



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Commodore floppy meghajtó megvalósítása FPGA-val

MSc ÖNÁLLÓ LABORATÓRIUM 2.

Készítette

Weisz Pál

Konzulens

Horváth Kristóf

2022. december 18.

Feladatkiírás

A hallgató feladata, hogy egy Commodore számítógépekhez készült floppy meghajtó egységet valósítson meg FPGA segítségével. A munka magába foglalja az eredeti hardver megismerését és annak áttervezését úgy, hogy azt az interfészarámköröket leszámítva meg lehessen valósítani modern technológiák felhasználásával. Cél még, hogy kész rendszer képes legyen valódi (azaz nem virtuális) floppy lemezek kezelésére egy PC floppy meghajtó felhasználásával.

Tartalomjegyzék

1. Bevezető	3
1.1. A feladat összefoglalása	3
1.2. Rendszerterv	4
1.3. Protokollok	5
1.3.1. A Commodore IEEE-488 soros busz	5
1.3.2. Floppy meghajtó	6
2. A hardver	8
2.1. Bemutatom a fejlesztőkártyát	8
2.2. Az interfészpanel	9
3. Digitális tervezés	10
3.1. Modulok	10
3.2. CPU	12
3.3. CIA	13
3.4. FDC	14
3.5. Órajel-generátor	15
3.6. Memóriák	15
4. Tesztelés, hibajavítás	16
4.1. Az interfészpanel tesztje	16
4.2. Verilog-modulok	17
4.3. A firmware	17
4.4. Emulátor	18
4.5. Tesztelés valódi környezetben	19
4.6. Záró gondolatok	22
Irodalomjegyzék	24
Függelék	25
F.1. Az eredeti 1581-es floppy meghajtó kapcsolási rajza	25
F.2. Az interféskártya kapcsolási rajza	30
F.3. Az interféskártya NYÁK-terve	33

1. fejezet

Bevezető

1.1. A feladat összefoglalása

Cél tehát, hogy egy szabványos IBM PC floppy meghajtót felhasználva FPGA segítségével egy olyan hajlékonylemezes egységet valósítsak meg, mely Commodore-számítógépekkel kompatibilis és a hozzájuk való floppy-lemezeket írni-olvasni képes.

Az önálló laboratórium alatt végzett munkám során egy Commodore 1581 típusú, 3,5"-es lemezmeghajtó prototípusát készítettem el ily módon. A kitűzött cél megvalósításához felkutattam ez eredeti meghajtó kapcsolási rajzát, szervizkönyvét, elolvastam a konstrukcióban alkalmazott alkatrészek adatlapjait. Megismerkedtem a Commodore számítógépes környezetben alkalmazott perifériák, interfések szabványaival, működésével is. Ezek alapján, valamint az általam használt FPGA-fejlesztőpanel és PC-s meghajtó jelszintjeinek, tulajdonságainak figyelembe vételével megterveztem egy interfészpanelt, mely az alaplapi vezérlő logikát implementáló FPGA-t illeszti a fizikai környezethez, vagyis a meghajtó hardveregységéhez és a számítógéppel való kapcsolatot biztosító soros porthoz.

Az eredeti 1581-es vezérlését egy komplett mikroprocesszoros rendszer látja el, melyet - funkcionálitását tekintve - egy az egyben az FPGA-n szintetizálva valósítottam meg. Az ehhez szükséges verilog-modulok egy részét korábbi projektekből emeltem át, a hiányzókat pedig én magam implementáltam.

A rendszert először modulonként, majd különböző integráltsági szinteken is teszteltem, először a Xilinx ISE szimulációs környezetében, majd pedig logikai analizátorral a kész hardveren is. Az FPGA-n megvalósított rendszer alkalmas az eredeti 1581-es firmware-ének futtatására, mely bináris állomány formájában rendelkezésemre állt. A kezdeti sikeres integrációs teszteket követően processzor belső állapotát nagyon körülmenyessé vált vizsgálni, és mivel a rendszer egy ponton érzékelhetően hibára futott, másféle megközelítéssel próbálkoztam: írtam egy emulátorprogramot, mely segítségével sikerült meghatároznom és kijavítanom a készként átvett CPU implementáció hibáját.

További teszteket követően számos egyéb hibát derítettem fel és javítottam ki. A félév végeztével sikerül elérnem, hogy a meghajtó képes legyen hibamentesen kommunikálni soros porton keresztül a valódi Commodore számítógéppel.

1.2. Rendszerterv

Első lépés az irodalomkutatás volt, valamint az elinduláshoz szükséges információk összegyűjtése. Az eredeti meghajtó áttervezéséhez nélkülözhetetlen megismerni, hogyan működik, milyen részekből épül fel az eszköz - ebben nagy segítségemre voltak a nyilvánosan elérhető dokumentációk, kézikönyvek.[12] Azért esett a választás épp a Commodore 1581 típusú floppy meghajtóra, mert ez ma is többé-kevésbé elérhető kis méretű 3.5"-es hajlékonylemezeket használ, belső felépítését tekintve pedig jól elkülöníthető részekre dekomponálható, szemben például a 1541-essel.



1.1. ábra. A Commodore 1581 típusú hajlékonylemezes meghajtó

A Commodore 1581-es meghajtó tulajdonképpen maga is egy "számítógép", hiszen a kiszolgálni kívánt számítógéppel való kommunikációt, és magát a lemezkezelést egy, a meghajtó alaplapján található diszkrét integrált áramköri elemekből felépített processzoros rendszer végzi. A 1581-es és a számítógép közötti kommunikáció egy Commodore IEEE-488 szabványú soros buszon zajlik.[21] A floppy lemezhez fizikailag egy belső író-olvasó egység segítségével lehetséges hozzáérni, mely egyébként szabványos, Amiga számítógépeknél is használatos ún. Shugart-interfésszel rendelkezik.[5] Ezzel szemben az általam megtervezett rendszerben a floppy író-olvasó egységet egy mai modern asztali IBM PC-vel kompatibilis floppy meghajtóval helyettesítettem, mivel ilyen berendezéseket ma sem lehetetlen beszerezni, szemben az eredeti konstrukcióban alkalmazott típusúval.

A szervizkönyv mellékleteként megtalálható a meghajtó eredeti kapcsolási rajza is, melyet a további tervezéshez az alábbi részekre bontottam:

- *Soros port*: a soros busz meghajtását és leválasztását szolgáló áramköri elemek
- *Floppy interfész*: a PC floppy meghajtó illesztéséhez szükséges bufferek, inverterek
- *FPGA*: az egész processzoros környezetet és minden más integrált áramköri elemet a szükséges kiegészítő logikákkal együtt ezen fogok megvalósítani

Így jól elkülöníthetővé váltak azok a részek, melyekhez kiegészítő áramkör fizikai elkészítése szükséges az egyes protokollok tulajdonságainak ismeretében.

1.3. Protokollok

Következő feladat annak az előkészítése volt, hogy az itt ismeretetett protokollokon szabványos módon történjen kommunikáció az alaplap és a PC floppy-meghajtó, valamint a számítógép között.

1.3.1. A Commodore IEEE-488 soros busz

A 8-bites Commodore asztali számítógépek az ún. Commodore IEEE-488 soros busz (vagy röviden IEC-busz) segítségével kommunikálnak a hozzájuk kapcsolt főbb perifériákkal, jellemzően nyomtatókkal vagy mágneses adathordozó alapú háttértárrakkal. Eredetileg az IEEE-488 szabványból alakították ki a költségek csökkentése végett (innen az elnevezése). Ez alapvetően egy fél-duplex szinkron soros adatátviteli interfész, viszont számos speciális megoldást alkalmaz a szabvány pl. handshake vagy a buszra kapcsolódó eszközök jelenlét-érzékelésének megvalósítására.[21]

1.1. táblázat. Az IEC-busz jelei

Pin	Jelölés	Funkció
1	<u>SRQ</u>	Service Request In
2	<u>GND</u>	Ground
3	<u>ATN</u>	Serial Attention In/Out
4	<u>CLK</u>	Serial Clock In/Out
5	<u>DATA</u>	Serial Data In/out
6	<u>RESET</u>	Serial Reset

Alacsony-aktív, nyitott kollektoros meghajtású vonalak ezek, 5V-os logikai jelszinttel. A számítógépez csatlakoztatott eszközöknek saját maguknak kell gondoskodniuk a tápellátásukról. Az egyes adatvonalakat minden két végükön felhúzó ellenállások zárlják le, a buszt nyitott kollektoros inverterek hajtják meg. Csatlakozó gyanánt 6 lábú DIN dugót alkalmaznak.

A soros buszra több eszköz is csatlakozhat, ezeket fizikailag daisy-chainelve lehetséges összekapcsolni egymással és a számítógéppel, emiatt szokásosan két csatlakozót építenek ki az egyes perifériákra. Az egyes eszközöket azonosítójuk alapján képes megcímezni a számítógép. Az adott azonosító DIP kapcsolók segítségével hardveresen konfigurálható, mely lemezmeghajtók esetén tipikusan 8 és 11 közötti egész értéket vehet fel.

A kommunikáció egységei 8-bites adatcsomagok, melyekben a legkisebb helyi értékű bit található az első helyen. A mintavételezés szinkron módon, a CLK jel felfutó élére történik. Ezek az adatcsomagok lehetnek vezérlő parancsok, vagy adatok. Vezérlő parancsokkal lehet a busz használatára vagy elengedésére utasítani az egyes perifériákat, vagy épp valamilyen irányú adatátvitelt kezdeményezni a megcímezett eszközzel. A vezérlő parancsok a buszon az ATN jel alacsony szintje mellett érkeznek mégpedig kizárolag a számítógép felől, innen lehet tudni, hogy nem egy általános adat átvitele történik.

A CLK és DATA vonalak kétirányúak, ezeket a floppy meghajtó is lehúzhatja. Ilyen módon jelzi a számítógépnek, hogy készen áll az adat fogadására vagy küldésére, de például bizonyos időkorláttal kiegészített nyugtázsra is használja ezt a módszert a szabvány.[6]

1.3.2. Floppy meghajtó

A feladat egyik nehézségét az adta, hogy a 1581-es meghajtó eredetileg másfajta floppy író-olvasót tartalmazott, mint amilyeneket az IBM-kompatibilis személyi számítógépekben használtak, használnak. Ez nem is a működési módjukban, sokkal inkább az IBM PC és a Shugart interfések közti különbségek nyilvánul meg:[17]

1.2. táblázat. *IBM/PC és Shugart interfésszel rendelkező hajlékonylemez-meghajtók lábkiosztása*

IBM/PC			Shugart		
Pin	Jelölés	Funkció	Pin	Jelölés	Funkció
2	REDWC	Density Select	2	DCD	Disk Change Detect
4	n/c	Reserved	4	key	no pin in this position
6	n/c	Reserved	6	DS3	Drive Select 3
8	INDEX	Index	8	INDEX	Index
10	MOTEA	Motor Enable A	10	DS0	Drive Select 0
12	DRVSB	Drive Select B	12	DS1	Drive Select 1
14	DRVSA	Drive Select A	14	DS2	Drive Select 2
16	MOTEB	Motor Enable B	16	MTRON	Motor On
18	DIR	Direction	18	DIR	Direction
20	STEP	Step	20	STEP	Step
22	WDATE	Write Data	22	WDATE	Write Data
24	WGATE	Write Enable	24	WGATE	Write Enable
26	TRK00	Track 0	26	TRK00	Track 0
28	WPT	Write Protect	28	WPT	Write Protect
30	RDATA	Read Data	30	RDATA	Read Data
32	SIDE1	Head Select	32	SIDE1	Side Select
34	DSKCHG	Disk Change	34	RDY	Ready

Nem tüntettem fel a táblázatban, de mindenkiépp érdemes megjegyezni, hogy minden páratlan számú lábat a földre kötnek. Mivel a gyakorlatban szalagkábelben szokás továbbítani ezeket a jeleket, zavarvédelem szempontjából előnyös, ha az áthallás csökkentése végett nem jelvezetékek haladnak közvetlenül egymás mellett, hanem felváltva követik egymást a földvezetékekkel.

