

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

A NES videojáték konzol FPGA alapú megvalósítása

DIPLOMATERV

Készítette
Varga Dominik

Konzulens
Raikovich Tamás

2023. november 25.

Tartalomjegyzék

Kivonat

Abstract

1. Bevezetés	1
2. Felhasznált eszközök	2
2.1. Altium Designer	2
2.2. Xilinx ISE	2
2.3. Logsys GUI	2
3. Nintendo Entertainment System ismertetése	3
3.1. NES hardver főbb komponensei	4
3.1.1. Komponensek kapcsolata - PPU és CPU adatbusz	6
3.2. Képalkotás - Picture process unit	6
3.2.1. PPU által generált kimeneti jel	7
3.2.2. Pattern táblák és paletták	8
3.2.3. A név táblák (Name Tables) és tulajdonság táblák (Attribute Tables)	10
3.2.4. Objektum Attribútum Memória (OAM)	11
3.2.5. PPU hardveres hibái (bug-ok)	11
3.3. 2A03 a NES fő vezérlő egysége	12
3.3.1. MOS 6502 - központi feldolgozó egység (Central Processing Unit, CPU)	12
3.3.2. Audió feldolgozó egység (Audio Process Unit, APU)	14
3.3.3. DMA	14
3.4. NES joypad	14
4. NES FPGA alapú újra gondolása	15
4.1. Képalkotás	15
4.2. Audio	16
4.3. Játékok tárolása	16
4.4. Kompakt hordozható méret	17
5. FPGA NES kártya ismertetése	18
5.1. Tápellátás	20
5.2. Órajel források	20
5.3. Memória - SRAM	20
5.4. Digital Analog Converter és erősítő	20
5.5. HDMI és I2C szint illesztő	22
5.6. A kártya bemenetei	22
5.7. MicroSD kártya	23

5.8. FPGA konfigurációs módok	23
5.9. Soros Flash memória	24
5.10. LOGSYS fejlesztői port	24
5.11. Nyomtatott áramköri terv	24
5.11.1. Réteg beállítások	24
5.11.2. Komponensek elhelyezése	25
5.11.3. HDMI adatvonalainak bekötése	27
5.11.4. FPGA táp vonalak kialakítása	28
5.11.5. FPGA NES 3D terve	29
6. FPGA tervezés	30
6.1. Rendszer blokkvázlat bemutatása	30
6.2. Működési órajel választása	30
6.3. Picture Process Unit	30
6.3.1. Eredeti rendrelés menete	30
6.3.2. VGA rendelés	30
6.3.3. Háttér renderelési állapot gép	30
6.3.4. Sprite rendering állapot gép	33
6.3.5. CPU által elérhető regiszterek és CPU adatbusz	33
6.3.6. PPU adatbusz és memória elérése	33
6.4. NES memória felépítése FPGA-ban	33
6.5. DMA	33
6.6. 6502 processzor működése	33
6.7. NES kontrollerek kezelése	33
7. A NES tesztelése	34
7.1. Rendszer szimulációk	34
7.2. Hardveres tesztek	34
7.2.1. Donkey Kong	34
7.2.2. Super Mario Bros.	34
7.3. A tesztek eredményeinek kiértékelése	34
8. Összefoglalás, jövőbeli tervezek	35
Köszönetnyilvánítás	36
Ábrák jegyzéke	37
Táblázatok jegyzéke	38
Irodalomjegyzék	38
Függelék	40
F.1. NES kártya alkatrész elhelyezési terve	40
F.1.1. Top	40
F.1.2. Bottom	41
F.2. Nyomtatott áramköri terv (2D transparent)	42
F.3. FPGA NES kártya kapcsolási rajza	43
F.3.1. Tápegység	43
F.3.2. HDMI és MicroSD kártya csatlakozó	44
F.3.3. DAC, erősítő és kontroller áramkörök	45
F.3.4. SRAM és SPI-Flash	46

F.3.5. FPGA OSC és JTAG	47
F.3.6. FPGA IO bankok	48

HALLGATÓI NYILATKOZAT

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

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

Budapest, 2023. november 25.

Varga Dominik
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomaternevnek. A sablon használata opcionális. Ez a sablon L^AT_EX alapú, a *TeXLive* T_EX-implementációval és a PDF-L^AT_EX fordítóval működőképes.

Abstract

This document is a L^AT_EX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* T_EX implementation, and it requires the PDF-L^AT_EX compiler.

1. fejezet

Bevezetés

Az 1983-ban megjelent Nintendo Entertainment System (NES) 8 bites videojáték konzol a maga korában igen népszerű volt. A hardverének kialakítása több későbbi, modernebb videojáték konzolra volt hatással, valamint számos kiemelkedő játékprogram erre a konzolra készült el először.

A NES viszonylag egyszerű, jól átgondolt hardvere lehetővé teszi annak az olcsóbb, kevesebb erőforrással rendelkező FPGA eszközökkel történő megvalósítását. Egy ilyen megvalósításnak több előnye is van, például ki lehet használni a modern megjelenítő interfések (VGA, DVI, HDMI, stb.), valamint az eredeti játékkazetta helyett alkalmazni lehet modern adattároló eszközöket is (SD kártya). Természetesen ezen „továbbfejlesztések” mellett az egyedi változat képes futtatni az eredeti játékprogramokat.

A feladat célja egy egyedi, FPGA alapú NES megvalósítás hardver és szoftver komponenseinek elkészítése.

2. fejezet

Felhasznált eszközök

2.1. Altium Designer

2.2. Xilinx ISE

A Xilinx ISE (Integrated Softver Environment) a Xilinx Inc. által kifejlesztett, széles körben használt szoftvercsomag volt. Átfogó eszközökészletet biztosított a digitális logikai áramkörök tervezéséhez, teszteléséhez és megvalósításához a Xilinx Field-Programmable Gate Array (FPGA) és Complex Programmable Logic Device (CPLD) eszközökkel.

A Xilinx ISE teljes körű megoldást kínált a digitális tervezéshez, beleértve a HDL (Hardware Description Language) tervezést, a szimulációt, a szintézist, az implementációt és az eszközprogramozást. Támogatta a különböző tervezési beviteli módszereket, beleértve a Xilinx Schematic Editor segítségével történő sematikus rögzítést és a HDL-alapú tervezést olyan nyelvekkel, mint a VHDL (VHSIC Hardware Description Language) és a Verilog.

2.3. Logsys GUI

3. fejezet

Nintendo Entertainment System ismertetése

A Nintendo Entertainment System (NES) egy otthoni videojáték-konzol, amelyet a Nintendo 1983-ban Japánban (Family Computer, röviden FamiCom néven) és 1985-ben Észak-Amerikában, Európában és Ausztráliában adott ki. Ez minden idők egyik legikonikusabb és legnagyobb hatású videojáték-konzolja.

A NES döntő szerepet játszott a videojáték-ipar újjáélesztésében az 1983-as észak-amerikai videojáték-válság után. Számos klasszikus és kedvelt játékot mutatott be, amelyeket a játékosok még ma is nagyra tartanak. A konzol sikere az erős játéktárnak, a felhasználóbarát kialakításnak és az innovatív marketingstratégiáknak köszönhető.



3.1. ábra. Nintendo Entertainment System

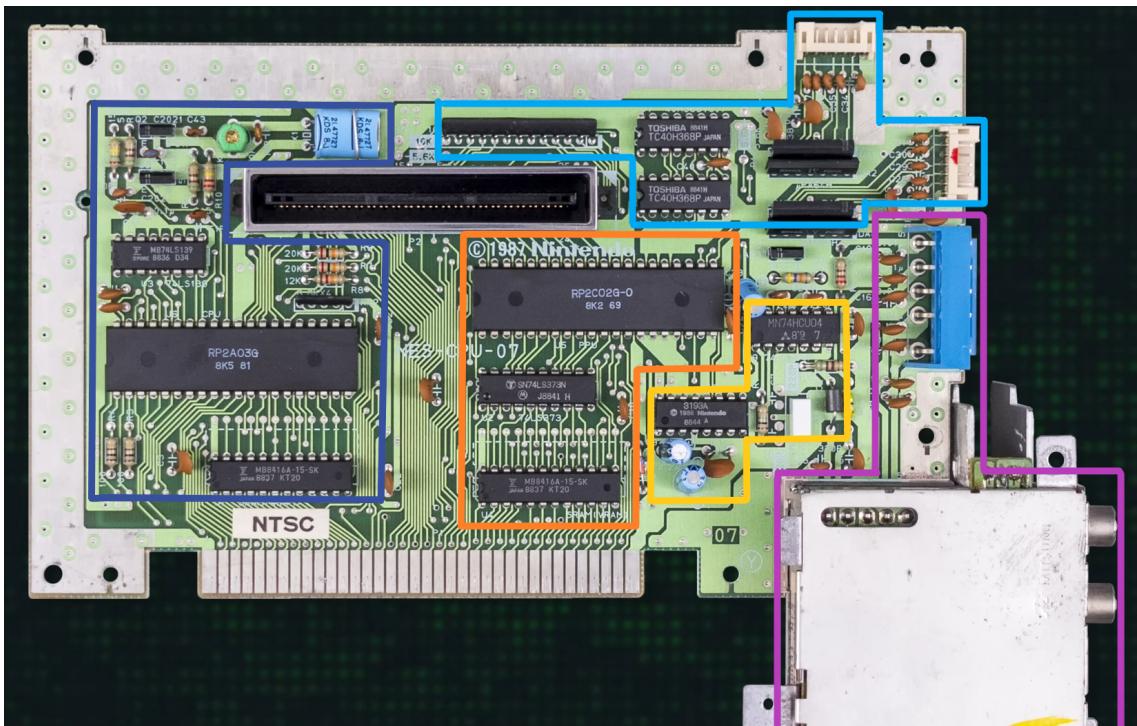
Az otthoni konzol 8 bites processzorral rendelkezett, és a játékok tárolására elsősorban kazettákat használt. Jellegzetes, téglalap alakú kialakítása volt, a játékkazetták behelyezésére szolgáló elülső betöltő mechanizmussal. A konzolhoz egy pár kontroller is tartozott, és bevezette a ma már ikonikus NES kontroller kialakítását, amely irányjelzővel, start- és választógombokkal, valamint az A és B gombokkal rendelkezett.

A konzolra megjelent legnépszerűbb és legnagyobb hatású játékai közé tartozik a Super Mario Bros., a The Legend of Zelda, a Metroid, a Mega Man, a Castlevania és még sok más játék. Ezek a játékok megalapoztak számos sikeres franchise-t, amelyek ma is virágognak.

A játékkonzol hardverének megismerésére, elsősorban a hivatalos wiki oldalt használtam. Az itt olvasható tartalmakat az évek során nagy részt az eredeti hardver visszafejtésével (reverse engineering) tárta fel, mivel a játékkonzol pontos adatlapjai, illetve időzítés és működési diagramjai nem lettek publikusak (a Nintendo tulajdonban vannak). Az oldalon szereplő adatokat a NESDev online közösség tartja karban, ezáltal az oldal pontos és helyes adatokat tartalmaz (ezeket a közösség rendszeresen felül vizsgálja). A NES eredeti hardverének áttekintéséhez és a chippek megismeréséhez (PPU, CPU) nagy segítséget nyújtott még a NESHacker youtube csatorna.

3.1. NES hardver főbb komponensei

A Nintendo Entertainment System hardverének áttekintéséhez célszerű az eredeti konzol alaplapjának tüzetes vizsgálata. Ezt a 3.2 képen láthatjuk, és ez alapján a komponensek öt fő csoportba sorolhatjuk.



3.2. ábra. Nintendo Entertainment System NTSC alaplap [4]

- Sötétkék, processzor:* A NES processzora és ennek kiegészítő áramkörei, komponensei. Ide tartozik természetesen az RP2A03G chip, ez a fő vezérlő egyégek két fő elemet tartalmaz : egy átalakított 6502-es processzort, illetve az APU co-processzort amely a hang generálásért felelős. Ezen a kék területen belül még megtalálható a WRAM (cpu alatt) amely egy két kilobájt méretű rendszer memória (system memory) kent volt használva, illetve az 74LS139-is (cpu-tól balra fent) amely a chip kiválasztó (chip select) jelek elő állításáért felelős hardver. Itt kapott még helyet a keret tetején látható kék oszcillátor kristály is és ennek segéd áram köre. Ez a komponens látta el az egész alaplapot órajellel.

- *Narancssárga, videó generálás:* Ebbe a kategóriába három komponens tartozik. Első az RP2C02G-O chip amely a videó generálás fő vezérlő egysége volt, ennek neve Picture Process Unit (PPU). A PPU alatt pedig a két memória chip látható, az alsó raktározta el a képgeneráláshoz szükséges adatokat (VRAM), a fölötté lévő egy adatpufferként funkcionált.
- *Citromsárga, CIC chip:* Itt található a CIC chip amely azért volt felelős, hogy nem licencelt játékokat ne lehessen a NES-el játszani. Illetve itt még egy invertáló áramkör kapott helyet (ezt nagyjából minden alaplapi komponens használta ha bit invertálásra volt szüksége).
- *Lila, kompozit kimenet:* Ezen a területen láthatók a kompozit jel elő állításáért felelős áramköri elemek és a táp bemenet szűrésért felelős kapacitások.
- *Világoskék, kontrollerek:* A fenti csatlakozó az egyes játékos kontroller portja a jobb oldali pedig a második játékosé. Itt még látható két dedikált invertáló komponens is, amelyek a kontrollerekből érkező adatok negálásával foglalkoztak.

A fenti elemek kívül az alaplapon még található középen egy bővítő csatlakozó is, illetve a kártya aljára lehet közvetlenül csatlakoztatni a játék kazettákat is. A játék kazetták multifunkcionális nyákok voltak és általában a 3.3 ábrán látható módon néztek ki.



3.3. ábra. Nintendo Entertainment System játék kazetta [2]

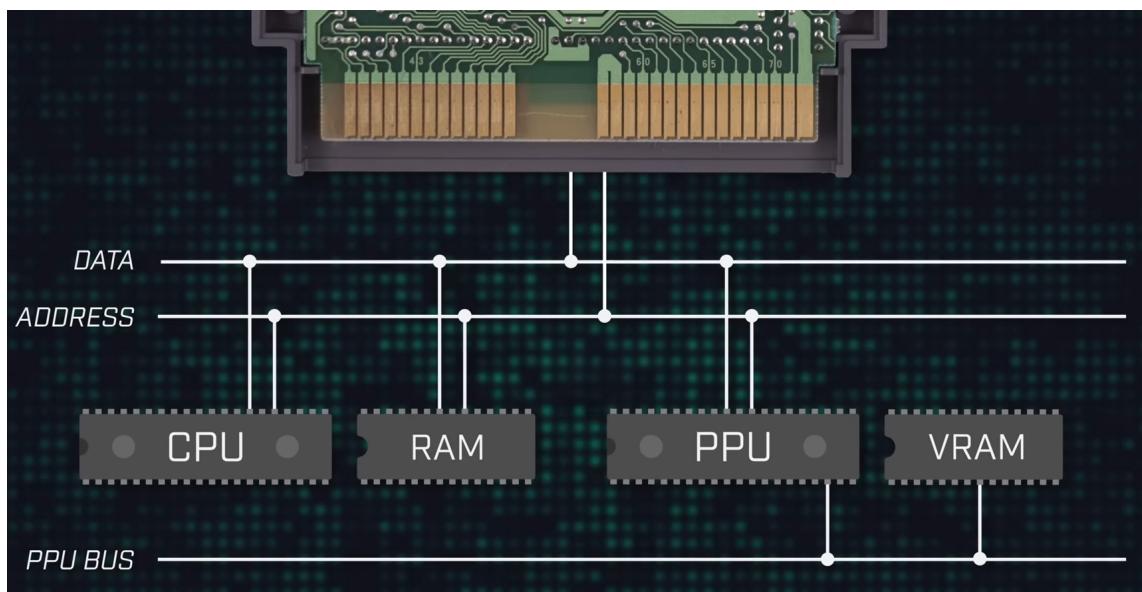
A játék kazetták fő alkotó elemei:

- *Program ROM, és Mapper-ek:* A játék szoftvert tartalmazó, kezdetben 16 kilobájt - 32 kbyte méretű ROM. Viszont a 32 kilobájnál nagyobb játékok esetén a ROM egy extra segéd chippel (Mapper) egészült ki, amelyen keresztül a NES megtudta címzni a nagyobb ROM területet.
- *Karakter ROM:* Általában 8 kilobájt méretű ROM. A megjelenítéshez szükséges csempe elemeket tárolja (3.6).

- *CIC chip*: Ez a komponens licencezte a játék szoftverét a NES konzol felé.

3.1.1. Komponensek kapcsolata - PPU és CPU adatbusz

A különböző komponensek mélyebb bemutatása előtt a NES hardverének adat kapcsolatát is át kell tekintenünk. A NES úgynevezett memory-mapped I/O architektúrával rendelkezik. Ez egy olyan technika amely a rendszer teljes memória területét feldarabolja nagyobb egységekre, és ezeket hardveres komponensekhez rendel. Ezek alapján az alaplapon található WRAM, a játék kártyán található Program ROM, illetve a PPU hét vezérlő regisztere és az APU vezérlőregisztere, mind a CPU 16 bites címterületén belül található és a CPU adatbuszon keresztül írható és olvasható. A NES-en belül található még egy PPU adatbusz is amely a VRAM-ot és a játék kazettán található karakter ROM-ot foglalja a PPU cím tartományba (3.1). Ezeket az adatkapcsolatokról a 3.4 sematikus ábra foglalja össze.



3.4. ábra. Komponensek közti kapcsolat (fent játék kazetta) [4]

3.2. Képalkotás - Picture process unit

A következőkben azt fogom bemutatni, hogy a NES hogyan tárol, dolgoz fel és jelenít meg sprite grafikákat. A Sprite egy 8x8-as pixel csempét jelent, ez a NES képalkotásának alappillére.

