



## DIPLOMATERVEZÉSI FELADAT

**Varga Dominik**  
Villamosmérnök hallgató részére

### A NES videojáték konzol FPGA alapú megvalósítása

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.

A hallgató feladatának a következőkre kell kiterjednie:

- A rendelkezésre álló források alapján ismerje meg a konzol belső felépítését, valamint az egyes részegységeinek (processzor, APU, PPU) működését
- Gondolja át, hogy az FPGA alapú megvalósításból adódóan milyen eltérések, módosítások szükségesek az eredeti változathoz képest
- Tervezze meg FPGA-t tartalmazó kártyát (kapcsolási rajz, nyomtatott áramkör)
- Valósítsa meg a NES hardvert az FPGA-ban
- Készítse el a használathoz szükséges szoftvert
- Próbálja ki az elkészült rendszert néhány játékprogrammal, valamint értékelje az elért eredményeket

**Tanszéki konzulens:** Raikovich Tamás, tanársegéd

**Külső konzulens:** -

Budapest, 2023.04.01.

.....  
Dr. Dabóczi Tamás  
tanszékvezető  
egyetemi tanár, DSc



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. december 13.

# Tartalomjegyzék

## Kivonat

### Abstract

<b>1. Bevezetés</b>	<b>1</b>
<b>2. Felhasznált eszközök</b>	<b>2</b>
2.1. Altium Designer . . . . .	2
2.2. Xilinx ISE . . . . .	2
<b>3. Nintendo Entertainment System ismertetése</b>	<b>3</b>
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) . . . . .	15
3.3.3. Direct Memory Access - OAM DMA . . . . .	16
3.4. NES kontroller . . . . .	17
<b>4. NES FPGA alapú újra gondolása</b>	<b>18</b>
4.1. Képalkotás . . . . .	18
4.2. Audio . . . . .	19
4.3. Játékok tárolása . . . . .	19
4.4. Kompakt hordozható méret . . . . .	20
<b>5. FPGA NES kártya ismertetése</b>	<b>21</b>
5.1. Tápellátás . . . . .	23
5.2. Órajel források . . . . .	23
5.3. Memória - SRAM . . . . .	23
5.4. Digital Analog Converter és erősítő . . . . .	23
5.5. HDMI és I2C szint illesztő . . . . .	25
5.6. A kártya bemenetei . . . . .	25
5.7. MicroSD kártya . . . . .	26
5.8. FPGA konfigurációs módok . . . . .	26

5.9. Soros Flash memória . . . . .	27
5.10. LOGSYS fejlesztői port . . . . .	27
5.11. Nyomtatott áramköri terv . . . . .	27
5.11.1. Réteg beállítások . . . . .	27
5.11.2. Komponensek elhelyezése . . . . .	28
5.11.3. HDMI adatvonalaiknak bekötése . . . . .	30
5.11.4. FPGA tár vonalak kialakítása . . . . .	31
5.11.5. FPGA NES 3D terve . . . . .	32
<b>6. FPGA tervezés</b>	<b>33</b>
6.1. Adatbuszok implementálása . . . . .	34
6.2. Működési órajel választása . . . . .	35
6.3. Picture Process Unit . . . . .	36
6.3.1. CPU által elérhető regiszterek és CPU adatbusz . . . . .	36
6.3.2. Háttér képalkotást és memória olvasást vezérlő állapot gép . . . . .	38
6.3.3. Mozgó csempe elemek "Sprite" képalkotás . . . . .	40
6.3.4. PPU adatbusz és memória elérése . . . . .	44
6.3.5. A PPU részéről történő kép generálás működése . . . . .	46
6.3.6. A VGA képalkotás . . . . .	48
6.4. NES memória felépítése FPGA-ban . . . . .	49
6.5. DMA - FPGA megvalósítása . . . . .	51
6.6. 6502 processzor működése az FPGA rendszerben . . . . .	52
6.7. NES kontrollerek kezelése . . . . .	53
<b>7. Az FPGA NES tesztelése</b>	<b>54</b>
7.1. Rendszer szimulációk . . . . .	54
7.2. Hardveres tesztek . . . . .	56
7.3. A tesztek eredményeinek kiértékelése . . . . .	59
<b>8. Összefoglalás, jövőbeli tervezek</b>	<b>60</b>
<b>Köszönetnyilvánítás</b>	<b>61</b>
<b>Ábrák jegyzéke</b>	<b>63</b>
<b>Táblázatok jegyzéke</b>	<b>64</b>
<b>Irodalomjegyzék</b>	<b>64</b>
<b>Függelék</b>	<b>67</b>
F.1. NES kártya alkatrész elhelyezési terve . . . . .	67
F.1.1. Top . . . . .	67
F.1.2. Bottom . . . . .	68
F.2. Nyomtatott áramköri terv (2D transparent) . . . . .	69
F.3. FPGA NES kártya kapcsolási rajza . . . . .	70
F.3.1. Tápegység . . . . .	70
F.3.2. HDMI és MicroSD kártya csatlakozó . . . . .	71
F.3.3. DAC, erősítő és kontroller áramkörök . . . . .	72
F.3.4. SRAM és SPI-Flash . . . . .	73
F.3.5. FPGA OSC és JTAG . . . . .	74
F.3.6. FPGA IO bankok . . . . .	75
F.4. NTSC PPU eredeti képalkotás . . . . .	76

## 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. december 13.

---

*Varga Dominik*  
hallgató

# Kivonat

A Nintendo Entertainment System (NES) egy ikonikus videójáték-konzol, amelyet a Nintendo eredetileg 1983-ban Japánban, Famicom néven (Family Computer) mutatott be, majd később, 1985-ben az Egyesült Államokban és más régiókban a Nintendo Entertainment System néven jelent meg.

A diplomatervem témaja, a NES viszonylag egyszerű, jól átgondolt hardverét, egy olcsóbb, viszonylag kevesebb erőforrással rendelkező FPGA eszközben történő megvalósítása. Ez lehetővé teszi a régebbi hardver tovább gondolását, modern megjelenítő interfések használatával (VGA, DVI, HDMI, stb.), valamint az eredeti játékkazetták helyett modern adattárolók segítségével.

Ehhez a diplomatervem első felében, részletesen meg kellet ismernem az eredeti hardver működését, majd ezt követően úgy tovább gondolni a hardvert, hogy az egyedi változatnak képesnek kell lennie futtatni az eredeti játékprogramokat. A tervezéseket követően, elkészítettem a NES tovább gondolt alaplaptát, amelyet egy olcsóbb spartan-6-os FPGA chippel láttam el. Ezek mellett a nyák természetesen rendelkezik modern megjelenítő és audio interfésszel, illetve háttér tárolóval is. Ezt követően elkezdtem a felújított NES hardver fejlesztését az ISE Design Suit fejlesztő programban, Verilog nyelven.

Végül az elkészült hardvert és FPGA modulok, teljes rendszer tesztje során, a tesztelt játékprogramok (Super Mario Bros., Donkey Kong) futtatása bizonyítja a projekt sikerét.

# Abstract

The Nintendo Entertainment System (NES) is an iconic video game console originally introduced by Nintendo in Japan in 1983 as the Famicom (Family Computer), and later released in 1985 in the United States and other regions as the Nintendo Entertainment System.

The subject of my thesis is the implementation of the relatively simple, well-designed hardware of the NES in a cheaper FPGA device with relatively fewer resources. This allows for further innovation of the older hardware, using modern display interfaces (VGA, DVI, HDMI, etc.) and modern data storage instead of the original game cartridges.

In order to design the hardware, I had to first learn in detail how the original hardware worked, and then design a custom version of the hardware that would be able to run the original game programs. After the planning, I created a customized motherboard for the NES, which I equipped with a cheaper Spartan-6 FPGA chip. Alongside these, the PCB has a modern display and audio interface, as well as background storage (SD card). Then I started developing the refurbished NES hardware in the ISE Design Suite development program in Verilog.

In the end, I tested the completed hardware and FPGA modules, with full system testing, by running the game programs (Super Mario Bros., Donkey Kong) as proof of the success of the project.

## 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 diplomatervem során, a NES viszonylag egyszerű, jól átgondolt hardverét valósítottam meg egy olcsóbb, viszonylag kevesebb erőforrással rendelkező FPGA eszközzel. Ennek a megvalósításnak több előnye is van, például ki lehet használni a modern megjelenítő interféseket (VGA, DVI, HDMI, stb.), valamint az eredeti játékkazetta helyett alkalmazni lehet modern adattároló eszközöket is (SD kártya). Természetesen ezen „továbbfejlesztések” mellett az egyedi változatnak képesnek kell lennie futtatni az eredeti játékprogramokat.

A következőkben a projekt során elkészült egyedi, FPGA alapú NES megvalósítás hardver és szoftveres komponenseinek bemutatását olvashatjuk. Először is az eredeti NES működését, majd a hardver tovább fejlesztését ismerhetjük meg. Ezt követően pedig, a FPGA NES továbbfejlesztett alaplapját, és az FPGA-ban elkészült komponensek működését és tesztelését láthatjuk.