Hasonlóan a soros porthoz, mindenkor fajta interfész esetén is 5V-os, open kollektoros meghajtású alacsony aktív jelekkel lehet találkozni, viszont nem előírás, hogy az eszköz tartalmazza a felhúzó ellenállásokat. Ezek PC esetén a számítógépben, jelen megoldásban pedig a 1581-es meghajtó alaplapján találhatóak.

Megfigyelhető, hogy a jelek többsége megfeleltethető egymásnak.[2] PC esetén egy szalagkábelre két meghajtó kapcsolható, ezeket az A és B kiválasztójelek segítségével lehet megkülönböztetni egymástól, míg a Shugart interfészen akár 4 különböző eszköz is kiválasztható a \overline{DS} jelek megfelelőjének földre húzásával.

Szembetűnő különbség ezen kívül még, hogy a Disk Change jel máshol van kivezetve a két esetben, a Density Select valamint a Ready jeleknek pedig nincs megfelelője a másik csatlakozón.

A Density Select jellel kapcsolatban egymásnak ellentmondó információkat találtam kutatásom során, valahol kimenetként, máshol bemenetként használják, de mindenkiépp a meghajtóban használt floppy lemez adatsűrűségével hozható kapcsolatba. Ennek oka, hogy fizikailag máshogy kell kezelní egy SD, egy DD vagy egy HD hajlékonylemezt.

Korszerű meghajtók ezt az adathordozó tokján található kivágás segítségével érzékelik, régi fajta (és főleg 5.25"-es) meghajtók esetén viszont ezzel a Density Select jellet írható elő, milyen adatsebességgel dolgozzon a meghajtó vezérlője. Az általam használt PC floppy meghajtóban például ez a jel nincs is kivezetve.[16] Mivel nekem csak HD floppy lemezeim voltak a teszteléshez, úgy orvosoltam az ebből fakadó eltérést, hogy leragasztottam a tokon a kivágást, így várhatóan sikerült elhitetni a lemezmeghajtóval, hogy DD lemez van benne.

A Ready jel előállításához mindenkorábban kiegészítő logika megtervezése kellett, mivel a 1581-es kapcsolási rajza szerint szükség van rá. Az eredeti író-olvasó egység adatlapja alapján ez a jel akkor aktív, ha van lemez a meghajtóban és emellett megfelelő sebességgel forog a motor.[3] Ha forog a motor, fordulatonként egy impulzus mérhető az Index jelen. Az általam használt PC-s meghajtóban ez az impulzussorozat kapuzva van, és csak akkor kerül ki a kimenetre, miután a motor elérte a megfelelő fordulatszámot. Ennek a felhasználásával előállítható a Ready jel, hiszen hardveresen biztosított, hogy a motor csak akkor forog, ha van a meghajtóban lemez.

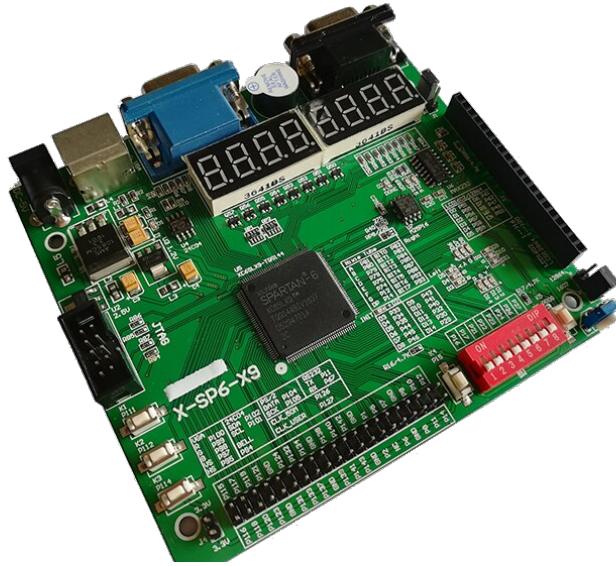
A Commodore 1581-es meghajtó két oldalú, 800kB kapacitású kétszeres adatsűrűségű (double-density, DD) lemezek kezelésére alkalmas. Az így formázott lemez oldalanként 80 sávot és 10 szektort tartalmaz fizikailag, a biteket MFM kódolással tárolja a rendszer. Mivel az így formázott lemez nem IBM kompatibilis, így a mai asztali számítógépekkel nem értelmezhető a tartalma. A feladatban használt PC floppy meghajtó reményeim szerint mégis tudja kezelni, hiszen az valójában csak alacsony szintű hozzáférést biztosít a lemezhez, a rajta lévő adatok feldolgozásáról az alaplapi processzoros rendszer gondoskodik.

2. fejezet

A hardver

2.1. Bemutatom a fejlesztőkártyát

A feladat megoldásához egy X-SP6-X9 jelzésű Xilinx Spartan 6 alapú FPGA-fejlesztőkártya állt rendelkezésemre. Számos szabadon felhasználható és széleskörűen konfigurálható IO kivezetésén kívül előnye, hogy tartalmazza az általános célú fejlesztésekhez szükséges perifériákat, nyomógombokat, kapcsolókat, LED-eket. Külső JTAG programozóval tölthető fel rá a bitstream, de egy 16MB-os SPI flash is helyet kapott, melyről szintén elvégezhető a konfiguráció. 5V-os tápellátással rendelkezik, viszont az IO lábak 3.3V-nak megfelelő CMOS logikai jelszinttel dolgoznak.



2.1. ábra. Illusztráció a megoldáshoz használt FPGA-fejlesztőpanelről

Mivel a soros busz és az író-olvasó egységek ként szolgáló PC meghajtó eltérő logikai jelszinteket használ, mint az FPGA, így mindenkor szükséges a valóságban realizálni egy szintillesztő áramkört. Ezt egy interfészpanel formájában terveztem és valósítottam meg.

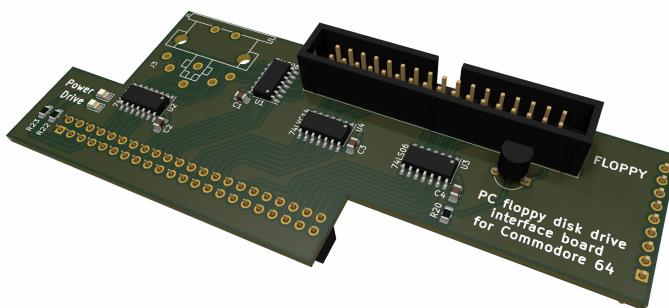
2.2. Az interfészpanel

Már a 1581-es kapcsolási rajzában is látható, hogy mind a soros port, mind a lemez-meghajtó esetén is meghajtóból származó fokozatokat, nyitott kollektoros buffereket vagy invertereket alkalmaztak, $1k\Omega$ -os felhúzó ellenállásokkal kiegészítve. Ezeket az áramköri elemeket a szintillesztésre való tekintettel a valóságban is realizálnom kellett.

Az interfészpanel tervezésekor az eredeti terveknek megfelelő tulajdonságú integrált áramköri alkatrészeket választottam azzal a kritériummal kiegészítve, hogy a szükséges jelszint-illesztési feladatot is képesek legyenek ellátni. Gazdaságossági okokból, valamint az áramkör egyszerűsítése végett az összes kimeneti vonalat 74LS06 típusú nyitott kollektoros inverterrel, a bemeneteket 5V toleráns bemenetű 74LVC14 típusú CMOS inverterrel kapcsoltam a rendszerhez. Ebből a megoldásból ugyan az következik, hogy az eredeti kapcsolással ellentétben néhány jel így invertáltan áll rendelkezésre, de ezek visszaállítását az FPGA-n megvalósított rendszer implementálásakor figyelembe vettetem.

Az illesztő áramkör kapcsolási rajza nem bonyolult, külön vettem rajta a soros porthoz és a floppy meghajtóhoz tartozó részeket, valamint helyet kapott rajta az eredeti 1581-es előlapján megtalálható két státuszjelző LED is. Az eredeti kapcsolási rajznak megfelelően a soros port kétirányú jeleit is különbözően elhelyeztem.

Az áramkörnek két tápfeszültségre van szüksége, egy 3.3V-osra, és egy 5V-osra, minden kettőt az FPGA-fejlesztőpanelről biztosítottam. Ehhez rendelkezésemre állt a fejlesztőpanel kapcsolási rajza és lábkiosztása. A nyomtatott áramkört úgy terveztem meg, hogy közvetlenül illeszkedjen az FPGA-panel tüskecsatornára. A huzalozás és a lábkiosztás megtervezésekor törekedtem arra, hogy a lehető legkevesebb kereszteződéssel elférjenek egymás mellett a vezetőszálak. Ergonómiai szempontokat is figyelembe vettetem a tervezés során, hogy az FPGA-kártya kialakításának megfelelően maradjon hely a programozó csatlakozó, a meghajtó azonosítójának beállítására használt DIP kapcsolók és a RESET gomb számára, és ne takarja el az interfészpanel ezeket a felületeket, miközben rajta van. Hasonlóképp helyeztem el a floppy meghajtó csatlakoztatására szolgáló IDC csatlakozót és a soros port számára a 6 lábú Tuchel aljzatot. A mechanikai kialakításkor nehézséget okozott, hogy nem volt pontos műszaki rajz az FPGA-panel oldalsó tüskecsatornáknak pontos helyéről, így azt becslés alapján tudtam csak pozicionálni, ez elég pontosan sikerült is.



2.2. ábra. A saját tervezésű interfészpanel 3D terve

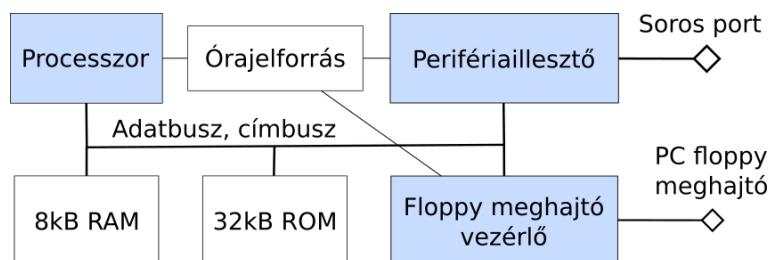
3. fejezet

Digitális tervezés

Az FPGA-ra a Xilinx ISE 14.3-as verzójú fejlesztői környezetének segítségével fejlesztettem a rendszert. Ez nem egy kimondottan korszerű ám szerencsére hibamentes szoftver. Azért ezt használtam mégis, mert a Spartan 6-os FPGA-kat a szoftver korszerű alternatívája (a Vivado) már nem támogatja. A leírófájlokat, modulokat és testbench-eket Verilog nyelven készítettem el, viszont helyenként szükséges volt VHDL-ben implementált részekkel is foglalkoznom később a hibakeresés során.

3.1. Modulok

A 1581-es meghajtót egy MOS 6502 típusú processzoron futó operációs rendszer irányítja. Az alaplapon megtalálható a firmware bináris állományát tartalmazó 32kB ROM és 8kB RAM. A perifériák illesztését egy 8520-as komplex interfész adapter (CIA), a lemez írólvasó egység vezérlését egy WD1772 típusú floppy meghajtó vezérlő (FDC) integrált áramkör végzi. Ezek képezik a főbb logikai egységeket, így kézenfekvő volt modulba szervezni őket, és így beilleszteni a projektbe.



3.1. ábra. Az FPGA-n megvalósított rendszer sematikus rajza

Korábbi projektek eredményét felhasználva rendelkezésemre álltak a CPU, a CIA és az FDC implementációi, ezeket átvettettem, esetenként továbbfejlesztettem munkám során. Az órajel-generátort, a RAM és ROM modulokat én készítettem el.

