérlő (pl. kurzormozgató), és mikor megjelenítendő szöveg adatokat küldeni.

A driver egy nagyméretű konstans definícióval kezdődik, mely tartalmaz minden olyan karakter sort, amelyet a tábla az LCD kijelzőn meg tud jeleníteni. Mikor a ~sendText(...) függvényt meghívjuk, paraméteréül annak a karaktersornak az indexét kell megadnunk, melyet ki akarunk írni.

Ezen felül van egy ~setCursor(...) nevű függvény is, melynek segítségével adott sor adott pozíciójára helyezhetjük a kurzort (mely egyébként a menüben való navigációnál fontos szerepet tölt be, mint arról a korábbiakban már volt szó).

Kényelmi szempontból fontos függvény a ~setPage(...), mely egy teljes oldal kijelzésére alkalmas, az első és a második sor szövege, illetve a kiíratás utáni kurzor pozíció is egyben meghatározható.

Végül, de nem utolsó sorban fontos még a ~sendTime(...) függvény is. Ez a függvény az aktuális kurzor pozícióból indulva ír ki egy időt, (xx)m(xx)s formátumban. Ennek a sakkóra kezelésénél, megjelenítésénél van nagy szerepe. A függvény 16 bites adatokkal dolgozik, paraméteréül a kiírandó időt másodpercben kell megadni.

4.2.5 Util

Ez a nyúlfarknyi modul kizárólag késleltetések generálását tartalmazza, a Delay_ms millisecundumban, a Delay_us pedig - egyértelmű módon - microsecundumban meghatározott ideig vár, maximum 255 adható meg paraméterül. A késleltetésnek sok helyen van fontos szerepe, például meghajtók kommunikációja során, vagy az animációk időzítésénél.

Maga a késletetés egy rövid ASM kód formájában, két, egymásba ágyazott cikluson alapszik. A 'ms' kód eredetije az *edaq24.c*fájlból származik [15] , ezt alakítottam át kissé a 'us' verzióhoz is.

4.3 Felhasználói modulok

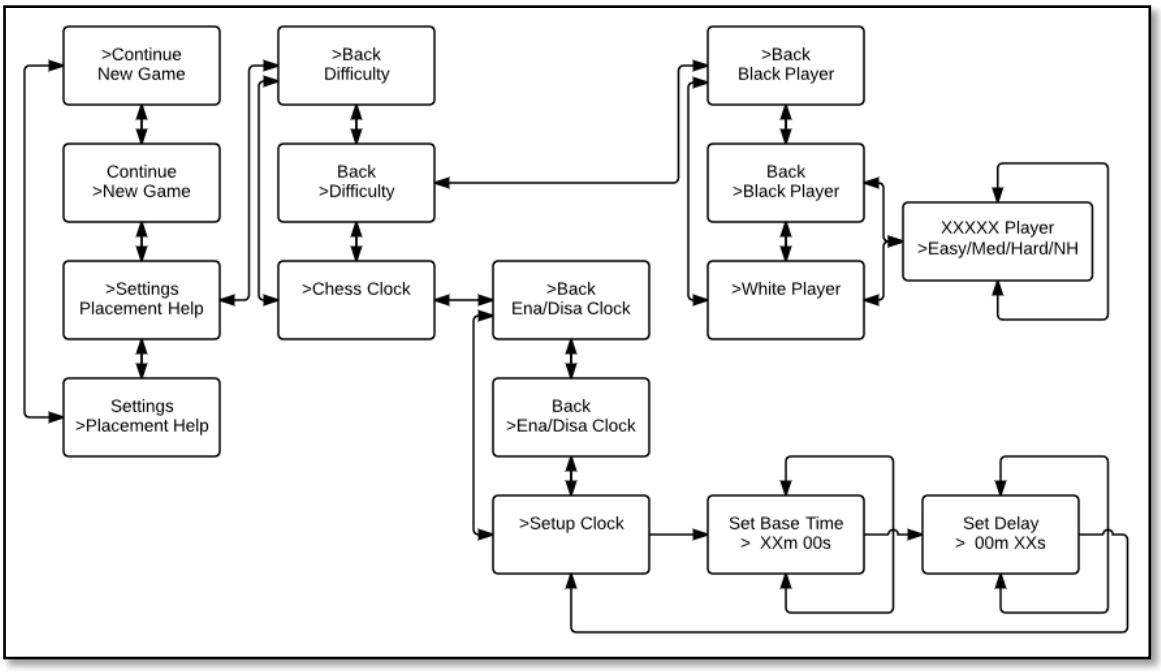
A felhasználói modulok közé a StartMenu, és a ChessGame sorolhatók. Ezek a modulok függőségeiket tekintve a korábban bemutatott driverekre épülnek, és a magasabb szintű feladatokat látják el. Méretüket tekintve sokkal nagyobbak, és összetettebbek a driver moduloknál (főleg a ChessGame).

4.3.1 Menu/StartMenu() függvény

A StartMenu() gyakorlatilag a főprogram. Amint a MainProgram befejezi a perifériák inicializációját, ez a függvény hívódik meg, mely aztán a felhasználóval a gombok, és az LCD kijelzőn keresztül tartja a kapcsolatot. A StartMenu() által nyújtott funkcionalitás a 2.2, 2.3 illetve 2.4 bekezdésekben már tárgyalásra került.

Szoftveres felépítését tekintve a lehető legegyszerűbb megoldásra törekedtem, hogy a menürendszer könnyen bővíthető, átalakítható legyen. "Szabvány" kódrészeket hoztam létre, melyek komplett menü ablakok megjelenítését, és viselkedését szabályozzák, és ezeket az építőelemeket használtam a teljes rendszer felépítéséhez.

A menü felépítése a 4.3.1.a ábrán látható. Az ábrán szereplő gráfban felfelé/lefelé a navigáló, oldalirányban az OK gomb segítségével lehet mozogni a menüpontok között. Az önmagukba visszatérő menük ún. "dinamikus" menük. (az ábrán a ">" jel a kurzor helyzete)



4.3.1.a ábra - A menürendszer felépítése

Az egyes menüpontok közötti mozgás programszinten a menüpontok kódjához rendelt címkék segítségével, goto utasítások által történik. Ez (megfelelő használat mellett) a lehető legnagyobb mozgásszabadságot nyújtja, bármilyen összetett menürendszer kialakítható. Egy tipikus menühöz tartozó kód a 4.3.1.b forráskód részletben látható:

A menüpont az LCD page beállításával (első/második kiírandó sor, majd a kurzor pozíciójának megadása) kezdődik.

```
M2_1: //Main menu, secondpage, Cursoronfirst line

      //Setpage
      LCD_setPage(2,3,0); //Settings
      Delay_ms(255);
      //Waitforbuttonto be pressed
      while(JHNS_col(button) != 0x08)
      {
            button = JHNS_isFieldChanged();
      }

      button = JHNS_row(button);

      switch (button) {
            case 0: //down
                  goto M2_2;
                  break;
            case 1: //ok
                  goto S1_1;
                  break;
            case 2: //up
                  goto M1_2;
                  break;
            break;
      }
```

4.3.1.b forráskód részlet
- Egy tipikus menüpont

Mint az látható, mivel a gombok érzékelése mindig 0-hoz viszonyított, a menüpontban található késleltetés (Delay_ms(255)) alapján egy gomb folyamatos nyomva tartása másodpercenként ~3-4 menüpont sebességgel történő vándorlást eredményez, amely tapasztalataim szerint éppen kényelmes.

Ezután a késleltetés után a menü addig várakozik, amíg valamely gombot le nem nyomták, majd a lenyomott gomb alapján lép tovább, vagy egy másik menüpontba, vagy hív valamilyen függvényt, például a PlacementHelp esetén.

A New Game, és a Continue menüpontok esetén az OK gomb lenyomása új játékot indít, mely a StartMenu(...) függvény utolsó címkéjéhez rendelt kódon keresztül történik meg. Ez azért fontos, mert maga a menü egy nagy while(1) ciklusban fut, és amennyiben egy játék megszakad (pl. menühívással), vagy bármilyen módon befejeződik (győzelem/döntetlen), a PlayChess(...) függvény a megszakítás okát jelző értékkel visszatér, a StartMenu() pedig az elejére ugorva újra indul a Continue/New Game menüponttal.

A játék megszakadásának oka alapján - amennyiben a játék nem menühívás miatt szakadt meg - beállítja a kezdő kijelző képet, és újra inicializálja a ChessGame.c dinamikus adattárolóit, beállítja a sakkórát, és új játéknak megfelelő konfigurációba állítja az elvárt tábla állást (if(!game_state){...} szakasz a StartMenu() elején).

Az ilyen, egyszerű menüpontoknál kissé bonyolultabbak az ún. dinamikus menüpontok, melyek valamilyen konfigurációs lehetőségek közül való választást tesznek lehetővé a felhasználó számára, például nehézségi fokozat beállítását, vagy a sakkóra alap/késleltetési idejének megadását.

Mint az a 4.3.1.c forráskód részletben látszik,az ilyen menüpontok egy rövid inicializációs szakasszal kezdődnek, mely csak a menüpontba való első belépéskor fut le. A menüpontnak van egy segédváltozója, mely a beállítás alatt lévő értéket tartalmazza. A felhasználó a fel/le gombok segítségével ennek a változónak az értékét manipulálja, majd a menü az LCD kijelző frissítése után egy belső visszatérési pontra, az inicializációs rész utánra tér vissza, és a szokásos késleltetés után vár egy újabb felhasználói inputot. Ha a felhasználó megnyomja az OK gombot, a segédváltozó értéke átkerül a megfelelő konfigurációs változóba, és a függvény visszatér valamilyen egyéb menüpontba (tipikusan oda, ahonnan jött).

```
//Chessclock, set Base Time.
// WARNING!Special menu point, entry here only!
CC_SB:
      //SetBase Time by minute incrememt/decrement
      //Default is 10m00s, min is 01m00s, max is 45m00s
      //Setpage
      LCD_setPage(24,18,1);
      y = 600;

CC_SB_1: //Return point for local use ONLY.

      LCD_sendTime(y);
      LCD_setLine(1,0);

      <<...Delay, waitforinput...>>

      switch (button) {
            case 0: //down - Decreasetime, and stayinmenu.
                  y-=60;
                  if(y<60) y = 60;
                  goto CC_SB_1;
                  break;
            case 1: //ok - Settime and go toDelaySettingpage
                  clock_config[0] = y;
                  goto CC_SD;
                  break;
            case 2: //up - Increasetime, and stayinmenu.
                  y+=60;
                  if(y>2700) y = 2700;
                  goto CC_SB_1;
                  break;
            break;
      }
```

4.3.2.1.a forráskód részlet
- 2. irány vizsgálatának kódja

4.3.2 ChessGame modul

Onnantól, hogy a New Game, vagy a Continue menüpont ki lett választva, a sakktáblát a ChessGame modul irányítja, egészen addig, amíg egy menühívás, vagy a játék vége miatt a Play_Chess(...) függvény vissza nem tér.Az itt lévő függvények felelősek azF1 mellékletben tárgyalt funkcionalitás megvalósításának döntő részéért.

Ahhoz, hogy a sakk játékot a tábla nyomon tudja követni, számos dinamikus adattárolóra van szüksége. A modul egy 8x8 byte mátrix formájában tartalmazza a tábla reprezentációját. Ez a 'state' változó, és ez az egyik legfontosabb dinamikus adattároló. A mátrix mezőiben tárolt értékek információval szolgálnak az adott mezőn álló báb színéről, típusáról, illetve minden báb egy, a saját oldalán belül egyedi azonosítóval is rendelkezik. Új játék indításakor ez a változó az 'init' konstans alapján kerül beállításra, mely a szabályos kezdőállást tartalmazza.