A NES főkomponensei közül a Picture Process Unit (későbbiekben PPU) felelős a konzol 8-bit-es grafikájának elő állításáért. A PPU egy a Nintendo által kifejlesztett speciális chip amely a processzor mellett működik, mint egy társprocesszor (co-processor), hasonlóan a napjainkban elterjedt videókártya-processzor pároszhoz.

A CPU-tól eltérően a PPU egy előre meghatározott grafikus műveleti parancs sorozatot hajt végre ciklikusan, nem lehet közvetlenül programozni. Saját memoriával rendelkezik amelyet a CPU képes módosítani, hogy ezzel megváltoztassa a grafika generálását. Ez a memóriaterület a következőképpen négy részre oszlik:

- *Pattern táblák*: Az első szekció tartalmazza a pattern táblákat, amelyek a nyers sprite-kép adatokat tartalmazzák az adott játékhoz. Két pattern tábla van a bal oldali és a jobb oldali tábla amelyek mindegyike 64 kilobyte-nyi memória. Együttesen

pedig 256 darab 8 x 8 pixeles csempét tárolnak. A memória ezen része általában közvetlenül a játék kazetta karakter ROM vagy RAM chipjére van leképezve.

- *Névtáblák (Nametables)*: A következő rész a PPU névtábláit tartalmazza, amelyek a háttérgrafikák kialakítására szolgálnak a játékhoz. Ezek 32x30-as rasszterben vannak felépítve, a rasszter minden egyes eleme egy 8x8 pixeles területet reprezentál a képernyőn. A cellák egyetlen byte-ot tartalmaznak, amely egy csempét címez meg a Pattern táblákban.
- *Paletták (Palettes)*: A harmadik rész az aktív szín paletták tárolására szolgál a játékhoz. A PPU képes több mint 50 különböző szín előállítására, de nem tudja az összes színt egyszerre használni egyidejűleg, ehelyett ezt a memória területet arra használják, hogy meghatározzunk nyolc aktív palettát amelyek egyenként négy színt tartalmaznak. Ebből a nyolc palettából, választhatunk színt a pixel-ek megjelenítése során.
- *Objektum Attribútum Memória (későbbiekben OAM)*: A PPU memóriának ez a része vezérli a játék előtérben lévő grafikájának megjelenítését. Ezek olyan dolgok, mint például Mario, Link, az ellenségek és az olyan effektek, mint a tűzgolyók és robbanások. Alapvetően bármi, ami a háttér grafika felett vagy néha alatta jelenne meg.

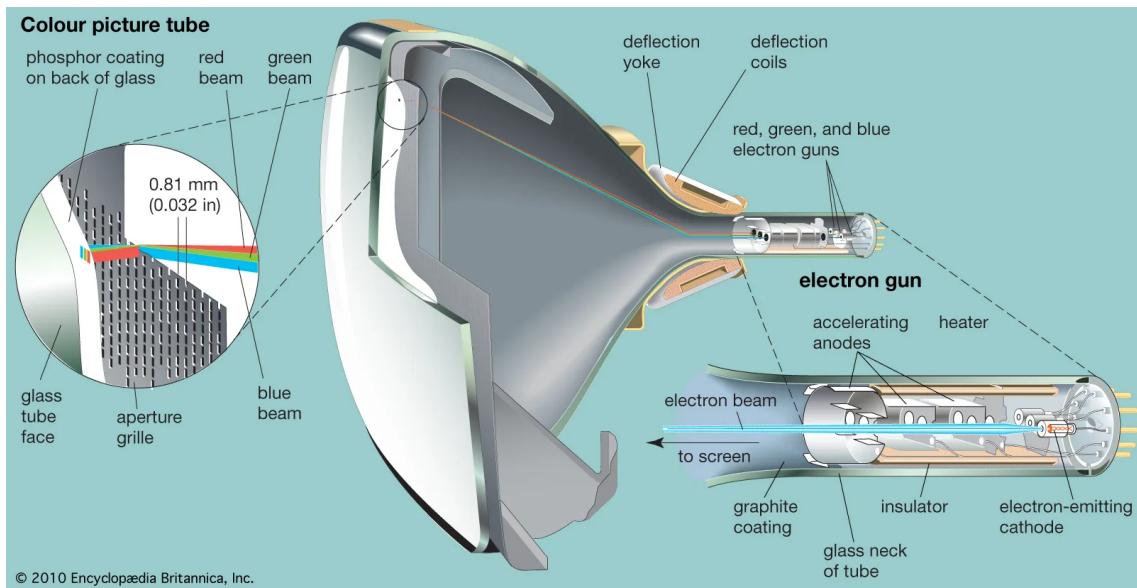
Tehát, mindez összegezve, úgy tekinthetjük a PPU-ra, mintha ez a négy jól elkülöníthető memória terület irányítaná ezt a segéd processzort. A Pattern táblák határozzák meg a nyers képadatokat. A névtáblák határozzák meg a háttér generálását. A színpaletták határozzák meg a használandó színeket és az OAM vezérli az előtérbe vagy háttérbe kerülő mozgó sprite-okat. Ezen felül a PPU további funkciókkal is rendelkezik, ezeket nyolc különböző regiszter írásával és olvasásával érhetjük el. Ezkről a regiszterekről az implementálás során a 6.3 fejezetben még olvashatunk.

3.2.1. PPU által generált kimeneti jel

Ebben a fejezetben bemutatom a PPU által generált jelet és ennek felhasználást a régi típusú CRT TV-kben. A CRT televízió (az eredeti TV) a modern lapos képernyők előfutára volt, alapvetően két fő komponensből épültek fel egy fluoreszkáló képernyőből és egy katódsugárcsőből.

A CRT működése röviden: a katódsugárcső egy pisztolyként funkcionál, amely elektrokat lő ki a képernyőre és amikor elég elektron találja el a képernyő egy bizonyos területét az világítani kezd. A televíziók kétféle típusban léteztek, fekete-fehérben vagy színesben. A fekete-fehér esetben egy elektronagyú szabályozta a képernyő pixeljeinek monokróm fényerejét, a színes esetben három különálló elektronagyú szabályozta a vörös, kék és zöld komponensek arányát, ezzel megalkotva a színes képet. A színes TV-k esetében is a három elektron sugár együtt mozgott végig a képernyőn, ezért a könnyebb megértés érdekében érdemes egy elektron sugárként gondolni ezekre.

A televízió működése során a bal felső sarokból kezdve úgy irányítja az elektron sugarat, hogy a teljes képernyőn véigfusson sorról sorra, amíg el nem éri a jobb alsó sarkát a képernyőnek. Ha egy sor végére érünk akkor az elektron sugarat vissza pozicionáljuk a sor elejére, ezt az időt horizontális szinkronizációnak nevezzük (horizontal blanking). Ha végigértünk egy képkockán a TV a katódsugárcsövet újra a felső sor bal oldalára állítja, ezt vertikális szinkronizációknak (vertical blanking) hívják. Ez képalkotási ciklus a TV működése közben rögzített időközönként ismétlődik, általában másodpercenként hatvan képkocka körül.



3.5. ábra. Katódsugárcsöves TV-k működése

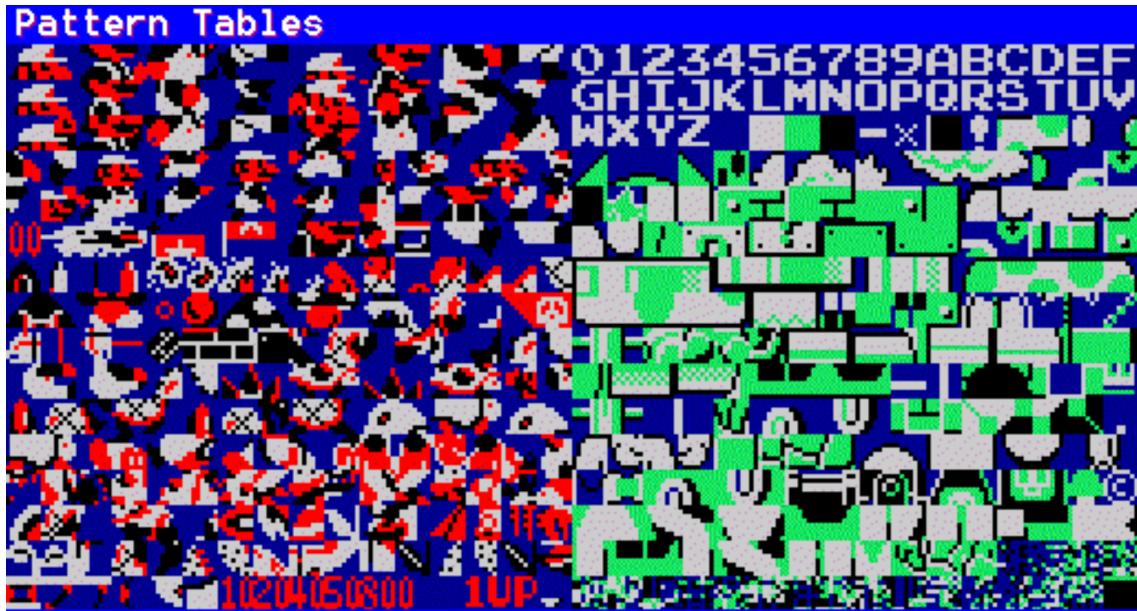
Miközben a elektronágynak mozog, a TV egy belső jel segítségével tudja szabályozni az elektronok kibocsátásának mértékét. Ez a jel megváltoztatja a szín fényerejét egy adott pozícióban (pixel-en), a pisztoly gyors mozgásának következtében, az eredmény egy folytonos animált kép képernyőn. Ezt a jelet kompozit jelnek nevezzük és általában egy rf antennáról vagy egy kábel dobozból származott, de a NES esetében ezt a jelet a PPU állítja elő.

A NES két típusú kompozit jelet tudott előállítani, attól függően, hogy a világ melyik területére gyártották. Ez azért következett be mert a CRT TV-knek két standard típusa terjedt el világszerte az NTSC és a PAL. Az NTSC-t elsősorban az Egyesült Államokban és Japánban használták, és 60 képkocka/másodperces sebességgel jelenítette meg kompozit jelet, ezek a TV-k összesen 525 képsorral rendelkeztek. A PAL-t elsősorban Európában, Afrikában és Dél-Amerikában használták és 50 képkocka/másodperc sebességgel futott, összesen 625 képsort jelenített meg.

A NES játékokat sosem programozták régió specifikusak, az egyetlen dolog ami változott területenként az a NES PPU-jának hardvere. Így a világ különböző területein ugyanaz a játék gyorsabban illetve lassabban futott, ez akár 17%-os különbséget is jelenthetett. A NES emulálás szempontjából az NTSC készülékekkel veszem alapul mivel ezeken gépek működését tárták fel részletesebben reverse engineering-gel.

3.2.2. Pattern táblák és paletták

A sprite képek, amelyek a PPU Pattern tábla memóriájában helyezkednek el, képezik az alapját minden grafikai megjelenítésnek. Ezek a 8 x 8 pixeles csempék alkotják a bonyolult háttereket, mozgó objektumokat és speciális effekteket. Alapvetően a sprite-ok tárolása hasonló módon történik mint egy modern számítógépek által használt kép esetében, mint például png. Mindkét tárolási formátumot kétdimenziós szám rácsként tudjuk elközelni, ahol minden egyes cellához tartozó érték egy pixelhez tartozó színt reprezentál, csak amíg a png több millió színt támogat addig a NES egy pixel-e csupán négy különböző színű lehet. Gyakorlatilag egy ilyen csempe nem is tárol szín adatokat, a pixel-eket reprezentáló számértékek referenciák egy éppen aktív szín paletta színére.



3.6. ábra. Super Mario Bros. Pattern táblái

A színpaletták memória területének írásával, nyolc különböző palettát állíthatunk be a grafikus megjelenítéshez, négyet a háttér megjelenítéséhez és a maradék négyet az előtér rendereléséhez. minden egyes paletta négy színt tárol, az első szín minden egyes palettán egy átlátszósági szín, ez azt jelenti, hogy ha a pixel szín értéke nullás indexel rendelkezik akkor az megjelenítés során minden esetben átlátszó lesz (függetlenül a palettába írt szín értéktől).

00	00	00	00	00	00	01	01
00	00	00	00	00	01	01	01
00	00	00	00	01	01	01	01
00	00	00	01	01	01	01	01
00	00	01	11	11	01	01	01
00	01	01	01	10	11	01	01
00	01	01	01	10	11	11	11
01	01	01	01	10	11	10	01

3.7. ábra. Gumba (Mario egyik ellenfele) bal felső csempe elem szín adatai [3]

A fentiek alapján, tehát egy pixel 0-3-ig vehet fel értéket, tehát 2 biten vagyunk képesek eltárolni az értékét. Egy 8 x 8 as csempe pedig 64 pixelt tartalmaz, ezért összesen 16 byte helyet foglal. Mivel a NES processzora egy 6502-es (8 bit-es) modell volt, ennek okán a legkisebb memóriaegység amit képes volt megcímzni az egy byte volt, így a csempe adatok ésszerű tárolásához egyedi megoldásra volt szükség.

Mivel a NES nem képes direkt a 2 bit-es számértékekkel dolgozni, ezért ezeket fel daraboljuk először logikailag magas és alacsony bitekre. Ezt követően elsőként a csempe alacsony 8 byte-ját tároljuk el majd ezt követően a magas nyolc byte-ot, így megkapjuk a fentebb kiszámolt 16 byte-os értéket. Ezáltal könnyel elérhetővé tettük a kép adatainkat a cpu számára, illetve a PPU-is helyesen képes ezeket megjeleníteni.

3.2.3. A név táblák (Name Tables) és tulajdonság táblák (Attribute Tables)

A név táblák alapvetően, ahogyan már fentebb olvashattuk egy-egy 32 x 30-as rácsként képzelhetők el, ahol minden egyes cella egy csempének felel meg (8 x 8 pixel). Ebből következően egy név tábla 256 x 240 pixelt tartalmaz, ez a CRT monitorok teljes képernyő területe. Egy rács elem, pedig egy, egy byte-os címet tartalmaz, amely az éppen aktív pattern tábla egy csempéjét címzi meg (mivel összesen 256 aktív csempénk lehet ezért elég egy byte a címzéshez).

Ahhoz, hogy a PPU-nk képes legyen egy játék során gyors és gördülékeny háttér változásokra több különböző eszköz fejlesztettek ki. Kezdve azzal, hogy a NES hardveren belül két név táblát helyeztek el, ezek okos címzésével és a hardver sajátosságainak kihasználásával (mirroring) képesek vagyunk az úgynévezett görgetés (scrolling) megvalósítására. Ez egyszerűen egy vertikális vagy egy horizontális finom lapozásnak írható le (egy-két sor pixel eltolásával, illetve megjelenítésével). Alapvető esetben a NES játékok vagy fix háttérrel rendelkeztek (lásd donkey kong), vagy csak horizontális vagy vertikális görgetést alkalmaztak (horizontális - Super Mario Bros.). Viszont a későbbiekben megjelentek komplexebb játékok is amelyek ezek vegyítését alkalmazták ilyen például a Metroid vagy a Legend of Zelda.

A mirroring az a jelenség, amikor két cím azonos memória területre mutat, ez azért fordulhat elő, mert a PPU nem teljes címzést használ (kevesebb bit-tel címez meg tartományokat). A név táblák esetében például egy 2 x 2-es rácsot szoktak képezni ennek segítségével, így kiterjesztve a két névtáblánk címzési tartományát. De ez a jelenség ez egész PPU memória felépítése során megfigyelhető.

Memória címek	Méret	Leírás
0000–0FFF	\$1000	Pattern tábla 1
1000–1FFF	\$1000	Pattern tábla 2
2000–23FF	\$0400	Név tábla 1
2400–27FF	\$0400	Név tábla 2
2800–2BFF	\$0400	Név tábla 3
2C00–2FFF	\$0400	Név tábla 4
3000–3EFF	\$0F00	A 2000–2EFF címterület mirror-ja
3F00–3F1F	\$0020	Szín paletta RAM indexek
3F20–3FFF	\$00E0	A 3F00–3F1F címterület mirror-ja

3.1. táblázat. A PPU memória kezelése (14 bit címek) mirroring jelenséggel

Minden egyes névtábla végén egy kisebb extra memória terület található, amelyet tulajdonság táblának (vagyis Attribute Table-nek) nevezünk. Ez egy kisebb táblázatként képzelhető el ahol minden egyes cellában egy byte adatot tárolunk, viszont ennek feloldása

bonyolultabb mint névtáblák esetében. Ez a 8 bit egy 4 X 4 csempéyi háttér terület szín palettáját határozza meg a következő képen:

- az első két bit a bal felső 2 x 2 csempe palettáját határozza meg,
- a második két bit a jobb felső 2 x 2-es terület palettáját határozza meg,
- a harmadik két bit a bal alsó 2 x 2-es terület palettáját határozza meg,
- végül pedig az utolsó két bit a jobb alsó terület színérért felelős.

Így a teljes képünket 8 x 8 ilyen byte-tal írhatjuk le.

3.2.4. Objektum Attribútum Memória (OAM)

Ez a memória terület 64 külön álló sprite tárolására képes. minden egyes OAM spritehoz négy byte adat tartozik. Az első byte meghatározza a vertikális vagy y koordinátáját a sprite-nak. A második byte azt határozza meg, hogy melyik 8 x 8-as csempe legyen megjelenítve az éppen aktív Pattern táblából. A harmadik byte segítségével különböző tulajdonságait vagyunk képesek befolyásolni a sprite-nak. Végül pedig az utolsó byte a horizontális vagyis x koordinátáját határozzam meg az objektumnak.