A rendszer 8-bites adatbuszt és 16 bites címbuszt használ, 2MHz-es rendszerórájelről jár minden a floppy vezérlő kivételével, mely 8MHz-es órajelet igényel. Ezen órajelek előállítása egy 4-bites számlálóval történt eredetileg, egy 16MHz-es oszcillátor jelének

leosztásával. Érdekesség, hogy az ehhez alkalmazott 74LS93-as IC konfigurálható biteszámú: 3 vagy 4 bites számlálót is lehetséges kialakítani a segítségével. Egy ebben a témaban végzett korábbi próbálkozás során a nem megfelelő konfiguráció jelenthetett hibalehetőséget.

Az eredeti kapcsolásban az egyes logikai egységekhez tartozó címdekódolást és engedélyező jelek előállítását kombinációs hálózattal kiegészített multiplexerek végzik, a memóriatérkép az alábbiak szerint alakul:

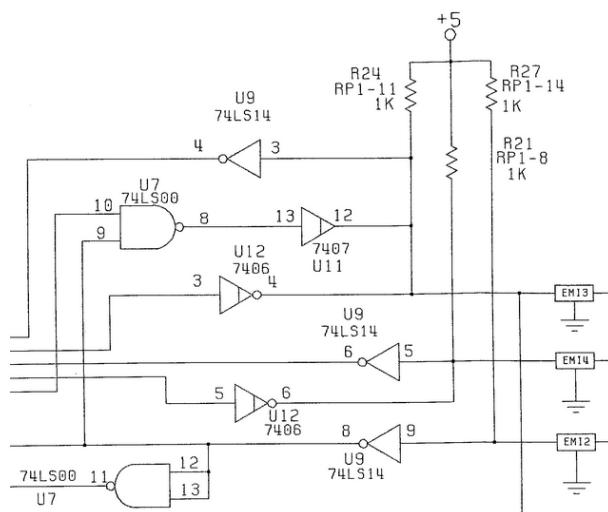
3.1. táblázat. Az 1581-es meghajtó processzoros rendszerének címtartomány-kiosztása

Eszköz	Címtartomány
RAM	0x0000 - 0x1FFF
CIA	0x4000 - 0x400F
FDC	0x6000 - 0x6003
ROM	0x8000 - 0xFFFF

A címdekóder leírása verilogban:

```
assign CSn_RAM = addr[15:13]!=3'b000;
assign CSn_ROM = !(rw && addr[15]);
assign CSn_CIA = addr[15:13]!=3'b010;
assign CSn_FDC = !(addr[15:13]==3'b011);
```

Mivel 1581-es meghajtóban gyakran használnak nyitott kollektoros meghajtású logikai elemeket, így kézenfekvő, hogy a felhasznált logikai kapuk számának csökkentése végett huzalozott módon valósítsanak meg bizonyos logikai függvényeket. Erre az FPGA-n technológiai okokból nincs lehetőség, ezért a huzalozott logikájú részeket átterveztem, hogy logikai kapukkal is le lehessen azokat írni.



3.2. ábra. Részlet az eredeti meghajtó kapcsolási rajzából.

Huzalozott logikai kapcsolat megvalósítása a soros busz $\overline{\text{DATA}}$ vezetékén, és a neki megfelelő verilog-leírás:

```
assign DATA_OUT = (~SP_OUT) || DATA_OUT_AUX || (ATN_IN && ATN_ACK);
```

Bizonyos VHDL modulok implementációjában előfordult, hogy kétirányúként definiáltak jeleket, például az adatbusz esetén. Ennek használata nem szerencsés, mert bár a valóságban tényleg kétirányú ugyanaz a vonal, az FPGA-ra két különböző irányú jelvezetékként szintetizálódik, ezért a könnyebb érhetőség és a többi modullal való kompatibilitás kedvéért már a leírófájlban szétbontottam két különböző vezetékre. Hasonlóképp előfordult ennek a fordítottja is, amikor az eredeti kapcsolás tartalmazott kétirányú adatforgalom szétválasztására használt multiplexert, melyet így el lehetett hagyni az implementációban.

3.2. CPU

A 6502-es CPU egy NMOS technológiával készült 8-bites processzor. Adatbusza 8 bites, címbusza 16 bites, így összesen 64kB memóriát képes megcímzni. Kevés belső regisztere van: egy akkumulátor (A), két indexregiszter (X és Y) melyek különböző címzési módokhoz, címtolásra is felhasználhatók, egy stack pointer (S) egy állapotregiszter (P), és egy 16-bites programszámláló (PC). A memória (virtuálisan) 256-bájtos blokkokba, lapokba van szervezve.

A stack hardveresen rögzített helyen, az első lapon, a 0x0100-as memóriacímtől kezdődik, és 0x01FF-ig tart. Külön figyelmet érdemel még a 0x00-s memórialap (zero-page) mely gyorsabban hozzáférhető, ha a parancsok dedikált címzési módú verziójával érjük el az ott tárolt változókat. Az állapotregiszter bitjei flag-ek, logikai és aritmetikai műveletek eredményeivel kapcsolatos tulajdonságokat jeleznek, a szokásos előjel- (N), nulla- (Z), átvitel- (C) és túlcordulás (V) biteken kívül tartalmaz egy BCD aritmetikát (D) és megszakításokat engedélyező bitet (I), valamint egy szoftveres megszakításkérés jelzésére szolgáló flag-et is (B). A reset vektor a 0xFFFFC 0xFFFFD címeken található, a megszakításvektor pedig a 0xFFFFE 0xFFFFF címeken - ezekről a helyekről töltődik be a programszámláló értéke a processzor indulásakor, illetve megszakításkérés esetén.[18]

A eredeti 6502-es processzor a ϕ_0 bemeneti órajelből két másik, fázisban egymáshoz képest 90°-kal eltolt és egymással nem átlapolódó órajelet állít elő a rendszer többi része számára. Ezek közül a ϕ_2 -t használja csak az eredeti terv alapján a meghajtó, mely az eredeti órajellel megegyező fázisú.

A processzor implementációját többször lecseréltem a fejlesztés során. Először az eredeti MOS 6502-es processzor tranzisztor-szintű felépítése alapján generált verilog-modullal próbálkoztam.[10] Sajnos az automatikusan generált leírófájl monolitikus felépítésű és gyakorlatilag átláthatatlan, így mivel eleinte nem sikerült működésre bírni és a belső jelek vizsgálata nagyon körülményesnek bizonyult, hamar lecseréltem az Arlet Ottens-féle verilog-6502 core-ra.[15] Ez az implementáció az eredetivel ellentétben az FPGA-n való megvalósíthatóság és a szemléletes működés kedvéért mikroprogramozott architektúrájú, a benne található állapotgép ráadásul aszinkron működésű. Ebből adódóan egy speciális, impulzusszerű órajelet igényel, hogy a processzormag mikro-utasításai biztosan le tudjanak futni az adott órajelciklusban. Technológiai megfontolásból FPGA-s rendszereken alapvetően szinkron logikai hálózatokat érdemes megvalósítani, így konzulensem

javaslatára elvettem ezt a megoldást is, és a fejlesztők körében gyakran alkalmazott T-65 nevű 6502 processzormag-implementációt használtam a továbbiakban.[8]

Ez a modul több kimenettel is rendelkezik, mint a 6502-es processzor, a nem használt jeleket bekötetlenül hagytam. Amelyik bemeneteket a kapcsolási rajz szerint az 5V-os tápfeszültségre kell felhúzni, azokat egész egyszerűen logikai magas szintre kötöttem.

3.3. CIA

A 8520-as komplex interfész adapter (Complex Interface Adapter, CIA) IC két 8-bites IO portot, két 16-bites időzítőt, egy valós idejű órát és egy dedikált soros port perifériát tartalmaz. A 1581-es alaplapján a CIA az egyedüli periféria, mely képes megszakításokat kérni, ha valamelyik időzítője lejár vagy valamilyen esemény következik be a nagy sebességű soros porton. A megszakításérés egyenként engedélyezhető, valamint a megszakítás forrása is lekérdezhető a CIA megszakításvezérlő regisztere segítségével. Hasonlóképp konfigurálhatók az időzítők, a valós idejű óra, és a két IO port paramétereit. A CIA A és B jelű IO portjaira csatlakoznak az eszköz azonosítóját kiválasztó DIP kapcsolók, a visszajelző LED-ek, valamint a soros busz jelei, irány szerint különválasztva. A csak bemenetként használt lábakra visszavezettem ugyanazon láb kimeneti irányú jelének értékét, hogy egy esetleges visszaolvasáskor jelen legyen rajta a valóságnak megfelelő logikai érték.

Érdekesség, hogy az IEC-busz jeleit az általános IO lábak segítségével is meg tudja hajtani, sőt alapvetően a 1581-es meghajtó ezt is használja az egyébként nagyobb sebességű dedikált soros periféria helyett. Ennek történelmi okai vannak, a Commodore 64-es számítógép egy tervezési hibából adódóan nem képes olyan gyorsan feldolgozni a nagysebességű soros porttal küldött adatot, ahogyan az ki tudná szolgálni. Később ezt javították, és a Commodore 128-as számítógép esetén már a soros busz SRQ jele szolgált (az eredeti specifikációtól eltérően soros órajelként) a gyors átviteli protokoll használatának kiegészítésére.[4]

A valós idejű órát nem az eredeti rendeltetési céljának megfelelően használja a meghajtó, viszont később a firmware-rel való ismerkedés során egy érdekes tervezési megoldásra derült fény: az alaplapot kétféle különböző típusú floppy vezérlővel is fel lehet szerelni, ez lehet WD1770 vagy WD1772. Az alaplapon egy átkötés beültetésével vagy elhagyásával lehet jelezni a firmware számára, melyik típusú IC szerepel a kapcsolásban, hiszen a két esetben eltérő rutinokat kell meghívnia a processzornak.

Mivel az összes általános célú IO láb foglalt már, a valós idejű órát léptető TOD láb vagy egy ellenálláson keresztül 5V-ra, vagy pedig a 2MHz-es rendszerórájelre kapcsolódik az átkötés jelenlététtől függően. Szoftveresen ez úgy ellenőrizhető, hogy ha egy adott idejű szoftveres várakozás után megváltozik a valós idejű óra regiszterének értéke akkor ott az átkötés, egyébként nincs.

A 8520-as CIA implementációját a C64-MiSTer elnevezésű projektből vettettem át. Ennek lényege, hogy egy egész Commodore 64 számítógép hardverleírását tartalmazza a MiSTer nevű nyílt forráskódú hardverkörnyezetre, melyen többféle retro számítógép és játékkonzol modern technológiákkal (FPGA) való megvalósítására terveztek. [14]

3.4. FDC

A hajlékonylemezes meghajtó vezérlő (Floppy Disk Controller, FDC) egy dedikált integrált áramkör, mely a floppy író-olvasó modul vezérlőjeit állítja elő közvetlenül, valamint ez fogadja a lemezmeghajtóról érkező nyers adatokat, visszajelzéseket is. Működését tekintve egy állapotgépet valósít meg, mely szoftver oldalról magas szintű parancsok regiszterbe írásával vezérelhető.[20]

A CIA-hoz hasonló módon, ehhez az IC-hez tartozó intellectual property-t is egy kész projektből emeltem át. A Suska projekt alapvetése, hogy egy Atari ST számítógép minden alkatrészét egyesével kicserélte a fejlesztő az annak megfelelő FPGA-s realizációjára.[9]

A kapcsolási rajz alapján érdekes módon vannak a floppy meghajtónak bizonyos jelei, melyek nem, vagy nem kizárolagosan az FDC chiphez vezetnek, ilyenek például a motorvezérlés vagy az írásvédettség ellenőrzésére szolgáló vonalak. Az is előfordul, hogy egyes kimenetek egyáltalán nincsenek is bekötve az IC-n, ezeket az hardverkonfiguráció során is bekötetlenül hagytam. Abban sincs egységesség, hogy a floppy meghajtó vezérlőjelei közül melyik használ ponált és melyik negált logikájú digitális jeleket. Ez különös figyelmet igényelt a a top modul huzalozásakor, mivel az interfészpanelen minden jelet invertálnak a szintillesztő áramkörök.