További fontos dinamikus adatok:

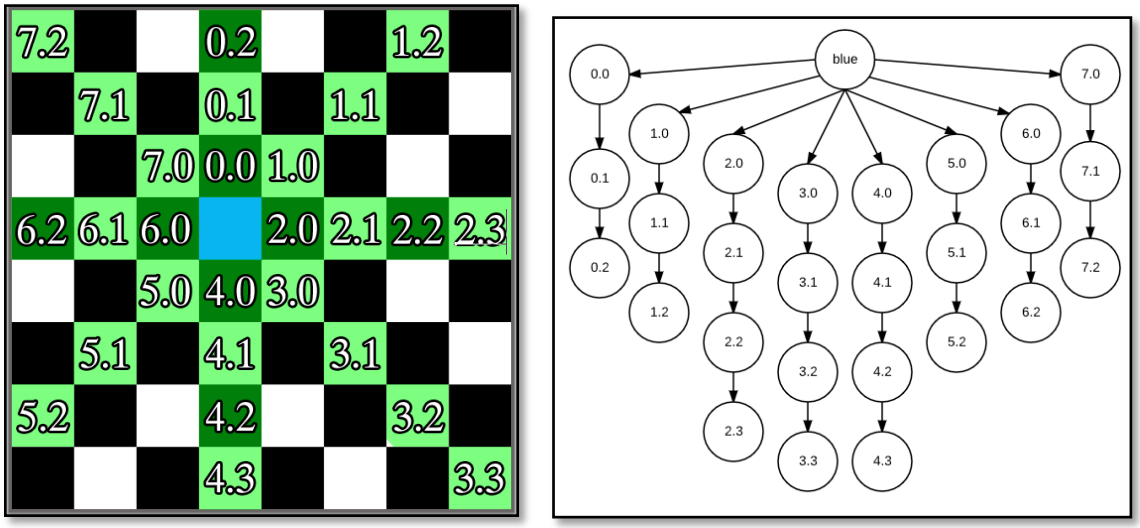
- **side**
  - Bit érték. Megmutatja, hogy mely oldal lépése következik.
- **valid_steps**
  - 16x8x8 bitmátrix, a gyakorlatban 16x8 byte, mely az aktuális oldal (max) 16 bábjára tartalmazza, melyek azok a mezők, ahova az adott báb léphet (egy-egy tábla reprezentáció bábonként). A 8x8 bitmátrixok és a bábok egymáshoz rendelése a bábok azonosítója alapján történik. A mapPossibleSteps() függvény tölti fel, és a clearDynamic() tisztítja ki minden lépésváltás után.
- **allowed**
  - 8x8 bitmátrix, a gyakorlatban 8 byte méretű tömb, (tábla reprezentáció), mely tartalmazza, mely mezőkön áll olyan báb, melynek van érvényes lépése. Ugyancsak a mapPossibleSteps() tölti fel, és a clearDynamic() tisztítja ki.
- **attackable és risky**
  - 8 byte méretű tömbök, ugyancsak 8x8 (tábla reprezentáló) bitmátrixok. Előbbi azon mezőket tartalmazza, melyek közvetlen veszélyben álló bábokat tartalmaznak, míg az utóbbi olyan mezőket, melyekre veszélyes lépni. Előbbit a find_Attackable(), utóbbit a find_Risky(...) függvények töltik fel. a find_Risky(...) paraméteréül egy konkrét báb azonosítóját kell megadni, tehát csak akkor lehet használni, ha már tudjuk, mely bábot emelte fel a játékos. Ezeket a mátrixokat is a clearDynamic() tisztítja ki.

A teljes modul részletezése túlságosan hosszú és - valljuk be - unalmas lenne, ezért az érdekesebb, kulcsfontosságú függvények bemutatásával folytatom. A következőkben bemutatásra kerül a senseCheck(...) függvény, mely eredetileg annak felderítésére szolgált, hogy a királyunk sakkban van-e, ám végül széles körben alkalmazásra került, beleértve a find_Risky(...), és a find_Attackable() függvények megvalósítását is. Ezután a senseCheck(...) "nagytestvérét", a mapPossibleSteps(...) függvényt fogom bemutatni, mely az egész program lelke, ennek segítségével vannak felderítve az érvényes lépések. Fontos funkció még a forceRight() is, mely kijelzi, ha a tábla fizikai konfigurációja eltér a 'state'-ben tárolt állapottól, és kikényszeríti, hogy a felhasználó a bábokat a megfelelő mezőkre helyezze. Rövid bemutatásra kerül a sakkóra funkció is, végül, a Play_Chess(...) függvény részletezésére kerül sor, mely a játékmenetet vezérli, és egyesíti a korábban vázolt függvények képességeit.

4.3.2.1 A senseCheck() függvény

A senseCheck(...) függvény a 'state' változó segítségével képes a tábla bármely mezőjéről eldönteni, hogy az adott mező támadás alatt áll-e vagy sem, sőt, egy kis hack alkalmazásával az is kideríthető segítségével, hogy mely mezők vannak védve valamely bábunk által, és melyek nem.

A függvény alapjában véve a szélességi keresés egy speciálismegvalósítása. A mező, melyre a függvényt meghívjuk, a fa gyökere, és minden, körülötte lévő mező a fa egy-egy ága lesz. Mivel a bábok mozgási irányai (a huszártól most eltekintünk) egyenes vonalak, így ezek az ágak mind egy-egy lánc lesz (4.3.2.1.a ábra).



4.3.2.1.a ábra - A senseCheck() által vizsgált state állapot, és a belőle képezhető gráf

Az algoritmus az egyes ágakat a 'rdy' byte megfelelő bitjeivel azonosítja. Amennyiben valamelyik irányhoz tartozó bit 0, az irány vizsgálandó, egyébként már lezárásra került valamilyen (később ismertetett) okból, és az algoritmus átugorja. A szélességi keresés során aztán a teljes függvényt átölelő for ciklus által lépésenként növelt i változó szerinti mélységben az ágakat a 4.3.2.1.a ábra alapján, 0->7 irányba haladva, az óramutató járásával megegyezően vizsgálja körbe. (Tehát először az x.0, majd az x.1 mezők következnek, és így tovább, ahol x 0->7 halad) Ha valamely irányban az algoritmus eléri a pálya szélét, akkor az irányt, és vele együtt az egész "égtájat" lezárja. Például abban az esetben, ha a 0 irányban elérjük a nyolcadik sort (kilépünk a tábla felső szélén), a 7,0,1 irányokat zárjuk le, és az algoritmus ezeket az irányokat többé nem vizsgálja.

Minden irányhoz tartozik egy vizsgálati kódrészlet (pl. amilyen a 4.3.2.1.b forráskód részletben szerepel), melybe a program csak akkor lép be, ha az adott irány még nyitva van.

```
if (!(rdy& 0x04))
{
    x = state[col+i][row];
    if(x){
          if ((x & 0x01) == side)
          {
                rdy |= 0x04;
          }
          else
          {
                if ((GET_TYPE(x) == 2)||(GET_TYPE(x) == 5)
                      || ((i == 1) && GET_TYPE(x) == 6))
                {
                      return 1;
                }
                else
                {
                      rdy |= 0x04;
                }
          }
    }
}
```

4.3.2.1.b forráskód részlet - 2. irány vizsgálatának kódja

Az irányt vizsgáló kódba lépve egy átmeneti tárolóba elmentjük a vizsgált mező tartalmát. Csak akkor vizsgálódunk tovább, ha a mező nem üres, egyébként lépünk a következő irányra. Ha a vizsgált mezőn a saját oldalunkon álló báb van, az irányt lezárjuk (hiszen az nem jelenthet veszélyt, és véd az adott irányból jövő támadástól). Egyébként megvizsgáljuk, hogy a vizsgált, immár ismerten ellenséges báb veszélyt jelent-e. A bemutatott kódrészletben, amely az egyik vízszintes irányt vizsgálja, ez az ellenőrzés egyszerű, ám kereszt irányoknál a gyalogok aszimmetrikus viselkedése már kissé bonyolultabbá teszi. Ha a talált báb veszélyt jelent, 1-el térünk vissza, ha nem, az irányt lezárjuk, hisz hiába ellenséges a talált báb, ha ártani nem árthat, az irányt pedig ugyancsak takarja a támadásoktól.

Természetesen a fent bemutatott szélességi keresésen túl még egy feladat hátra van, meg kell vizsgálni, nincs-e olyan huszár, mely veszélyt jelenthet. Ennek megkönnyítésére a modul tartalmazza azon mezők relatív helyzetét, ahonnan huszár támadhat. Ezeket a mezőket a 4.3.2.1.a forráskód részletben bemutatotthoz hasonló módon vizsgáljuk végig.

A kód ismeretében már könnyű felismerni, hogyan használható a függvény védett mezők keresésére is (átmenetileg egyszerűen megfordítjuk a 'side' változót).

4.3.2.2 A mapPossibleSteps(...) függvény

A mapPossibleSteps(...) függvény (mint azt neve is elárulja) felelős az érvényes lépések feltérképezéséért, illetve az érvényes lépéssel rendelkező bábok kijelöléséért. Felépítését tekintve erősen emlékeztet a senseCheck(...) algoritmusára, hiszen itt is egy nagyon hasonló szélességi keresést valósítunk meg.

A függvényt a 'state' bármely mezőjére meghívhatjuk, azonban csak akkor fut le, ha a mezőn egy, az éppen lépés előtt álló félhez tartozó báb áll. Működésének első néhány lépése, (ha a mező az előbbi feltételnek eleget tesz), hogy **a bábot, melynek a lépéseit fel szeretnénk térképezni, egy átmeneti tárolóba, x-be** eltárolja, helyét pedig a későbbi vizsgálatok helyes működésének érdekében kinullázza **(mintha x-et felemeltük volna)**.

Itt is van egy 'rdy' változó, melynek feladata azonos a senseCheck() függvényben betöltött szerepével, tehát az irányok lezárását adminisztrálja. Az inicializációs szakaszban a vizsgált mezőn álló báb típusa alapján lezárjuk azokat az irányokat, amelyekbe a báb biztosan nem léphet (például a diagonális irányokat bástya esetén).

Ezután kezdődik az a for ciklus, mely a senseCheck()-ben is volt, az i változó itt is a keresési mélységet adja, és itt is lezárjuk az égtájakat, ha a tábla széléhez érünk.

Az első igazán nagy különbség az egyes irányok vizsgálatának kódjában található (4.3.2.2.a forráskód részlet).

Az irány vizsgáló kód első lépéseként y-ba kimentjük a vizsgált mező tartalmát, majd annak alapján eldöntjük, lezárandó-e az irány. Ilyen lezáró ok lehet, ha a mezőn egy szövetséges báb áll, vagy egy király (ami ugye nem üthető le), de egyéb ok is lehet, például a kiemelt kódrészletben `(!y&& (type == 1))`akkor zárja le az irányt, ha x gyalog, de keresztirányban nincs előtte támadható ellenséges báb.

Ha az irányt nem zártuk le, akkor a vizsgált mezőt helyettesítjük x-el, majd megvizsgáljuk, hogy az így átalakított state-ben x királya sakkba került-e(a király koordinátáira meghívjuk a senseCheck(...) függvényt) **- gyakorlatilag lépést szimulálunk. Ha a király sakkba került, a lépés a sakk szabályai szerint érvénytelen, ha nem, a lehetséges lépést elmentjük az x-hez tartozó valid_steps mátrix megfelelő mezőjébe, és az allowed mátrixban is regisztráljuk, hogy x-nek van érvényes lépési lehetősége.** (Megj.: Ez a módszer abban az esetben is tökéletesen működik, ha éppen sakkhelyzet van, hiszen csak azoknak a bábóknak lesz lépési lehetősége, melyek megszüntethetik a sakkot, míg a matt/patthelyzet felismerése is hasonlóan egyszerű, hisz az allowed mátrix üres marad a mapPossibleSteps() lefutása után is.)

Ezután y-t visszatesszük a helyére. Az utolsó fontos momentum, hogy az irányt még ekkor is lezárhatjuk, történetesen abban az esetben, ha a fenti mező vizsgálatakor y nem üres mező volt, hanem egy ellenséges báb. Ekkor ugyanis egy ütést "szimuláltunk le", tehát ebben az irányban x már nem léphet távolabbra a vizsgált irány mentén.