Az előbb bemutatott byte-ok közül az egyetlen bonyolultabb működésű a harmadik, ebben az estben is a különböző bit-ek más és más működést kódolnak. Az nulladik és az első bit egy előtér szín palettát választanak ki a sprite számára. A következő három bit (3, 4 és 5) nem használtak. Ezt követően az ötödik bit határozza meg, hogy a sprite a háttér elé vagy mögé kerüljön (ha értéke nulla akkor a háttér fölé fog kerülni, ha pedig egy mögé). Végül pedig a hatodik illetve hetedik bitek azt határozzák, hogy a sprite pixel-ei horizontálisan vagy vertikálisan helyet cseréljenek (tükrözve legyenek), nulla esetén a sprite eredeti formájában marad, egy esetén, pedig meg tükröződik. Ez egy nagyon hasznos tulajdonság, hiszen így a játék fejlesztők rengeteg memória területet spórolhattak a Pattern táblákból. Mivel több olyan objektumot, karaktert vagy effektet is tervezhettek amelyek vagy horizontálisan vagy vertikálisan szimmetrikusak voltak (erre az egyik legjobb példa a felvétő gomba a Super Mario Bros. videojátékból).

3.2.5. PPU hardveres hibái (bug-ok)

Mivel hardveres emulálást készítünk, ezért nem lehetünk el az eredeti hardver hibái mellett. Az esetek többségében a NES játékok fejlesztői kihasználták ezeket az hazárdokat a játékfejlesztések során. Tehát ha az eredeti játékokkal szeretnénk játszani ezekkel is maradéktalanul meg kell ismerkednünk.

A PPU leghíresebb ilyen hibája, a sprite túlcordulás kezelése (Sprite Overflow Bug). Ez a OAM feldolgozása és megjelenítése közben alakulhat ki a NES-ben. Ennek megisméréséhez először is érdemes áttekinteni, hogy a PPU milyen állapotgép alapján dolgozza fel a OAM-ot. A PPU-n belül található egy másodlagos OAM amely nyolc sprite eltárolására képes, ennek a nyolc sprite-nak a megjelenítése történik a NES egy 256 pixeles sorban. minden egyes sorral ebbe a memória területbe töltjük be az éppen aktív sprite-okat, ennek következtében a 64 sprite közül csupán nyolcat tudunk egy sorba megjeleníteni. Ezek alapján az OAM megjelenítés a következő négy lépésből áll:

1. először is megtisztítjuk a másodlagos OAM-ot,
2. majd végigvizsgáljuk a teljes OAM-ot és kiválasztjuk azokat a sprite-okat amiket a következő sorban meg kell jeleníteni (ezekből az első nyolcat),

3. ha a megtaláltuk a 8 megjelenítendő sprite-ot akkor sajnos egy hibás implementációval végig vizsgálja az eszköz, hogy van-e még aktív sprite a sorban (ha talál ilyet beállítja a Sprite Overflow Flaget),
4. végül pedig a PPU feltölti a megjelenítéshez szükséges regisztereket az éppen aktív spritok adataival.

A következő sorban megjelenítendő aktív sprite-ok kiválasztása, az objektumok y értéke alapján történik az OAM-ban tárolt prioritási sorrend alapján. A fent említett rendszerhiba a harmadik lépésben következhet be, amikor a PPU megtalálta a nyolcadik sprite-ot, ezt követően elkezdi vizsgálni a maradék OAM területet, viszont ennek vizsgálata csak az első esetben következik be helyes tulajdonság szerint (y érték). Ezt követően a sprite-ok tulajdonságaiban diagonálisan haladunk (tehát a második keresés a Pattern tábla cím alapján történik, és későbbiekben így tovább halad az objektum négy byte-ján keresztül), így olyan esetben is bejelezhet a sprite túlcsordulást jelző flag (Sprite Overflow Flag), amikor ez nem is történt meg. Erre a hibára több híres játék is épített az egyik leghíresebb a The Legend of Zelda, de például a Ninja Gaiden és Castlevania sorozatokban is meg jelent.

Egy másik kevésbé ismert hazárd az OAMADDR regiszterrel kapcsolatban találtak meg. Ez általában akkor jött elő amikor nem a DMA vezérlőt használták az OAM frissítésére, hanem a szimpla byte elérését. Ilyenkor néha elő fordult, hogy egy bájtot rossz OAM területre másolt az eszköz, ezzel beszennyezve az OAM-ot. Ennek a bug-nak későbbi kompatibilitási okai voltak.

3.3. 2A03 a NES fő vezérlő egysége

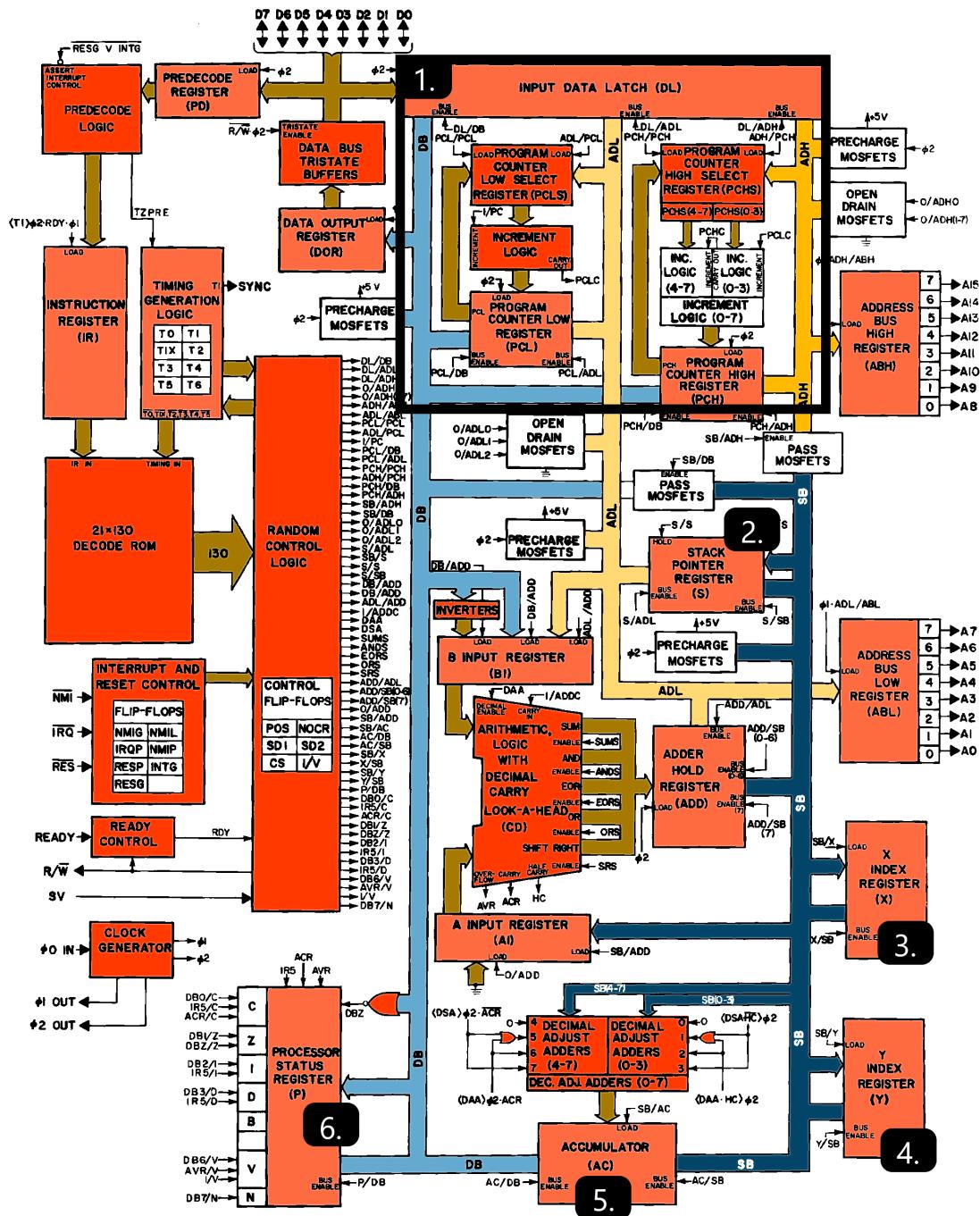
A NES videojáték konzol egy komplex több feladatot ellátó fő végrehajtó egységgel rendelkezik, melynek neve 2A03 (RP2A03[G] NTSC konzolokban). A chip három fő hardveres elemből áll. Először is a kor legjobban elterjedt CPU-jának a MOS 6502-esnek egy módosított változatát tartalmazza, emellett helyet kap még az APU tárprocesszor (co-processzor) is, amely a hang generálásért felelős végrehajtó egység, illetve az adatok gyorsabb másolását elősegítő DMA egységet. A következőkben ezt a három hardveres elemet ismerhetjük meg kicsit részletesebben.

3.3.1. MOS 6502 - központi feldolgozó egység (Central Processing Unit, CPU)

A CPU alapja a MOS Technology 6502-es 8 bites architektúrával rendelkező mikroprocesszora. A processzort Chuck Peddle és Bill Mensch amerikai mérnökök terveztek és először 1975-ben mutatták be. Már a kezdetektől nagy sikernek számított kompakt és egyszerű designja és jó programozhatósága miatt. Egyszerűsége miatt sokkal kevesebb tranzisztorban elfért (3510 tranzisztor) mint vetélytársai az Intel és Motortollától. Anyagköltsége miatt nagyon kedvező áron kezdhették el forgalmazni (csupán 25 dollár). A leghíresebb felhasználásai az Apple 1, 2, illetve a Commodore 64. Több játék konzol is felhasználta a chip architektúráját saját CPU-ik kialakításához (NES, SNES, Atari 2600).

A processzor egy órajel bemenettel rendelkezik ϕ_0 , amelyet a NES esetén egy külső kvarc kristály hajt meg 1.789773 MHz frekvencián (3.2). Ebből a beérkező órajelből állít elő a processzor két, eltolt fázisú órajelet ϕ_1 és ϕ_2 . Ezzel több szinkronizációs lehetőséget engedve egy órajelenbeliül a külső hardvereknek. A 6502 8 bites architektúrája azt jelenti, hogy a processzorunk egy byte adatot képes feldolgozni/módosítani egy órajel lefutása alatt. Tehát 8 bites adatbusszal és egy 16 bites címbusszal rendelkezik. Az 2A03-as model-

leken a Nintendo mérnökei annyiban módosították az eredeti MOS architektúrát, hogy letiltották a cpu decimális módját.



3.8. ábra. A MOS 6502 egyedi nyolcbites architektúrája

A 6502 8 bites architektúrája hat regiszterrel rendelkezik. Ebből három (A, X, Y) általános programozási célokot tölt be. Hárrom pedig speciális belső információk tárolására szolgál (PC, SP, SR). Ezek elhelyezkedését és logikai kapcsolatát láthatjuk a 3.8 képen.

1. *Program számláló (Program Counter, PC):* 16 bites regiszter, amely a program jelenlegi címét tartalmazza (ennek a regiszternek a mérete határozza meg a cím tartományt)

2. *Verem mutató (Stack Pointer, S)*: 16 bites regiszter, viszont felső nyolc bitje fix decimális egy (0000 0001) értékkel rendelkezik. Tehát valójában egy 8 bites regiszter, amely a verem aktuális címére mutat.
3. *X index regiszter*: 8 bites index regiszter programozás során fölfelé és lefelé is számolhatunk vele. Általában adat indexelésre használjuk, de aritmetikai művelet is végrehajtható rajta.
4. *Y index regiszter*: 8 bites index regiszter használata megegyezik a az X index regiszter használatával.
5. *Akkumulátor (Accumulator, A)*: 8 bites regiszter fő szerepe az adat tárolás, ha bármilyen műveletet végzünk a processzoron annak eredménye ebben a regiszterben kerül eltárolásra.
6. *Processzor Státusz Regiszter (P)*: 8 bites regiszter, a processzor státusz flagjeit tárolja a 3.9 ábrán látható módon.



3.9. ábra. A MOS 6502 processzor státusz regisztere (P)

opkódok
memória map

3.3.2. Audió feldolgozó egység (Audio Process Unit, APU)

5 sound channel

3.3.3. DMA

sprite/ dmc mode

3.4. NES joypad

4. fejezet

NES FPGA alapú újra gondolása

A diplomaterv projektem egyik fő célja, hogy egy hardveres emulátort készítsek a Nintendo Entertainment System játékkonzolhoz. Alapvetően emulátort lehet készíteni szoftveres illetve hardveres módon. A szoftveres emulátorok alapja az eredeti hardver szimulálása egy magasabb programozási nyelven íródott szoftver segítségével (python, java), viszont ezek az emulátorok az esetek többségében nem tudják az eredeti hardver időzítésit pontosan betartani, ezért nem teljesen autentikus a játék élmény. Hardveres emulálásnál általában egy FPGA chipet használnak és ebben implementálják az eredeti hardveres működéseket. Az általunk választott megvalósítás sokkal idő igényesebb, viszont ennek segítségével képesek vagyunk az eredeti eszköz leg pontosabb emulálására. Illetve egy erősebb FPGA segítségével modernizálhatunk egy régebbi konzolt is, persze bármilyen hardveres módosítás esetén (amely eltér az eredeti eszköztől), át kell gondolnunk, hogy a meg változott körülmények között is képesek leszünk-e futtatni az eredeti szoftvereket a hardveren.

Ebben a fejezetben azt fogom bemutatni, hogy milyen hardveres változtatásokat/fejlesztéseket eszközöltem, az eredeti NES-hez képest, hogy egy friss, modernebb megjelenést adjak az eredeti konzolnak.

4.1. Képalkotás

Már az előző 3.2.1 fejezetben olvashattuk, hogy a NES egy kompozit jelet állított elő CRT televíziók számára. Ez talán a játék konzol legelavultabb része, hiszen napjainkban már ezt a televízió típust nem is lehet beszerezni. Ezeket teljes mértékben leváltották a VGA alapú (DVI, HDMI, DisplayPort csatlakozóval ellátott) különböző méretű és felbontású lapos TV-k. Ezeknek a megvalósításoknak rengeteg előnye van az analóg megjelenítéssel szemben. Az egyik legjelentősebb, hogy ezen keresztül képesek vagyunk torzításmentes átvitelre.

A NES eredeti felbontása 256 x 240 pixel és 60 Hz (NTSC). Ez a méret sajnos nem felel meg a modern VGA szabványoknak, úgy hogy ezen a téren módosítanunk kell a PPU képgenerálásán. A módosítások során a VGA adatok generálását helyeztem elő térbe. A legkisebb VGA kép méret amivel érdemes dolgozni és a modern TV-k és monitorok támogatnak az a 640 x 480 pixel és 60 Hz, szerencsénkre ebbe a méretbe pontosan elfér a kétszeres NES kép méret. Ebből eredeztetve, ha egy pixel-t 2 x 2 pixel-el reprezentálunk, akkor 512 x 480 pixel méretű képet kapunk ennél bonyolultabb megoldás is létezik (bilineáris interpoláció), de ez a legegyszerűbb ezért kezdetben ezt implementálom. Ez a változtatás lehetővé teszi, hogy NES nyomtatott huzalozott kártyáján modern HDMI csatlakozót helyezzék el.

Illetve ebből a módosításból következőleg a rendszerünk működési frekvenciája is meg változik, a VGA jelünk pixel óra jele 25 Mhz lesz, az RGB adatok pedig 250 Mhz-el lesznek

továbbítva a TV vagy monitor felé. Ez azt jelenti, hogy a PPU-nk működési órajelét is meg kell változtatni (eredetileg körülbelül 5.37MHz NTSC modellben). A NES nyomtatott huzalozott kártyáján ezért egy magasabb órajelforrást kell elhelyezni. Az órajel pontos beállítására pedig az FPGA chip DCM (Digital Clock Manager) és PLL (Phase Locked Loop) funkcióit használhatjuk.

4.2. Audio

Az audió feldolgozó egység (Audio Process Unit, APU), egy öt külön álló hangsávot kezelő analóg komponens, mely a NES eredeti hangját, nem lineáris keverés segítségével állította elő ezekből a csatornákból. Ezt a jelet a hardveres emulálás során mi egy digitális jelként állítjuk elő az FPGA-val. Ahhoz, hogy eredeti eszköz hangját élvezni tudjuk ezt először egy DAC segítségével analóg jelékké kell konvertálnunk, végül pedig egy megfelelő erősítőn át egy Jack csatlakozóra kivezethetünk. Az erősítő típusa attól függ, hogy fülhallgatón vagy pedig hangszórón szeretnénk, hogy ezek az ikonikus dallamok megszólaljanak. Egy jó kompromisszumos megoldás a kártyát fülhallgató erősítővel ellátni mivel hangszórókból léteznek olyan modellek (belül erősítővel rendelkezők), amelyek ezzel az gyengébb erősítővel is működnek (így minden a két opció fennáll a konzol használatra). Az eredeti konzol mono hangzással rendelkezett viszont mi ezt jelet a DAC működése miatt, minden a két fülre kivezethetjük, ezzel sztereó hangzást imitálva (ettől még természetesen ez nem lesz valódi sztereó jel).

Mivel a NES nyomtatott huzalozott kártyáján már a képalkotás miatt elhelyezek egy HDMI csatlakozót, ezért egy kisebb I2C szint illesztő komponens segítségével a HDMI hang csatornái is bekötethetők. Így lehetővé téve, hogy egy esetleges későbbi fejlesztés során a TV/monitor beépített hangszóróján szólaljon meg játékkonzolunk.

4.3. Játékok tárolása

A NES játékok programkódja és Pattern tábla adatai az esetek többségében a játék kazetta saját program és karakter ROM-jában helyezkedtek el. Ez a megoldás rengeteg extra területet emészteni fel a nyomtatott huzalozott kártyán (a kazetta befogadó egységet rá kéne tervezni az eszközre), illetve az összes teszt játéknak fizikailag a birtokomba kell lennie ehhez (és egyéb hardveres teszteket nehezen lehetne megvalósítani).