A PC-s floppy meghajtó és a meghajtóvezérlő IC közt némi kiegészítő logikára volt szükség a READY jel előállításához. A Shugart-kiosztású meghajtók specifikációja alapján sejtettem, hogy a READY jel előállítható egy SR-flipflop segítségével a motor engedélyező és az INDEX jeleket bemenetként felhasználva. Ezt alátámasztotta az a tény, hogy léteznek kifejezetten arra a célra készült adapter-áramkörök, hogy PC-s floppy meghajtókat illesztenek Shugart-interfészről használó számítógépes rendszerekbe.[19] Ezekben a kapcsolásokon egy 4 darab 2 bemenetű NOR kaput tartalmazó chip található, melynek működését a szabadon elérhető NYÁK-tervek alapján visszafejtettem, és kiderült, hogy valóban egy SR-flipflopot és két invertert valósít meg.

A READY jelet előállító logikai függvény verilog-implementációja:

```
reg RDY;
initial RDY <= 1'b1;
always @(posedge clk) begin
    if(!rstn)
        RDY <= 1'b1;
    else begin
        case({MOTB, INDEX})      //motor on? index?
            2'b00: RDY <= 1'b1;
            2'b01: RDY <= 1'b1;
            2'b10: RDY <= RDY;
            2'b11: RDY <= 1'b0;
        endcase
    end
end
```

3.5. Órajel-generátor

Az többi modulhoz szükséges órajeleket ezzel a modullal állítottam elő. 16MHz-es rendszerórajelet feltételeztem, melyet a floppy vezérlő egy az egyben megkapott, a többi modul számára az ebből leosztott 2MHz-es ϕ_2 jelet adtam. A processzor implementációjából kihagyták az eredeti órajel-kondicionáló részt, mely a ϕ_0 -ból állítja elő a kétfázisú rendszerórajelet, de erre egyszerűbb nincs szükség ebben a megoldásban, másrészt úgyis az órajel-generátor modul a felelős az összes szükséges órajel előállításáért.

A frekvenciaosztást egy egyszerű bináris számláló végzi. Mivel az Arlet-féle processzor-implementációhoz aszimmetrikus órajelre volt szükség, így a 2MHz-es órajel-impulzusokat komparálással állítottam elő, és ezen a későbbiekben sem módosítottam, mivel ugyanolyan jól működött.

A CIA implementációjának érdekes módon szüksége van egy invertált ϕ_2 jelre is, ezt nem az órajel-modulban állítottam elő, hanem a bekötésnél vezettem rá az negált órajelet.

A floppy vezérlő modul specifikációja szigorúan előírja a 16MHz-es órajel használatát, mivel az FPGA-n való megvalósítás miatt belső állapotgépének léptetéséhez és a vezérlésbe iktatott időzítések pontos megvalósításához nagyobb frekvenciára van szükség, mint az eredeti hardver esetén.

3.6. Memóriák

A meghajtó eredeti kapcsolásában aszinkron RAM és ROM chipet alkalmaztak, az adat- és címbusz meghajtását engedélyező jeleket címdekóderrel állítják elő. Én szinkron logikai vezérléssel biztosítottam a hozzáférést minden kétfajta memóriához. A RAM és a ROM is bájtos szervezésű, azaz egy memóriacímmel egy bájt összes bitjéhez hozzá lehet férfni. Ha olvassuk a memóriákat, akkor ezek a bitek kikerülnek az adatbuszra, ha írjuk a RAM-ot, bekerül az adatbuszon lévő bájt az adott című helyre.

Megvalósítás szempontjából tulajdonképpen minden memória egy-egy nagy regisztertömb. Arra kell külön figyelni, hogy amikor nincsenek megcímezve, ne hajtsák meg adatbuszt. Ez verilogban a Z logikai állapot használatával valósítható meg.

Mivel a ROM tárolja a 1581-es firmware-ét vagyis a CPU-n futó programot, így azt a rendszer konfigurálásakor bele kell azt valahogyan tölteni. Ehhez a \$readmemh függvényt használtam, mely egy megadott helyen található bináris állományával inicializálja a paraméterként megadott regisztertömböt.

A bináris állomány rendelkezésemre állt, ugyanarról a weboldalról elérhető, ahol a 1581-es szervizkönyvét is megtaláltam. Mivel a verilog hex fájlt vár a \$readmemh paramétereinként, át kellett alakítanom a bináris állományt a megfelelő formátumúra. Linux alatt a következő bash-szkripttel végeztem ezt el:

```
for FILE in *.bin; do
    hexdump -v -e '1/1 "%02X\n"' $FILE > ${FILE%.bin}.mem;
done
```

4. fejezet

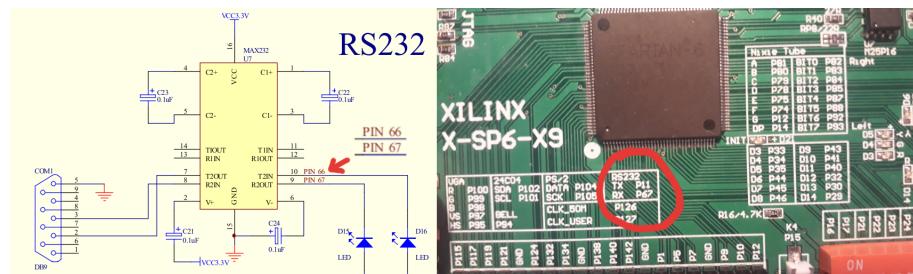
Tesztelés, hibajavítás

A feladat hátralevő részében az elkészült hardver-elemek tesztelésével, az esetleges hibák felderítésével és azok kijavításával foglalkoztam.

4.1. Az interfészpanel tesztje

Miután megérkezett a gyártásból az interfészpanel amit terveztem, elvégeztem az alkatrések beforrasztását, és egy példaprojektet készítettem abból a célból, hogy magának a panelnek a működését ellenőrizni tudjam. Ehhez elkészítettem az FPGA-panel tüskesorának hozzárendelését a verilog-modul jeleihez egy UCF-fájl segítségével. Az interfészpanel 5V-os jeleire egy próbapanel segítségével LED-eket illesztettem, melyeket a tesztprojektben sorban egymás után villantottam fel, így megbizonyosodhattam a szintillesztő áramkör helyes működéséről. Ekkor vettet észre azt is, hogy a meghajtó foglaltságát jelző zöld színű stáruszjelző LED-et fordított polaritással forrasztottam rá az interfészpanelre, mivel az alkatrészen megtévesztő módon szerepelt a polaritásmegjelölés - így ezt kijavítottam.

A lábkiosztás megtervezésekor törekedtem arra, hogy az FPGA panel olyan lábait használjam, melyeknek nincs alternatív funkciója, ezzel csökkentve annak a valószínűségét, hogy a fejleszőpanelen található áramköri elemek vagy perifériák módosítják az áramköröm tervezett viselkedését. A teszt során nyilvánvalóvá vált, hogy az FPGA-panel specifikációjától eltérő módon a P11-es lábnak más funkciója is van, mégpedig az integrált soros port TX vonalát is ez táplálja. Mivel ez nincs használatban szerencsére nem okozott gondot, csak bosszantó, hogy világít az alaplapon egy extra led, valamint ezek után kevésbé megbízható dolog pusztán a dokumentációra támaszkodni.



4.2. Verilog-modulok

Először a saját készítésű verilog-modulokhoz készítettem külön-külön tesztkörnyezeteket, és működésüket az ISE szimulátora segítségével végeztem. Először az órajelgenerátort, majd a memória-modulokat teszteltem, végül pedig egy olyan összeállítást is elkészítettem, ahol a RAM és a ROM is a rendszer buszaira kapcsolódik, és ellenőriztem az engedélyező logikát valamint azt, hogy a megfelelő memóriacímek használatakor nem hajtanak-e szembe.

Mivel úgy tűnt, hogy helyesen működnek ezek a modulok, elkezdtem a rendszer többi részének integrációját, egyesével vettem hozzá a projekthez a processzor, a CIA és a floppy vezérlő implementációjának egy-egy példányait.

4.3. A firmware

Miután a processzor is bekerült a top modulba, a tesztelést a ROM-ba betöltött firmware bináris segítségével végeztem: a processzor ekkor már az eredeti 1581-es meghajtóra készült kódott futtatta, így biztos lehettem benne, hogy a szoftverben nincs hiba. A bináris állományt már csak azért sem lett volna érdemes piszkálni, mivel tartalmaz egy ellenőrzőösszeget is, amivel a program indulásakor a ROM tartalmának sértetlenségét ellenőrzi. Érdekes tapasztalat volt úgy fejleszteni, hogy egy hibamentes, kész szoftver alatt kellett megvalósítani azt a hardvert, mely az eredeti specifikációnak megfelelő működést hivatott biztosítani.

További nehézséget jelentett, hogy magának a firmware-nek a működése csak az eredeti, assemblyben megírt forrásfájlok kommentjeiből, valamint egy német nyelven kommentezett dissassembly listing fájlból volt kideríthető.[7] Természetesen a 1581-es meghajtó felhasználói kézikönyvében számos információ utal arra, hogyan kell működnie bizonyos esetekben a meghajtónak, a részletes okokat csak a hardver és a firmware átfogó ismeretében lehet tudni.

A szimulátor segítségével nyomon követhető volt a program futása, az FPGA-panel IO lábai mellé a processzor és a rendszer belső jeleit is felvettettem a megjelenített hullámformák közé, így láthatóvá váltak a buszműveletek, a programszámláló és a különböző regiszterek értékei is. A program futása során először különböző hardveres teszteket hajt végre: inicializálja a perifériákat, leellenőrzi a ROM sértetlenségét az ellenőrzőösszeg alapján, különböző értékekkel tölti fel a RAM-ot majd vissza is olvassa azokat. Az ekkor fellépő hibajelenségeket a hiba okától függően a visszajelző LED-ek megadott számú felvillanásával lehet azonosítani.

Miután lefutottak a tesztek, elalszik a 'Drive' feliratú státusz visszajelző LED, és a firmware elindít egy operációs rendszert. Az ütemezőt az interfész adapter Timer B időzítője hívja meg, másodpercenként 100-szor. Az operációs rendszer prioritásokkal megadott JOB-okat futtat, egyszerre legfeljebb 8-at képes nyilvántartani. Futáskor a JOB kódokat a prioritásuknak megfelelő sorrendben fix memóriacímeken tárolja, a feladat befejezésekor pedig a visszatéréskor adott kód (hibakód) kerül a helyére.

A szimuláció kezdeti szakasza alapján működni látszott a processzor, így megpróbáltam az FPGA-ra szintetizálva is kipróbálni a rendszert. Azt tapasztaltam, hogy sosem aludt el a zöld LED, és valóban nem is jutott túl az inicializálási állapotba a meghajtó. Mivel a hardveren nem állt rendelkezésre debugolásra vagy legalábbis a program futásának nyomon követésére alkalmas interfész, visszatértem a szimulációhoz. Hosszabb ideig futtatva azt, kiderült, hogy végtelen ciklusba kerül a program és egy olyan változó értékének 0-ra csökkenésére vár, amelyet a program további futása során nem módosít semmi.