Az alább bemutatott kód ezen felül néha speciális esetek kezelésével is kiegészül, ilyen például a gyalogok kezdő lépésének kezelése, mikor kettőt is léphetnek. Ezen felül általánosan, ha gyalogot, vagy királyt vizsgálunk, melyek nem léphetnek korlátlan távolságra, a szélességi keresés az 1 mélységű mezők vizsgálata után befejeződik.

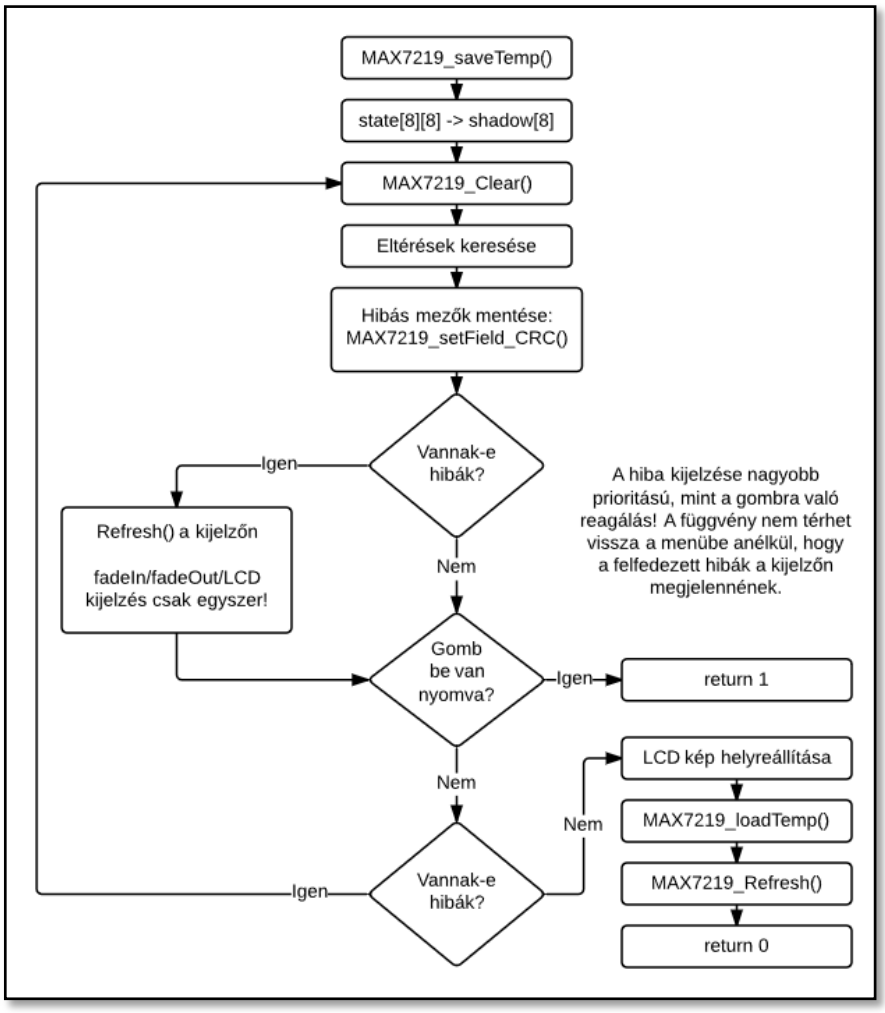
Természetesen itt sem fejeződik be a függvény a szélességi kereséssel, hiszen hátra van még a huszárok lépéslehetőségeinek vizsgálata, és a speciális lépési lehetőségek, például a sáncolás vizsgálata. Ez utóbbi egy speciális változó alapján történik, mely a játék közben regisztrálja, mely játékosnak mely sáncolási lehetőségei érhetőek még el, és ennek alapján végzi el az ellenőrzést. Ezen funkciók részleteit - főleg hely hiányában - már nem szeretném kifejteni.

```
if (!(rdy& 0x02))
{
    y = state[col+i][row+i];
    if((y && (IS_WHITE(y) == side)) || (!y && (type == 1))
           || (GET_TYPE(y) == 6) )
    {
          rdy |= 0x02
    }
    else
    {
          //replace dest. field with x
          state[col+i][row+i] = x;
          //if x is the king, refresh its position catche
          if(GET_TYPE(x) == 6) find_King();

          //check if the king got in check
          if(!senseCheck(k_col,k_row))
          {
                //If the step is valid, its saved in valid_steps
                valid_steps[id][row+i] |= (1<<(7-(col+i)));

                //and the piece becomes available to move
                allowed[row] |= (1 << (7-col));
          }
          //dest field gets restored
          state[col+i][row+i] = y;
          //if the step is a capture, the direction gets closed
          //En-passant dummies doesn't affect moves
          if(y && (GET_TYPE(y) != 7))
                rdy |= 0x02;
    }
}
```

4.3.2.2.a forráskód részlet
- 1. irány vizsgálatának kódja

4.3.2.2 A forceRight() függvény

A forceRight() függvény a sakkjáték során számos alkalommal van meghívva, hiszen ez biztosítja, hogy a táblán a bábok a 'state' mátrixban tároltnak megfelelő pozícióban álljanak. Ezen felül, mint az már korábban is említésre került (4.2.2.1 bekezdés), az isFieldChanged() használata előtt mindig frissíteni kell a Jhns_Cntr modul 'shadow' mátrixát a getColState() segítségével. A forceRight() ezutóbbi függvényt használja a tábla ellenőrzésére, és teszi egyúttal alkalmassá a Jhns_Cntr modult arra, hogy az később a felhasználói inputokat helyesen kezelhesse. A függvény működése a 4.3.2.2.a ábrán látható.



4.3.2.2.a ábra - A forceRight() függvény működése

Nem meglepő módon a forceRight() függvényben is van egy 'shadow' 8x8 tábla reprezentáló bitmátrix. Ez gyakorlatilag egy kivonat, mely a 'state' mátrixból keletkezik, és megmutatja, mely mezőkön kell bábunak állni, és melyiken nem. A függvény ezután sorra veszi a tábla sorait, XOR művelettel képzi a beolvasott fizikai állapot, és a shadow különbségét, majd kijelzi, ha eltérést talál.

A függvény mindaddig fut, amíg bármilyen "hiba", az elvárt konfigurációtól való eltérés található a táblán. Az egyetlen, ami megszakíthatja, a menü valamely gombjának megnyomása. Ha a függvény futása közben újabb eltérések jelennek meg a táblán, ezek is kijelzésre kerülnek.

A hibajelzés kétirányú. Ha a függvény hibát észlel, először az LCD kijelzőn megjelenik a "PlacementError" hibaüzenet, majd egy rövid animáció kíséretében a kijelzőn pirossal kiemelésre kerülnek a hibás mezők. Azért, hogy a kijelző és az LCD feletti "hatalomátvétel" a Play_Chess() számára transzparens legyen, a függvény meghívásának pillanatában a saveTemp() segítségével lementi a hibajelzés előtti képernyőképet, majd befejeződésekor helyreállítja azt a loadTemp() segítségével, és az LCD kijelzőn is visszaállítja a meghívása előtti képet (save/loadTemp() - 4.2.3 bekezdés, 43. o).

4.3.2.3 A sakkóra működése

Ez egy viszonylag kisméretű almodul. Működésének alapja a Timer0 túlcsordulásos megszakítása. Két fontos adattároló egység tartozik hozzá a ChessGame modulban. Egyik a 'cclock[3]' tömb, mely a játékosok hátralevő idejét tartalmazza másodpercben, illetve a Delayt, azaz azt a késleltetést/növekményt, melyet a játékos időtöbbletként megkap.

Ezek az adatok a Menu-ben el vannak tárolva, játék közben is módosíthatók, azonban a ChessGame csak az initGame() függvényen keresztül kaphatja meg őket, új játék indításakor. (Ez azt jelenti, hogy bármely módosítás a sakkóra beállításain csak új játékkal érvényesíthető, és a sakkóra mindig megtartja állapotát, amíg ez az újra inicializáció meg nem történik, függetlenül attól, hogy játék közben hányszor engedélyezzük/tiltjuk az óra működését.)

A másik fontos szabályzó/adattároló a Play_Chess(...) paraméteréül kapott 'clock_enabled' érték, mely meghatározza, hogy az óra működjön-e játék közben, vagy sem.

A Timer úgy van felkonfigurálva, hogy körülbelül másodpercenként 8 megszakítást okozzon (25Mhz / 48 / 2^16 = 7,95Hz). A megszakítás kezelő rutin kódja a 4.3.2.3.a forráskód részletben látható. A kód egy számláló (divider) segítségével a 8Hz megszakítási frekvenciából ~1Hz - et hoz létre, és az aktuális játékos hátralevő ideje ezzel a frekvenciával csökken.

A sakkóra csak akkor fut (a megszakítás csak akkor van engedélyezve), ha a játékosnak épp gondolkodási ideje van, tehát akkor, amikor a tábla felhasználói interakciót vár egy báb felemelése, vagy lépés megtétele során, egyéb esetben, például a forceRight() működése, vagy a menü használata közben nem.

```
voidchessClock_IRQ_Handler(void) __interrupt 1
{
    staticunsignedchardivider = 0;

    divider++;
    if(divider == 8)
    {
          divider = 0;

          if(cclock[side]>0)
                cclock[side]--;

          LCD_setLine(0,0);
          LCD_sendText(6+side);
          LCD_setLine(1,0);
          LCD_sendTime(cclock[side]);
          LCD_setLine(1,0);
    }
}
```

4.3.2.3.a forráskód részlet
- A sakkóra megszakítási rutinja

4.3.2.5 Egyéb fontos segédfüggvények

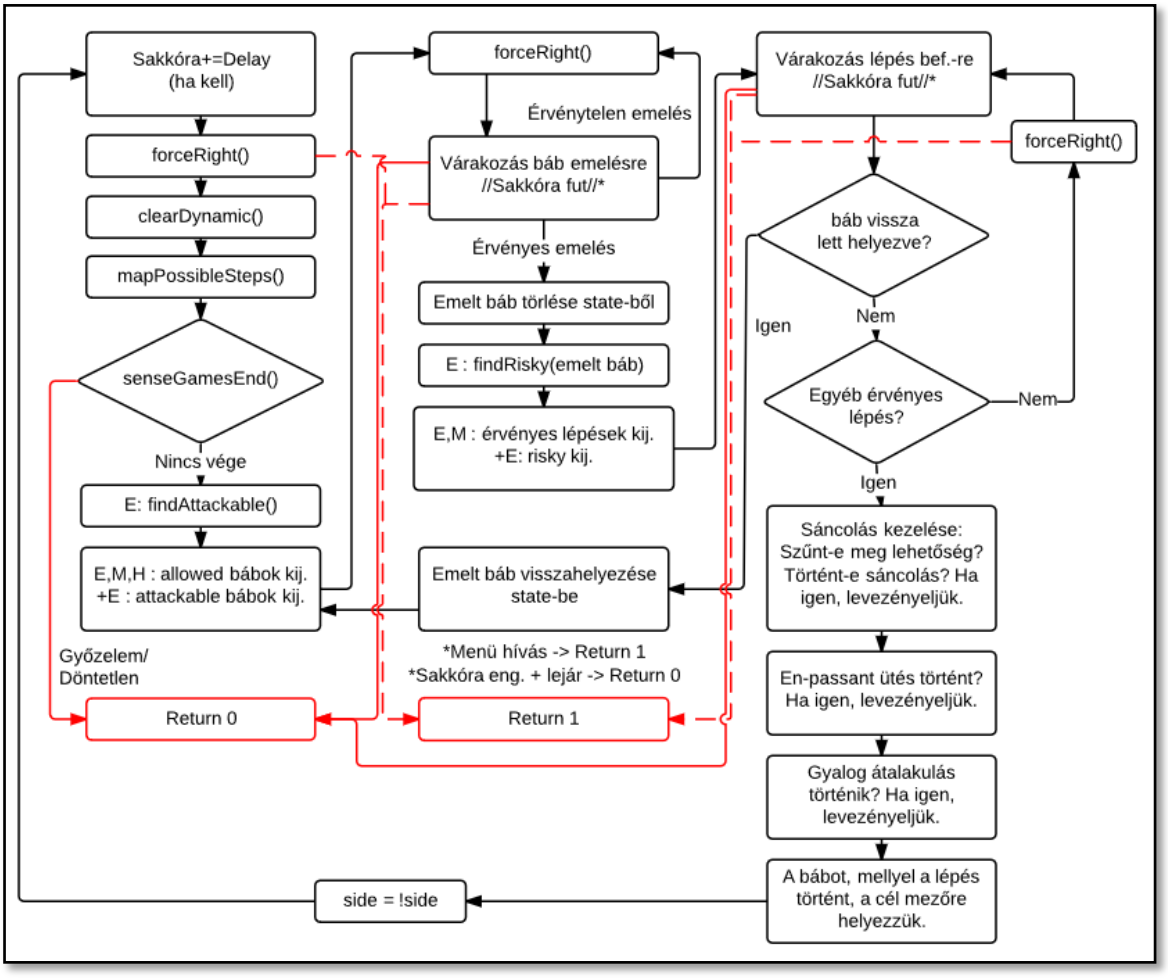
Egyéb, fontos függvények még:

- **clearDynamic(),** mely a dinamikus adatokat törli lépésváltások esetén (pl. a 'valid_steps', 'allowed', 'attackable', 'risky');
- **gameInit(),** mely új játék esetén inicializálja az adatokat;
- **displayWin(...)** mely a paraméterétől függően jelzi ki a győzelmi, vagy a döntetlenhez tartozó animációt, LCD képet;
- **senseGamesEnd(),** mely a state, és az allowed mátrixok alapján képes detektálni, ha az F1.6.3.1 melléklet bekezdésben vázolt döntetlen feltételei teljesülnek, vagy az egyik fél mattot, esetleg pattot kapott, majd ennek megfelelően hívja meg a displayWin(...) függvényt;
- **findRisky(...), findAttackable(),** melyek a veszélyben lévő bábokat, ill. a veszélyes lépéselehetőségeket térképezik fel;
- **placementHelp(),** mely a bábok helyes elhelyezésében segít.

Ezen függvények tárgyalása hely hiányában elmarad, de működésük véleményem szerint nem is olyan összetett, hogy érdemes lenne őket részletesen kifejteni.

4.3.2.5 A Play_Chess(..) függvény működése

A Play_Chess(...) függvény a StartMenu(...) függvényhez hasonlóan, moduljának fő funkciója, mely a korábban bemutatott függvényekre épülve vezényli le a sakk játékot. A függvény fő tevékenységeit a 4.3.2.4.a ábra vázolja fel. Az ábrán a fekete nyilak a program normál működését jelentik, míg a vörös nyilak visszatérési folyamatra mutatnak. A szaggatott vörös vonal menü hívást, míg a folytonos vörös vonal szabályos játék befejeződést jelent (mely győzelemmel/döntetlennel/sakkóra lejártával következhet be.) Ha egy folyamat előtt nagybetűs jelzők szerepelnek (pl. "E : ", vagy "E,M,H : "), azok nehézségi fokozatokra utalnak, tehát a folyamat akkor és csakis akkor játszódik le, ha a játékoshoz tartozó nehézségi fokozat E, azaz Easy, M, azaz Medium, vagy H, azaz Hard, vagy ezek valamilyen kombinációja. Rövidítések: eng. = engedélyezve, kij. = kijelzése, bef. = befejezés. A program a bal felső sarokban a "Sakkóra+=Delay" lépéssel indul.



4.3.2.4.a ábra - A Play_Chess(...) függvény folyamatai

Mivel az ábra véleményem szerint jól követhető, és pontjainak nagy része már korábban tárgyalásra került, ezért igyekszem azon pontok kifejtésére szorítkozni, melyekről eddig nem, vagy csak nagyon kevés szó esett (sáncolás, en-passant, átalakulás).

Sáncolás kezelése

Sáncolás folyamata - függelék F1.5.8.1.A sáncolás kezelése két részre bontható. Egyfelől a Play_Chess függvény maga egy globálisváltozóban tartja karban, mely sáncolási lehetőségek érhetőek még el. A játék elején mindkét játékos sáncolhat mind királynő, mind király oldalon (ha a feltételek adottak). Amennyiben a játékos valamely bástyájával lép, azon az oldalon már nem sáncolhat, hasonlóképpen, ha királyával lép, minden sáncolási lehetősége megszűnik.

A mapPossibleSteps() az előbb említett változó figyelembe vételével a valid_steps mátrixban feltünteti a sáncolási lépéseket is, mint érvényes lépéseket. Ha a játékos úgy dönt, ezen mezők valamelyikére lép királyával, a lépés folyamodványaképp a Play_Chess(...) függvény áthelyezi a bástyát a király túloldalára, és kinullázza a játékos sáncolási lehetőségeit.

En-Passant lépés kezelése

En-Passant folyamata - függelék F1.5.8.2.Ha a játékos gyalogjával a kezdő sorból kettőt lép előre, a Play_Chess(...) függvény a gyalog mögé egy ún. En-Passant bábot, dummy-t helyez. A báb csak az ellenség következő lépésének idejére létezik, utána kitörlődik. Az ellenség bábjai közül kizárólag a gyalogok érzékelik ezen en-passant dummy létezését, és képesek leütni is. Ha ez megtörténik, a Play_Chess a dummy-hoz tartozó gyalogot is kitörli a 'state' mátrixból.

Gyalog-átalakulás/Promoting

Gyalog-átalakulás folyamata - függelék F1.5.8.3.Ha egy gyalog eléri az ellenfél alapsorát, a kijelzőn megjelenik, mely bábra változtathatja meg a játékos a gyalogját. A játék addig nem folytatódhat, amíg a választás meg nem történik. (Az LCD kezelését ekkor a Play_Chess(...) függvény, és nem a Menu modul végzi.) Ha megtörtént a választás, a gyalog típusát azonosító bitjei megváltoznak a kiválasztott típusra, és a játék folyhat tovább. A báb természetesen kicserélhető, hogy formájában is új típusát tükrözze, de a tábla számára ez nem fontos.

5 Összefoglalás

Röviden összefoglalva a következő eredményeket értem el:

- Az egyedileg módosított bábok pozíciójának követése reed mátrix segítségével.
- Látványos, interaktív visszajelzés a játékos számára, RGB LEDek segítségével. A LEDek bevilágítják a mezőket, illetve az üveg sakkfigurákat.
- LCD kijelzőn megjelenített menü, illetve visszajelzés játék közben.
- Konfigurálható Fischer típusú sakkóra.
- A sakk szabályainak betartása, rugalmasan konfigurálható segítségnyújtás a lehetséges lépések, a biztonságos, vagy kockázatos lépések kijelzésével.
- Hibás lépések megakadályozása.
- Segítségnyújtás a bábok elhelyezésében a játék teljes folyamán.
- Játék végének felismerése - sakk, matt, patt, sakkóra lejárta esetén, vagy ha egyik félnek sincs matt adó ereje.
- Hardveres alapok USB kapcsolat kialakításához, de a felépítés modularitásából adódóan a későbbiekben új, nagyobb teljesítményű vezérlő panel is kialakítható.

A működésről szóló részletes leírást és használati utasítást, mint az már korábban is említve volt, a melléklet F1 fejezete tartalmazza. Az eredetileg még tervben lévő, ám a megvalósításig el nem jutott funkciókról egy rövid összesítés található a melléklet F2 fejezetében. Mind a forráskód, mind a tervezés során elkészült dokumentumok megtalálhatók a CD mellékleten.

5.1 Jövőbeli továbbfejlesztési lehetőségek

Úgy gondolom, MSc szakdolgozatom keretében, vagy egyetemen kívüli tevékenység során érdemes lenne még a projekttel foglalkozni, továbbfejleszteni.

Elsősorban a kimaradt funkcionalitás megvalósítása lehet cél, például a vezeték nélküli működés, az USB (vagy akár Bluetooth) kommunikáció biztosítása, de nagyobb fejlesztések is elképzelhetőek, például nagyobb intelligencia beépítése. Raspberry PI, ARIETTA G-25, vagy egyéb, viszonylag nagy számítási kapacitású és memóriájú modul segítségével akár mesterséges intelligencia is adható a táblának, lehetséges lenne a számítógép ellen, vagy akár interneten keresztül mások ellen is játszani.

További érdekességként az eszköz speciális sakk szabálykészletek alapján való működésre, vagy egyéb táblajátékokra is használható lehetne. Igazából a design továbbvitelének csak a pénz, és a képzelet szabhat határt.

Irodalomjegyzék

Felhasznált irodalom

[1] Wikipedia - Chess ill. Wikipédia - Sakk szócikkek
elérhető: http://en.wikipedia.org/wiki/Chess és http://hu.wikipedia.org/wiki/Sakk
(megtekintve: 2014.11.15.)
[2] ChessHouse.com - ElectronicChessComputers
elérhető: http://www.chesshouse.com/electronic_chess_s/5.htm
(megtekintve: 2014.11.15.)
[3] Purdue ECE Senior Design - LED ChessBoardprezentáció
elérhető: http://www.youtube.com/watch?v=2Q8ZjUIQhE4
[4] A sakkjáték szabályzata
elérhető: http://chess.hu/blog/wp-content/uploads/2014/06/LAWS-OF-CHESS-2014.pdf
[5]ASMT-YTB7-0AA02 LED Datasheet
elérhető: http://www.farnell.com/datasheets/1772736.pdf
[6] CD54HC4017, CD74HC4017 Johnson CounterDatasheet
elérhető: http://www.ret.hu/DataSheets/78_SMD_CTTL/TI_006/cd74hc4017.pdf
[7] MAX7219/MAX7221 Datasheet
elérhető: http://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf
[8] DOG SERIES 3.3VLCD ModuleDatasheet
elérhető: http://www.lcd-module.com/eng/pdf/doma/dog-me.pdf
[9] FT232R USB UART ICDatasheet
elérhető: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
[10] C8051F410/1/2/3MCU Datasheet
elérhető: http://www.keil.com/dd/docs/datashts/silabs/c8051f41x.pdf
[11] Aleena Emmanuel, ArunPrabhakar, Arvind Shankar, Raji James:
RGB LED Project
elérhető: http://rgb.kitiyo.com/pics/7219_64LED_ckt.png
[13] Randy Rasa:
MAX7219 LED Driver (SourceCode)
elérhető: http://ee.cleversoul.com/max7219-source.html
[14] SDCC CompilerUserGuide
elérhető: http://sdcc.sourceforge.net/doc/sdccman.pdf

"Beszállítók" - cégek, vállalkozások

[12.1] ROBTRON ELEKTRONIK TRADE KFT.
honlap: http://www.ret.hu/Page.aspx
[12.2] FDH Kft.
honlap:http://fdh.hu/
[12.3]MikroPAN KFT.
honlap:http://www.mikropan.hu/
[12.4] OptinKft**.**
honlap:http://optin.hu/
[12.5] Selmeczky és Társa Kft.
honlap: http://www.selmeczky.hu/
[12.6] Perfect Profil Kft.
honlap: http://www.perfectprofil.hu/
[12.7] Régió Játékkereskedelmi Kft.
honlap: http://www.regiojatek.hu/
[12.8] Szűcs Zoltán, Magnet Planet webáruház
honlap: http://magnetplanet.eu/

Nyilatkozat

Alulírott Mérnök Infomatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Tanszékcsoport Műszaki Informatika Tanszékén készítettem, Mérnök Informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Tanszékcsoport könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum

Aláírás

Köszönetnyilvánítás

Szeretném megköszönni témavezetőm, Mingesz Róbert, és Mellár János segítségét, főleg a hardveres elemek kiválasztása, a nyomtatott áramkörök tervezése, ellenőrzése, hibaelhárítása során, szakdolgozatom elkészítése alatt bármikor bizalommal fordulhattam hozzájuk.