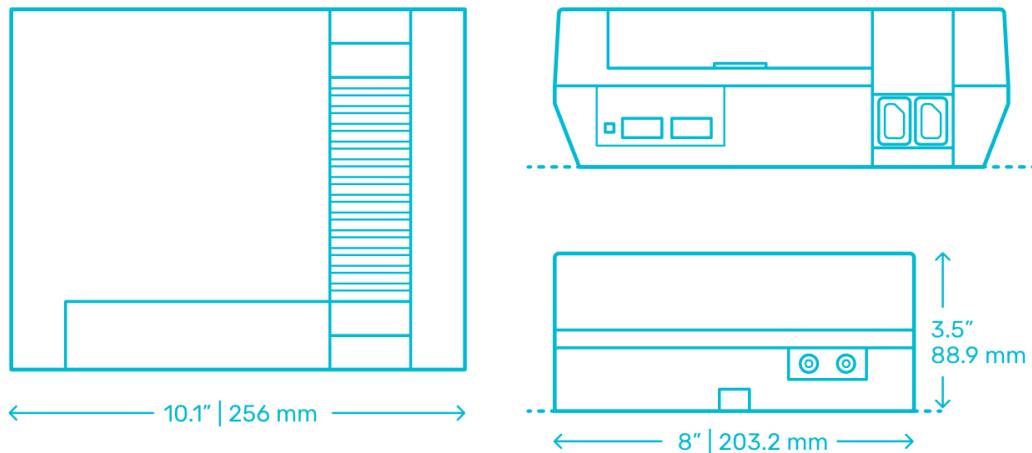
Ennek következtében egy új megoldást kellett kitalálni arra, hogy az emulált NES hardvert megfelelő módon ellássam játékokkal. Az egyik kézen fekvő megoldás, hogy a hordozó kártyára el lehet helyezni egy statikus RAM modult amelynek elég nagynak kell lennie ahhoz, hogy a játék kártyák karakter és program ROM-ját egyszerre tartalmazza. A legnagyobb NES játék 768 KB memória területtel rendelkezett és ez a Kirby's Adventure volt, ehhez képest az összes többi játék 512 KB-os vagy ennél kisebb volt. A NES hardver indítás előtt ezt a SRAM-ot fel kell töltenünk a játékkal, majd a hardver indítását követően a PPU és CPU innen fog dolgozni.

Annak érdekében, hogy több játékot is képesek legyünk tárolni érdemes egy nagyobb méretű háttértárat is tervezni az eszközre, amely a játékokat fogja tartalmazni, erre ideális lehet egy SD kártya. A projekt kezdeti szakaszában ezt még nem használom, viszont az esetleges jövőbeli fejlesztések miatt érdemes, már most a kártyára tervezni egy ilyen olvasót.

A NES hardver játék megjelenítéshez szükséges belső memória területeit, pedig az FPGA-ba kialakítható Blokk/LUT ram-ba helyezhetjük helyezhetjük el (Név táblák, OAM, másodlagos OAM).

4.4. Kompakt hordozható méret

A NES újra tervezésének egyik fő aspektusa a méret csökkentése. Az eredeti konzol 256 mm hosszú, 203.2 mm széles és 88.9 mm vastag volt. Ezt a modern nyomtatott áramkörök tervezésével, és a komponensek kis méretével sokkal kisebb területre csökkenhető. Ezzel kompakt hordozható kialakítást kölcsönözve a játék konzolnak.



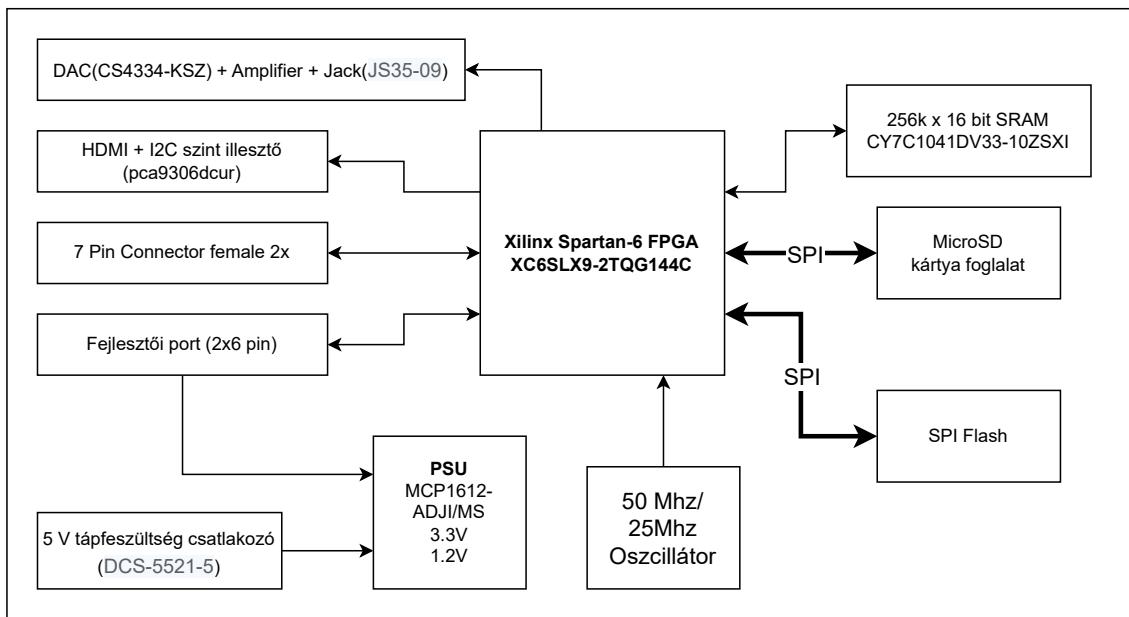
4.1. ábra. A NES játék konzol dimenziói

A kompakt méret mellett szerettem volna megőrizni az eszköz nosztalgia faktorát, ezért a kártyámon elhelyeztem két eredeti NES játék kontroller csatlakozót (kooperatív játékok miatt). Ez természetesen az eredeti hardver kontrollereivel kompatibilis.

5. fejezet

FPGA NES kártya ismertetése

A NES hardverének újragondolásából az 5.1 ábrán látható blokk diagramot készítettem, ez a nyomtatott huzalozott kártyák tervezésének első lépése. Már itt érdemes feltüntetni a különböző áramköri elemek közti kommunikációs utakat (busz típusokat), illetve ezek irányát. Ez alapján a diagram alapján, pedig elkezdődhet a különböző komponensek keresése, ezt követően át kell gondolnunk ezek fogyasztását és feszültség szintjeit ezekből az adatokból pedig megtervezhető a kártya táp ellátása is (a mi esetünkben már kiegészítettem ezzel a blokk diagramot).



5.1. ábra. NES kártya blokkdiagramja

A komponensek, közül az FPGA chip kiválasztása a legnehezebb, ennek menete általában az, hogy megpróbáljuk felmérni a hardverünk méretét és ez alapján választunk megfelelő méretű chip-et. A mi esetünkben az egyik legkomplexebb elem a 6502-es 8-bites processzor amely méretét az OpenCores weboldalon található nyílt forráskódú hardver tervek alapján megbecsülhetjük körülbelül 1000 LUT-ra. Mivel NES hardver működéséhez három fő komponens kell (CPU, APU, PPU) ezek méretét egyesével felülbecsülhetjük a legkomplexebb alkatrész méretével, így összesen 3000 LUT-ot kapunk. Ehhez még érdemes a VGA jel elő állítását (HDMI jel kódolása), illetve az audió jel kezelését még hozzá számolni, erre is jó felső becslés az 1000 LUT. Végül még érdemes tartalékkal is számolni ezért egy 5000-6000 LUT-al rendelkező FPGA chip valószínűleg elég nagy ahhoz, hogy a

teljes projekt elférjen benne. Fontos kritérium még, hogy DCM-el illetve PLL-el rendelkezzen a chip az egyedi órajelek előállítása érdekében (például a 250 MHz a VGA bit-ek kiadásához), ezen kívül még Blokk-RAM-ra is szükség lesz legalább akkorára mint a NES hardverének belső memóriája (például Név táblák 2 kilobájt).

A feladat megvalósításához egy Spartan-6-os Xilinx FPGA állt a rendelkezésemre, amely megfelel a fent említett összes elvárásnak. Ez a chip nagyban meggyorsította a nyáktervezés menetét is, mivel az egyetemi Spartan-6-os fejlesztő kártyák fő komponense is ez az FPGA volt (erről itt olvashatunk részletesen [6]). Ezt a fejlesztő kártyát vettem alapul a NES kártyám fejlesztő portjának kialakítása, az SPI flash bekötése, illetve a táp vonalak kialakítása során is.

A kártyán az alábbi komponensek találhatók:

- *FPGA*: Xilinx XC6SLX9-2TQG144C típusú Spartan-6-os FPGA, amely lehetővé teszi összetettebb logikák és mikroprocesszoros rendszerek megvalósítását. Az eszköz főbb jellemzői:
 - 5720 darab 6 bemenetű LUT és 11440 darab flip-flop
 - 32 darab 18 kilobites blokk-RAM
 - 16 darab DSP48A1 blokk (elő összeadó, 18 x 18 bites előjeles szorzó és akkumulátor)
 - 4 darab DCM (Digital Clock Manager) és 2 darab PLL (Phase Locked Loop) modul
- *Memóriák a program és az adatok tárolására*:
 - Egy 256k x 16 bites (512 kB), 10 ns-os aszinkron SRAM (Cypress CY7C1041DV33-10ZSXI)
 - Egy 32 Megabites SPI buszos soros FLASH memória (Atmel AT25DF321A), amely konfigurációs memóriaként is szolgál az FPGA számára
- *Egy MicroSD memóriakártya foglalat*:
 - Teljes MicroSD kártya protokoll
 - Egyszerű SPI protokoll
- *Beviteli eszközök*:
 - Két eredeti 7 lábas NES (GamePAD) kontroller csatlakozók
 - Reset és PROG gombok
- *Képfeldolgozás*:
 - HDMI csatlakozó
 - I2C szint illesztő (PCA9306DCUR típusú), a HDMI audió vonalainak illesztéséhez
- *Audio*:
 - Digitális-analóg átalakító (DAC), CS4334-KSZ típusú
 - 100mW Erősítő TS486IST típus (félhallgatókhöz)
 - CUI SJ1-3553NG 3.5mm Jack csatlakozó
- *Tápegységek*: MCP1612-ADJI/MS típusú szinkron Buck konverterek
- *Egy 50 MHz-es oszcillátor*
- *Csatlakozó a LOGSYS fejlesztői kábel számára*

5.1. Tápellátás

A tápellátás kialakítása a Logsys Spartan-6-os fejlesztői kártyáéhoz hasonló [6]. A NES kártya 5 V-os táfeszültségről működik. Ezt a tápellátást, vagy a fejlesztő kábelről kapja az eszköz, vagy egy külső 5 V-os forrásból. A külső egyenfeszültségű forrás Shottky diódával védtem, illetve a kártyát töltést jelző zöld led-del is elláttam (PWR).

Az 5 V-os forrást két azonos típusú step down (Buck) konverterrel 3.3 V-ra és 1.2 V-ra konvertálom. Alapvetően az FPGA működéséhez kell a két feszültség szint. A 3.3 V az I/O vonalakért, DCM, PLL, és konfigurációért felelős, az 1.2 V pedig az FPGA belső magjának kell. Ezt a két tápvonalat az FPGA dokumentációja alapján (a tág lábaihoz közel) elláttam a megfelelő mennyiségű csatoló (coupling) és hidegítő (bulk) kapacitásokkal a stabil működés végett. A 3.3 V-ot a kártyán található többi alkatrész is használja (SRAM, MicroSDkártya, kontrollerek, a fejlesztő kábel is megkapja mint JTAG referencia feszültség ként stb.). Illetve a kártya a HDMI csatlakozó, DAC és erősítő komponensek esetén, a tápellátás 5 V-ját is felhasználja. A tápellátás schematik rajzát az F.3.1 függelékben láthatjuk.

5.2. Órajel források

A NES kártyán a Logsys fejlesztő kártyához hasonlóan, egy 50 MHz-es oszcillátort helyeztem el. Az FPGA, vagy a fejlesztő portról érkező CLK-től kapja az órajelét, vagy ezt az 50 MHz-es CLK-t használja. Ahhoz, hogy az FPGA használhassa ezeket az órajeleket egy-egy órajel bemeneti lábára (GCLK) kellett ezeket bekötni. Az oszcillátor segéd áramkörét az F.3.5 függelékben láthatjuk.

5.3. Memória - SRAM

A választott asszinkron SRAM mérete 256 k x 16 bit (byte-okban mérve 512 k), ez a méret megfelel a 4.3 fejezetben tárgyalt játék méreteknek (csak egy NES játék nem fog beleférni ebbe a RAM-ba). A választott memória előnye, hogy 10 ns elérési idejű asszinkron, statikus RAM, egyszerű kezeléssel. Ez azért jelent előnyt a konzol hardveres emulálása szempontjából, mert nem fogja ennek működését befolyásolni a memória elérési idő (a későbbiekben az elérést igazíthatjuk az általunk választott időzítéshez). DRAM esetén sokkal több időzítési paraméterrel kell számolni ez természetesen jóval megnehezíti a időzítés kritikus hardver létrehozását.

Az SRAM 18 bites címmel rendelkezik, amellyel megcímhetők a 2 byte-osával az adataink (256 k cím). A RAM-ból kiolvasott, illetve beírandó adatokat pedig a 16 bit-es adat vonalak olvasásával és írásával érhetjük el. Az SRAM vezérlésétől függően kiolvasható vagy írható egyszerre mind a 16 bit, de van byte-os elérési mód is.

Az SRAM szabványos vezérlési felülettel rendelkezik. Tehát következő vezérlő jelek segítségével tudjuk irányítani (ezek mindegyike negált logikájú): chip engedélyezés CSn, írás engedélyezése WEn, olvasás engedélyezés OEn, alsó byte engedélyezés LBn, végül pedig a felső byte engedélyezés UBn. A memória olvasási és írási idő diagramjait az [5] adatlapon olvashatjuk, kiegészítő áramkörét pedig az F.3.4 függelékben láthatjuk.

5.4. Digital Analog Converter és erősítő

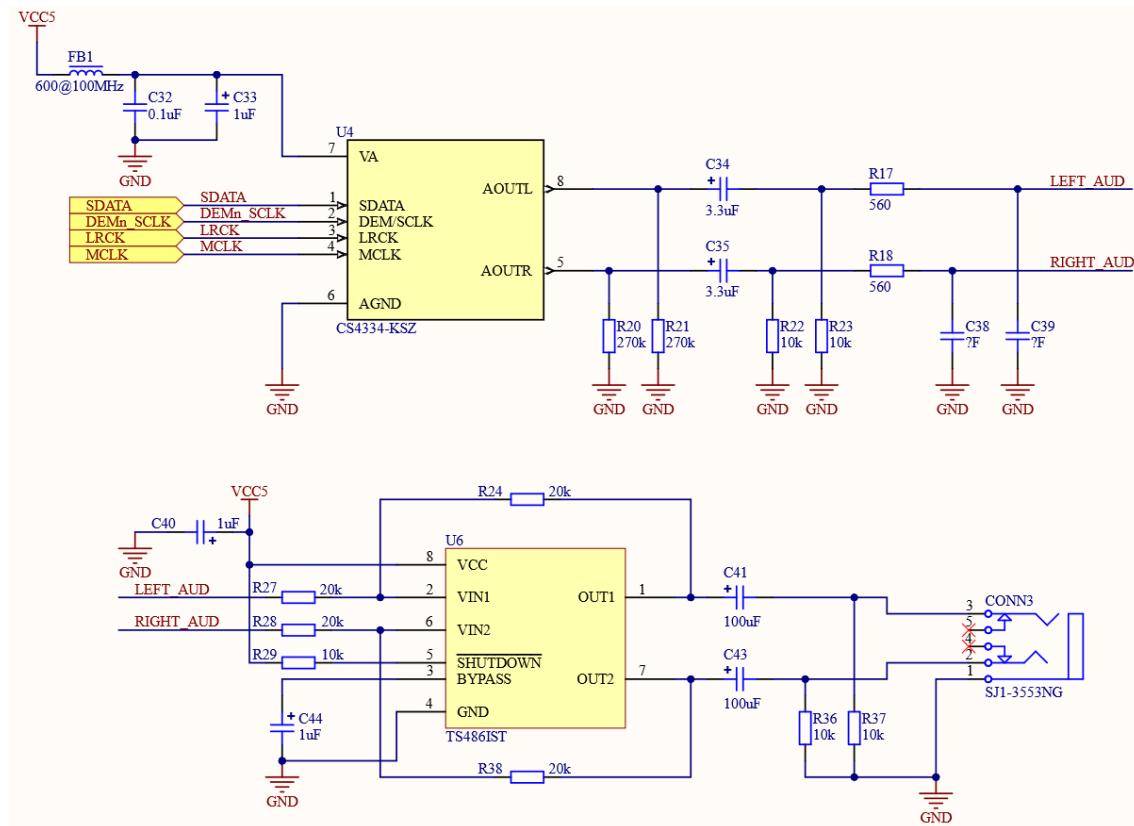
A NES kártya egyetlen analóg alkatrészekre támaszkodó része a DAC-ot követő erősítő áramkör, ennek részletes megértéséhez tekintsük meg az 5.2 ábrát, amely az F.3.3 függelékből lett kiemelve.

A DAC komponenst egy 100MHz-en 600 Ω -as Ferrite Bead-el védem, az esetleges 5 V-os tápvonalról beszűrődő zajokkal szemben, ide szűrési okokból tantál és kerámia kondenzátorokat is helyeztem. Az alkatrész az FPGA által elő állított mono digitális "hang" jelet, átalakítja és kiadja mind a két kimenetén. Ezek az analóg jelek fognak az erősítést követően, a Jack csatlakozó bal és jobb fülhöz menő lábára csatlakozni.