Az assembly listing alapján kiderítettem, hogy a szóban forgó változó egy szoftveres várakozáshoz használt számláló, melynek értékét az időzítő lejártával kiszolgálásra kerülő megszakítás rutin csökkenti eggyel. A probléma, hogy nem jut sose érvényre ez a megszakítás. Mivel a jelenség okának felderítéséhez ezen a ponton már nem volt kényelmes pusztán a szimulátor környezetben vizsgált jelek alapján kitalálni, mi okozza a program futásában fellépő eltérést (annak tudatában, hogy a program viszont biztosan jól működne egy hibamentes hardveren), más megközelítéssel próbálkoztam.

4.4. Emulátor

Az ugró utasítások és ciklusok nyomon követése nagyon megnehezítette a debugolást, mivel az ISIM programban csak lineárisan követhető a program futása. Ráadásul a memória tartalma csak a szimuláció végével elért állapotot tükrözi, a kurzorok állításával nem kérdezhető le a szimuláció egy korábbi időpontjához tartozó állapot. Emiatt a változók és a stack tartalmának vizsgálata, valamint az eltérések észrevétele csak egy újabb szimuláció segítségével volt megtehető, ami viszont jelentősen megnövelte a hibakereséssel töltött időt.

Ahhoz, hogy egyáltalán a program futásának nyomon követését hatékonyabban tudjam végezni, írtam egy emulátor programot C-ben, mely a rendszer funkcionális működésének megfelelően viselkedik. Léteznek ugyan online elérhető 6502-es processzorhoz készült disassemblyer és emulátor eszközök, melyek viszont elsősorban demonstrációs célúak, így nem bővíthetők vagy fejleszthetők tetszőlegesen. Az emulátor elkészítéséhez mindenkor szükséges volt megismernem a 6502-es processzor belső felépítését és működését, utasításkészletét, címzési módjait, és hogy melyik művelet milyen egyéb belső állapotokat változtat meg. Ebben segítségemre volt a 6502-es processzor programozói kézikönyve[1], valamint az utasításkészlet online példákkal kiegészített interaktívan kezrehető változata.[13]

Az emulátor program ugyanazt a bináris állományt futtatja, mint amit a ROM-ba is betölöttem, sőt felépítését tekintve a verilog-ban leírt rendszert tükrözi azzal az eltéréssel, hogy minden perifériához mint memória fér hozzá, a róluk beolvasható értékeket pedig az olvasás pillanatában állítottam elő a valóságnak (és a szimulációhoz) megfelelő módon.

Az emulátor segítségével lehetők a programba töréspontok, de más-milyen típusú feltétel esetén megállítható a program futása, és szemléletes módon megtekinthető a stack vagy a zero-page tartalma, hasonlóképp a CPU és a többi periféria regisztereinek is. A

hardveres megszakítás-kéréseket szintén a szimuláció és a kiváltó okok ismeretében az emulátorban szoftveresen valósítottam meg.

Az emulátor, a listing fájl, valamint papír és ceruza használatával sikerült visszakövetnem, hogy a kívánt állapot elérési láncában hol megy félre a program, és milyen értékek stack-re kerülésének köszönhető, hogy ez megtörténik.

Kiderült, hogy a processzor verilog-moduljában hibás a szoftveres megszakítást jelző flag implementációja, ugyanis a specifikációtól eltérő módon akkor is ugyanolyan értékkel kerül fel a stack-re ha hardveres megszakítás érkezik, mintha szoftveres. Fontos megjegyezni, hogy a státuszregiszterben nem változik meg sosem ennek a flag-nek az állapota, csak a megszakítás érvényre jutásakor kerül beállításra a státuszregiszter stack-re kerülő másolatában.

Miután megkerestem és javítottam a hibát a processzormag leírásában, a meghajtó sikeresen el tudta végezni a inicializációt a szimuláció szerint, és a valóságban is.

4.5. Tesztelés valódi környezetben

Ettől kezdve a meghajtó eseményvezérelt működését feltételeztem, így nem szimulátor vagy emulátor segítségével, hanem egy valódi, Commodore 64-es számítógéppel, PC-s floppy meghajtóval kiegészített teszkörnyezetben figyeltem meg az FPGA-panel működését. A vezetékeken jelen lévő jelszinteket egy Digilent Analog Discovery 2 típusú műszer 16 csatornás logikai analizátor funkciója segítségével mértem. A logikai analizátor úgy kapcsoltam a rendszerhez, hogy házilag kiegészítettem a floppy meghajtóhoz vezető szalagkábelt, és az interfészpanelt a számítógéppel összekötő soros buszt egy-egy extra hüvelyorszárral.

A meghajtó bekapcsolásakor figyelmes lettem arra, hogy a hajlékonylemez forgató motor egy adott ideig engedélyező parancsot kap majd leáll. Szimuláció szerint is ez a működés volt elvárható, viszont a floppy vezérlő állapotgépét vizsgálva figyelmes lettem arra, hogy mintha nem megfelelő állapotba kerülne a rendszer.



4.2. ábra. Szimuláció: a floppy-vezérlő hibás állapot-átmenete

Az FDC-nek adatlapja szerint a 0xD0 (Force Interrupt) parancs hatására IDLE állapotban kellene maradnia, hiszen így történik meg a periféria alaphelyzetbe állítása, ehenyett INIT, majd SPINUP állapotba kerül. (Érdekességképp megjegyzem, hogy ezután a floppy-vezérlő hat index-impulzus megérkezésére várakozik, hogy megbizonyosodjon arról, valóban forog a motor. Mivel azonban a floppy meghajtó motor engedélyező jele nem az FDC, hanem a CIA felől van meghajtva, ez nem következik be. Helyette egy másik leállási feltétel lép érvénybe, timeout-ol a rendszer, majd hibajelzés nélkül folytatja a programvégrehajtást.)

A jelenség oka az volt, hogy az FDC command regiszter írása szinkron, viszont az állapotgép "aszinkron", és már a chip select hatására továbblép, nem várja meg, hogy érvényre jusson az újonnan beérkező parancs. Megoldást jelentett a problémára, hogy a command regiszter írását az órajelhez szinkronizáltam:

```
CMD_WR <= true when
  CSn = '0' and A1 = '0' and A0 = '0' and RWn = '0' and rising_edge(CLK)
else false; -- Command register write.
```

Másik, nagyobb horderejű probléma akkor jelentkezett, amikor megpróbáltam parancsokat küldeni a számítógépről a meghajtónak. A soros buszon zajló adatforgalom könnyebb olvashatósága kedvéért írtam egy szkriptet a logikai analizátor felhasználói felületéhez, mely a Commodore IEEE-488 protokoll-specifikáció szerint beérkező handhsake-jelek és adatbitek alapján állítja vissza az adatokat olvasható formátumú hexadecimális értékekkel.

A Commodore számítógépről olyan parancsokat küldtem, melyek a floppy meghajtó működésének vizsgálatára alkalmasak: ilyen volt például a lemez formázása vagy a lemezen szereplő fájlok kilistázása. Ez utóbbinak természetesen akkor van a gyakorlatban értelme, ha valóban egy megfelelőképp formázott lemez van a meghajtóban, viszont az én floppy lemezeim eredetileg IBM PC-ken való használat számára voltak megformázva. A Commodore 64-es számítógépen BASIC nyelven lehetséges programokat írni és parancsokat végrehajtani, így a lemezkezelés is ezek segítségével történt.

Például a buszon lévő egyes eszközök megszólítására és az adott parancs elküldésére az OPEN15,8,15,"PARANCS":CLOSE15 szekvencia használható. A paraméterekben szereplő számok rendre a (tetszőleges 8-bites) csatornaszám, a meghajtó azonosítója (8), és a másodlagos címe.[11] A kiadott parancs bájtonként kerül elküldésre, az adatfolyamot egy nyitó- és egy záró vezérlőkód keretezi.

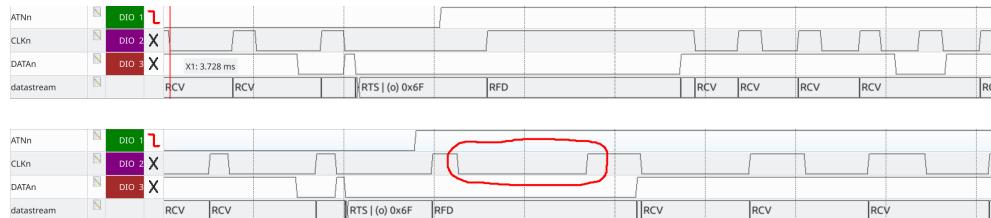
4.1. táblázat. Az IEC-busz vezérlőkódjai

Kezdőcím	Parancs leírása és paraméterei
\$20	LISTEN + azonosító
\$3F	UNLISTEN
\$40	TALK + azonosító
\$5F	UNTALK
\$60	OPEN CHANNEL/DATA + másodlagos cím
\$E0	CLOSE + másodlagos cím
\$F0	OPEN + másodlagos cím

A soros kommunikáció tesztelése végett a felhasználó kézikönyvben javasolt programrészletet használtam. Ennek segítségével lekérdezhető a meghajtó aktuális állapota, kiolvasható az esetlegesen fennálló hibakód. Érdekesség, hogy a meghajtó bekapcsolása vagy újraindítása után hibakód helyett a firmware verzióját küldi vissza.

```
10 OPEN15,8,15
20 INPUT#15,EN,EM$,ET,ES
30 PRINT EN,EM$,ET,ES
40 CLOSE15
RUN
```

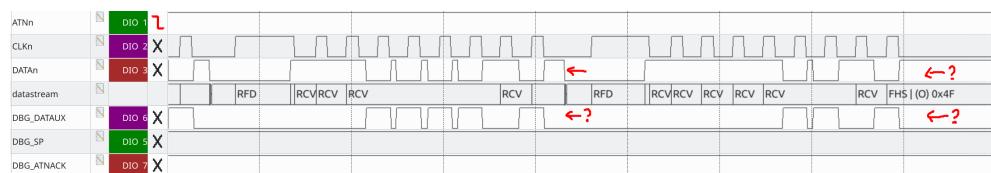
Azt tapasztaltam, hogy a parancsot elküldi a számítógép a meghajtónak, viszont válaszként nem érkezik meg az elvárt 73, "COPY RIGHT CBM DOS V10 1581" szöveg, helyette lefagyott a meghajtó, és nem fogadott több parancsot a következő újraindulásig. Kíváncsiságból elvégeztem egy referenciaimréést egy SDrive64 1564 típusú, SD2IEC SD-kártya adapterhez hasonló periféria eszközzel, amivel tudtam tesztelni, milyen - protokollra jellemző magas szintű - válaszokat várhatok az FPGA-n megvalósított meghajtóról.



4.3. ábra. Eltérés a handshake-jelek között

A két esetben eltérést tapasztaltam, mégpedig a kommunikáció irányának megváltoztatására szolgáló handshake-jelek tekintetében. A floppy meghajtó nem húzza le a busz CLK vonalát, ellentétben a specifikációval és az SD-kártya olvasóval, mely az ábrán is látható.

Annak megkülönböztetésére, hogy az adott pillanatban melyik eszköz hajtja meg a kétirányú adat- és órajel vonalat, külön állapotjelző jelek kivezetése volt szükséges az FPGA-ról, melyeket az interfészpanel szabadon maradt lábaira illesztettem, és mértem a logikai analizátorral.