## 2. fejezet

# Felhasznált eszközök

### 2.1. Altium Designer

Az Altium Designer egy teljes körű elektronikai tervező automatizációs szoftvercsomag (EDA), amelyet nyomtatott áramköri lapok (PCB-k) tervezésére alkalmaznak. Egységes környezetben kombinálja a kapcsolás rajz és a PCB-elrendezés tervezését, ezzel lehetővé téve a tervezőknek a grafikus áramköri tervek gyors és hatékony létrehozását. A szoftver támogatja a hierarchikus tervezést és valós idejű kapcsolások ellenőrzését a schematic készítésekor. A PCB-elrendezés során előnyös funkciói közé tartozik az interaktív nyomvonalkövetés, a 3D-s PCB-vizualizáció és a különböző tervezési szabályok ellenőrzése.

Az Altium Designer integrált megközelítése lehetővé teszi a tervezők számára, hogy egyetlen platformon belül hozzák létre és optimalizálják elektronikai terveiket. A szoftver nemcsak a tervezési folyamatot egyszerűsíti, hanem magában foglalja a szimulációs és elemző eszközöket is, amelyek segítik a tervezőket az elektronikai rendszerek teljesítményének és megbízhatóságának előzetes értékelésében.

### 2.2. Xilinx ISE

A Xilinx ISE (Integrated Softver Environment) a Xilinx Inc. által kifejlesztett, széles körben használt szoftvercsomag volt. Átfogó eszközök készletet 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.

### 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 [5]

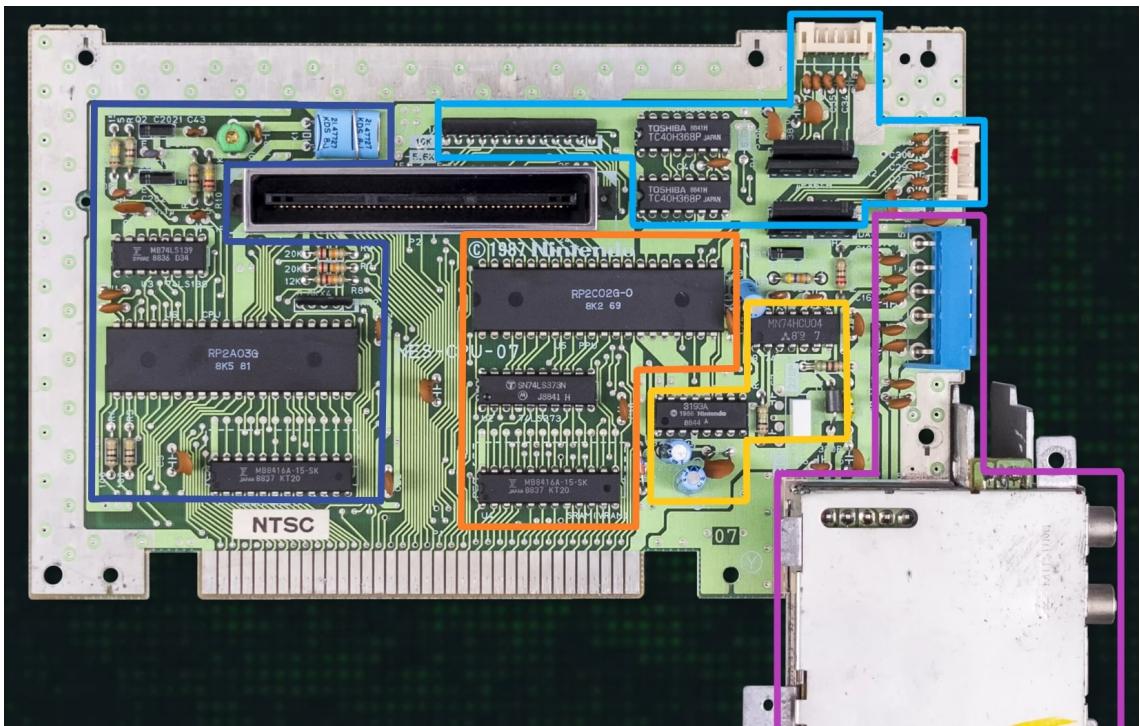
Az otthoni konzol 8 bites processzorral rendelkezett, és a játékok tárolására elsősorban kazettákat használt. Jellegzetes, téglalap alakú kialakítása volt, a játékkazetták behelyezésére szolgáló elülső betöltő mechanizmussal. A konzolhoz egy pár kontroller is tartozott, és bevezette a ma már ikonikus 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 [13]. Az itt olvasható tartalmakat az évek során nagy részt az eredeti hardver viaszfejtésével (reverse engineering) tárták 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 kutatáshoz sok segítséget nyújtott Nathon Altice által írt könyv [1]. Ez rengeteg fundamentális információt nyújt a NES hardverén belül található chipekről. A NES hardverének áttekintéséhez és a chipek általános megismeréséhez (PPU, CPU) nagy segítséget nyújtott még a NESHacker youtube csatorna [17]. Amely páratlanul pontos információkkal és képi adatokkal szolgált az eszköz működéséről.

### 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 [16]

- Sötétkék, processzor:* A NES processzora és ennek kiegészítő áramkörei, komponensei. Ide tartozik természetesen az RP2A03G chip, ez a fő vezérlő egyégy 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) ként 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.

- *Naranccsárga, 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 szűréséért 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 [14]

A játék kazetták fő alkotó elemei:

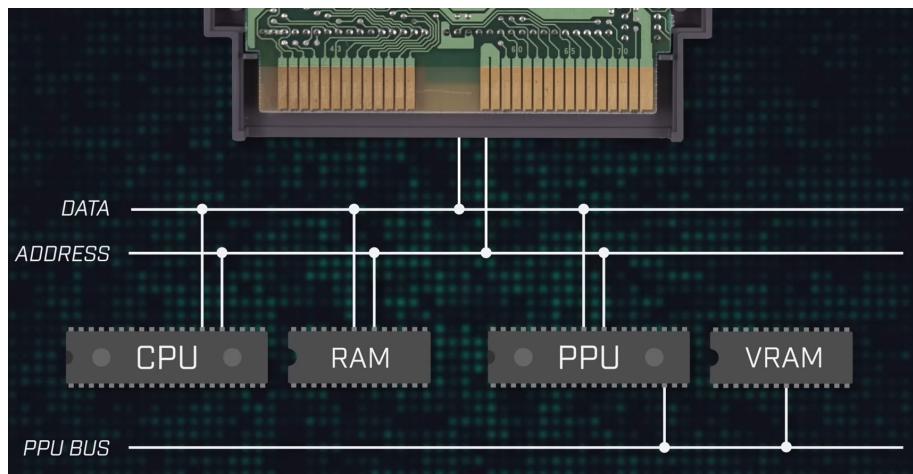
- *Program ROM, és Mapper-ek:* A játék szoftvert tartalmazó, kezdetben 16 kilobájt - 32 kbyte méretű ROM. Viszont a 32 kilobájtnál nagyobb játékok esetén a ROM

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 licencelte 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) [16]

## 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 (coprocessor), hasonlóan a napjainkban elterjedt videókártya-processzor pároshoz.

A CPU-tól eltérően a PPU egy előre meghatározott grafikus műveleti parancs sorozatot hajt végre ciklikusan, nem lehet közvetlenül programozni. Saját memóriával 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 rászterben vannak felépítve, a rászter 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.

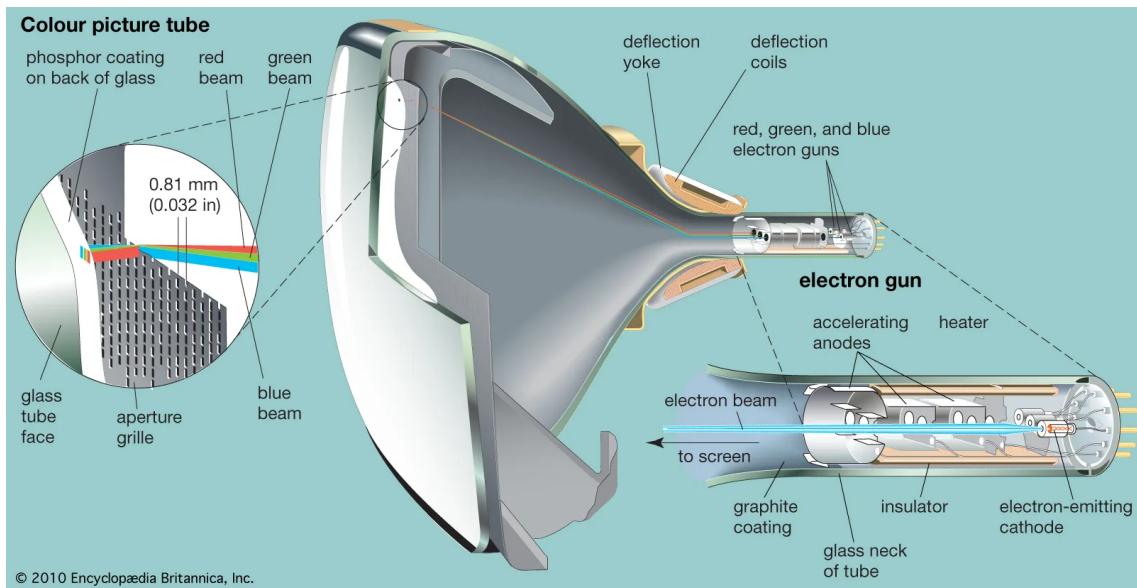
Tehát, mindez összegezve, úgy tekinthetjük a PPU-ra, mintha ez a négy jól elkülöníthető memória terület irányítaná ezt a segéd processzort. A Pattern táblák határozzák meg a nyers képadatokat. A névtáblák határozzák meg a háttér generálását. A színpaletták határozzák meg a használandó színeket és az 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. Ezekről 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 elektronágyú szabályozta a képernyő pixeljeinek monokróm fényerejét, a színes esetben három különálló elektronágyú 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 végigfusson 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 szinkronizációt (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 [3]