Köszönettel tartozom szakmai gyakorlatom konzulensének, Kovács Tamásnak, aki sokat segített a forrasztás rejtelmeinek megismerésében, helyet, és jó minőségű eszközöket/anyagokat biztosított a hardveres megvalósítás során.

Ezúton szeretném megköszönni Szekretár Kingának, hogy a LED/Reed panelek légvezetékeinek elkészítésében segítségemre volt.

Ugyancsak köszönettel tartozom Szépe Tamásnak, aki hasznos ötletekkel és tanácsokkal látott el a ház/műszerdoboz megtervezése során, és különösen nagy köszönettel tartozom Rabi Zsoltnak, aki segítséget nyújtott a tervek finomhangolásában, és a tanszéki CNC laborban terveim alapján elkészítette a műszerdobozt.

Végül, de nem utolsó sorban szeretném megköszönni szüleimnek, hogy anyagi és lelki támogatást adtak a projekt megvalósításához.

Függelék

F1 A sakk szabályai, megvalósított funkciók részletes ismertetése

Avagy használati utasítás a sakktábla használatához

F1.1 Előszó - jelölésmagyarázat, források

A következőkben ismertetni kívánom a sakk szabályait, melyek forrásául a Wikipédia megfelelő, forrásmegjelöléssel ellátott magyar és angol szócikkeit, illetve a Chess.hu hivatalos szabályzatát használtam. Ezek a kivonatok dőlt betűvel szerepelnek, és általában vízszintes vonal választja el őket a szöveg többi részétől. Bizonyos bekezdések után ismertetem, hogy az általam megvalósított interaktív sakktábla hogyan jelzi / kényszeríti ki az adott szabály betartását. Néhány, leginkább versenysakkban alkalmazott szabály esetén, (mivel az eszköz alapvetően nem ilyen célokat szolgál), a tábla nem működik teljesen összhangban a szabályokkal, ezeket az eseteket külön kiemelem.

F1.2 A menürendszer általános használata

A szabályok megismerése előtt célszerű megismerni, hogyan működik a megvalósított menü rendszer. A menü a tábla áram alá kapcsolásakor, illetve egy játék befejeződése után automatikusan, egyéb esetben a felhasználói interfészhez tartozó gombok bármelyikének megnyomásakor jelenik meg az LCD kijelzőn. A kiválasztott menüpontot mindig a villogó kurzor jelzi. A menüben a felhasználói interfészen található felfelé/lefelé mutató gombokkal lehetséges navigálni, vagy az éppen beállítás alatt álló értéket módosítani, míg a középső, 'OK' gombbal lehetséges a menüpontba belépni, vagy az éppen beállítás alatt lévő értéket elfogadni.

F1.3 Nehézségi beállítások

A következőkben mindig utalni fogok arra, hogy az egyes kijelzett 'segítő funkciók' milyen nehézségi beállítás mellett jelennek meg. Az egyes játékosokhoz tartozó nehézségi beállítások a menüben a Settings/Difficulty alatt határozhatók meg. Ezen beállítás megkezdett játék közben is módosítható.

F1.4 Új játék indítása, kezdeti lépések, általános funkciók

---

*"A játék kezdetén világosnak és sötétnek ugyanannyi figurája van: 1–1**király*** **(King)**, 1–1**vezér***(Queen) *(alternatív neve: „királynő"), 2–2**bástya*** **(Rook)** *(alternatív neve: „torony"), 2–2**huszár***(Knight) *(alternatív neve: „ló"), 2–2**futó*** **(Bishop)***és 8–8**gyalog*** **(Pawn)** *(alternatív neve: „paraszt").*

A sakktábla bekapcsolásakor, vagy egy játék befejeződése után a LEDes mátrixon (a továbbiakban kijelző) a világos, illetve a sötét játékos oldalán 2-2 sor (1,2,7,8 sorok) zöldeskéken világít. Ha vannak mezők, melyeken báb áll a középső, zöldeskék színnel nem megvilágított sorokban, a báb alatti mező kéken világít. A tábla a bábok mozgatására nem reagál (F1.4.a / 1 ábra).

Az LCD kijelzőn megjelenik a >Continue/New Game felirat. Ekkor ezek közül akármelyiket is választjuk, a tábla a sakk szabályainak megfelelő kezdőállapotot állítja be, és azon mezőket, amelyeken ezzel nem összhangban lévő, helytelenül elhelyezett báb van, piros fénnyel jelzi, emellett az LCD kijelzőn megjelenik a "PlacementError" üzenet (F1.4.a / 2 ábra). Ha egy futó játékot szakítunk meg, és a menüben a New Game menüpontot választjuk, ugyanide jutunk. A jelzett bábok megfelelő mezőre helyezésével a hibajelzés mezőnként megszűnik. Ha minden hibajelzés megszűnt, az LCD kijelzőn megjelenik az éppen lépés előtt álló játékos (ez esetben White Player), és - ha engedélyezve van - a sakkóra.

A tábla kizárólag a bábok jelenlétét képes érzékelni, típusát nem. Ha nem vagyunk biztosak abban, hogy a táblán minden báb a neki megfelelő helyre került, a következőt tehetjük:

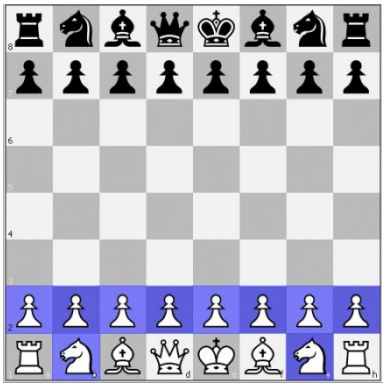
A menü megjelenítése után, felfelé léptetve a kurzort, majd a PlacementHelp menüpontot kiválasztva, az OK gomb nyomogatásával végigléphetünk a különböző bábokon (pl. LCD kijelzőn megjelenik a "White Player/Pawn", azaz fehér gyalog, üzenet, majd a "White Player/Rook, és így tovább). A kijelzőn pedig azon mezők világítanak zölden, amelyeken az LCD-n megnevezett bábóknak állniuk kellene. Miután az összes báb pozícióját megmutatta a funkció, visszatér a menübe.

Ez a lehetőség természetesen a játék során bármikor előhívható, és amíg fut, - hasonlóan bármely más menü funkcióhoz - az aktuális játékos sakkórája is áll.

Amennyiben a hozzá rendelt nehézségi fokozat enyhébb, mint "No Help", játék közben az éppen lépés előtt álló félnek azon bábjai alatt, melyeknek van a szabályokkal összhangban lévő lépéslehetősége, a kijelző kéken világít (F1.4.a / 3 ábra). Amennyiben a nehézségi fokozat könnyű (Easy), a közvetlen veszélyben lévő bábuk alatti mező ezen felül pirosan (is) világít. Közvetlen veszélyben akkor van egy báb, ha az ellenség által támadás alatt van, de saját báb nem védi, tehát leütése az ellenfél számára nem hordoz veszélyt. Amennyiben egy bábnak van érvényes lépése, és veszélyben is van, a mező lila színű (kék+piros) lesz.

Bármely, érvényes lépéssel rendelkező báb felemelése esetén, amennyiben a nehézségi fokozat közepes (Medium), vagy annál enyhébb, a bábhoz tartozó érvényes lépéslehetőségek a kijelzőn zöld színnel lesznek megjelölve. Amennyiben a nehézségi fokozat könnyű (Easy), a veszélyesnek ítélt lépések sárga színnel kiemelésre kerülnek a lehetséges lépések közül (ezen lépések esetén a báb közvetlen veszélybe kerülhet).

Amint az a korábbiakban is említésre került, az LCD kijelzőn, ha a tábla a konfigurációnak megfelel, a menü nem aktív, és nincs promóció előtt álló gyalog sem, játék közben mindig az aktuálisan lépés előtt álló játékos (White player vagy Black player), illetve, amennyiben a sakkóra engedélyezve van, a számára hátralevő gondolkodási idő jelenik meg.



F1.4.a ábra[2]
Bal felső / 1:
- Alapállapot jelzése
Jobb felső /2:
-PlacementError jelzése
Bal alsó /3:
- Helyes konfig., világos lépése következik. A nehézségi fokozat enyhébb mint No Help

F1.5 Bábokra vonatkozó szabályok, lehetséges lépések

A következőkben az egyes bábok, illetve az egyes bábokhoz tartozó lépés lehetőségek kerülnek ismertetésre. Természetesen a tábla e szabályok mindegyikét ismeri, és betart(at)ja.

F1.5.1 Király

Kezdő pozíciója: világos - e1, sötét - e8
Bármely irányban (vízszintesen, függőlegesen, átlósan) léphet, de csak egy mezőt, vagyis csak közvetlen szomszédos mezőre léphet, kivéve ha sáncol (később részletezve lesz, a különleges lépések között). A király mozgását korlátozza, hogy sakkba nem léphet, azaz nem állhat olyan mezőre, amelyen támadás alá kerülne. A király nem üthető, de üthet. Ha megtámadják, akkor meg kell szüntetni a támadást. (lásd. 2.6.2 bekezdés)

Mind a Vezérre, mind a Bástyára, mind pedig a Futóra igaz, hogy a szabályok által meghatározott irányokba tetszőleges távolságra léphetnek, addig, amíg egy másik báb az útjukba nem kerül. Ha ez a báb ellenséges, azt üthetik, tehát átvehetik a helyét a táblán. Ha barátságos, eggyel előbb meg kell állniuk. Általános szabály, hogy bármely mezőn egy időben kizárólag egy báb állhat.

F1.5.2 Vezér

Kezdő pozíciója: világos - d1, sötét - d8

Bármely irányban léphet, bármennyi mezőt, mindaddig, amíg a tábla széléhez nem ér, vagy egy másik báb útját nem állja. Tág mozgáslehetőségéből adódóan a vezér a sakkjáték legerősebb figurája.

F1.5.3 Bástya

Kezdő pozíciói: világos - a1 és h1, sötét - a8 és h8

Bármennyi mezőt léphet, de csak függőleges és vízszintes irányban, az átlókon nem. Részt vesz a sáncolásban.

F1.5.4 Futó

Kezdő pozíciói: világos - c1 és f1, sötét - c8 és f8

Átlós irányban léphet, bármennyi mezőt. Mindkét játékosnak az egyik futója csak a sötét, a másik futója csak a világos mezőkön közlekedik.

F1.5.5 Huszár

Kezdő pozíciói: világos - b1 és g1, sötét - b8 és g8

A róla elnevezett „lóugrásban" lép: vízszintesen két mezőt, majd függőlegesen egyet (vagy fordítva). Mivel a huszárnak nincs konkrét menetiránya, csak kiindulási és érkezési mezője (azaz nem „halad", hanem ugrik), nincs értelme „útjában álló bábról" beszélni. Természetesen, ha a lépése céljául szolgáló mezőn ellenséges báb áll, azt ütheti, ha saját, a lépés érvénytelen. A huszár lépéslehetőségeit ezen felül egyedül az korlátozza, ha a tábla szélén vagy a sarokban áll. Sarokmezőről a huszár üres táblán is mindössze két helyre léphet, a centrumban nyolcra.

F1.5.6 Gyalog

Kezdő pozíciói: világos - 2 sor minden mezőjén, sötét - 7 sor minden mezőjén

Kizárólag előre léphet. A kiindulási helyéről mind a nyolc gyalog tetszés szerint egy vagy két mezőt léphet előre, a továbbiakban azonban lépésenként mindig csak egy mezőt haladhat előre. Ütni azonban csak átlósan tud, szintén csak egy mezőnyi távolságra.

Van egy hozzá kapcsolódó kivételes ütési lépés, az en passant, továbbá "különleges lépése", az átváltozás.

F1.5.7 Lépés és ütés folyamata

A játékot mindig világos kezdi, és tetszése szerint – de a szabályoknak adta kereteken belül – valamelyik figuráját áthelyezi egy másik mezőre. Ezt lépésnek hívjuk. Ezt követően sötét lép egyet. A játékosok felváltva lépnek, passzolásra nincs lehetőség („lépéskényszer"). Ha egy mezőn az ellenfél bábja áll, azt egy szabályos lépéssel ki lehet ütni: az ellenfél figuráját levesszük a tábláról, és saját, odalépő bábunkat tesszük a helyére. Saját figura kiütésére nincs lehetőség.