A DAC két kimenetét, egy egy 3.3 uF-os csatoló kondenzátorral választom el az analóg résztől. Az áramkörben található C38, C39-es kondenzátorok és R27, R28-as ellenállások egy alul áteresztő szűrőt valósítanak meg. A kondenzátorok értékét, pedig a következő képlet alapján határoztam meg (az ellenállások értéke kötött volt az erősítő miatt):

$$C = \frac{R + 560\Omega}{4} * \pi * F_s * (R * 560\Omega) \quad (5.1)$$

Itt F_s az általunk választott audió jel frekvenciája az 560 Ω pedig a soros ellenállás értéke. Ezek alapján a két kondenzátorom értéke 3.3 nF vagy 2.7 nF lehet, mivel így 48 KHz-hez közeli értéket kapunk a frekvenciára. Az összes kerámia kondenzátornak, amely az analóg áramkör része NP0 (vagy C0G) dielektrikummal kell rendelkeznie, mivel ezeknek nincs piezoelektromos tulajdonságuk.



5.2. ábra. NES audió jelért felelős áramkörök

Az erősítő komponensnek egy fázis fordító erősítőt választottam, amely erősítése az R24 és R38 ellenállásokkal módosítható a következő képletek alapján:

$$Gain_{LINEAR} = -\frac{R_{FEED}}{R_{IN}} \quad (5.2)$$

$$Gain_{dB} = 20 * \lg\left(\frac{RFEED}{RIN}\right) \quad (5.3)$$

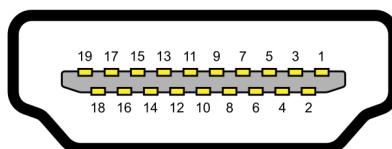
Az áramkörben jelenleg nem állítottam be erősítést az RFEED és RIN ellenállások értékét 20 kΩ-nak határoztam meg. Ez természetesen a hardveres tesztelés során cserélhető és állítható.

5.5. HDMI és I2C szint illesztő

A NES nyomtatott huzalozott kártyáján elhelyeztem egy HDMI csatlakozót és a körülötte elhelyezkedő áramkör segítségével felkészítettem VGA jelek kiadására. A teljes áramkör sematikus ábráját (schematic) az F.3.2 függelékben láthatjuk.

A jövőbeli fejlesztések miatt elhelyeztem a nyákon még egy I2C jel szint illesztőt is, amely segítségével az FPGA 3.3 V-on működő lábait, a HDMI audió jel kiadásáért felelős lábaihoz (SCL/SDA) illesztettem. Így a kártya támogatja a TV-k és monitorok beépített hangszóróit is.

Az alábbi ábrákon láthatjuk és olvashatjuk egy HDMI anya aljzat pin és láb kiosztását:



5.3. ábra. aljzat anya

Funkció	Láb	Funkció	Láb
TMDS Data2+	1	TMDS Clock Shield	11
TMDS Data2 Shield	2	TMDS Clock-	12
TMDS Data-	3	CEC	13
TMDS Data1+	4	Reserved	14
TMDS Data1 Shield	5	SCL	15
TMDS Data1-	6	SDA	16
TMDS Data0+	7	DDC/CEC Ground	17
TMDS Data0 Shield	8	+5 V Power	18
TMDS Data0-	9	Hot Plug Detected	19
TMDS Clock+	10		

5.1. táblázat. HDMI lábkiosztás

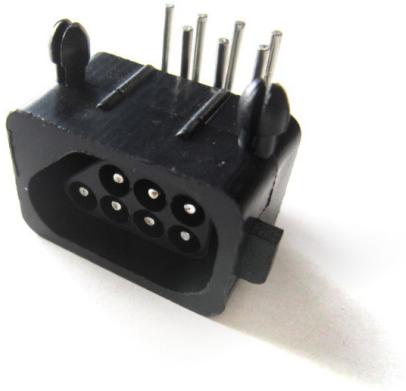
Mivel az FPGA nem minden I/O láb pára képes differenciális jelek küldésére, ezért figyelni kell, hogy a csatlakozót az FPGA melyik oldalához közel helyezem el. A HDMI szabvány az adat vonalak között 100 Ω-os impedancia különbséget ír elő (15% os toleranciával), ezért érdemes ezeket az vezetékeket minél rövidebben/egyszerűbben megoldani (FPGA bekötési oldalhoz közel).

Az alkatrészeim védelmére a HDMI 5 V-os tápellátását egy 100 mA-es biztosítékon (poli fuse) vezettem keresztül, ez túláram esetén véd. A csatlakozó fém burkolatát egy 1 MΩ-os ellenállással és egy 1 nF-os (1 kV-os) kapacitással földeltem, ez HDMI kimenetek esetén ideális.

5.6. A kártya bemenetei

A NES nyomtatott huzalozott kártyáját az eredeti játékkonzol kontroller csatlakozóival lát-tam el. Az eredeti kontroller a NES 5 V-járól működik, viszont a benne található parallel-soros átalakító (shift regiszter) adatlapja alapján 3.3 V-ról is működik. Ez azért fontos, mert így nem kell extra jelszint illesztő IC a kártyára, működtethetem az adat fogadást és az órajel küldést az FPGA I/O lábairól (3.3 V).

Ennek a kontrollernek egyedi hét lábas anya csatlakozója, van amit az 5.4 ábrán láthatunk.



5.4. ábra. NES kontroller csatlakozó

Ebből a hét lábból kettőnek csak rögzítési szerepe van, kettő a tápellátásért felelős és a maradék három lábon keresztül történik a kontrollerben található regiszter olvasása, vezérlése és a működési órajel küldése. A kártyára ebből a csatlakozóból kettőt helyeztem el a kooperatív játékok végett. A kontroller portok sematikus ábráját (schematic) az F.3.3 függelékben láthatjuk.

A pcb-n két gomb is helyet kapott az egyik az FPGA és ezáltal a NES reset gombja (RST), a másik pedig az FPGA újrakonfigurálását elindító nyomógomb (PROG). Az RST gomb pergésmentesítésért az FPGA a felelős. Ezek mindegyikét az F.3.5 függelékben láthatjuk.

5.7. MicroSD kártya

A MicroSD kártya csatlakozót teljes interfésszel tudtam implementálni, mivel az FPGA-nak még sok szabad I/O lába maradt. Ez azt jelenti hogy teljes SD kártya protokollt is megtudok valósítani a NES fejlesztő kártyán, az egyszerűbb soros SPI kommunikációt mellett/helyett. A MicroSD kártya segéd áramkörét az F.3.2 függelék sematikus rajzán láthatjuk.

Itt érdemes az áramkör ki-bekapcsolásáért felelős P-Mosfet-es áramkört megnézni, ez azért szükséges, mert a fent említett két protokoll közötti váltáshoz áramtalanítanunk kell az csatlakozót. A tápellátás elvétele mellett a felhúzó ellenállások tápellátását is elvesszük kikapcsolás során. Egyedül a CD kártya detektáló lábtól nem vesszük el, mivel ez csak azt jelzi, hogy van-e SD kártya a csatlakozóban (nem része a fent említett kommunikációs protokolloknak). Ezt be/ki kapcsolási eseményt az FPGA egy I/O lábának segítségével kontrollálhatjuk.

Az FPGA védelme érdekében elhelyeztem a az SD kártya CLK lábára egy $33\ \Omega$ -os soros ellenállást, ezt a PCB layout tervezése során a lehető legközelebb helyeztem el az FPGA-hoz.

5.8. FPGA konfigurációs módok

A NES kártyának is, a LOGSYS Spartan-6 FPGA kártyához hasonló módon [6] két konfigurációs módja van. Az FPGA-t felprogramozhatjuk a fejlesztőportban található JTAG interféssz segítségével, illetve felkonfigurálhatja saját magát a kártyán található soros FLASH memóriából is. A konfigurációs módok között egy rövidzár segítségével (jumper) váltha-

tunk a LOGSYS-es kártyához hasonlóan. Ennek működését az 5.2 táblázatban olvashatjuk, illetve az F.3.6 függelékben láthatjuk.

Jumper állása	Konfigurációs mód	Leírás
	JTAG	Az FPGA-t a JTAG interfacen keresztül kell felkonfigurálni.
	SPI	Az FPGA az SPI buszos soros FLASH memóriából konfigurálja fel magát a tápfeszültség bekapcsolása vagy a PROG gomb megnyomását követően.

5.2. táblázat. Fejlesztői port bekötése

5.9. Soros Flash memória

A NES kártyán egy Atmel AT25DF321A típusú, 32 Megabit-es SPI busszal rendelkező soros FLASH memóriát helyeztem el. Ez a komponens az FPGA számára konfigurációs memóriaként szolgál (tehát a NES hardverének binárisát fogja tartalmazni). Ennek a komponensnek az elhelyezése, bekötése és ezáltal működése is megegyezik a Logsys Spartan-6-os fejlesztői kártyán található Flash-el [6]. A soros Flash bekötését, pedig az F.3.4 függelékben láthatjuk.

5.10. LOGSYS fejlesztői port

A fejlesztői port kialakítása teljes mértékben megegyezik a Logsys fejlesztő kártyán található port-al, ennek köszönhetően az FPGA felprogramozása történhet a MIT tanszéken tervezett egyedi fejlesztő kábellel. Ennek részletes bemutatása a [6]-os dokumentáció része. A fejlesztői port kialakítását a ?? képen látható és FPGA bekötését pedig a ?? táblázatban olvasható, illetve az F.3.5 függelékben látható. Az FPGA sikeres felkonfigurálását egy zöld LED-el jelzem (DONE).

5.11. Nyomtatott áramköri terv

A schematik megalkotását követően, a pcb tervezés következő fázisa a layout (pcb rajzolat) elkészítése. Itt természetesen figyelembe kell vennünk az eddig meghatározott célokat 4.4, miszerint egy kompakt hordozható eszközöt tervezünk. Egy nyomtatott áramkör mérete nagyban függ a réteg felépítésétől, illetve a kiválasztott komponensek méretétől. Tehát minél több rétegből épül fel egy pcb és minél modernebb alkatrészeket használunk, annál nagyobb felületi alkatrész sűrűség érhető el. Természetesen ezekkel arányosán az ár is nő. Viszont mivel először egy prototípust fejlesztek, ezért egy kompromisszumos megoldást kellett választanom.

5.11.1. Réteg beállítások

A Logsys Spartan-6-os fejlesztő kártyán már láthattuk [6], hogy a választott FPGA chip TQFP tokozása miatt, két rétegű nyákon behuzalozható. Az FPGA NES kártya tervezésekor ez egy fő szempont volt, mivel így érdekes mérnöki megoldásokat kellett alkalmaznom a tág bekötése során, illetve a kártya elkészítési költségét is csökkenteni tudtam.

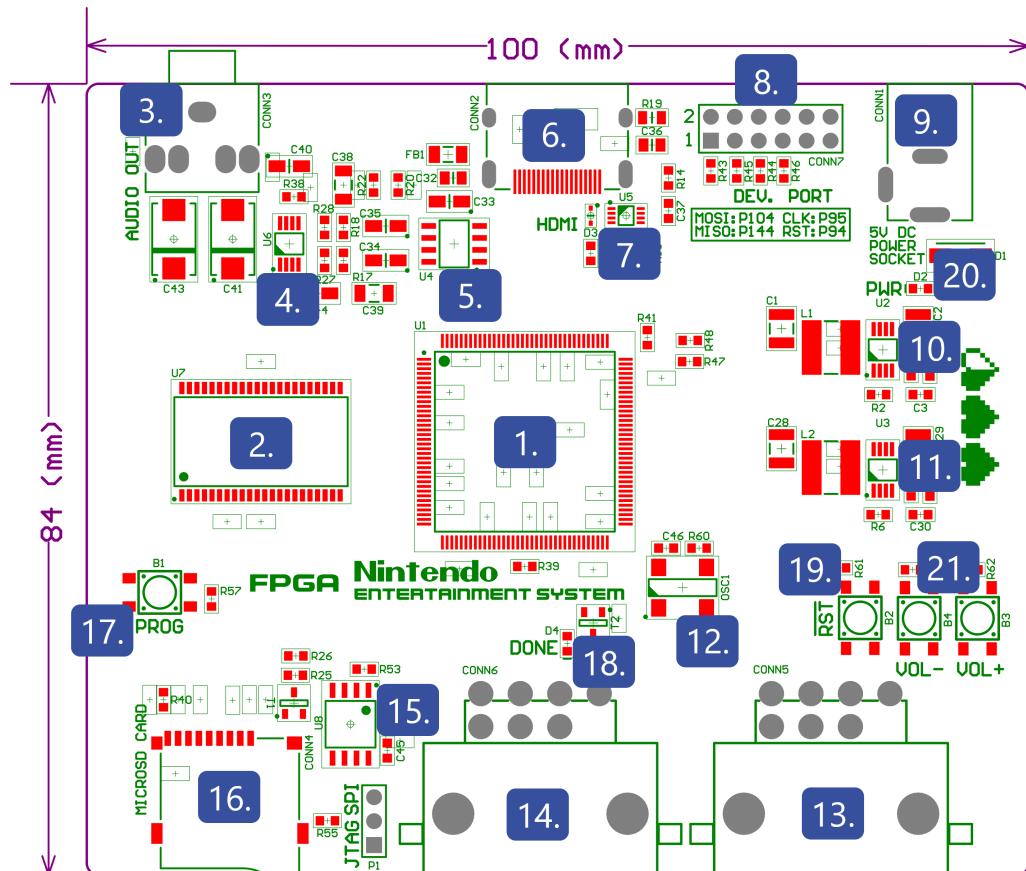
A prototípus kártyákat a jlpcb nevű kínai nyomtatott áramkör gyártó fogja gyártani. Ahhoz, hogy az Altium tervező program képes legyen bonyolultabb számítások (differential pair routing) és szimulációk készítésére, a réteg felépítést meg kell adnunk a PCB-nk számára. Ez a kínai gyártó által szolgáltatott információk által [1] alakítottam ki, ez az 5.5 ábrán látható.

#	Name	Material	Type	Weight	Thickness	Dk
	Top Overlay		Overlay			
	Top Solder	Solder Resist	Solder Mask		0.015mm	3.8
1	Top Layer		Signal	1oz	0.035mm	
	Dielectric 1	FR-4	Dielectric		1.5mm	4.5
2	Bottom Layer		Signal	1oz	0.035mm	
	Bottom Solder	Solder Resist	Solder Mask		0.015mm	3.8
	Bottom Overlay		Overlay			

5.5. ábra. Az FPGA NES kártya réteg beállításai

5.11.2. Komponensek elhelyezése

A rétegbeállításokat követően, a komponensek és segéd áramkörei elhelyezése következett. Ennek segítségével tudtam meghatározni végül, hogy az FPGA mely I/O bankjaihoz/lába-ihoz fogom vezetni a különböző komponensek kivezetéseit, illetve ez határozta meg pcb-m méreteit is. Az alkatrészek és segéd áramkörei rajzolatát az F.1 függelékben láthatjuk, de a könnyebb áttekinthetőség érdekeben elkészítettem a következő ábrát:



5.6. ábra. A top layer alkatrész elhelyezési terve

FPGA NES kártya főbb alkotó komponensei:

1. Xilinx XC6SLX9-2TQG144C típusú FPGA
2. 256k x 16 bites (512 kB), 10 ns-os aszinkron SRAM (Cypress CY7C1041DV33-10ZSXI)
3. 3.5mm Jack csatlakozó (CUI SJ1-3553NG)
4. 100mW Erősítő (TS486IST fülhallgatókhöz)
5. Digital Analog Converter (DAC), (CS4334-KSZ)
6. HDMI csatlakozó
7. 2C szint illesztő (PCA9306DCUR)
8. Csatlakozó a LOGSYS fejlesztői kábel számára
9. 2.1mm 12 V / 5 A DC táp csatlakozó (DC10A)
10. 3,3 V feszültséget előállító tápegység
11. 1,2 V feszültséget előállító tápegység
12. 50 MHz-es oszcillátor
13. 7 pines NES GamePAD kontroller csatlakozók anya aljzat 1
14. 7 pines NES GamePAD kontroller csatlakozók anya aljzat 2
15. 32 Mbites SPI buszos soros FLASH (Atmel AT25DF321A)
16. MicroSD kártya foglalat
17. Az FPGA újrakonfigurálását indító gomb (PROG)
18. Az FPGA sikeres felkonfigurálását jelző LED (DONE)
19. Az FPGA kézi reset gomja (RSTn)
20. A bekapcsolt tápfeszültséget jelző LED (PWR)
21. A NES hangerő szabályzó gombjai (VOL-, VOL+)

Mivel kézi forrasztással és hőfűvő segítségével fogjuk a kártyát összeállítani, ezért törekedtem arra, hogy az összes komponens (és ezek segéd áramkörei) a előlap (top) oldalon helyezkedjen el, illetve a hátoldalra (bottom) csak azonos méretű elemek kerüljenek. Egyedül a HDMI csatlakozó biztosítéka került a hátoldalra, ami nem 0603-as méretű lett.

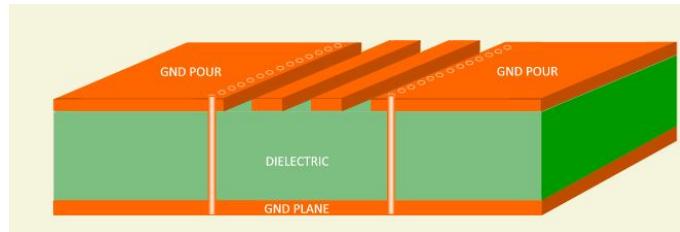
Ezt követően a PCB alkatrészek bekötésével foglalkoztam, itt a alsó és felső réteget is jel/föld (signal/GND) rétegként alakítottam ki. Ez azt jelenti, hogy jel vezetékeket és táp vonalak bekötését követően az egész nyák felszínén föld kitöltést hoztam létre, így növelte a jelek integritását. A alsó és a felső föld rétegeket, pedig via-k segítségével kötöttem össze (via stiching). A két réteget egyszerre az F.2 függelékben láthatjuk, az itt használt réteg ábrázolási mód az Altium designer átlátszó 2D módja, amely betekintést enged a föld kitöltések alá. A felső réteg jeleit és komponensek pad-jei vörös színnel vannak jelölve, a alsó réteg pedig kék színnel.