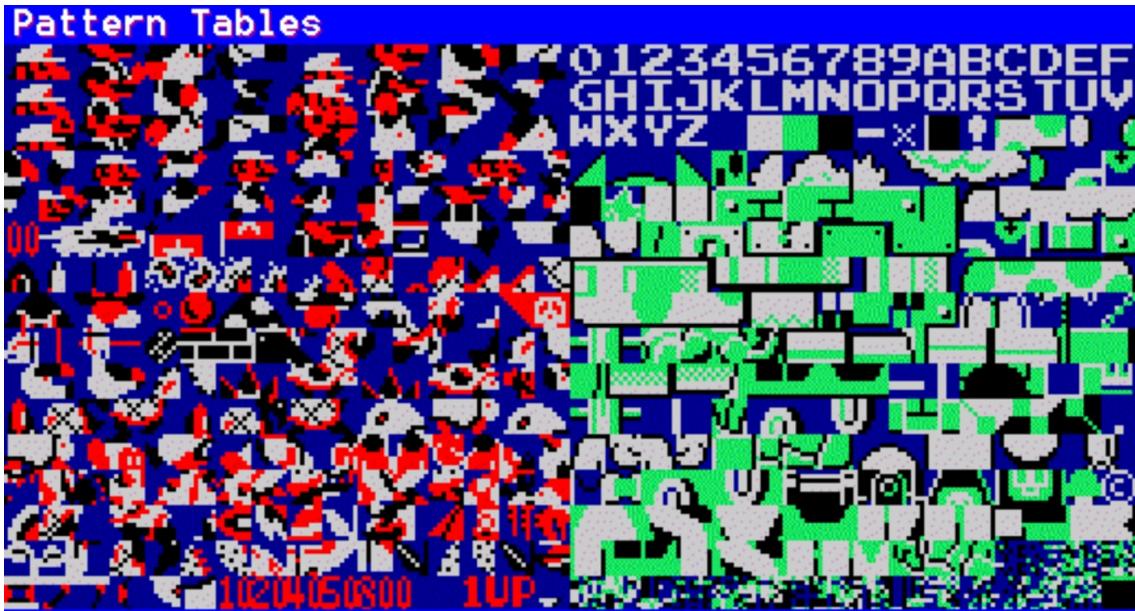
Miközben a elektronágú mozog, a TV egy belső jel segítségével tudja szabályozni az elektronok kibocsátásának mértékét. Ez a jel megváltoztatja a szín fényerejét egy adott pozícióban (pixel-en), a pisztoly gyors mozgásának következtében, az eredmény egy folytonos animált kép képernyőn. Ezt a jelet kompozit 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 [2]

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 [15]

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 bájtos 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-ai 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

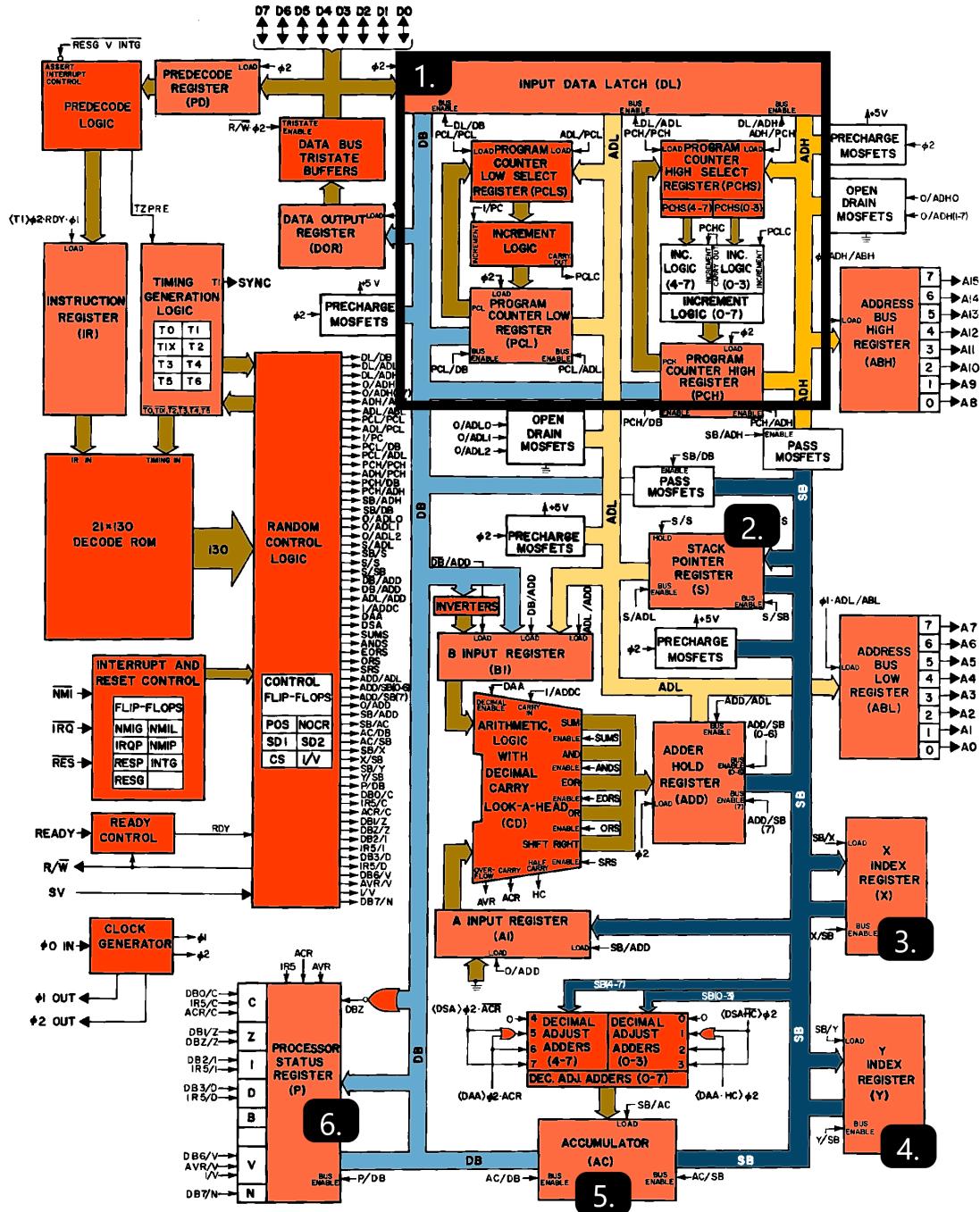
A NES videojáték konzol egy komplex több feladatot ellátó fő végrehajtó egységgel rendelkezik, melynek neve 2A03 (RP2A03[G] NTSC konzolokban). A chip három fő hardveres elemből áll. Először is a kor legjobban elterjedt CPU-jának a MOS 6502-esnek egy módosított változatát tartalmazza, emellett helyet 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  $\phi_0$ , 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  $\phi_1$  és  $\phi_2$ . 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 módját.



3.8. ábra. A MOS 6502 egyedi nyolcbites architektúrája [23]

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) értékkal 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) [4]

Az eredeti MOS 6502 tizenhárom címzési móddal rendelkezett, többek között akkumulátoros, beleértett (implied), közvetlen (immediate), relatív, abszolút X és Y szerinti, ezek között voltak ritkábban és gyakran használt címzési módok is. Nem minden címzési módban van lehetőségünk az összes utasítás használatára.

A processzorunk utasítás készlete 56 alapértelmezett utasításból állt. Az utasítások műveleti kódjai egy bájtosak és a különböző címzési módokkal 151-nyi kód helyet foglalnak el a rendelkezésre álló 256-ból. A maradék fennálló utasítás helyet általában a processzor gyártók foglalt (reserved) utasítás területékként tartják fent. Ezek használata során előre nem definiált jelenségek történnek a 6502-ben. Ez azért fontos a NES szempontjából, mert a játék fejlesztők az 2A03-ra történő fejlesztés során többször is használtak, ilyen úgy nevezett illegális utasításokat a program ROM helytakarékkossága szempontjából. Ilyen például a Beauty and the Beast (E) (1994) amely a \$80 műveleti kódot használja (ez egy 2-bájtos NOP utasításnak felel meg) vagy a F-117A Stealth Fighter amely a \$89-ot (hasonlóan egy 2-bájtos NOP). Ahhoz, hogy teljes hardveres emulálást készítsünk ezeket az illegális utasításokat is le kell implementálnunk. Az 3.10 ábrán a 2A03-ban található 6502 teljes utasítás készlete látható. A táblázat horizontális tengelye a 8 bites műveleti kódok ofszetje, a vertikális pedig a kezdő címe (így a oszlop és a sor érték összegéből megkaphatjuk az opkódot). A piros utasítások a program végrehajtást befolyásolják, a zöldök a különböző ALU utasítások, a kékek pedig az úgynevezett olvasási és írási utasítások (read-modify-write, RMW)). Az összes szürke utasítás pedig illegális utasítás, ezek többsége a kék és zöld utasítások valamilyen kombinációja. A táblázatban még az is látható, hogy az adat utasítás milyen címzési módban használható, illetve milyen adatokat várnak el bemenetként.