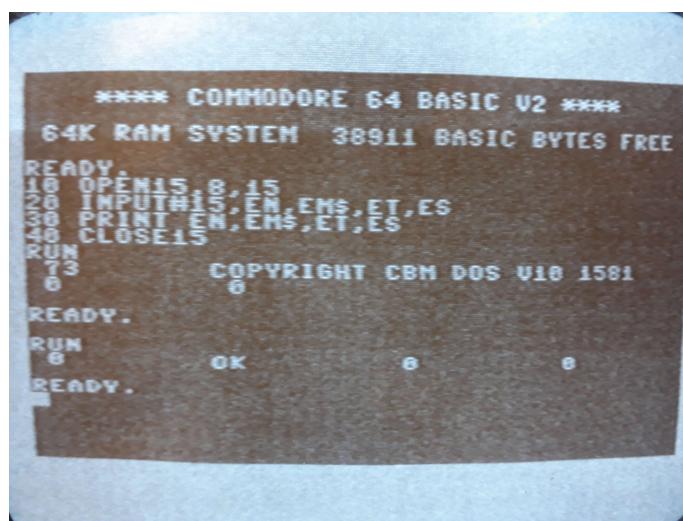


4.4. ábra. Megszakad az adatfolyam

Látható, hogy a Commodore 64 egyszer csak már nem húzza le a DATA vonalat. A nyilakkal jelölt helyen nem húzza le a vonalat a periféria (ez a DBG_DATAIN jel segítségével ellenőrizhető), tehát a számítógépnek kellett volna vezérelnie az adatbuszt a specifikáció megfelelően. Látható tehát, hogy a parancsot sikeresen fogadta a meghajtó, és meg is kezdte a válasz elküldését, melyet viszont valamilyen ok folytán nem tudott befejezni.

Egy konzultáció során felmerült a gyanú, hogy esetleg nem megfelelő órajel-frekvenciával hajtom meg a CIA-t, mivel a soros adatátvitelt az azon található Timer A periféria ütemezni. Ezt a mérési eredmények alá is támasztották. Elkerülte a figyelmem, hogy az FPGA panelen rendelkezésre álló órajelforrás 16MHz helyett 50MHz-es órajellel látja el a rendszert. Ugyan a soros busz szinkron adatátvitelt biztosít, a protokoll specifikáció tartalmaz olyan helyzeteket, melyek során kritikus betartani a pontos időzítést.

A megoldás egyben egy feladatot is rejtett magában, ugyanis mivel az 50MHz nem egész számú többszöröse a 16MHz-nek, így egyszerű osztással nem lehet azt előállítani belőle. Ehhez a Spartan 6 FPGA-ban található PLL modullal egészítettem ki az órajel-generátor verilog-modult, melyet az adatlapja alapján felkonfiguráltam és le is szimuláltam, mielőtt kipróbáltam a fizikai hardveren.[22]



4.5. ábra. Sikeres státusz-kiolvasás eredménye a képernyőn

Ezzel a megoldással már helyesen működött a soros port, viszont a meghajtót még mindig nem sikerült rábírnom a lemez megformázására. A parancs megérkezik a meghajtóra, és látszólag el is indul a formázás folyamata, azonban miután az író-olvasó fej ahelyett, hogy a szélső sávra lépés után egyesével belépkedne a lemez belső része felé némi várakozás után teljes sebességgel középre mozog és ott is marad, ezután pedig ismételten nem lehet megszólítani a meghajtót.

4.6. Záró gondolatok

A féléves munka végeztével ezeket az eredményeket sikerült elérnem. Megpróbálkoztam még a formázás során ütemezésre kerülő job működésének megismerésével, valamint a fej által írt/olvasott adatok dekódolására - a soros porthoz hasonlóan - készíteni egy szkriptet a logikai analizátor szoftveréhez. Idő hiányában egyelőre nem jártam sikkerrel.

Önálló laboratóriumi munkám során a mérőki feladatok egy új perspektíváját ismertem meg, főleg a reverse-engineering és a hardvertesztes terén. További lehetőségeknek fenntartom a 1581-es meghajtó befejezését, valamint a megszerzett tudás és tapasztalatok ismeretében hasonló témaiban tovább fejlődni.

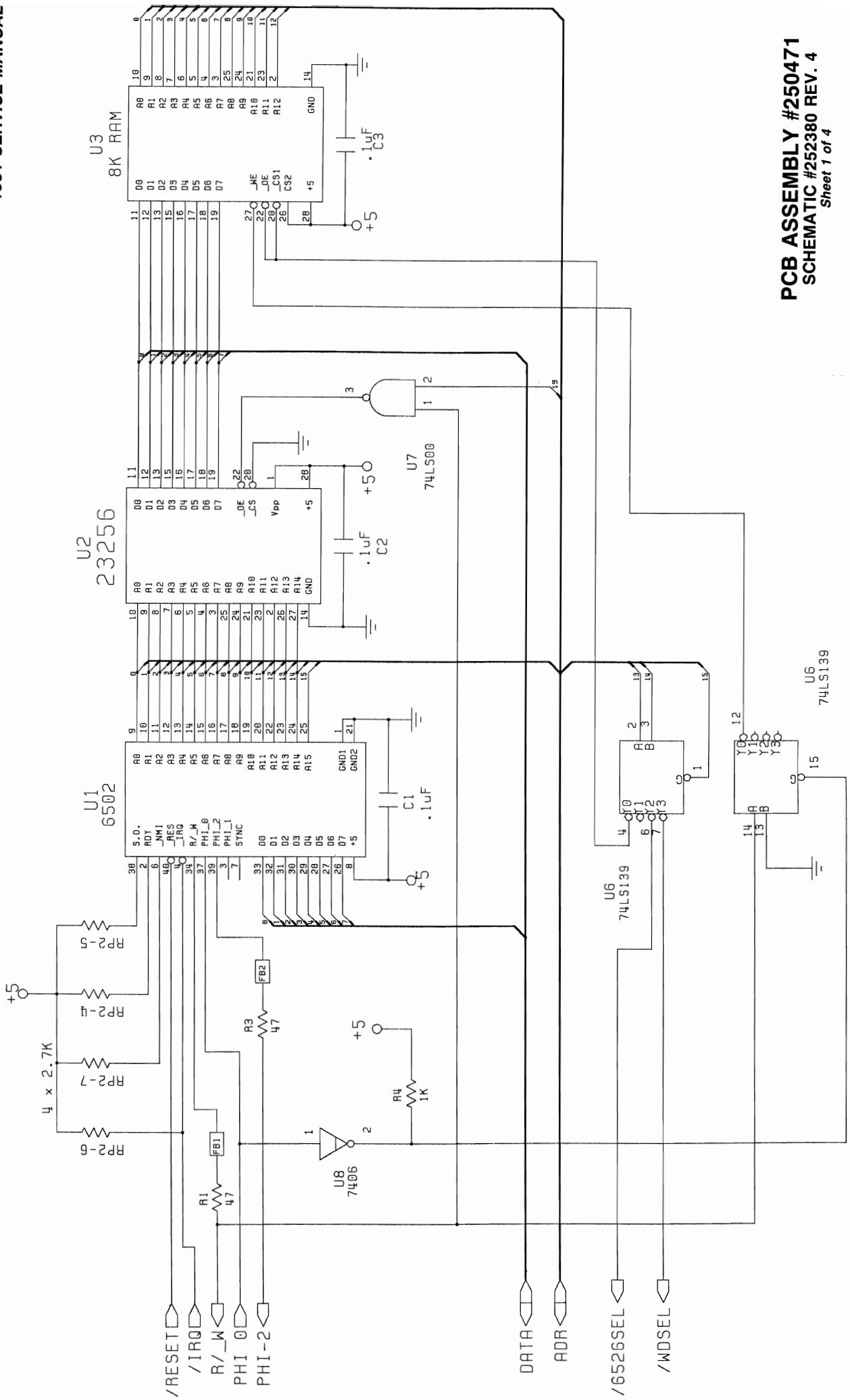
Irodalomjegyzék

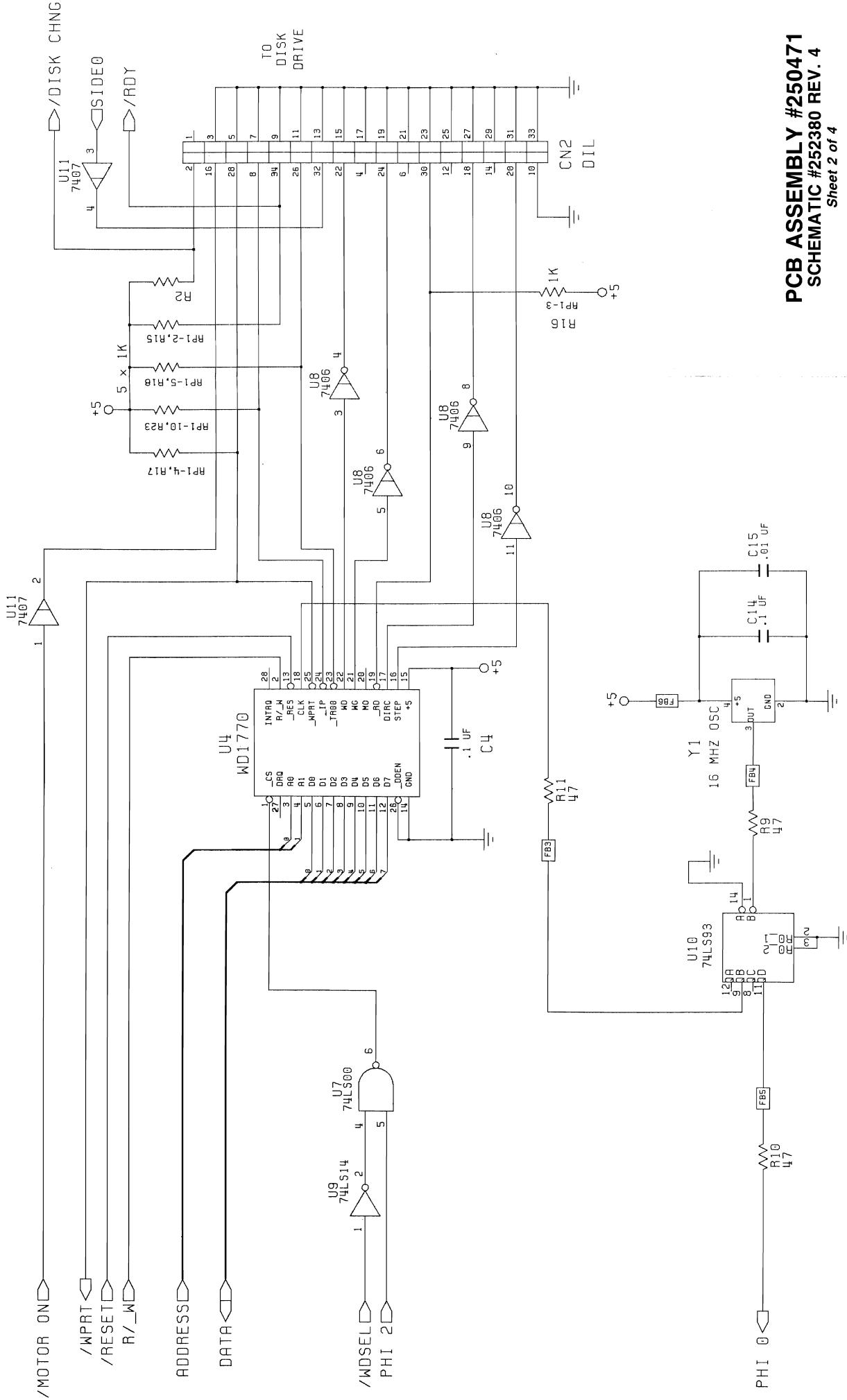
- [1] Bluechip. Rockwell 6502 programmers reference. <https://www.csh.rit.edu/~moffitt/docs/6502.html>, October 2003. [Online; accessed 15-dec-2022].
- [2] Thomas Bräse. Floppy disk drive / bus interface. <https://retrocmp.de/fdd/general/floppy-bus.htm>, 2022. [Online; accessed 15-dec-2021].
- [3] CHINON INDUSTRIES, INC. *Chinon F-354C 135TPI Double-Sided 3.5 In Specifications*. URL: http://www.bitsavers.org/pdf/chinon/Chinon_F-354C_135TPI_Double-Sided_3.5_In_Specifications.pdf.
- [4] COMMODORE SEMICONDUCTOR GROUP, Postscript-conversion by Markku Alén. *6526 COMPLEX INTERFACE ADAPTER (CIA)*, February 2001. URL: http://archive.6502.org/datasheets/mos_6526_cia_recreated.pdf.
- [5] NEC Corporation. *FDI036A 3.5" FLOPPY DISK DRIVE PRODUCT DESCRIPTION*, 1985. URL: <https://raw.githubusercontent.com/MiSTER-devel/AtariST-MiSTER/master/rtl/fdc1772/NEC%20FD1036%20Floppy.pdf>.
- [6] J. Derogee. *IEC dissected - IEC-bus documentation as used for the 1541-III*, February 2008. URL: <http://www.zimmers.net/anonftp/pub/cbm/programming/serial-bus.pdf>.
- [7] Ninja/The Dreams. All about your 1581-online help v0.13. <http://unusedino.de/ec64/technical/aay/c1581/>, 2002-2005. [Online; accessed 15-dec-2022].
- [8] FreeCores. t65. <https://github.com/freecores/t65>, 2005. [Online; accessed 15-dec-2022].
- [9] Experiment-S / Wolfgang Förster. Ein in vhdl modellierter open source ip-core mit atari st(e) funktionalität. <https://download.experiment-s.de/Configware/2K21A/rtl/vhdl.7z>, 2004. [Online; accessed 15-dec-2022].
- [10] Andrew Holme. Verilog 6502. <http://www.aholme.co.uk/6502/Main.htm>, 2016. [Online; accessed 15-dec-2022].
- [11] Commodore Electronics Limited. *1581 Disk Drive User's Guide*, 1987. URL: http://www.zimmers.net/anonftp/pub/cbm/manuals/drives/1581_Users_Guide.pdf.