5.11.3. HDMI adatvonalainak bekötése

A HDMI vonalak kialakításáról már az 5.5 fejezet során olvashattunk. Viszont mivel a réteg kialakítás során a két rétegű megvalósítást választottunk és a PCB gyártó (jlcpcb) nem biztosít két réteg esetén impedancia kontrollálásra réteg felépítést. Ezért a HDMI szabványban leírt 100Ω -os impedancia különbséget (15% os toleranciával) csak egy speciális differenciális pár bekötés típusával tudjuk biztosítani. Ez a típus a Dual Strip Coplanar Waweguide Grounded, ez azt jelenti, hogy differenciális pár bekötés mellett figyelnünk kell arra is, hogy a pár jobb és bal oldalán fix távolságra föld kitöltést helyezzünk el. Az itt található két réteg földjét előre meghatározott távolságokon via fence-szel látjuk el (ezt az elrendezést az 5.7 ábrán láthatjuk). A via-k közti maximális távolságot a következő képlettel számolhatjuk ki:

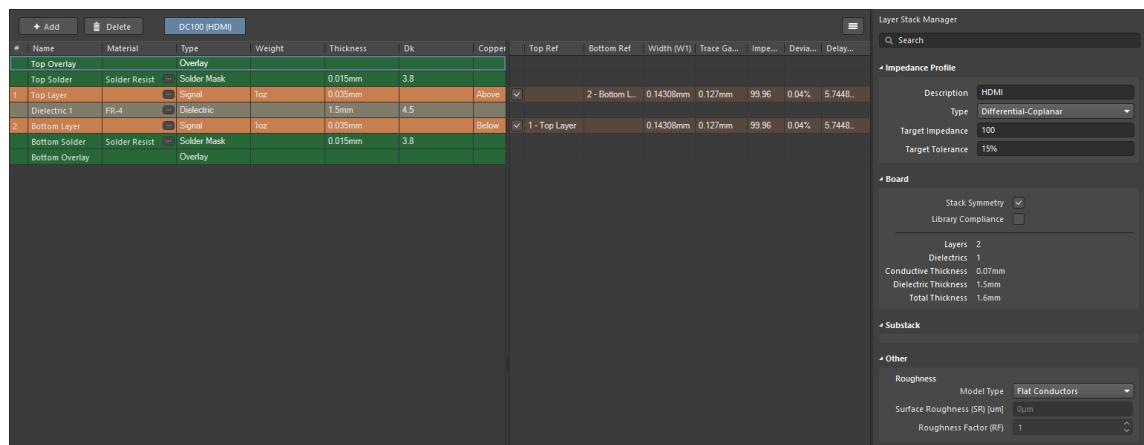
$$S(via) = \frac{\lambda}{20} = \frac{c}{20 * f * \sqrt{\epsilon_r}} \quad (5.4)$$

Az 5.4 képletben található λ a differenciális jelünk hullámhossza (a részletesebb képletben pedig: c a fény terjedési sebessége, f a jelünk frekvenciája, ϵ_r pedig az anyag dielektromos állandója). A mi esetünkben ezen a vezeték páron 250 Mhz-es jeleket fogunk küldeni, ennek hullámhossza körülbelül 1,151 m, ebből kiszámított két via közti maximális távolság 0.058 m (ennél természetesen választhatunk kisebb értéket is).



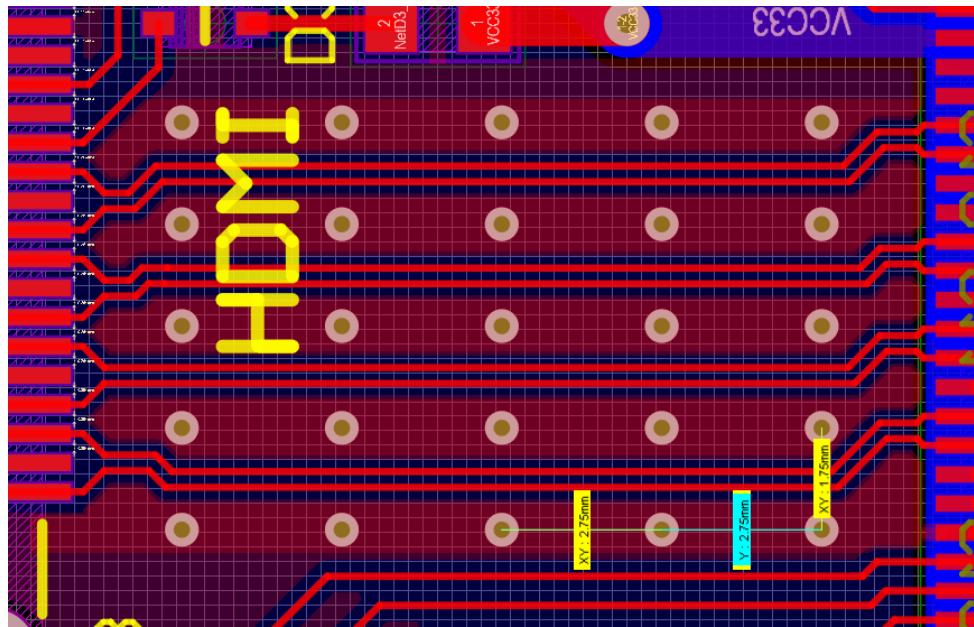
5.7. ábra. Dual Strip Coplanar Waweguide Grounded felépítése

Az Altium designer segítségével van lehetőségem a GND réteg és vezető pár közti távolság kiszámítására (később az ecad szoftver ez alapján fogja elhelyezni a vezetékeket). Ezek a beállítások az alábbi ábrán láthatók:



5.8. ábra. Coplanar differential pair routings Altium beállításai

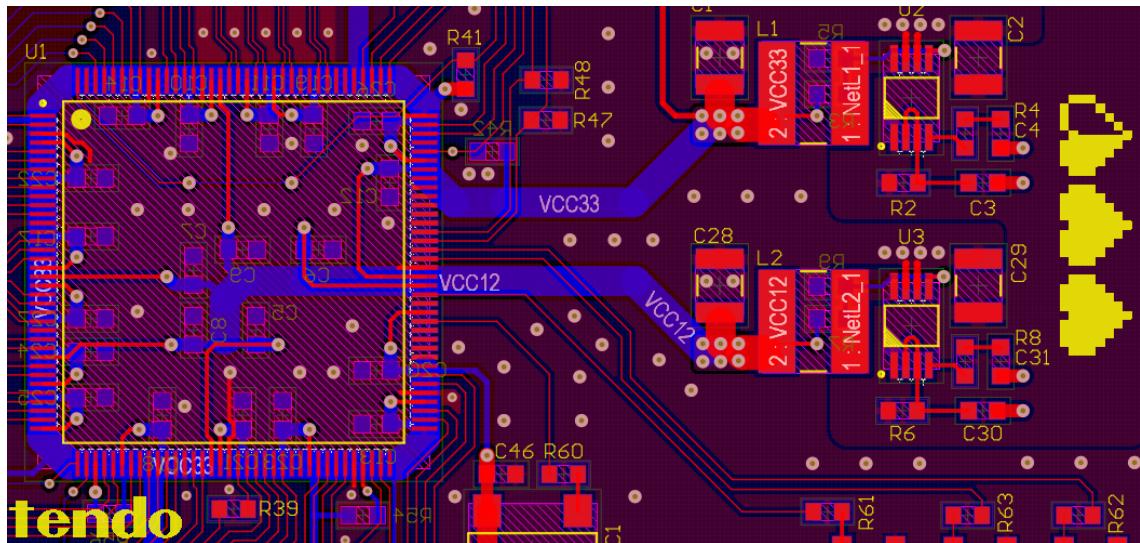
A FPGA NES kártya HDMI jeleinek bekötésére 2,75 mm-es via fence méretet választottam (körülbelül a fentebb kiszámolt érték 20-ad része) ez részletesen az 5.9 ábrán látható, amely az F.2 függelékből lett kiemelve.



5.9. ábra. A HDMI adatvonalainak bekötése

5.11.4. FPGA táp vonalak kialakítása

Ahhoz, hogy a Spartan-6-os FPGA chip-et két rétegen teljes mértékben ki tudjuk használni egy speciális tápellátási módszerhez kellet folyamodnunk (Logsys-es fejlesztő kártya táp vonalai is hasonlóan vannak kialakítva). Ennek alapja az volt, hogy az alsó oldalról érkezik a 3.3 V és az 1.2 V-is. A 3.3 V-ot az FPGA lábai alatt végig vezetjük (innen indul ki a többi 3.3 V-os komponens tápellátást is) egy helyen be engedve az 1.2 V-ot a belső magja számára. Ezzel a rendszerrel az előnye, hogy így az összes csatoló (coupling) és hidegítő (bulk) kapacitás az FPGA alatt kaphat helyet, ezzel szabadon tartva a top réteget az FPGA I/O bankjainak és JTAG interfészének bekötésére.

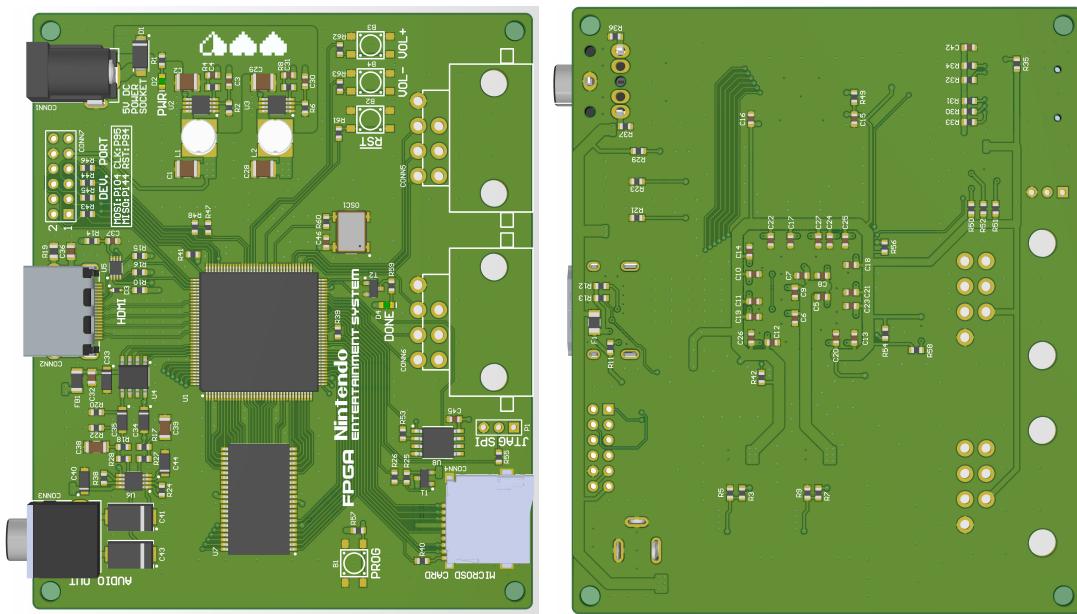


5.10. ábra. FPGA tápellátásának kialakítása

A chip tápellátását az 5.10 ábrán láthatjuk, a kártya teljes tápellátását pedig az F.2 függelékben figyelhetjük meg.

5.11.5. FPGA NES 3D terve

A kártya tervezését követően, az Altium designer lehetőséget nyújt PCB 3D tervének megtekintéséhez (ez a komponensek elhelyezésénél, illetve nyák foglalatok/tokok tervezésénél is hasznos). Ez egy teljes képet nyújt a nyomtatott áramkör kialakításáról és jövőbeli kinézetéről. Az FPGA NES 3D tervét a következő (5.11) ábrán láthatjuk:



5.11. ábra. 3D PCB rajzolat

6. fejezet

FPGA tervezés

6.1. Rendszer blokkvázlat bemutatása

6.2. Működési órajel választása

6.3. Picture Process Unit

6.3.1. Eredeti rendrelés menete

6.3.2. VGA rendelés

6.3.3. Háttér renderelési állapot gép

```
1  always @ (posedge clk)
2  begin
3      if (rst)
4          bgrender_state <= SLEEP;
5  else
6      case (bgrender_state)
7          SLEEP: begin
8              if ((x_rendercntr == FIRST_SCANLINE_PIXEL)
9                  && ((y_renderingcntr >= START_OF_VBLANK_ROW) && ~(y_renderingcntr == PRERENDERING_ROW)))
10                 bgrender_state <= VBLANK;
11             else if ((x_rendercntr == FIRST_SCANLINE_PIXEL) && oddframe && (y_renderingcntr ==
12 FIRST_RENDERING_ROW))
13                 begin
14                     ppu_addr_fetch <= ppu_nt_addr;
15                     nametable_rd_sel <= 1'b1;
16                     rd_req_reg <= 1'b1;
17
18                     bgrender_state <= NT;
19                 end
20             else if (x_rendercntr == FIRST_SCANLINE_PIXEL)
21                 begin
22                     ppu_addr_fetch <= bg_lsb_addr;
23
24                     bgrender_state <= IDLE;
25                 end
26             else
27                 bgrender_state <= SLEEP;
28         end
29     IDLE: begin
30         if (x_rendercntr[2:0] == BG_NEXT_STEP_CONDITION)
31             bgrender_state <= NT;
32         else
33             begin
34                 ppu_addr_fetch <= vram_addr_reg;
35                 bgrender_state <= IDLE;
36             end
37     NT: begin
```

```

38         if ((x_rendercntr == END_OF_BG_RENDERING_LINE)
39             || (y_renderingcntr == PRERENDERING_ROW) && oddframe && (x_rendercntr ==
40                 ODDFRAME_END_OF_BG_RENDERING_LINE))
41             begin
42                 nametable_read_reg <= 1'b0;
43
44                 bgrender_state <= SLEEP;
45                 end
46             // ODDFRAME_END_OF_FIRST_NT is good here because x_rendercntr will always be higher then this
47             just in the first line
48             else if ((x_rendercntr == ODDFRAME_END_OF_FIRST_NT) || (x_rendercntr == START_OF_LAST_NT
49             ))
50                 begin
51                     // commands get here for NT fetch
52                     nametable_read_reg <= 1'b0;
53
54                     bgrender_state <= NT;
55                     end
56             else if (x_rendercntr[2:0] == BG_NEXT_STEP_CONDITION)
57                 begin
58                     // commands get here for AT fetch
59                     nametable_read_reg <= 1'b0;
60
61                     bgrender_state <= AT;
62                     end
63             else
64                 begin
65                     // commands get here for NT Load
66                     if (x_rendercntr[2:0] == MEM_READ_START)
67                         begin
68                             ppu_addr_fetch <= ppu_nt_addr;
69                             nametable_rd_sel <= 1'b1;
70                             rd_req_reg <= 1'b1;
71                         end
72                     else if (x_rendercntr[2:0] == MEM_READ_CONTROLL_OFF)
73                         begin
74                             nametable_rd_sel <= 1'b0;
75                             rd_req_reg <= 1'b0;
76                         end
77                     else if (x_rendercntr[2:0] == MEM_READ_TAKE)
78                         nametable_read_reg <= 1'b1;
79
80                     bgrender_state <= NT;
81                     end
82                 end
83             AT: begin
84                 if (x_rendercntr[2:0] == BG_NEXT_STEP_CONDITION)
85                     begin
86                         // commands get here for BG_LSB fetch
87                         attribute_read_reg <= 1'b0;
88
89                         bgrender_state <= BG_LSB;
90                         end
91                     else
92                         begin
93                             // commands get here for AT Load
94                             if (x_rendercntr[2:0] == MEM_READ_START)
95                                 begin
96                                     ppu_addr_fetch <= ppu_at_addr;
97                                     attribute_rd_sel <= 1'b1;
98                                     rd_req_reg <= 1'b1;
99                                 end
100                            else if (x_rendercntr[2:0] == MEM_READ_CONTROLL_OFF)
101                                begin
102                                    attribute_rd_sel <= 1'b1;
103                                    rd_req_reg <= 1'b0;
104                                end
105                            else if (x_rendercntr[2:0] == MEM_READ_TAKE)
106                                begin
107                                    attribute_read_reg <= 1'b1;
108                                end

```

```

107
108
109         bgrender_state <= AT;
110     end
111 end
112 BG_LSB: begin
113     if (x_rendercntr[2:0] == BG_NEXT_STEP_CONDITION)
114     begin
115         // commands get here for BG_MSB fetch
116         bg_lsb_read_reg <= 1'b0;
117
118         bgrender_state <= BG_MSB;
119     end
120     else
121     begin
122         // commands get here for BG_LSB load
123         if (x_rendercntr[2:0] == MEM_READ_START)
124         begin
125             if (sprite_read)
126                 ppu_addr_fetch <= sprite_lsb_addr;
127             else
128                 ppu_addr_fetch <= bg_lsb_addr;
129             bg_lsb_rd_sel <= 1'b1;
130             rd_req_reg <= 1'b1;
131         end
132         else if (x_rendercntr[2:0] == MEM_READ_CONTROLL_OFF)
133         begin
134             bg_lsb_rd_sel <= 1'b0;
135             rd_req_reg <= 1'b0;
136         end
137         else if (x_rendercntr[2:0] == MEM_READ_TAKE)
138         begin
139             bg_lsb_read_reg <= 1'b1;
140         end
141
142
143         bgrender_state <= BG_LSB;
144     end
145 end
146 BG_MSB: begin
147     if (x_rendercntr[2:0] == BG_NEXT_STEP_CONDITION)
148     begin
149         // commands get here for NT fetch
150         bg_msb_read_reg <= 1'b0;
151
152         bgrender_state <= NT;
153     end
154     else
155     begin
156         // commands get here for BG_MSB load
157         if (x_rendercntr[2:0] == MEM_READ_START)
158         begin
159             if (sprite_read)
160                 ppu_addr_fetch <= sprite_msb_addr;
161             else
162                 ppu_addr_fetch <= bg_msb_addr;
163             bg_msb_rd_sel <= 1'b1;
164             rd_req_reg <= 1'b1;
165         end
166         else if (x_rendercntr[2:0] == MEM_READ_CONTROLL_OFF)
167         begin
168             bg_msb_rd_sel <= 1'b0;
169             rd_req_reg <= 1'b0;
170         end
171         else if (x_rendercntr[2:0] == MEM_READ_TAKE)
172         begin
173             bg_msb_read_reg <= 1'b1;
174         end
175
176         bgrender_state <= BG_MSB;
177     end
178 end

```

```

179 VBLANK: begin
180   if (x_rendercntr == END_OF_BG_RENDERING_LINE)
181     bgrender_state <= SLEEP;
182   else
183     bgrender_state <= VBLANK;
184   //ppu_addr_fetch <= vram_addr_reg;//vram_addr_reg
185   end
186   default:
187     bgrender_state <= IDLE;
188   endcase
189 end
190

```

6.3.4. Sprite rendering állapot gép

6.3.5. CPU által elérhető regiszterek és CPU adatbusz

6.3.6. PPU adatbusz és memória elérése

6.4. NES memória felépítése FPGA-ban

6.5. DMA

6.6. 6502 processzor működése

6.7. NES kontrollerek kezelése

7. fejezet

A NES tesztelése

7.1. Rendszer szimulációk

7.2. Hardveres tesztek

7.2.1. Donkey Kong

7.2.2. Super Mario Bros.

7.3. A tesztek eredményeinek kiértékelése

8. fejezet

Összefoglalás, jövőbeli tervezek

A Diplomatervezés 1 során sikerült foglalkoznom a projekt irodalom kutatásával, és NES nyomtatott áramkörének FPGA alapú újratervezésével, illetve a PPU hardverének felépítésével és felújításával.