+00	+04	+08	+0C	+10	+14	+18	+1C	+01	+05	+09	+0D	+11	+15	+19	+1D	+02	+06	+0A	+0E	+12	+16	+1A	+1E	+03	+07	+0B	+0F	+13	+17	+1B	+1F
00	BRK	NOP	PHP	NOP	BPL	NOP	CLC	NOP	ORA	ORA	ORA	ORA	ORA	ORA	ORA	STP	ASL	ASL	ASL	STP	ASL	NOP	ASL	SLO	SLO	ANC	SLO	SLO	SLO	SLO	
d	a	*	d	d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d	a		d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x				
20	JSR	BIT	PLP	BIT	BMI	NOP	SEC	NOP	AND	AND	AND	AND	AND	AND	AND	STP	ROL	ROL	ROL	STP	ROL	NOP	ROL	RLA	RLA	ANC	RLA	RLA	RLA	RLA	
a	d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d	a		d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x			
40	RTI	NOP	PHA	JMP	BVC	NOP	CLI	NOP	EOR	EOR	EOR	EOR	EOR	EOR	STP	LSR	LSR	LSR	STP	LSR	NOP	SRE	SRE	ALR	SRE	SRE	SRE	SRE			
d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d	a		d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x				
60	RTS	NOP	PLA	JMP	BVS	NOP	SEI	NOP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	STP	ROR	ROR	ROR	STP	ROR	NOP	ROR	RRA	RRA	ARR	RRA	RRA	RRA	RRA	
d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d	a		d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x				
80	NOP	STY	DEY	STY	BCC	STY	TYA	SHY	STA	STA	NOP	STA	STA	STA	STA	NOP	STX	TXA	STX	STP	STX	TXS	SHX	SAX	SAX	XAA	SAX	AHX	SAX	TAS	AHX
#i	d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		#i	d	a	d,y	a,y	(d,x)	d	#i	a	(d),y	d,y	a,y	a,y			
A0	LDY	LDY	TAY	LDY	BCS	LDY	CLV	LDY	LDA	LDA	LDA	LDA	LDA	LDA	LDA	LDX	LDX	TAX	LDX	STP	LDX	TSX	LDX	LAX	LAX	LAX	LAX	LAX	LAS	LAX	
#i	d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		#i	d	a	d,y	a,y	(d,x)	d	#i	a	(d),y	d,y	a,y	a,y			
C0	CPY	CPY	INY	CPY	BNE	NOP	CLD	NOP	CMP	CMP	CMP	CMP	CMP	CMP	CMP	NOP	DEC	DEX	DEC	STP	DEC	NOP	DEC	DCP	DCP	AXS	DCP	DCP	DCP	DCP	
#i	d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		#i	d	a	d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x			
E0	CPX	CPX	INX	CPX	BEQ	NOP	SED	NOP	SBC	SBC	SBC	SBC	SBC	SBC	SBC	NOP	INC	NOP	INC	STP	INC	NOP	INC	ISC	ISC	SBC	ISC	ISC	ISC	ISC	
#i	d	a	*	d	x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		#i	d	a	d,x	a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x			

**3.10. ábra.** A 2A03-ban található 6502 teljes utasítás készlete [12]

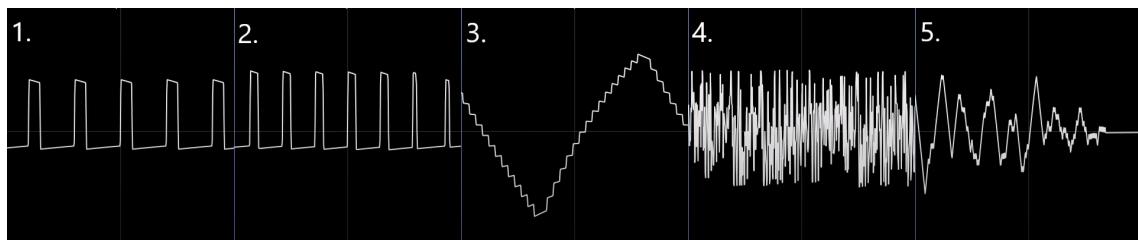
Az 3.1.1 fejezetben már olvasottak alapján a 2A03 chip esetén is foglalkoznunk kell a memory mapped I/O architektúrával, tehát a különböző hardveres komponensek elérésével. A 6502, 16 bites címtartománnal rendelkezik, ez pedig a NES tervezői 3.2 táblázatban látható módon osztották fel. Itt is jól megfigyelhető a PPU-ban már bemutatott tükrözés (mirroring) jelensége.

Cím tartomány	Méret	Eszközök
\$0000 - \$07FF	\$0800	2 KB belső memória
\$0800 - \$0FFF	\$0800	\$0000 - \$07FF tükröképe (Mirrors)
\$1000 - \$17FF	\$0800	\$0000 - \$07FF tükröképe (Mirrors)
\$1800 - \$1FFF	\$0800	\$0000 - \$07FF tükröképe (Mirrors)
\$2000 - \$2007	\$0008	PPU regiszterei
\$2008 - \$3FFF	\$1FF8	\$2000 - \$2007 tükröképei (Mirrors) 8 bájtonként ismételve
\$4000 - \$4017	\$0018	APU és I/O regiszterek
\$4018 - \$401F	\$0008	Olyan APU és I/O funkciók amik alapértelmezetten ki vannak kapcsolva
\$4020 - \$FFFF	\$BFE0	Játék kártya területek, először Mapper regiszterek, ezt követően program ROM és Program RAM

**3.2. táblázat.** A CPU memória címtartománya (16 bit címek) mirroring jelenséggel

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

Az APU is egy PPU-hoz hasonló társprocesszor (coprocessor), tehát a CPU a chip belső regiszterein keresztül képes elérni és változtatni működését (a CPU adatbuszon keresztül 3.2). Az eszköz belsejében öt digitális jel generátor található, amelyek szimultán működnek a chip belsejében. A regiszterek segítségével ezeknek a jelgenerátoroknak a működését tudjuk befolyásolni (frekvencia, amplitúdó (milyen hangos az adott csatorna), illetve az adott csatorna ki és bekapcsolása). A csatornák részletesebb vizsgálatához tekintsük meg a 3.11 ábrát.



**3.11. ábra.** Az audió feldolgozó egység hang sávjai [9]

1. *Első négyzetgel generátor kimenete (pulse<sub>1</sub>):* Az első és második pulzus generátor a NES melodikus dallamaiért volt felelős. A két pulzus jel generátor a korabeli számítógépek jellegzetes Y hangjára hasonlított.
2. *Második négyzetgel generátor kimenete (pulse<sub>2</sub>):* A pulzus generátorok felhasználása megegyezik az első pulzus generátoréval.
3. *Háromszöggel generátor:* A legtöbb esetben a basszus hangok képzésére használták. Itt érdekes még megfigyelni, hogy a háromszög generátor nagy lépésekben dolgozik a 3.11 ábrán is jól láthatók ezek (összesen 32 lépéssel rendelkezik, amely minden egyikhez 4 bites értéket rendel). Illetve egy szinusz jelet próbál generálni a nagyobb lépésekkel (ebből csak fűrész jel lesz). Ezzel egy enyhe, tompa hangokat tartalmazó basszust nyújt a NES játékoknak.
4. *Zaj generátor:* Ez a csatorna egy fehér zaj generátor, ez a korabeli TV-k jel nélküli (no-signal) hangjához hasonlított leginkább. A hang mérnökök ezt a zajt ütőszökhöz és dobokhoz használták legtöbbször.
5. *Delta modulációs csatorna:* Ez a hangcsatorna egy delta-kódolt egy bites kimenetet működtet vagy a hardverben található hét bites számlálót töltetheti. A kimeneten képes előre beállított DPCM minták lejátszására (ez egy veszteség mentes kódolási típus). Ennek segítségével letudtak játszani beszédhangokat, dallamokat, ütőszöket. Viszont ügyelni kellett rá, hogy csak rövid idő tartalmú minták lejátszása volt lehetséges a hardverben található puffer mérete miatt. Használata befolyásolta az OAM DMA működését.