A sakktábla az érvényes lépések között természetesen mindig kijelzi az ütési lehetőségeket is. Amennyiben egy bábbal ütünk egy másikat, előbb az ütést végző, majd a cél mezőn található bábot kell felemelni. Ekkor a cél mező pirossá válik addig, amíg az ütő báb a leütött báb helyére nem kerül.

F1.5.8 Különleges lépések kezelése

A sakkban három különleges lépés van:

F1.5.8.1 Sáncolás (avagy a szlengben: rosálás)

"A király és az egyik bástya együttes lépése. Ezt világos is, sötét is mindössze egyszer teheti meg a játék során. A király a kiválasztott bástya felé lép két mezőt, a kiválasztott bástya pedig a király által átlépett, tehát a királlyal szomszédos túloldali mezőre kerül. Ha a király a királyszárnyra sáncol, azt rövidsáncnak (írott jele: 0-0), ha a vezérszárnyra, hosszúsáncnak (írott jele: 0-0-0) hívjuk.

A sáncolás csak akkor lehetséges, ha az alábbi feltételek mindegyike igaz:

1. *a király még nem lépett a játszma folyamán;*
2. *az a bástya, amellyel sáncolni szeretnénk, még nem lépett a játszma folyamán;*
3. *a király és a bástya között nem áll sem saját, sem ellenséges báb;*
4. *a sáncolás előtt a király nem áll sakkban, a sáncolással nem kerül sakkba, és a sáncolás közben nem halad át olyan mezőn, amelyet ellenséges báb támad."*
5. *természetesen, az a bástya, mellyel sáncolni szeretnénk, nem lehet átalakult gyalog.*

Sáncolás esetén a király felemelésekor az érvényes lépések között megjelennek a szabályok által megengedett sáncolási lehetőségek. Amennyiben a királyt egy érvényes, sáncolási lépéshez tartozó cél mezőre léptetjük, a tábla a sáncoláshoz használt bástya alatti, illetve a király melletti mezőt piros fénnyel emeli ki. Ekkor a bástyát fel kell emelni (ennek hatására alatta a mező elsötétül), és a másik kiemelt mezőre kell helyezni.

F1.5.8.2 Ütés menet közben (en passant)

"Ha a kiindulási mezőjéről kettőt lépő gyalogunkkal áthaladunk egy ellenséges gyalog ütésmezején, akkor az ellenfél gyalogunkat a következő lépésben – de csakis akkor – leütheti. Az ütés pontosan úgy történik, mintha az ütött gyalog csak egyet lépett volna."

Ebben az esetben az ütést végző gyalog lehetséges lépései között megjelenik az ellenfél gyalogja által "átugrott" mező is, mint lépési lehetőség. Ha a gyaloggal ide lépünk, a tábla a leütött gyalogot pirosan jelzi, amíg el nem távolítjuk azt a tábláról.

F1.5.8.3 Átalakulás

"Ha a gyalog áthaladt az egész táblán és eljutott az ellenfél alapsorába, átváltozik tisztté. Ez úgy történik, hogy a gyalogot levesszük a tábláról, és a helyére állítunk egy (a táblán levő állományon kívüli) vezért, bástyát, futót vagy huszárt, tetszés szerint. A játékosok többnyire a vezér figuráját választják, mivel az a legerősebb. Ugyanakkor adódhat olyan állás, amikor érdemes más figurát fölvenni. Az alapsorra való belépés (vagy beütés), a gyalog (és esetlegesen a leütött figura) levétele a tábláról, az új figura felhelyezése – mindez egyetlen lépésben történik, mégpedig szükségszerűen: ha a gyalog elérte az utolsó sort, kötelező az átváltozás."

Ha egy gyaloggal az ellenfél alapsorába lépünk, a lépés befejeződése előtt az LCD kijelzőn a következő jelenik meg: Promoteto/>Rook. A navigáló gombokkal a megszokott módon választhatunk, hogy milyen egyéb tisztre szeretnénk lecserélni a gyalogot, ha a Bástya nem megfelelő. Az OK gomb megnyomásakor a tábla automatikusan rögzíti a báb új típusát. Amennyiben van más készletből származó figuránk, vagy a leütött figurák között szerepel az, amelyre a gyalogot átváltoztattuk, természetesen a figura lecserélhető, hogy új titulusát reprezentálja.

F1.6 Egyéb fontos szabályok, és megjelenésük a játék során

F1.6.1 Fogott bábu lép, letett bábu marad

"Franciául: pièce touchée. (am. 'megérintett bábu'). Az alább említett kivételtől eltekintve ha a lépésre következő játékos megérinti saját figurái egyikét, köteles azzal lehetőség szerint szabályos lépést tenni. Ha az ellenfél figuráját érinti meg, lehetőség szerint köteles azt kiütni. Továbbá a megtett lépést tehát nem lehet visszavonni, azaz az elengedett bábu már nem mozdítható. Ez alól a versenyző csak akkor mentesül, ha az érintés előtt Igazítok! felkiáltást tesz, majd ezt végre is hajtja. Nem hivatalos játszmáknál e szabály alkalmazásáról előzetes megegyezés alapján döntenek. Ez alól csak egy kivétel van: ha olyan figurát érintett meg a játékos, amellyel nem lehet szabályos lépést megtenni."

Az interaktív sakktábla használata esetén a fenti szabály speciális változata érvényes. Egyfelől, felemelni/elmozdítani csak olyan, saját bábot lehet, melynek érvényes lépése van, tehát ellenséges bábu nem emelhető fel, kizárólag akkor, ha egy azt ütni képes saját bábunkat már előtte felemeltük. Ez egyben befejezett lépésnek számít, saját bábunkat a leütött/felemelt báb helyére KELL helyezni, és automatikusan az ellenség következik. Lépés nem vonható vissza! Hasonlóan igaz ez akkor is, ha nem ütünk, tehát abban az esetben, ha egy saját bábunkat felemeltünk, majd egy érvényes lépés keretében letettünk, a lépés végleges, és az ellenfél következik. Azonban abban az esetben, ha egy érvényes lépéssel rendelkező bábot felemelünk, még NEM kötelező lépni vele. A figura kiindulási mezője a lépés idejére kék színű jelzést kap. Ha a bábot felemeltük, de mégsem szeretnénk lépni vele, a fent említett kék mezőre visszahelyezve úgy folytatható a játék, mintha fel sem emeltük volna a bábot. Bármely egyéb esetben (pl. érvénytelen báb felemelése, lehelyezése, a tábla által helytelen elhelyezés miatt nem érzékelt báb, stb.) az elvárt táblaállásnak nem megfelelő mezők pirosan ki vannak emelve, az LCD kijelzőn a PlacementError üzenet jelenik meg, mely csak akkor tűnik el, ha a tábla az elvárt állapotba kerül. Ha nem tudjuk, mi az elvárt állapot, a korábban már említett PlacementHelp menüpont segítségével deríthetjük ki. Fontos megjegyezni, hogy az ütés, vagy sáncolás során megjelenő piros figyelmeztetések esetében ugyancsak ez a hibaüzenet jelenik meg a kijelzőn, amíg a lépést be nem fejeztük.

F1.6.2 Sakkadás

"Ha az ellenfél lépésével olyan helyzet alakul ki a táblán, hogy amennyiben megint ő következne lépéssel, kiüthetné a királyunkat, akkor sakkot kaptunk*, másként fogalmazva a királyunk sakkban áll. A* sakkot *kötelező megszüntetni. Ez háromféleképpen történhet:*

1. *Kilépünk a sakkból, vagyis királyunkkal olyan mezőre lépünk, amelyet nem támad ellenséges báb.*
2. *Leütjük a sakkot adó bábot.*
3. *Közbehúzunk egy másik figurát a támadó báb és a királyunk közé. Ez a megoldás értelemszerűen nem jöhet szóba, ha huszártól kaptunk sakkot, vagy közvetlenül szomszédos mezőről."*

A tábla konkrétan nem hívja fel a figyelmet a sakk tényére. Ha sakkba kerülünk, csak azon lépések megengedettek, melyek a sakk megszüntetésére irányulnak.

F1.6.3 A játék célja

"A sakkjáték célja olyan állás elérése, amelyben az ellenfél királya sakkban (ütésben) van, de nincs olyan mező, ahová szabályosan léphetne, és más szabályos módon sem tudja a sakkot kivédeni (azaz közbehúzni egy másik figurát, vagy leütni a támadó bábot). Ha ez történt, a király mattot kapott*. Az eredeti játékszabályok szerint a játékot akkor nyeri meg valaki, ha ellenfelének mattot ad."*

Amennyiben valamely fél mattot kap, a győztes oldal bábui alatt rövid animáció után fehéren világítanak a mezők, minden más mező elsötétül, a sakkóra megáll. Ezután a bábuk elmozdítására a tábla már nem reagál, a felhasználói interfész bármely gombjának megnyomása esetén a tábla alaphelyzetbe áll (azonos állapot, mint a tábla áram alá helyezésekor).

F1.6.3.1 Döntetlen kezelése

Döntetlen számos esetben előfordulhat. Lehetséges közös megegyezés alapján döntetlenné nyilvánítani a játékot annak bármely pontján. Lehet patthelyzet is, amely akkor alakul ki, ha a soron következő játékos számára nem létezik szabályos lépés, azonban királya nincs sakkban. Továbbá amennyiben valamely fél kérelmezi, döntetlenné nyilvánítható egy játék, ha a tábla háromszor kerül - ugyanazon játékos lépése után - azonos konfigurációba. Az előfordulásoknak nem kell közvetlenül egymás után történniük. Ugyancsak kérvényezhető döntetlen, ha az utolsó 50 lépés során nem következett be ütés, vagy lépés valamely gyaloggal (a szabálynak több változata is létezik, versenysakkban van 100 és 75 lépéses szabály is). Végül, de nem utolsó sorban döntetlenné válik egy meccs abban az esetben, ha az egyik fél számára sem áll rendelkezésre elég báb ahhoz, hogy az ellenfélnek mattot adhasson.

Abban az esetben, ha patthelyzet alakul ki, vagy egyik fél számára sem áll rendelkezésre megfelelő állomány ahhoz, hogy mattot adhasson, a tábla automatikusan befejezi a játékot, és döntetlent hirdet. Ekkor a táblán álló valamennyi báb alatt rövid animáció után fehéren világítanak a mezők, minden más mező elsötétül, a sakkóra megáll. Ezután a bábok elmozdítására a tábla már nem reagál, a felhasználói interfész bármely gombjának megnyomása esetén a tábla alaphelyzetbe áll (azonos állapot, mint a tábla áram alá helyezésekor).

Egyéb eseteket (pl. 50 lépéses szabály) a tábla nem vesz figyelembe, mivel azokat a játékosnak kell követnie/észrevennie, illetve kérvényeznie, de mivel a tábla eleve nem verseny célokat szolgál, e szabályok kielégítését adó játékhelyzet előfordulására az esély alacsony.

F1.6.4 Játék időre - a sakkóra konfigurációja és használata

"Annak érdekében, hogy egy-egy játszma ne nyúljon a végtelenségbe, a versenyeken sakkórát használnak. A sakkóra olyan készülék, amely két óraszerkezetet tartalmaz. Az egyik óra leállítása elindítja a másikat és viszont. Lépésének megtételét követően a játékos egy gomb lenyomásával leállítja saját, egyben elindítja ellenfele óráját.

A sakkóra segítségével meghatározható, hogy a játékosoknak mennyi gondolkodási idő álljon rendelkezésükre a játék során. Akinek letelik a gondolkodási ideje (a hagyományos, analóg óránál: leesik a zászlója), elveszíti a játszmát, ugyanúgy, mintha mattot kapott volna. Ez alól kivétel, ha ellenfelének már nincs a táblán matt adó bábja."