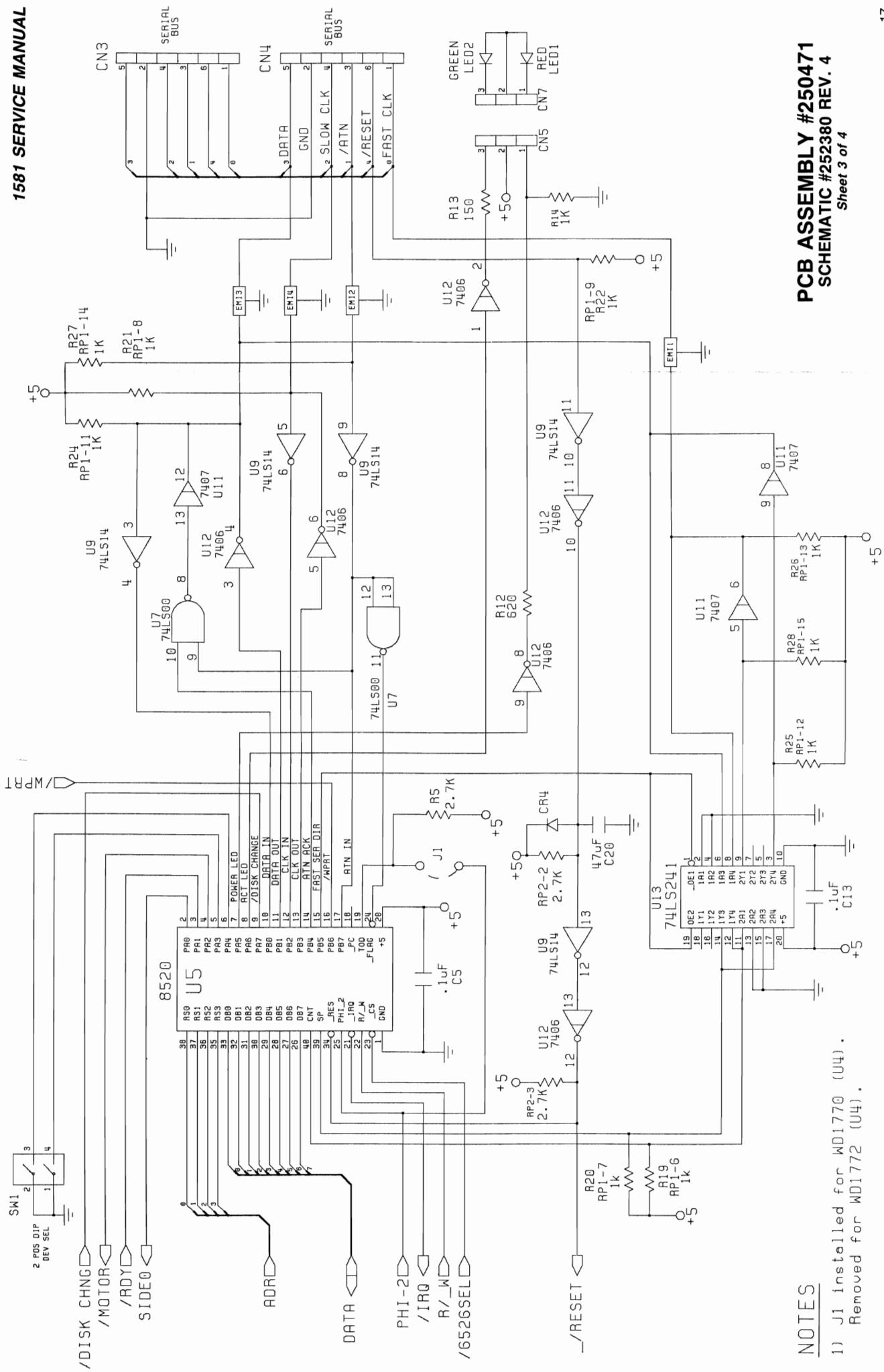
- [12] Commodore Electronics Limited. *SERVICE MANUAL 1581 3.5 DISK DRIVE*, June 1987. URL: [http://www.zimmers.net/anonftp/pub/cbm/schematics/drives/new/1581/1581_Service_Manual_314982-01_\(1987_Jun\).pdf](http://www.zimmers.net/anonftp/pub/cbm/schematics/drives/new/1581/1581_Service_Manual_314982-01_(1987_Jun).pdf).
- [13] mass:werk. 6502 instruction set. https://www.masswerk.at/6502/6502_instruction_set.html. [Online; accessed 15-dec-2022].
- [14] MiSTer-devel. C64_mister. https://github.com/MiSTER-devel/C64_MiSTER/tree/master/rtl/iec_drive, 2019. [Online; accessed 15-dec-2022].
- [15] Arlet Ottens. verilog-6502. <https://github.com/Arlet/verilog-6502>, 2016. [Online; accessed 15-dec-2022].
- [16] Sony Corporation. *MPF920-Z 3½" Floppy Disk Drive*, 2004. URL: <https://pro.sony/s3/cms-static-content/operation-manual/4668278111.pdf>.
- [17] Sven Olaf Kamphuis, Malvineous, Kikinou, Peter Bye, archyx et. al. Floppy diskdrive pinout and wiring. https://old.pinouts.ru/HD/InternalDisk_pinout.shtml, 2019. [Online; accessed 15-dec-2022].
- [18] Synertek. *SY6500 8-Bit Microprocessor Family*. URL: <https://www.princeton.edu/~mae412/HANDOUTS/Datasheets/6502.pdf>.
- [19] toms01. 1581-pc-drive-adapter. <https://gitlab.com/toms01/1581-pc-drive-adapter/-/tree/master/>, 2020. [Online; accessed 15-dec-2022].
- [20] WESTERN DIGITAL CORPORATION, Edited By Jean Louis-Guérin. *WD1772 Floppy Disk Formatter/Controller*, January 2015. URL: http://info-coach.fr/atari/documents/_mydoc/WD1772-JLG.pdf.
- [21] Wikipedia contributors. Commodore bus — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Commodore_bus, 2022. [Online; accessed 04-dec-2022].
- [22] XILINX. *Spartan-6 FPGA Clocking Resources User Guide (UG382)*, June 2015. URL: <https://docs.xilinx.com/v/u/en-US/ug382>.