Az öt szimultán működő jel generátorak egy-egy digitál-analóg konverter változtatja analóggá a kimenetét. A konverterek egymástól nem lineárisan függő elemek, amik befolyásolhatják az egyes csatornák amplitúdóját. Ezeket a jeleket az APU analóg keverője alakítja egy analóg kimenetté a későbbiek során. A fentebbekben leírt nem linearitást és az analóg csatornák keverését a 3.1, 3.2 és 3.3 egyenletekkel tudjuk legjobban közelíteni.

$$Output = Pulse_{out} + Tnd_{out} \quad (3.1)$$

$$Pulse_{out} = \frac{95.88}{\frac{8128}{pulse_1+pulse_2} + 100} \quad (3.2)$$

$$Tnd_{out} = \frac{159.79}{\frac{1}{\frac{triangle}{8227} + \frac{noise}{12241} + \frac{dmc}{22638}} + 100} \quad (3.3)$$

### 3.3.3. Direct Memory Access - OAM DMA

A 2A02-on belül egy korai DMA megvalósítás is helyet kapott. Ennek szerepe az adatok fix RAM területre való másolása, gyorsabban mintha a 6502-nek kellene ezzel foglalkoznia. A CPU-nak négy órajel ciklusra lenne szüksége egy adat kiolvasására és másolására, a DMA ezt fele annyi idő alatt (két órajel ciklus) eltudja végezni, viszont csak előre fixált területről tudunk olvasni és írni. Ez a modern DMA megvalósításokhoz képest egy egyszerűbb és primitívebb eszköz, mivel amikor ezen keresztül másolunk a DMA elveszi a CPU adatbuszát és ready jelét így a CPU egy blokkolt állapotba (stand by) a másolás idejére.

Napjainkban a DMA-k már nem blokkolják a mikrokontrollerek/processzorok működését, így nem veszítve CPU számítási kapacitást.

A NES esetén a DMA szerepe az Objektum Attribútum Memória gyors feltöltése a mozgó csempe adatokkal. minden esetben ha a DMA-t aktiváljuk (belő regiszterén keresztül) a teljes OAM RAM feltöltésre kerül. Egy írási/olvasási ciklus során, a CPU által beállított WRAM címről (általában \$0200), egy bájtnyi adatot olvas az eszköz. Majd az olvasott adatot a CPU címtartományában található \$2004-es PPU OAMDATA regiszterbe írja. Ezt a ciklust ismétli (növelve a WRAM címszámláló értékét) a DMA, mindaddig amíg a teljes memória végére nem érünk.

Ez a hardveres komponens szoros kapcsolatban áll az APU DMC csatornájával. Mivel ez a csatorna képes szüneteltetni DMC elérés során a DMA másolását négy órajelciklus idejéig.

### 3.4. NES kontroller

A Nintendo Entertainment System videó játék konzolhoz különlegesebbnél különlegesebb bemeneti eszközök születtek az évek során. Ebben a dokumentumban viszont a modern értelemben vett a kontrollerek atyját szeretném jobban bemutatni. Ez pedig a NES gamepad vagy NES standard kontroller néven vált ismerté a 3.1 ábrán is ez látható.

Az eszköz működése rendkívül egyszerű, hardvere csupán egy paralel-soros léptető (shift) regiszterből áll. Összesen nyolc gomb található a kontrolleren, ezt nyolc gombot mintavételezi a CPU (a címtartományában található regiszteren keresztül 3.2.). A mintavételezés pillanatában a lenyomott gombok nullás, a többi gomb pedig egyes értékkel szerepelnek a léptető (shift) regiszter bemenetén. A mintavételezést követően pedig a 2A03-tól kapott órajel hatására, bitenként kiolvasásra kerül a mintavételezett érték. A két kontrollerből beérkező értékek még egy hardveres negáláson esnek át az alaplapon 3.1, a könnyebb feldolgozhatóság érdekében.

## 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).

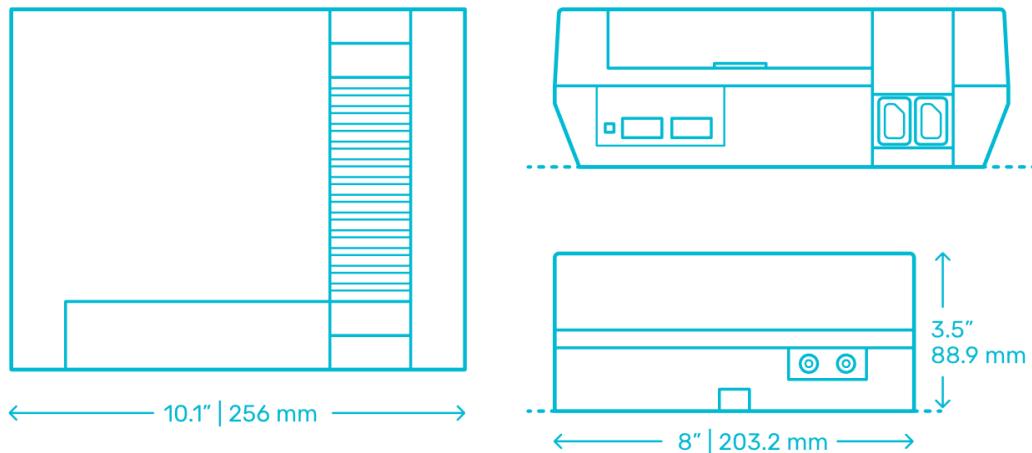
Ennek következtében egy új megoldást kellett kitalálni arra, hogy az emulált NES hardvert megfelelő módon ellássam játékokkal. Az egyik kézen fekvő megoldás, hogy a hordozó kártyára el lehet helyezni egy statikus RAM modult amelynek elég nagynak kell lennie ahhoz, hogy a játék kártyák karakter és program 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.



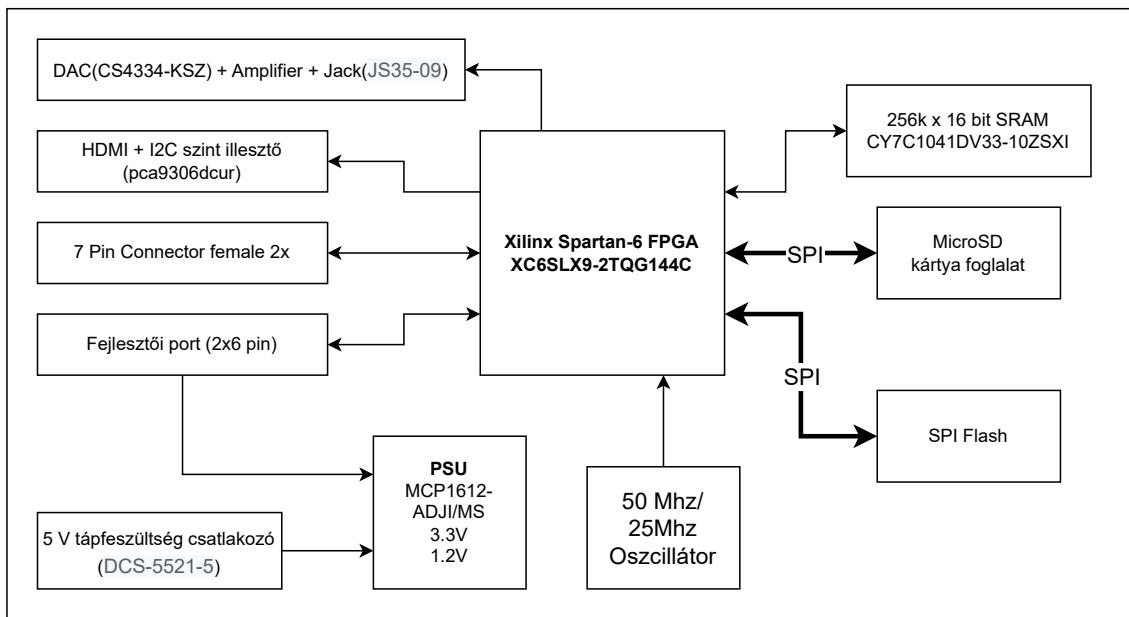
4.1. ábra. A NES játék konzol dimenziói [6]

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 érdekes 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 részletesebben [21]). Ezt a fejlesztő kártyát vettetem 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 [21]. A NES kártya 5 V-os tágfeszültségről 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 táp 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 a [19] adatlapon olvashatjuk, kiegészítő áramkörét pedig az F.3.4 függelékben láthatjuk.