Az interaktív sakktábla lehetőséget ad arra, hogy a menüben a Settings/ChessClock/SetupClock menüpont alatt sakkórát konfiguráljunk. Ez egy Fischer típusú, növekményes sakkóra, amely a következőt jelenti:

A játékosok rendelkezésére áll egy, a teljes játék idejére vonatkozó alap gondolkodási idő, mely a SetupClock menü indításakor megjelenő első pontban adható meg, SetBase Time/10m00s formájában. A navigációs gombokkal ezt az időt lehetséges 1 perc és 45 perc közötti tetszőleges értékre beállítani, perces pontossággal. Az OK gomb megnyomása után állíthatjuk be a növekményt. A kijelzőn a SetDelay/00m05s jelenik meg, ezt az időt 0 másodperc és 30 másodperc közé állíthatjuk, másodperces pontossággal. Az OK gomb megnyomása után visszatérünk a Settings/ChessClock menübe. A sakkóra engedélyezéséhez az EnableClock menüpontot kell kiválasztanunk, majd megnyomnunk az OK gombot. Ezután a menüpont DisableClock-ra vált, lehetővé téve a sakkóra letiltását a későbbiekben. Az új beállítások csak újonnan megkezdett játékban érvényesülnek. A sakkórán alapból megjelenik az alap gondolkodási idő, plusz a növekmény, mely minden körben hozzáadódik a maradék időhöz. Ha a játékos a növekmény lejárta előtt befejezi a lépést, a növekmény hozzáadódik a maradék rendelkezésre álló időhöz. A sakkóra nem jár, ha a tábla szabálytalan elrendezésű, vagy a menü meg van nyitva, értékét megőrzi mindaddig, míg új játékot nem kezdünk (ez azt is jelenti, hogy a sakkóra leállítása, majd újraindítása esetén a sakkóra onnan fut tovább, ahol korábban le lett állítva). Abban az esetben, ha valamelyik félnek elfogy az ideje, az automatikusan az ellenfél győzelmét okozza.

F2 Tervezett, de nem megvalósított funkcionalitás

A tervezett funkcionalitásból néhány elem sajnos idő hiányában kimaradt. Bár a hardveres felépítés támogatja e funkciókat, kihasználásuk csak a tábla egy későbbi verziójában lesz lehetséges.

F2.1 USB kapcsolat kialakítása, AI biztosítása USB kapcsolaton keresztül

Az eredeti tervek szerint a sakktábla egy USB kapcsolaton keresztül minden lépés után elküldte volna a tábla aktuális pozícióját egy számítógépnek, ahol valamilyen mesterséges intelligencia segítségével (kezdetnek pl. alfa-béta vágással) a helyzetet felmérve, majd az eredményt visszaküldve a következő lépésben lehetséges lett volna optimális lépést ajánlani. Bár a hardverelemek között megtalálható minden ehhez szükséges periféria, a sakktábla jelen verziójának megvalósítása felemésztette a fejlesztésre rendelkezésre álló időt.

F2.2 Vezeték nélküli működés, akkumulátor

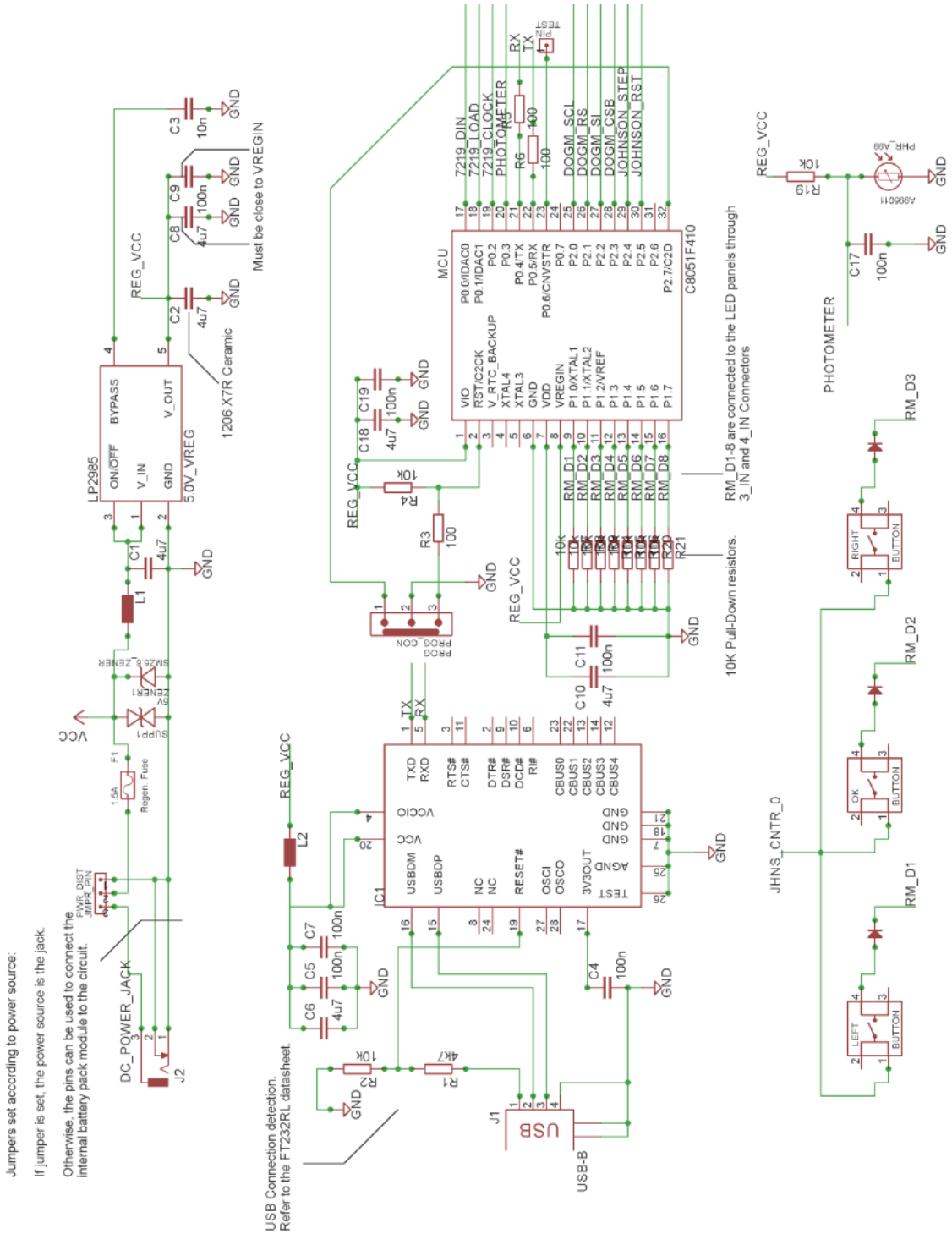
A tábla áramellátását biztosító rendszer úgy lett kialakítva, hogy az eszköz képes legyen saját, belső áramforrásról is üzemelni. Sajnos a megvalósítás során egyfelől az áramkörben több (tervezési, és máig ismeretlen eredetű) hibára is fény derült, másfelől az akkumulátoros megvalósítás nagyban növelte volna az eszköz hardveres költségét.

Sajnos a tervezés során ugyancsak elsiklottam néhány apró dolog felett, mely az energiagazdálkodást támogathatja, például az LCD háttérvilágítása sajnos nem kapcsolható ki, illetve nincs be/kikapcsoló gomb, és ez sajnos a felhasználói interfész gombjaival sem valósítható meg.