Függelék

F.1. Az eredeti 1581-es floppy meghajtó kapcsolási rajza



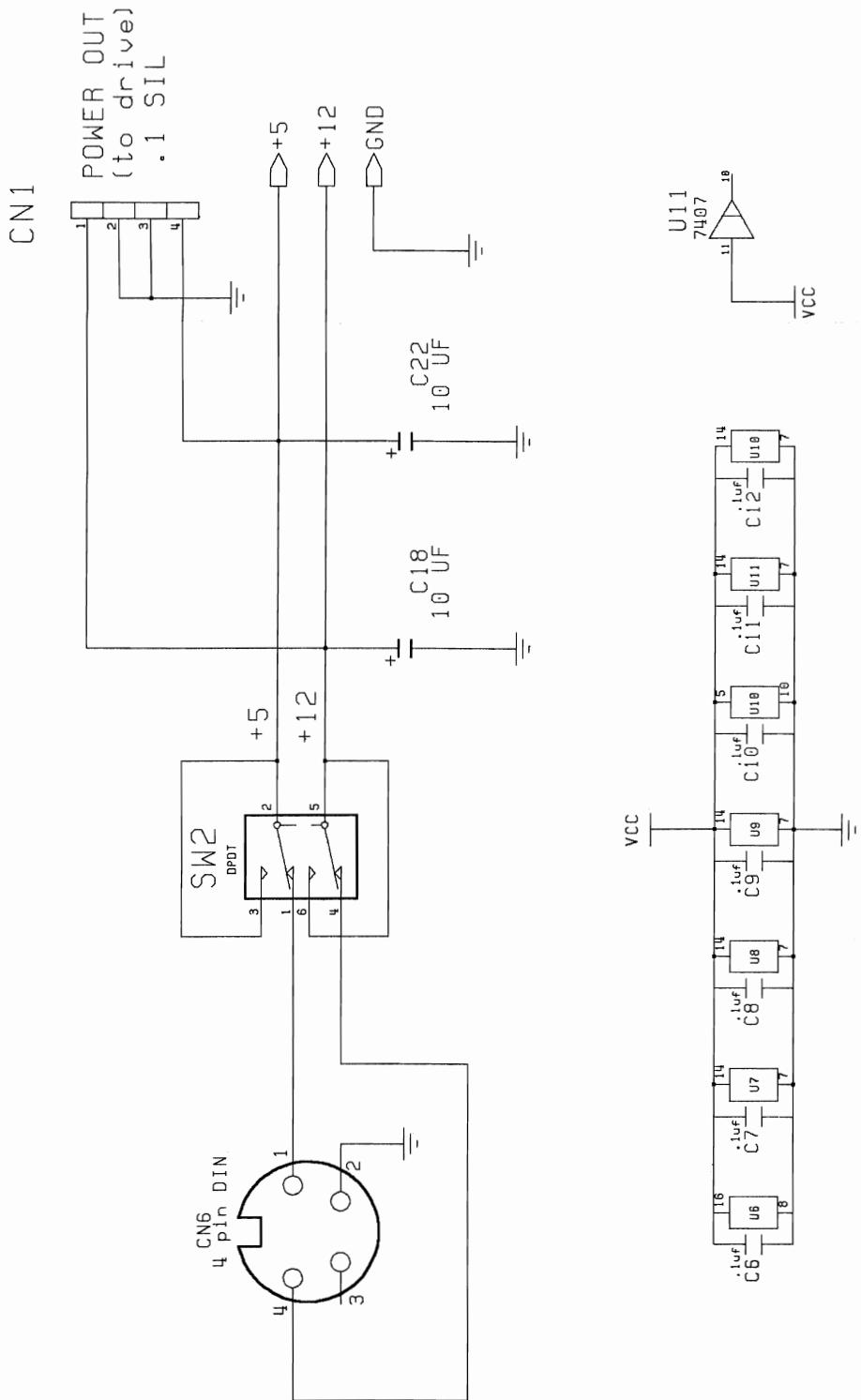




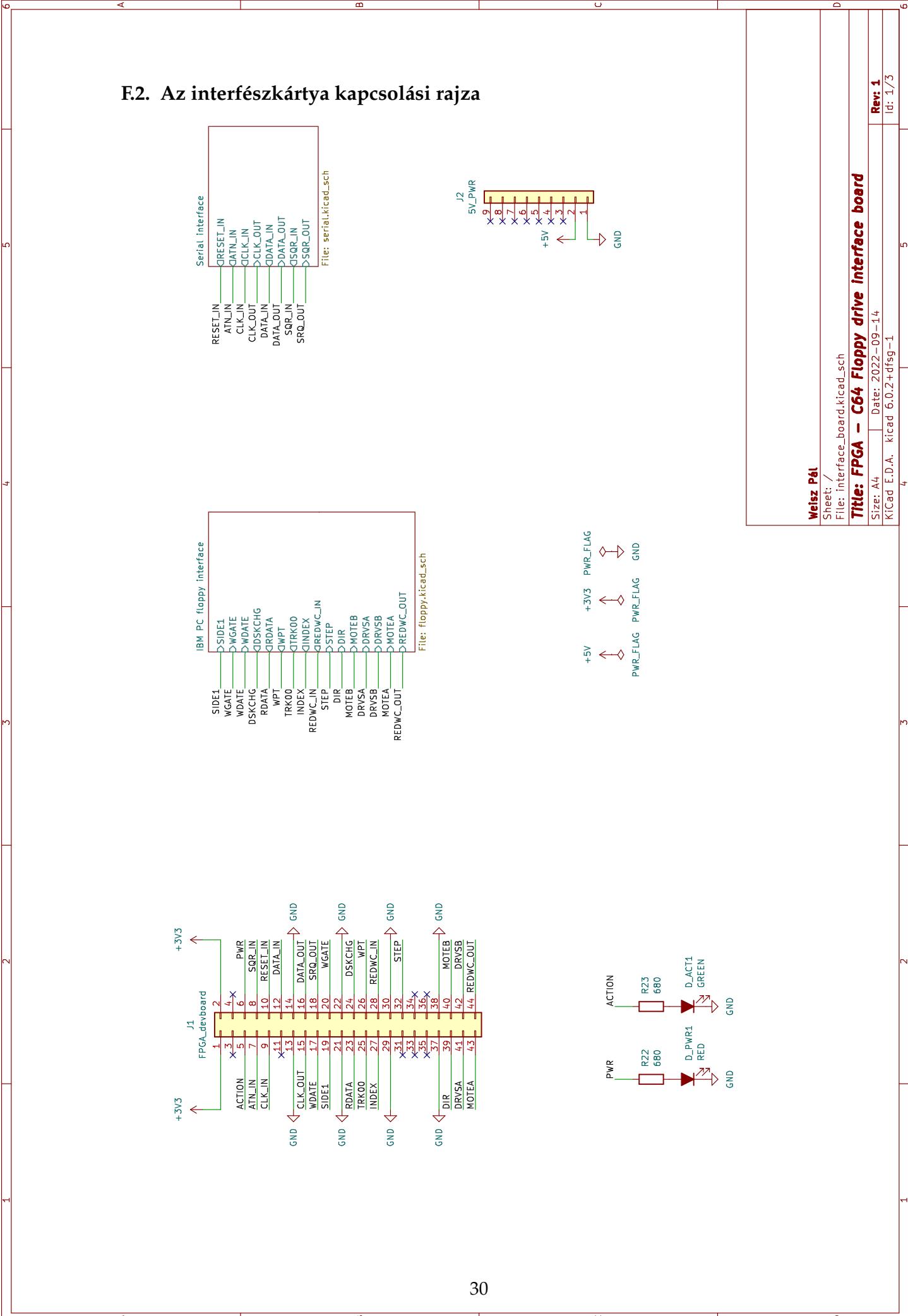
NOTES

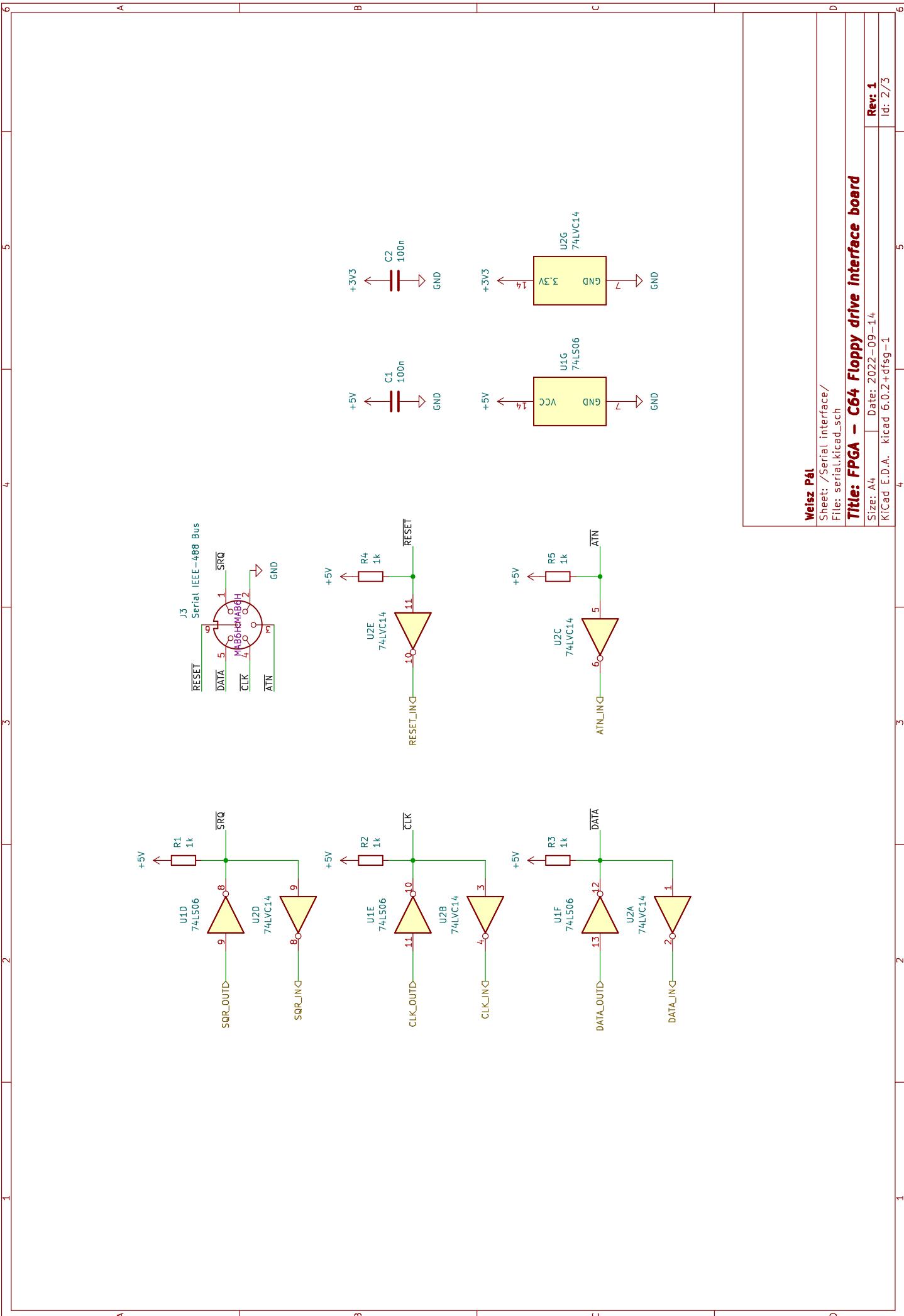
- 1) J1 installed for WD1770 (UW).
2) Removed for WD1772 (UH).

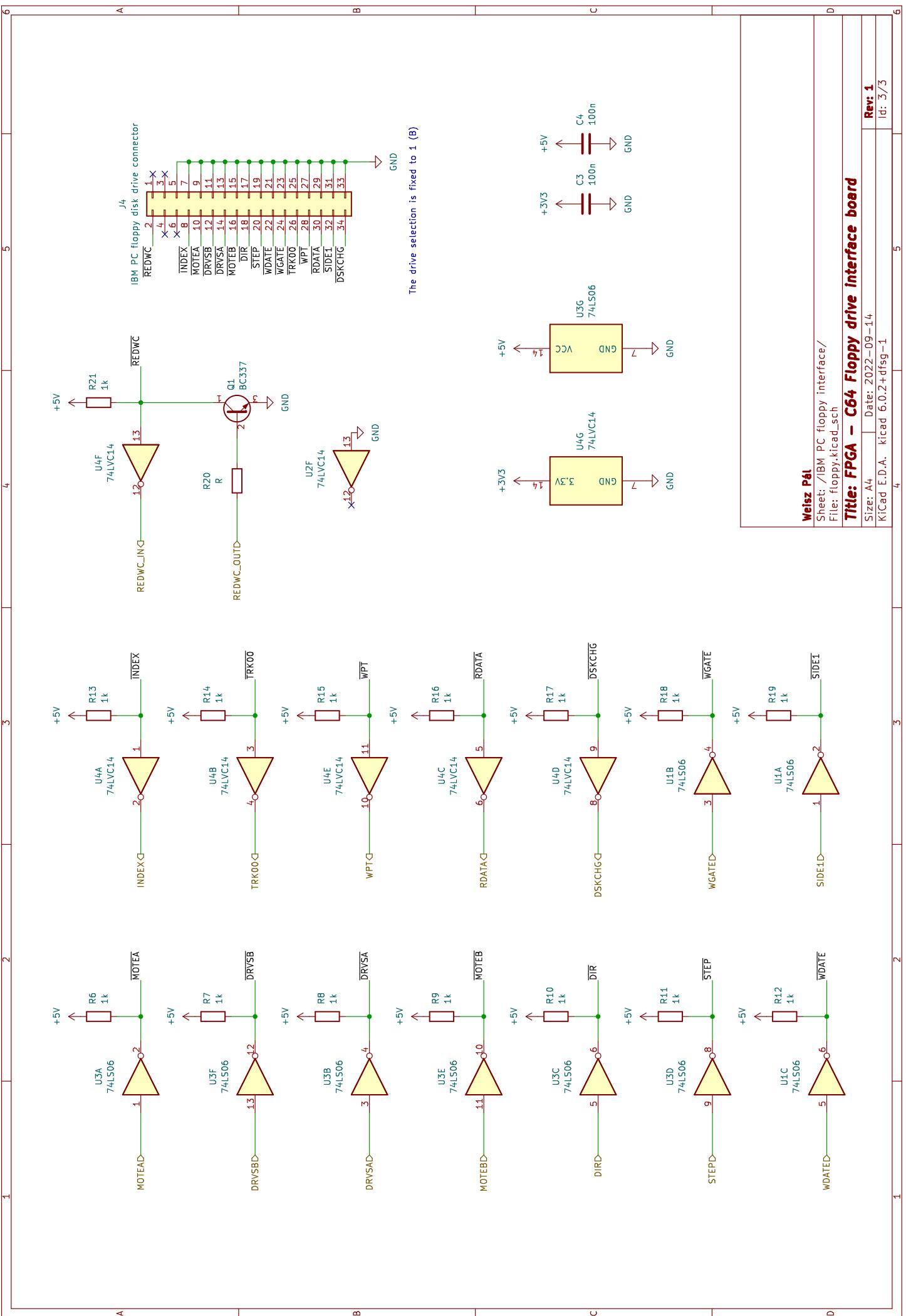
PCB ASSEMBLY #250471
SCHEMATIC #252380 REV. 4



PCB ASSEMBLY #250471
SCHEMATIC #252380 REV. 4
Sheet 4 of 4







F.3. Az interfészkártya NYÁK-terve