## 5.4. Digital Analog Converter és erősítő

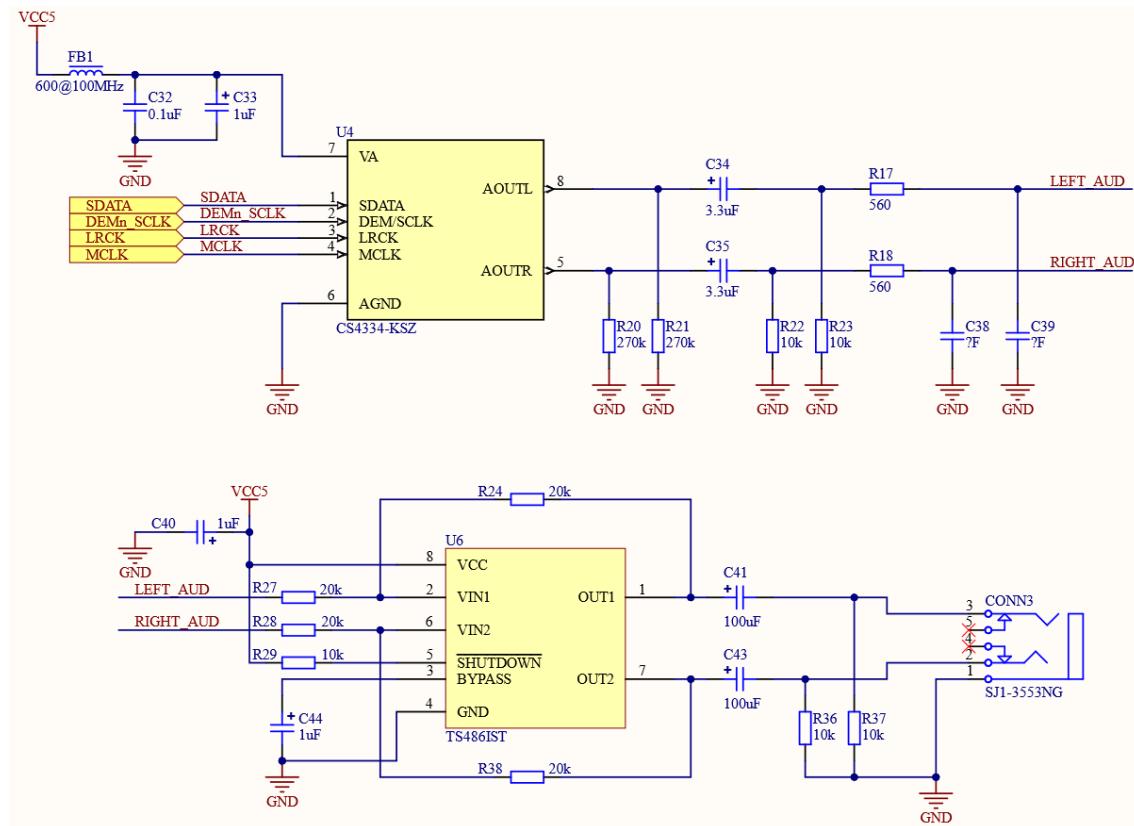
A NES kártya egyetlen analóg alkatrészekre támaszkodó része a DAC-ot követő erősítő áramkör, ennek részletes megértéséhez tekintsük meg az 5.2 ábrát, amely az F.3.3 függelékből lett kiemelve.

A DAC komponenst egy 100MHz-en 600  $\Omega$ -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  $\Omega$  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:

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

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

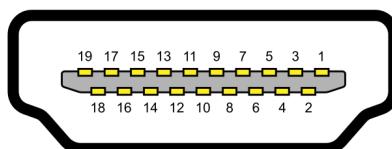
Az áramkörben jelenleg nem állítottam be erősítést az RFEED és RIN ellenállások értékét 20 kΩ-nak határoztam meg. Ez természetesen a hardveres tesztelés során cserélhető és állítható.

## 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 [10]

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

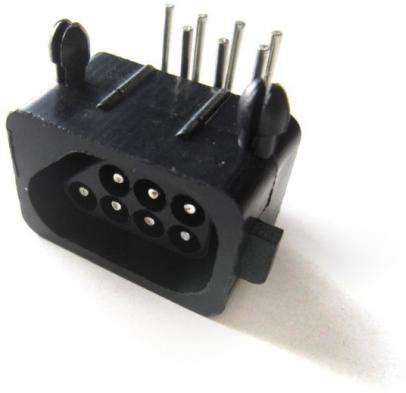
Mivel az FPGA nem minden I/O láb pára képes differenciális jelek küldésére, ezért figyelni kell, hogy a csatlakozót az FPGA melyik oldalához közel helyezem el. A HDMI szabvány az adat vonalak között 100 Ω-os impedancia különbséget ír elő (15% os toleranciával), ezért érdemes ezeket az vezetékeket minél 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 [20]

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 mellet 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 [21] 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 [21]. 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 [21]-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 [21], 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.

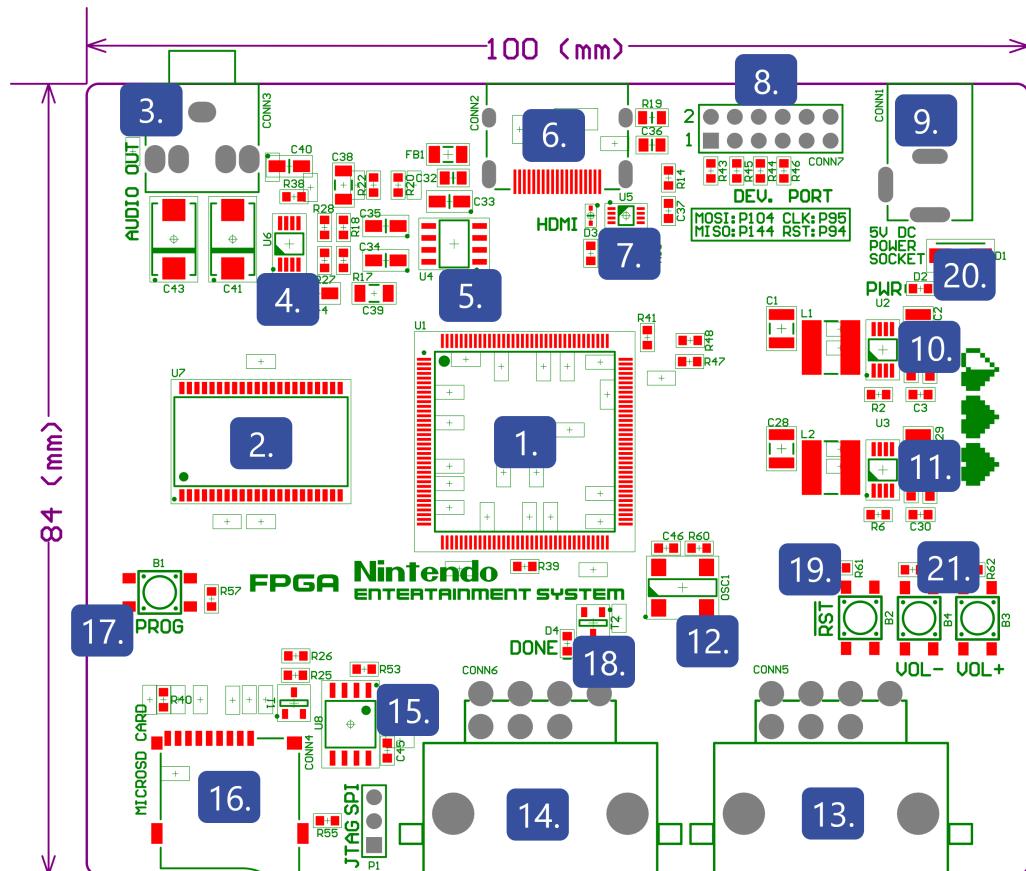
A prototípus kártyákat a jlpcb nevű kínai nyomtatott áramkör gyártó fogja gyártani. Ahhoz, hogy az Altium tervező program képes legyen bonyolultabb számítások (differential pair routing) és szimulációk készítésére, a réteg felépítést meg kell adnunk a PCB-nk számára. Ez a kínai gyártó által szolgáltatott információk által [7] 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.

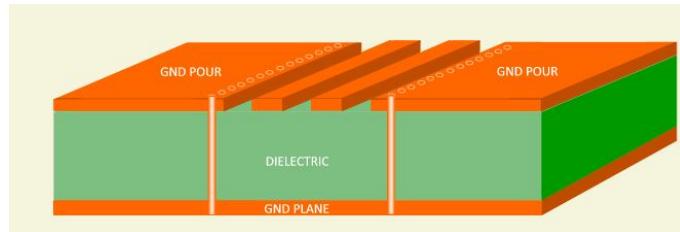
Ezt követően a PCB alkatrészek bekötésével foglalkoztam, itt a alsó és felső réteget is jel/föld (signal/GND) rétegként alakítottam ki. Ez azt jelenti, hogy jel vezetékeket és táp vonalak bekötését követően az egész nyák felszínén föld kitöltést hoztam létre, így növelte a jelek integritását. A alsó 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  $100 \Omega$ -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:

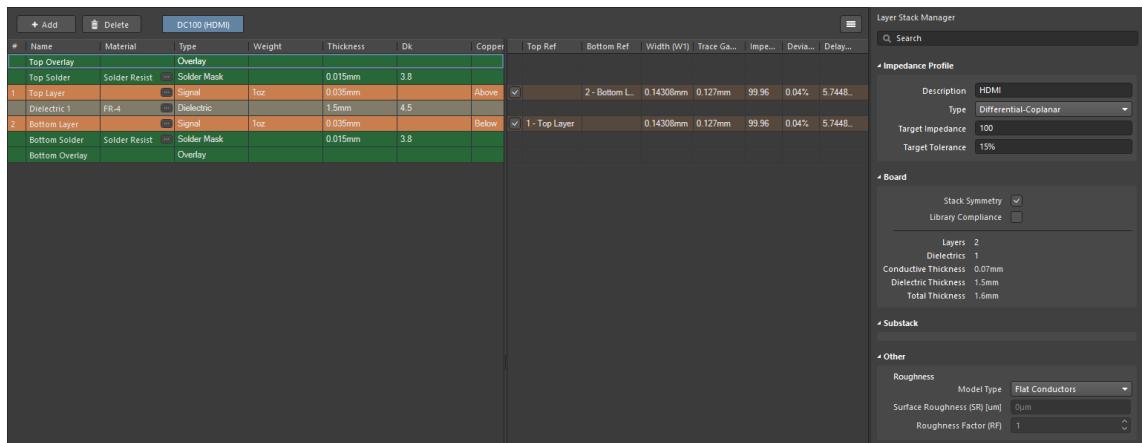
$$S(via) = \frac{\lambda}{20} = \frac{c}{20 * f * \sqrt{\epsilon_r}} \quad (5.4)$$

Az 5.4 képletben található  $\lambda$  a differenciális jelünk hullámhossza (a részletesebb képletben pedig:  $c$  a fény terjedési sebessége,  $f$  a jelünk frekvenciája,  $\epsilon_r$  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).