A Diplomatervezés végeztéig még meg kell ismernem a CPU és APU hardverét is részletesen, illetve ezeket a PPU-val együtt meg kell terveznem a spatan-6-os FPGA-ba. Ezt követően pedig tesztelni kell ezeket az éles hardveren.

A projekt hosszútávú tervei közé tartozik az MicroSD kártyáról történő játék betöltés és minél több NES játék mapper-ének lefejlesztése, ezzel teljes körű hardveres emulálást készítve a Nintendo Entertainment System-hez.

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Ábrák jegyzéke

3.1. Nintendo Entertainment System	3
3.2. Nintendo Entertainment System NTSC alaplap [4]	4
3.3. Nintendo Entertainment System játék kazetta [2]	5
3.4. Komponensek közti kapcsolat (fent játék kazetta) [4]	6
3.5. Katódsugárcsöves TV-k működése	8
3.6. Super Mario Bros. Pattern táblái	9
3.7. Gumba (Mario egyik ellenfele) bal felső csempe elem szín adatai [3]	9
3.8. A MOS 6502 egyedi nyolcbites architektúrája	13
3.9. A MOS 6502 processzor státusz regisztere (P)	14
4.1. A NES játék konzol dimenziói	17
5.1. NES kártya blokkdiagramja	18
5.2. NES audió jelért felelős áramkörök	21
5.3. aljzat anya	22
5.4. NES kontroller csatlakozó	23
5.5. Az FPGA NES kártya réteg beállításai	25
5.6. A top layer alkatrész elhelyezési terve	25
5.7. Dual Strip Coplanar Waweguide Grounded felépítése	27
5.8. Coplanar differential pair routings Altium beállításai	27
5.9. A HDMI adatvonalainak bekötése	28
5.10. FPGA tápellátásának kialakítása	28
5.11. 3D PCB rajzolat	29

Táblázatok jegyzéke

3.1. A PPU memória kezelése (14 bit címek) mirroring jelenséggel	10
5.1. HDMI lábkiosztás	22
5.2. Fejlesztői port bekötése	24

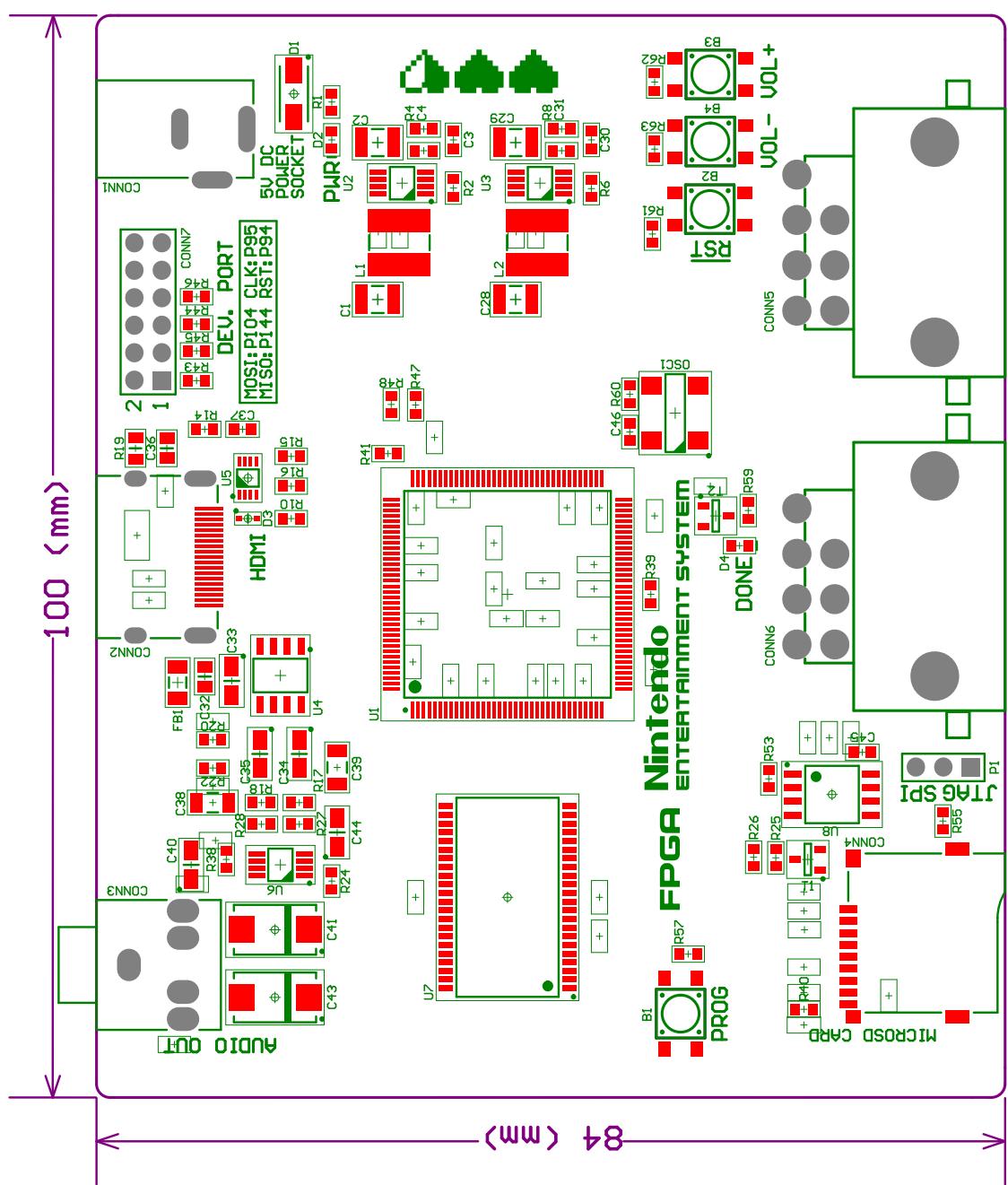
Irodalomjegyzék

- [1] JLCPCB: *PCB Manufacturing and Assembly Capabilities*. JLCPCB, 2023. 06. <https://jlcpcb.com/capabilities/pcb-capabilities>.
- [2] NesHacker youtube csatorna: Nes carts explained (2023. október 21.). <https://www.youtube.com/watch?v=GssRNEaKoPw>.
- [3] NesHacker youtube csatorna: Nes graphics explained (2023. október 21.). https://www.youtube.com/watch?v=7Co_8dC2zb8.
- [4] NesHacker youtube csatorna: Nes hardware explained (2023. október 21.). <https://www.youtube.com/watch?v=mMq4FFUnBPc>.
- [5] Cypress Perform: *CY7C1041DV33 - 4-Mbit (256 K × 16) Static RAM*. Cypress Perform, 2023. 06. https://datasheet.lcsc.com/lcsc/1804240720_Cypress-Semicon-CY7C1041DV33-10ZSXI_C32338.pdf.
- [6] Raikovich Tamás: *LOGSYS SPARTAN-6 FPGA KÁRTYA (V2.1) FELHASZNÁLÓ ÚTMUTATÓ*. Budapesti Műszaki és Gazdaságtudományi Egyetem, 2023. 06. http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_SP6_FPGA_Board.pdf.

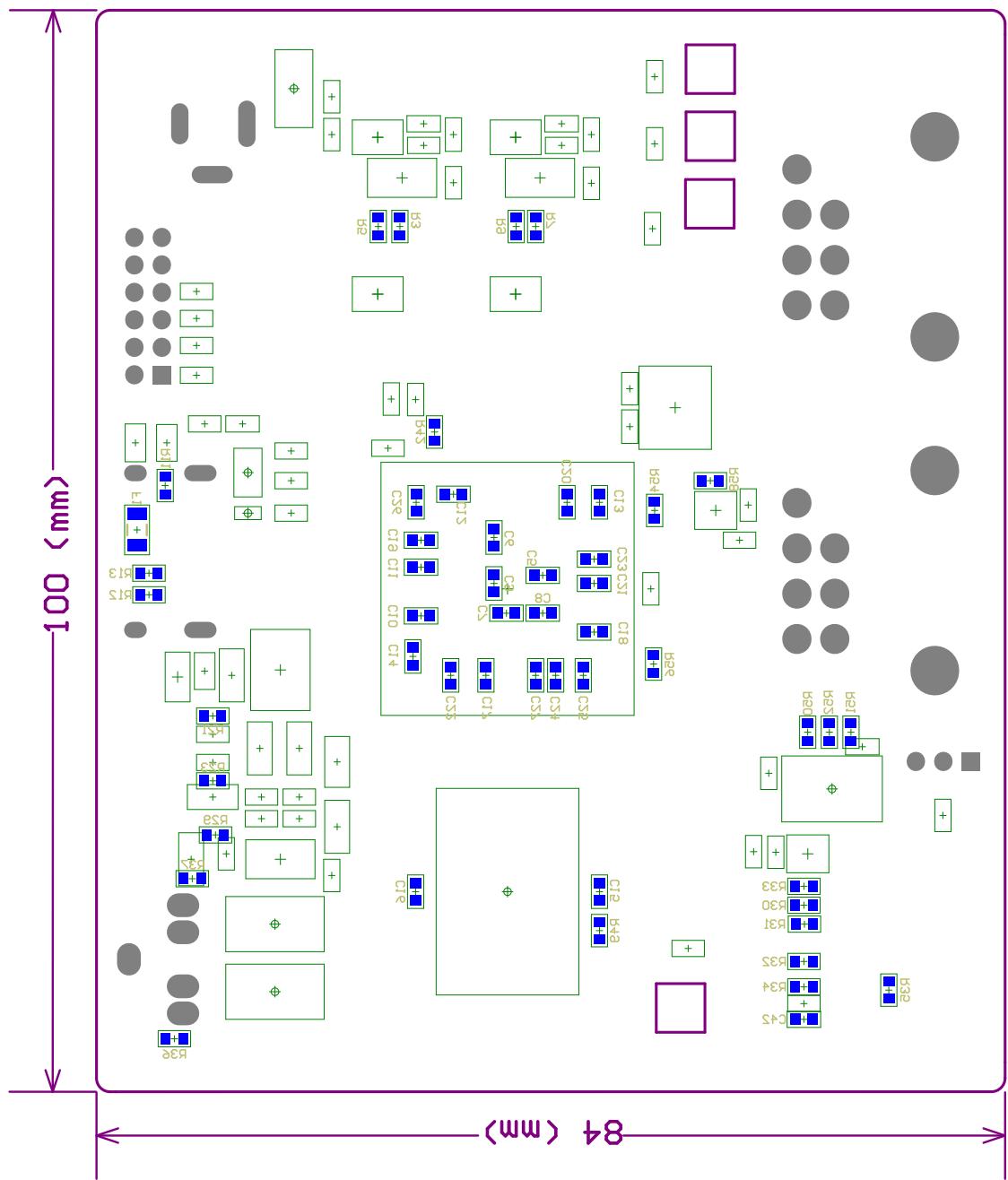
Függelék

F.1. NES kártya alkatrész elhelyezési terve

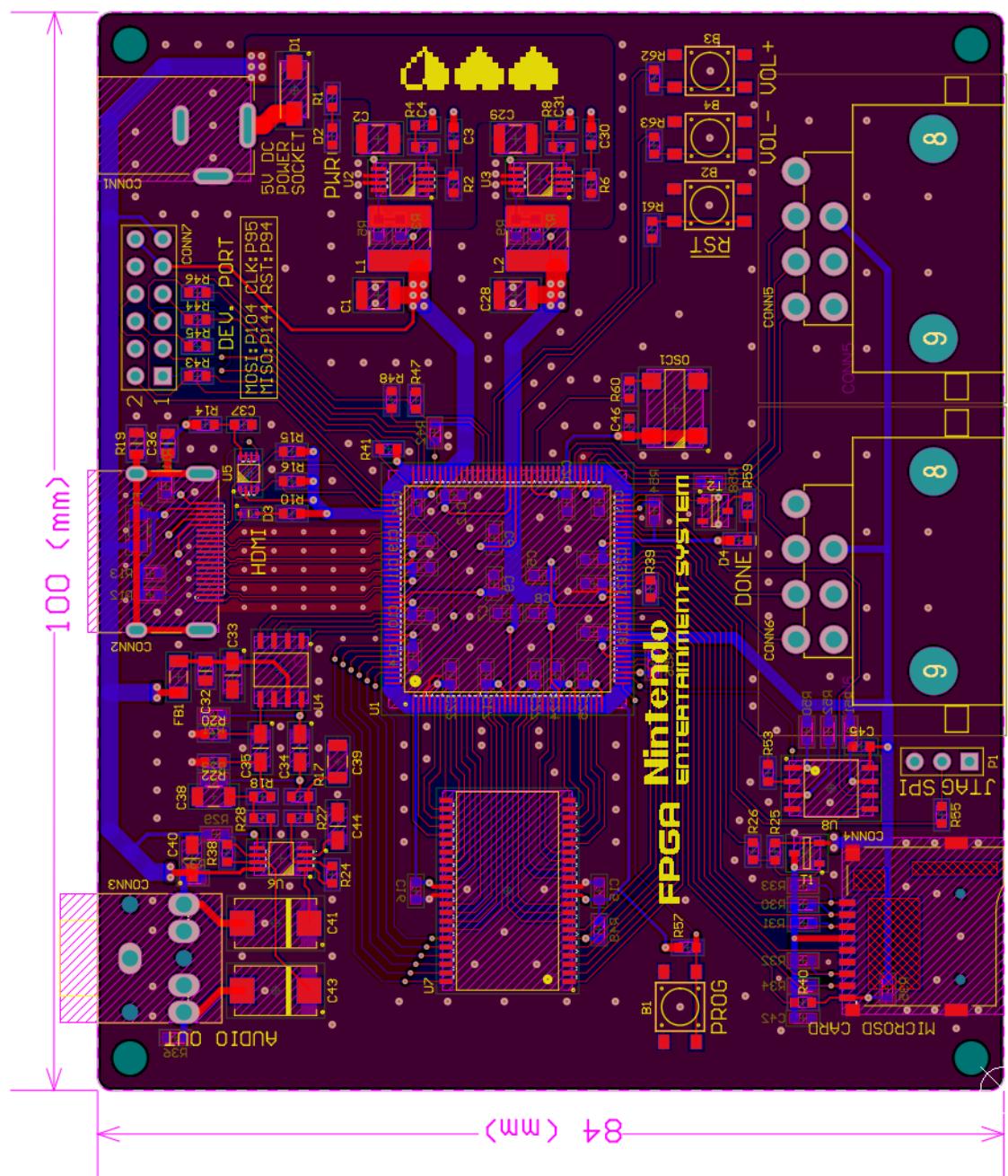
F.1.1. Top



F.1.2. Bottom

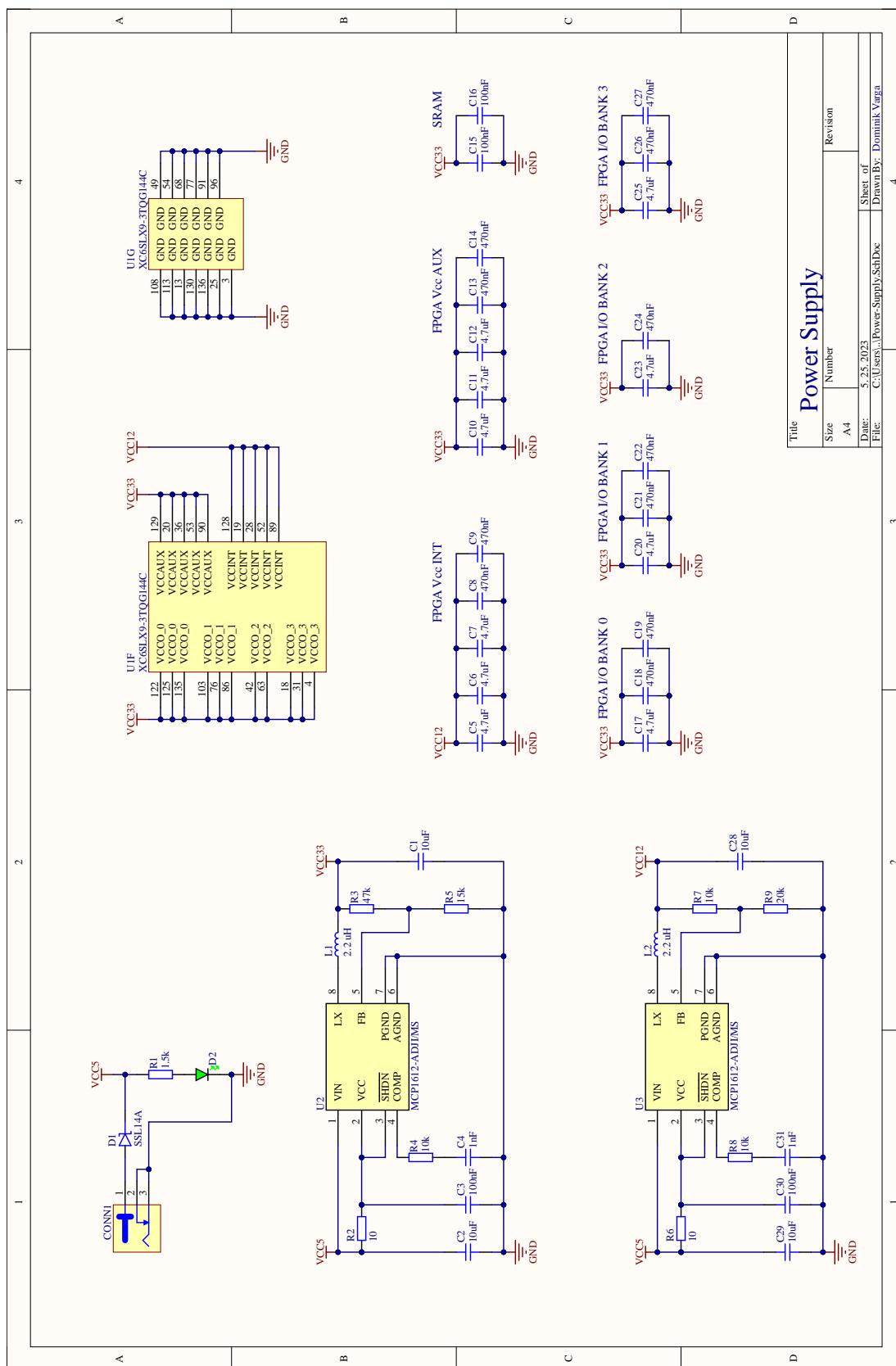


F.2. Nyomtatott áramköri terve (2D transparent)