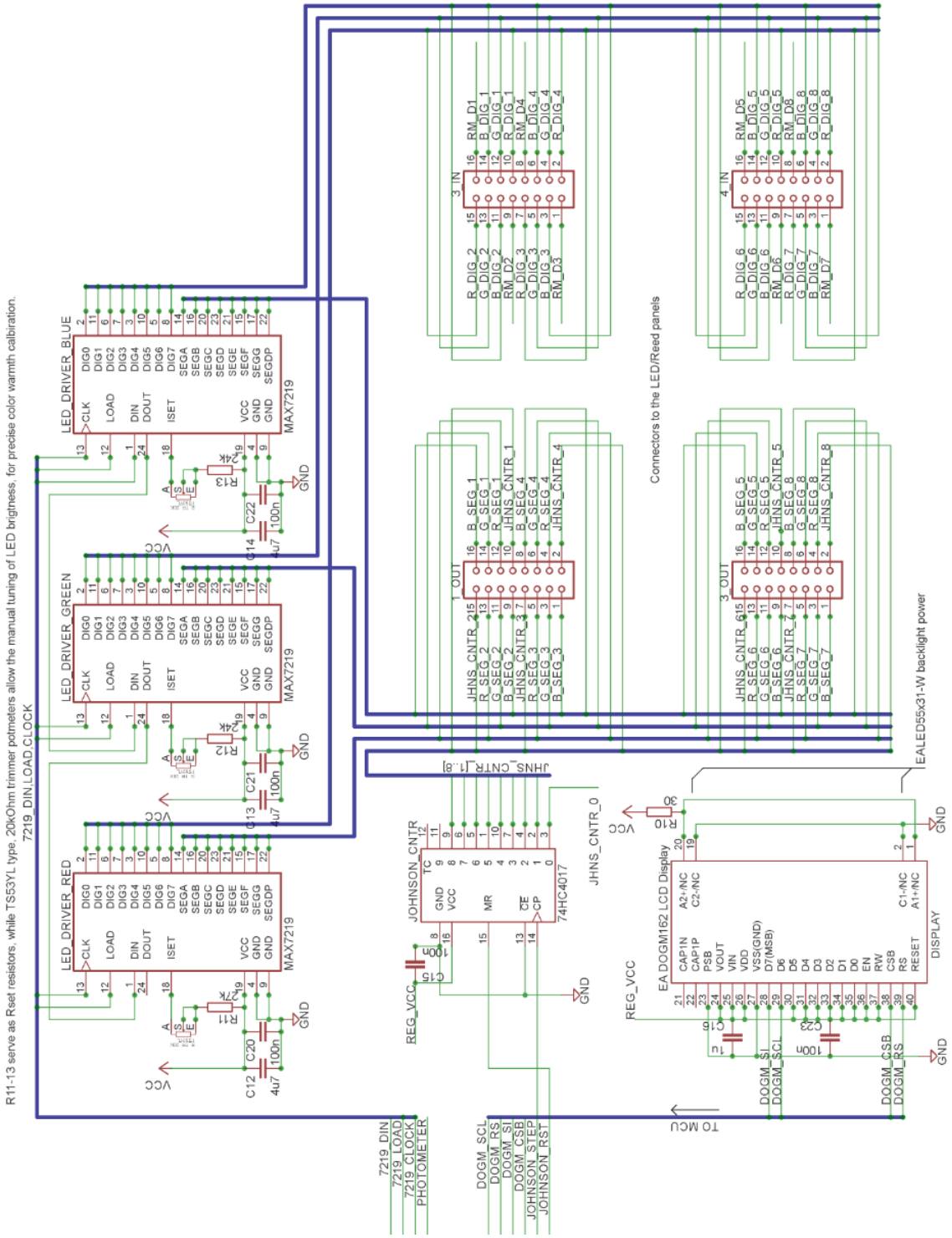
F3 Kapcsolási rajzok, NYÁK tervek

F3.1 Control Panel kapcsolási rajza

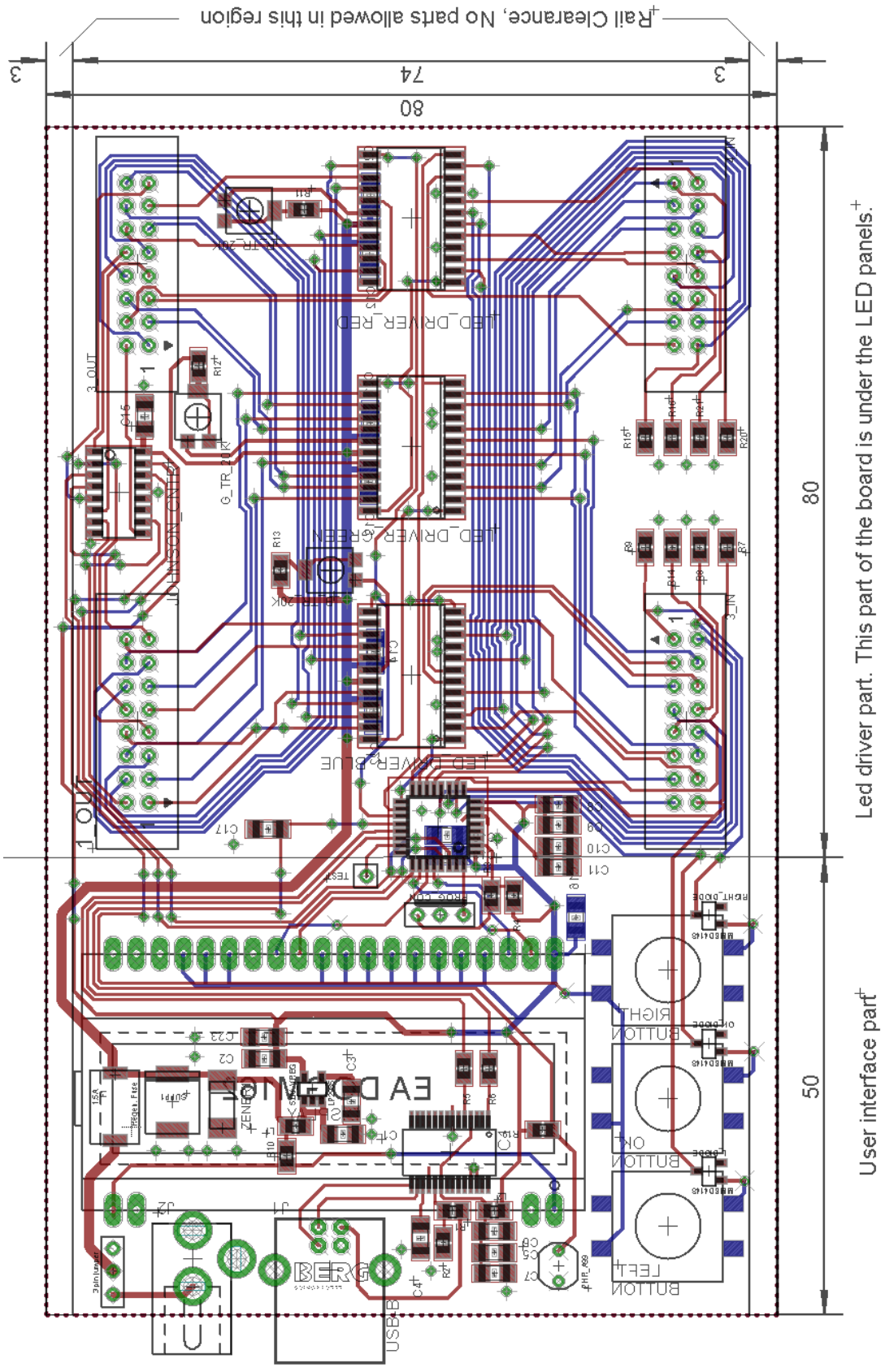
(1. rész - Táprendszer, gombok, USB, MCU)



(2. rész - LED meghajtók, dekád számláló, csatlakozók, LCD)

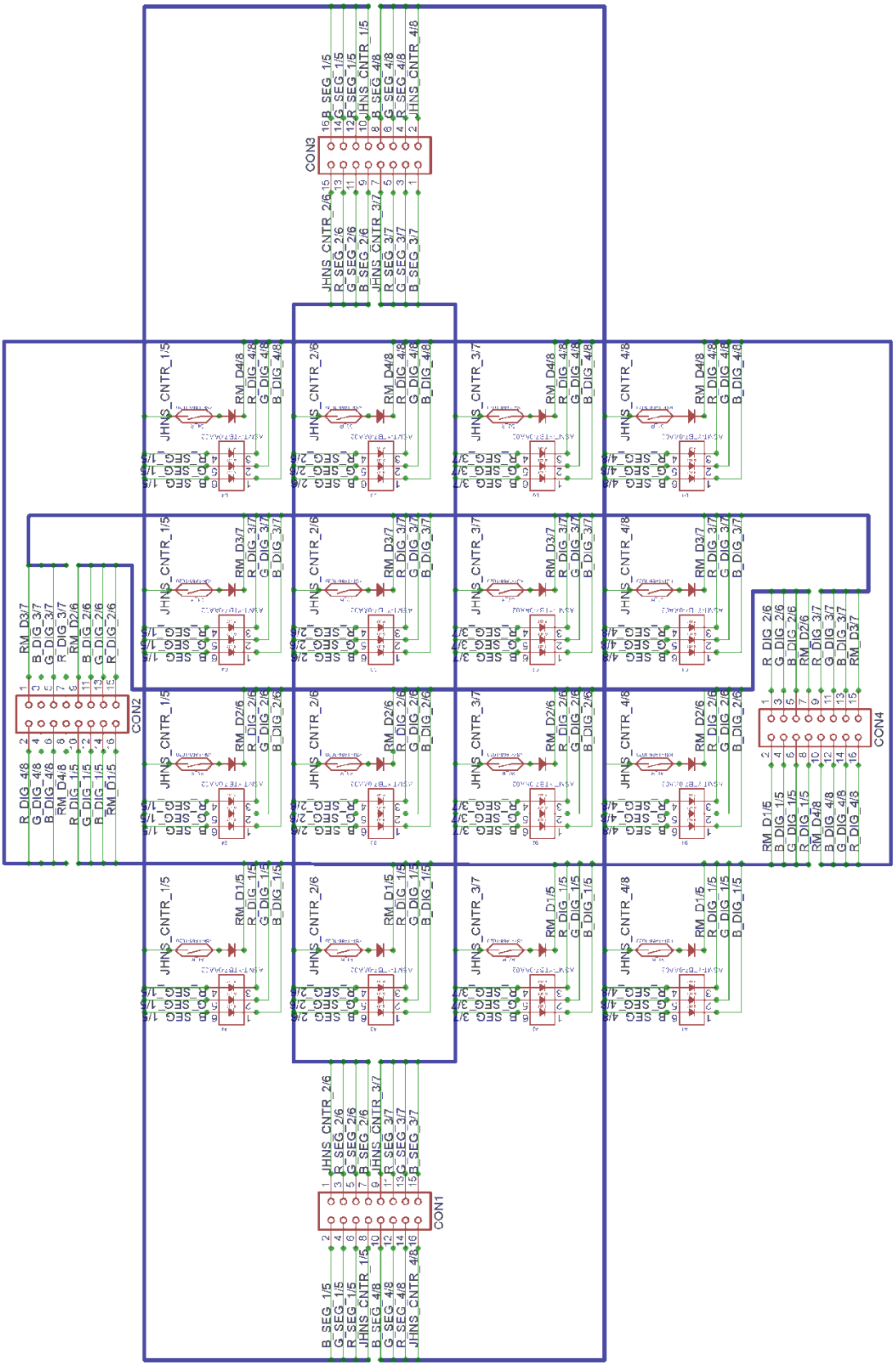


F3.2 Control Panel NYÁK terve

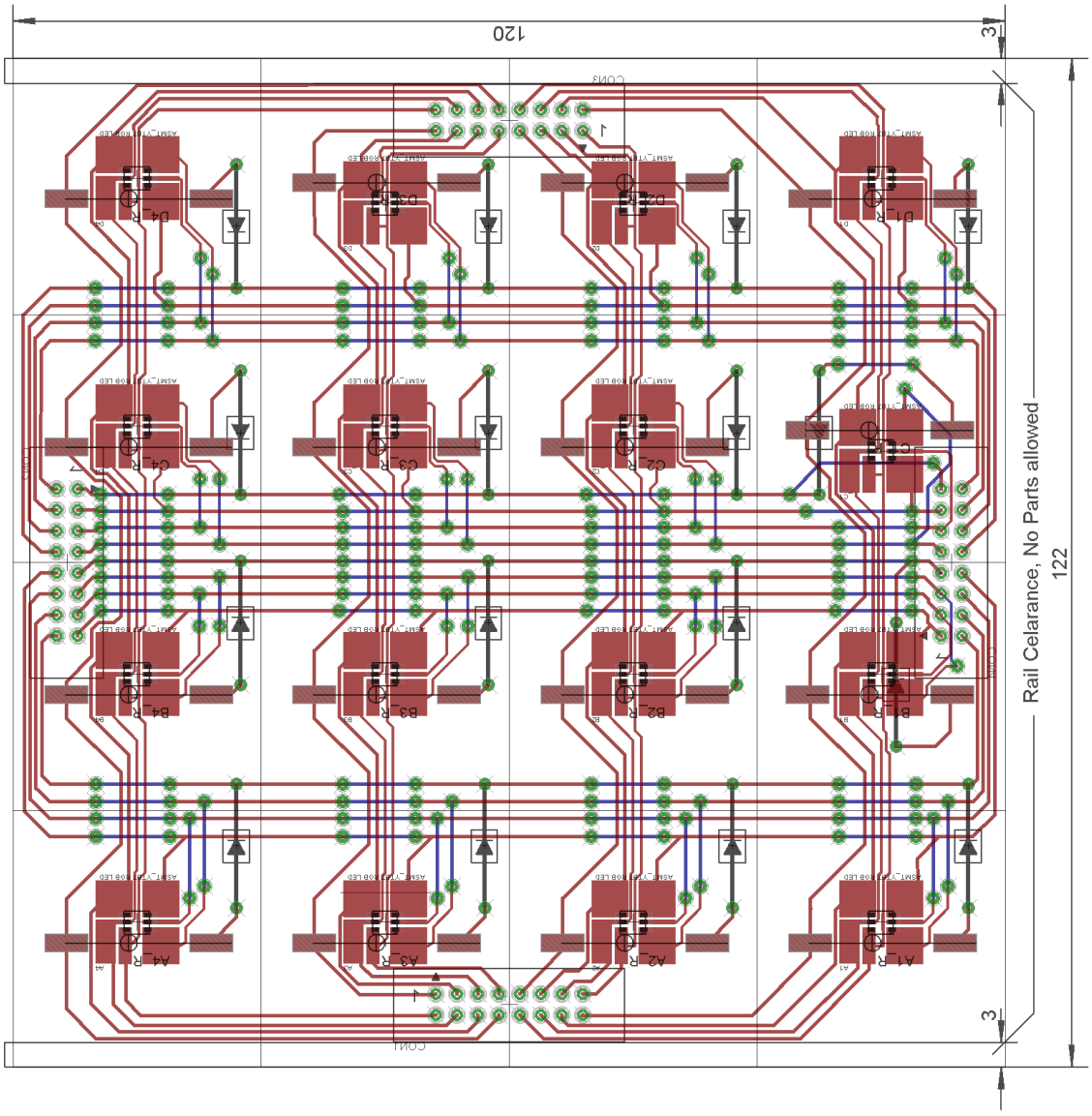


F3.3 LED/Reed panel kapcsolási rajza

(CD mellékletben lévő rajzhoz képest kissé átalakított rajz, hogy jobban illeszkedjen a nyomtatási elrendezéshez)



F3.4 LED/Reed panel NYÁK terve



F4 Egyéb táblázatok, ábrák

F4.1 Az MCU pin kiosztása

Pin Number	Role	Target
P0.0	DIN	MAX 7219
P0.1	LOAD	
P0.2	CLK	
P0.3	~~Ambient Light input~~	~~Photoresistor~~
P0.4	RX	FT232
P0.5	TX	
P0.6	Test PIN	Test input/output
P0.7	-	
P1.0	D0	Reed matrix inputlines
P1.1	D1	
P1.2	D2	
P1.3	D3	
P1.4	D4	
P1.5	D5	
P1.6	D6	
P1.7	D7	
P2.0	SCL	EA_DOGM LCD SPI
P2.1	RS	
P2.2	SI	
P2.3	CSB	
P2.4	Step	4017BD Johnson C.
P2.5	Reset	
P2.6	-	
P2.7	C2D	Programming connector