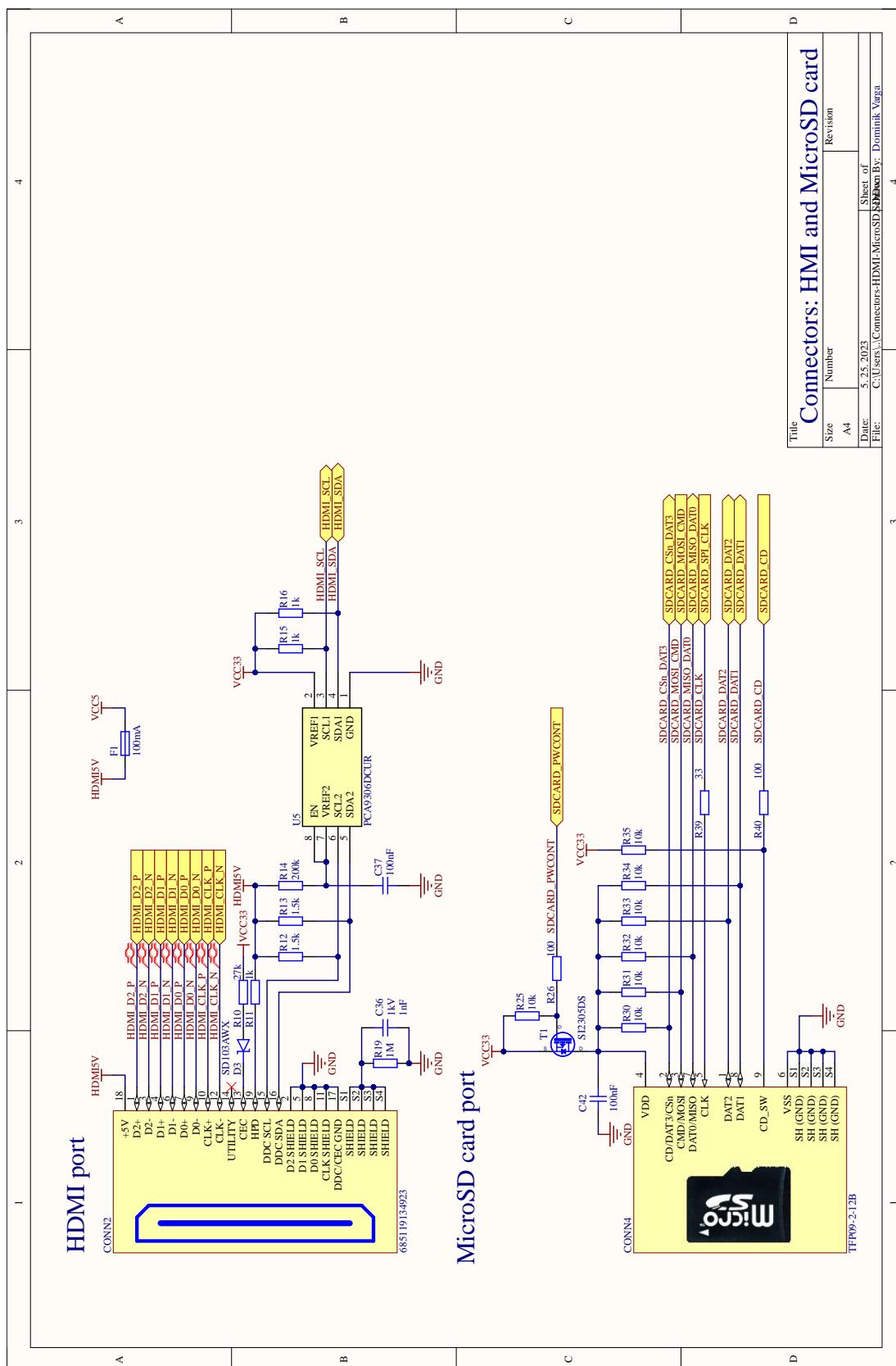


F.3. FPGA NES kártya kapcsolási rajza

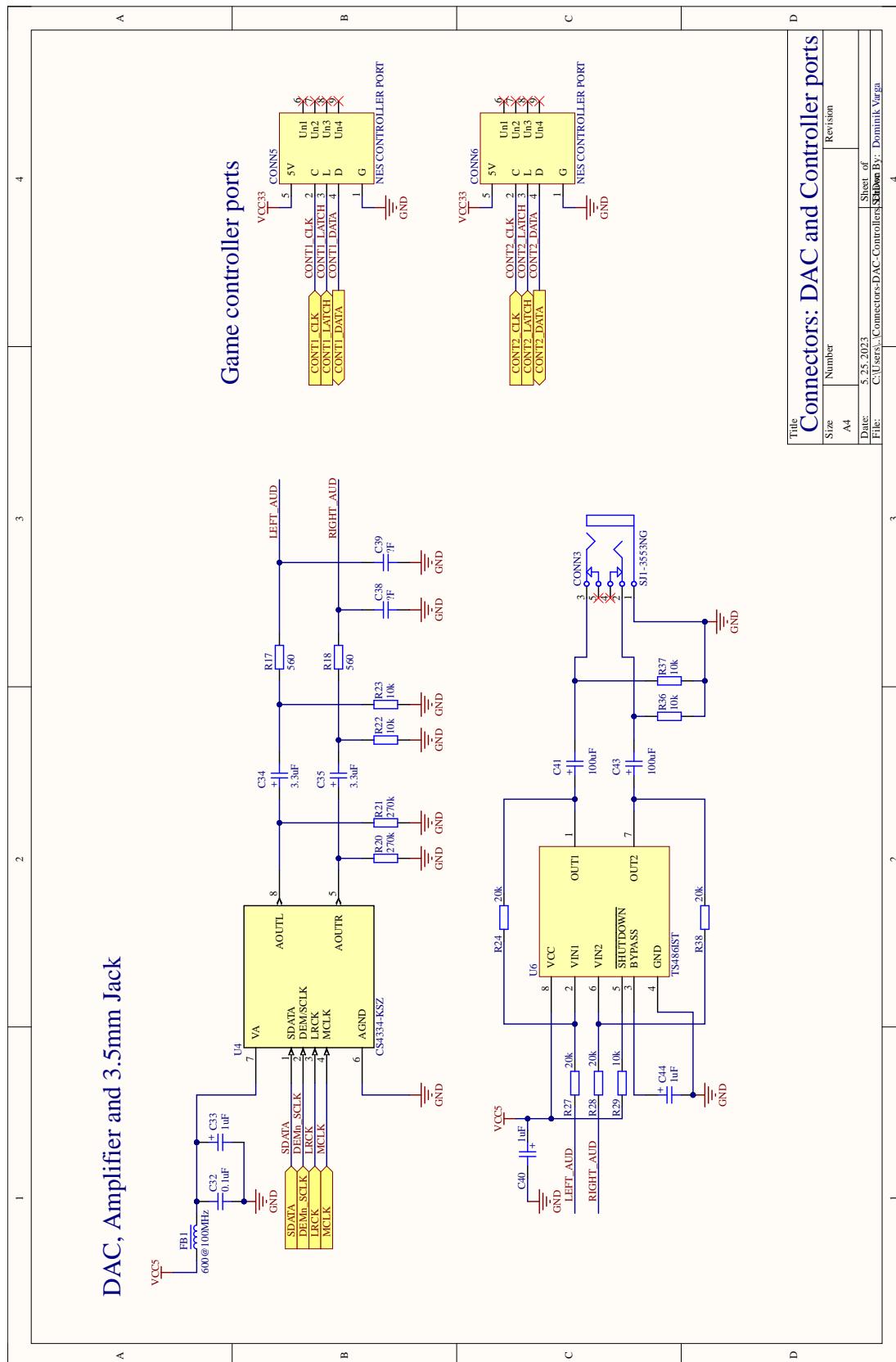
F.3.1. Tápegység



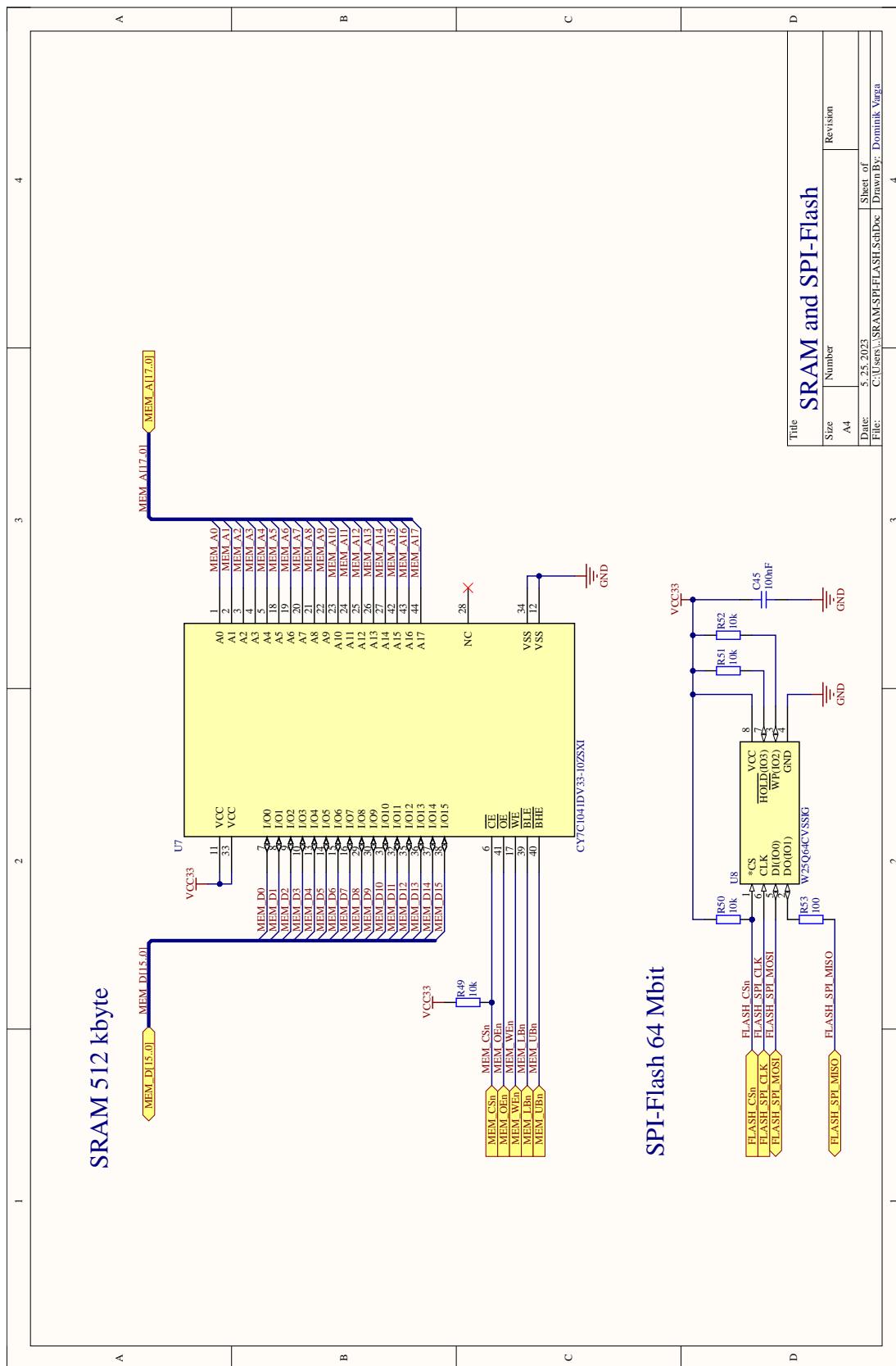
F.3.2. HDMI és MicroSD kártya csatlakozó



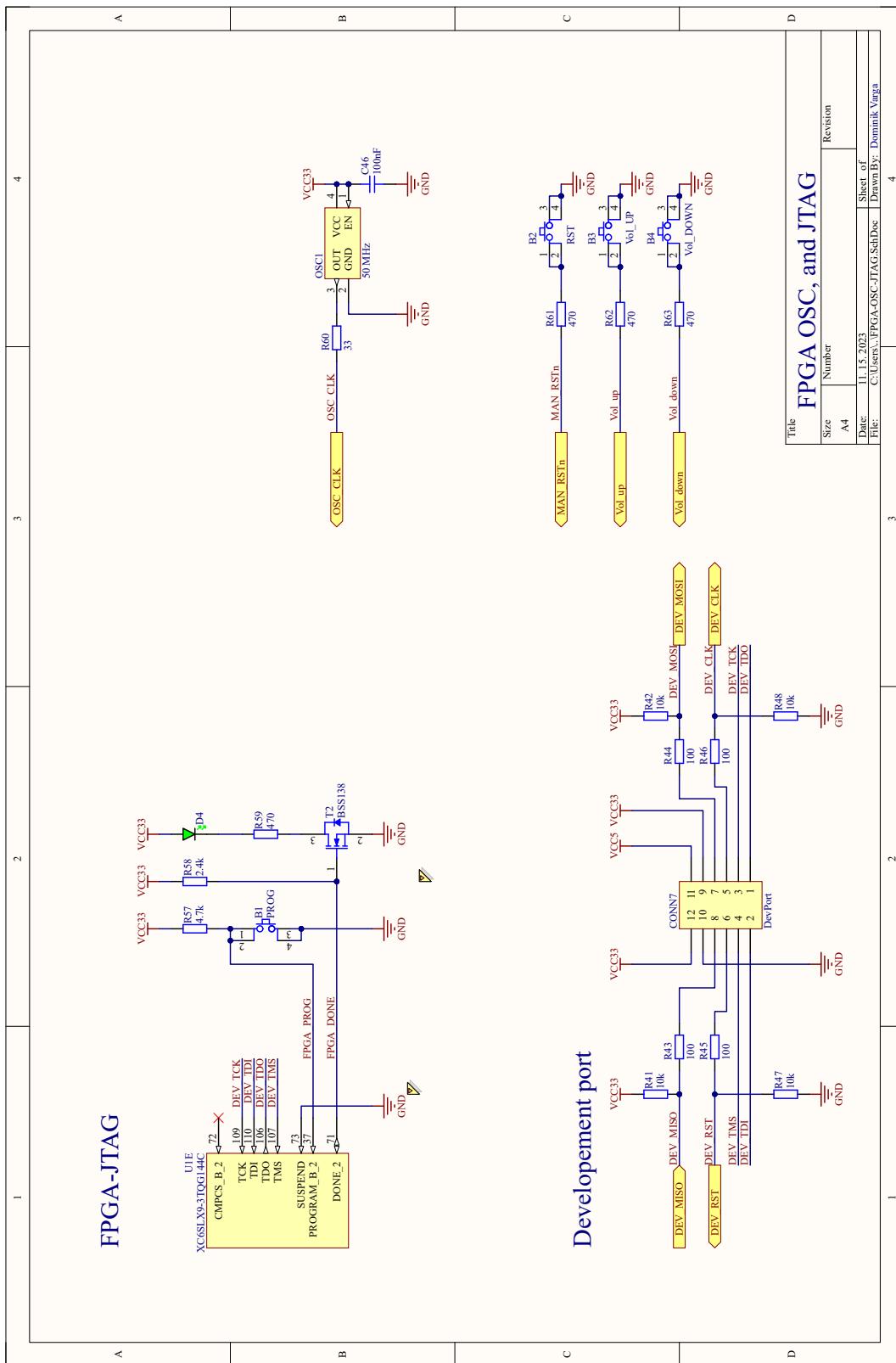
F.3.3. DAC, erősítő és kontroller áramkörök



F.3.4. SRAM és SPI-Flash



F.3.5. FPGA OSC és JTAG



F.3.6. FPGA IO bankok