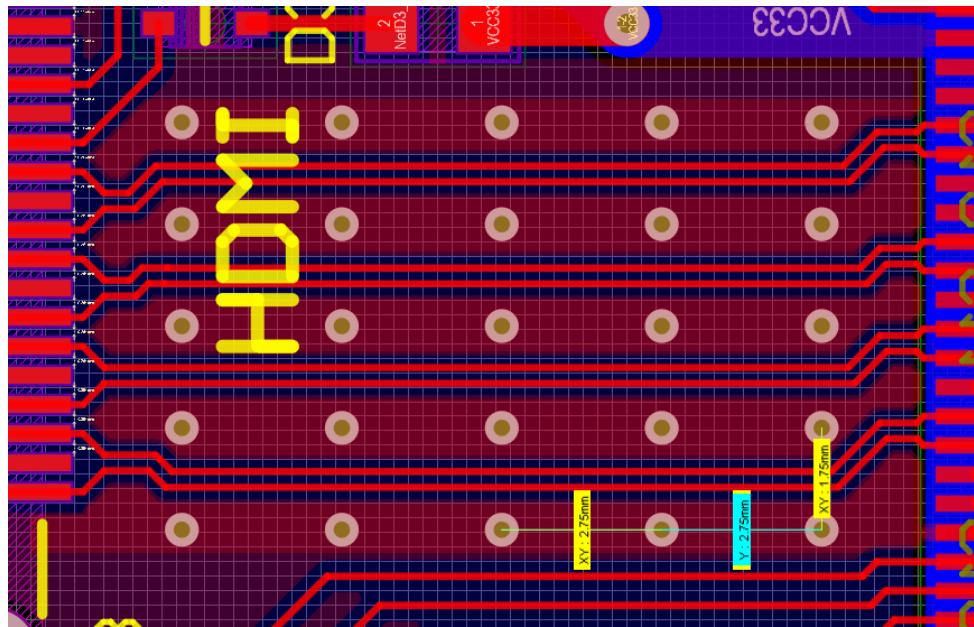
**5.7. ábra.** Dual Strip Coplanar Waweguide Grounded felépítése [8]

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

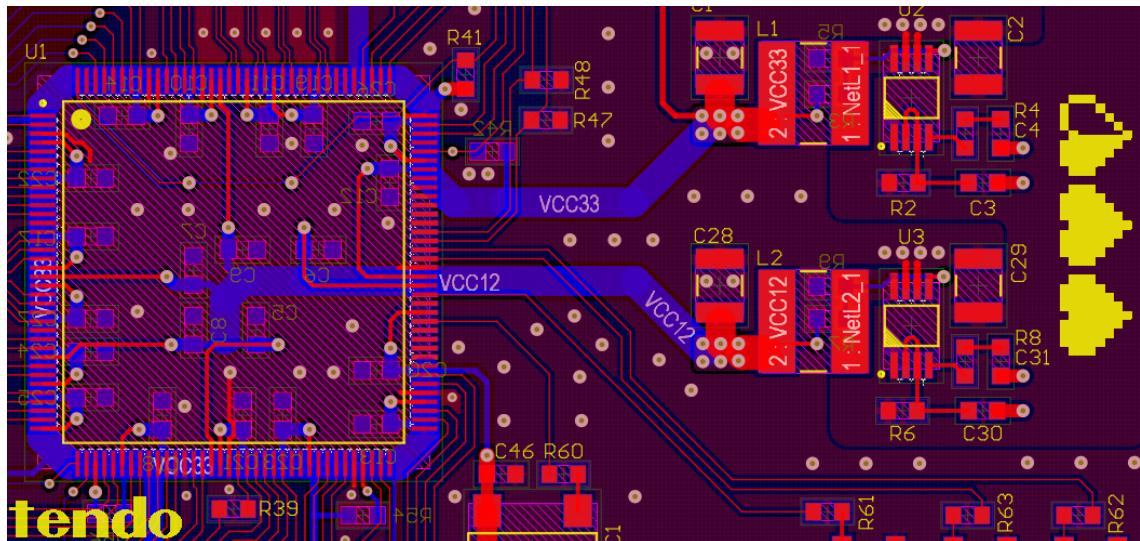
A FPGA NES kártya HDMI jeleinek bekötésére 2,75 mm-es via fence méretet választottam (körülbelül a fentebb kiszámolt érték 20-ad része) ez részletesen az 5.9 ábrán látható, amely az F.2 függelékből lett kiemelve.



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 hidegitő (bulk) kapacitás az FPGA alatt kaphat helyet, ezzel szabadon tartva a top réteget az FPGA I/O bankjainak és JTAG interfészének bekötésére.

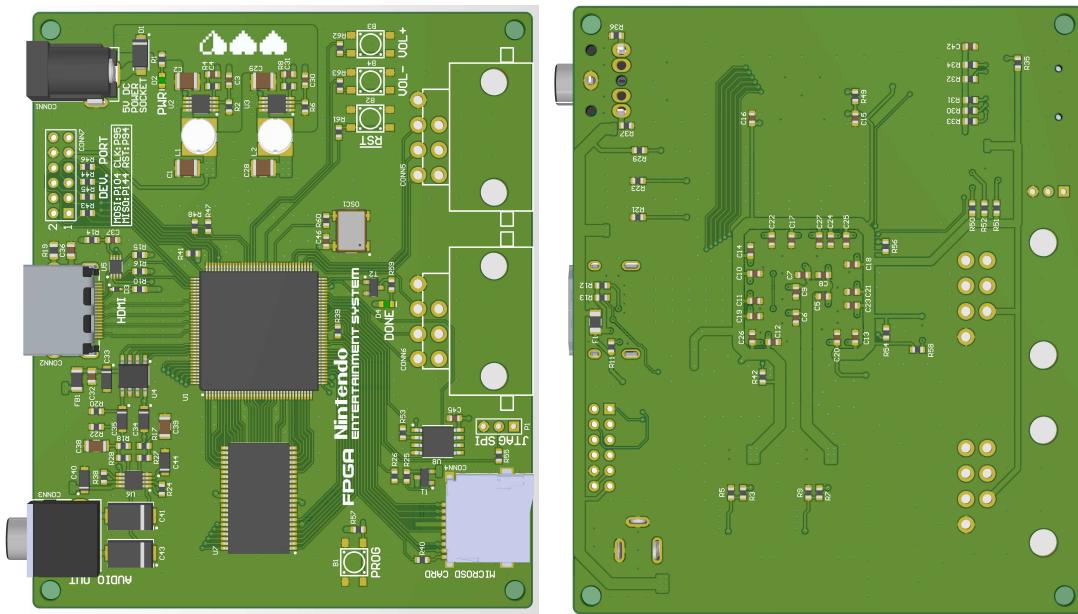


5.10. ábra. FPGA tápellátásának kialakítása

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 tervé

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:



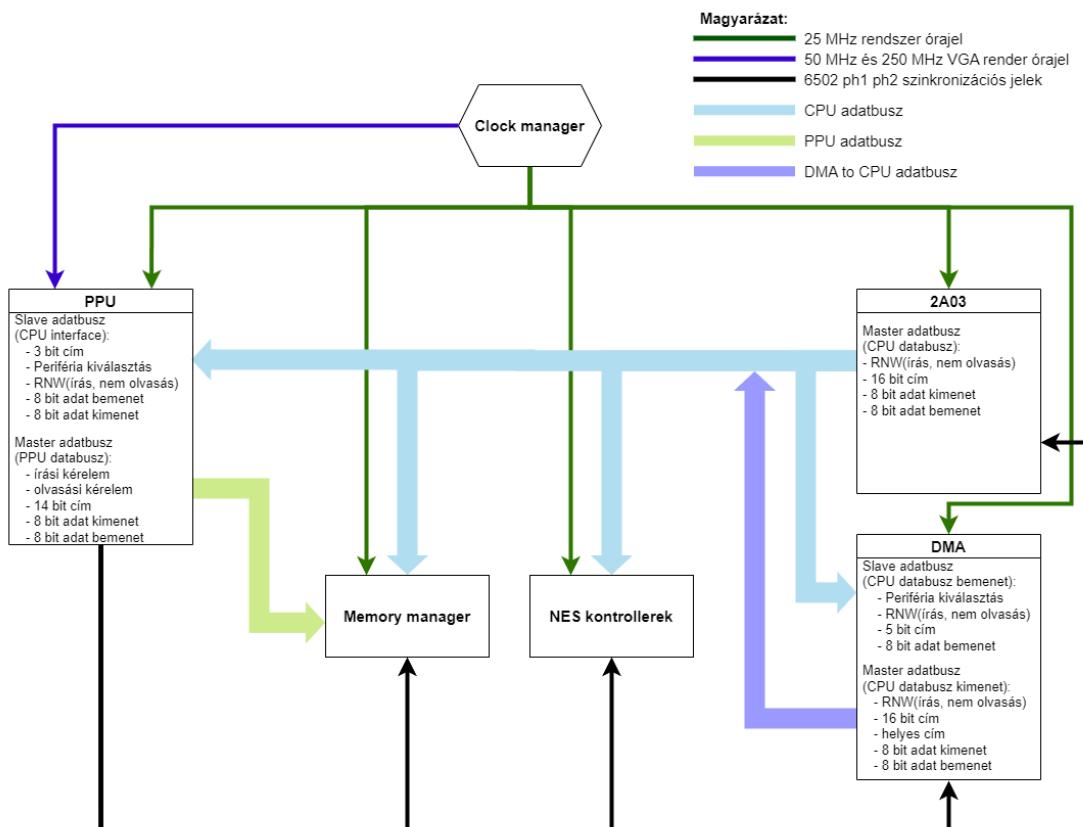
### 5.11. ábra. 3D PCB rajzolat

## 6. fejezet

# FPGA tervezés

A Nintendo Entertainment System hardveres komponenseinek újratervezése során az ISE Design Suit szoftvert és a Verilog hardver leíró nyelvnek választottam. A tervezés fő szempontja a beláthatóság, céludatosság és tesztelhetőség volt, ez elengedhetetlen egy nagyobb projekt fejlesztése során. A meghaladóan fejlesztett chipeknek az összehangolt és hibátlan (az eredeti NES hardverrel megegyező) működését követően, léptem tovább a következő fejlesztési egységre.

A következőkben a diplomamunka során tervezett és tesztelt hardveres komponensek működését és felépítést fogom bemutatni, az úgynevezett "top-down" elv alapján. Ennek jelentése, hogy először a teljes rendszer kapcsolatát mutatom be, majd ezt követően térek ki a különböző elemek részletes működésére.



**6.1. ábra.** Megvalósult FPGA projekt (nes\_top) működési diagramja

Az FPGA tervezés első fázisában a különböző funkciókat ellátó hardveres elemeket (a NES különböző chipjeinek) modulokra osztottam. Ezeket a modulokat és a köztük lévő logikai kapcsolatot láthatjuk a 6.1 ábrán.

A modulok között két fő logikai adatkapcsolatot látható, az első az kommunikációs buszok (vastag nyilak), ezek többféle vezetéket foglalnak magukba (adat, cím, kiválasztó jelek), ezek pontos tartalmát a nyilak kezdőpontjánál láthatjuk. A második pedig a rendszer órajelek. Ide tartoznak a rendszer órajelből elő állított,  $\phi_1$  és  $\phi_2$  szinkronizációs jelek is.

## 6.1. Adatbuszok implementálása

A NES eredeti belső logikai kapcsolatait alapul véve, két fő adatbuszt különböztetünk meg a CPU és a PPU adatbuszt.

A PPU adatbusz struktúrájában egyszerűbb csupán a memória menedzser modullal kommunikál. A modulban található név tábla (NT) memóriát írja és olvassa, illetve a karakter ROM tartalmát olvassa.

A CPU adatbusz működése a DMA miatt egy fokkal bonyolultabb. Alap esetben a CPU az adatbuszán keresztül éri el a címtartományában található különböző perifériák regisztereit/memóriáit. A slave interfészek implementálása során ügyeltem arra, hogy a különböző hardveres komponensek minél egyszerűbben fogadják a CPU adatbuszát (ezzel egyszerűsítve a szintetizálálandó hardveren), ez a 6.2 hardverleírásban is látható. Viszont a DMA működése során, (helyes CPU cím kiadásával) átveheti a processzor adatbuszát, ilyenkor ő végez memória és regiszter hozzáféréseket a perifériákon. A helyes cím kiadása mellett a DMA elveszi a processzor Ready jelét is, ezzel blokkolva CPU működését és megakadályozva a busz több periféria általi meghajtását. A különböző modulok kimeneti jelei úgy lettek megtervezve, hogy csak megfelelő időzítés mellett adjanak ki adatokat (különben nullát), így a 6.1 hardverleírásban is látható, hogy a különböző kimenetek egyszerű logikai vagy kapcsolatával tudjuk az adatbuszunkat összegezni.

```

1 // CPU data in
2 wire [7:0] cpu_din = (memory_manager_cpu_din | ppu_to_cpu_din | controller_cpu_din);
3
4 // DMA and CPU outputs
5 assign cpu_addr = (dma_cpu_valid) ? (dma_cpu_addr) : (nes6502_addr);
6 assign cpu_dout = (dma_cpu_dout | nes6502_dout);
7 assign rnw = ~(~dma_cpu_rnw | ~nes6502_rnw);
```

### 6.1. hardverleírás részlet. CPU adatbuszt befolyásoló DMA jelek

A különböző interfészek egyszerűsítése során a 3.2 táblázatban és a 3.2.3 fejezetben bemutatott tükrözés "mirroring" jelenségét használtam ki. Ennek alapja hogy csupán a címregiszter adott bitjeit figyeljük, például PPU regiszterek esetén az alsó három bitet. Ezzel párhuzamosan pedig megpróbálunk egy megfelelő kiválasztó jelet generálni a chiphez, PPU esetén a felső három bit fixálja a \$2000 - \$3FFF-ig tartó cím tartományt. Ilyen egyszerűsítést alkalmaztam a kontrollerek és DMA interfész esetén is (6.2).

```

1 // PPU Slave interface bemenet
2 .slv_mem_addr(cpu_addr[2:0]), // register interface reg select (#2000-#2007)
3 .slv_mem_cs((cpu_addr[15:13] == 3'b001)), // register interface enable (#2000 - #3FFF just
4   when it is active)
5
6 // Controller inputs
7 .cpu_data_in(cpu_dout[0]), //just one bit the $4016 write strobe
8
9 // DMA Slave databus
10 .slv_mem_select((ag6502_addr[15:14] == 2'b01)),
    .slv_mem_addr(ag6502_addr[4:0]),
```

### 6.2. hardverleírás részlet. A CPU adatbusz interfészek egyszerűsítése

## 6.2. Működési órajel választása

Egyik első feladat FPGA tervezés során a hardver átgondolását követően, a rendszerórajel kiválasztása. Erről a fő órajelről fogjuk működtetni a szinkron tárolóinkat, memória elérésünket és az összes perifériánk időzítésének is, ez lesz az alapja. A NES megvalósítása során használtam két specifikusabb órajelet is a rendszer órajel mellett, ezeket a TMDS jelek előállítása során használtam fel. Ezek a kiemelt órajelek a többi FPGA-n található jelről elkülönített helyen, a globális órajel osztó hálózatban folynak, ez biztosítja rendszerünk számára, hogy minden SLICE-hoz ugyanakkor érjen el az órajel. Ebbe az órajelosztó hálózatba csatlakozik a külső oszcillátorunk is, illetve a PLL és DCM FPGA-n belüli hardveres elemek, amelyek lehetővé teszik az órajelek frekvenciájának változtatását.

A NES kártyán egy 50 MHz-es külső órajel forrást helyeztem el. A 4.1 fejezetben leírtaknak alapján a rendszer órajelemet a megjelenítésnek megfelelően kell megválasztani, ezt figyelembe véve a működési és egyben a pixel órajelemet is 25 MHz-nek választom. A beérkező 50 MHz-ból a PLL helyes beállítása során elő állítok 25 MHz-et és 250 MHz-et. A VGA kép generálásához szükségem lesz mind a három órajelre, a 25 MHz a pixel órajelem (erre a jelre kell előállítanom a 24 bites RGB értékeimet), az 50 MHz az 5 bites párhuzamos-soros átalakítókhöz kellene (strobe jelek), végül pedig a 250 MHz-el fogom kiadni a biteket a HDMI differenciális adatvezetékei számára.

```

1 reg clkgen_cnt_en_clr;
2 wire clkgen_cnt_en_set = (x_rendercntr == (ODDFRAME_END_OF_BG_RENDERING_LINE + 4));
3 always @(*)
4 begin
5     if (background_enabled && oddframe && (y_renderingcntr == PRERENDERING_ROW))
6         clkgen_cnt_en_clr <= (x_rendercntr == ODDFRAME_END_OF_BG_RENDERING_LINE);
7     else
8         clkgen_cnt_en_clr <= 1'b0;
9 end
10
11 // clock generation enable
12 reg clkgen_cnt_en;
13 always @ (posedge clk)
14 begin
15     if (rst || (x_rendercntr == END_OF_BG_RENDERING_LINE) || clkgen_cnt_en_clr)
16         clkgen_cnt_en <= 1'b0;
17     else
18         if ((x_rendercntr == FIRST_SCANLINE_PIXEL) || clkgen_cnt_en_set)
19             clkgen_cnt_en <= 1'b1;
20 end
21
22 //clock generation timer
23 reg [3:0] clkgen_cnt;
24 always @ (posedge clk)
25 begin
26     if (rst)
27         clkgen_cnt <= 4'd0;
28     else
29         if (clkgen_cnt_en)
30             if (clkgen_cnt == 4'd11)
31                 clkgen_cnt <= 4'd0;
32             else
33                 clkgen_cnt <= clkgen_cnt + 4'd1;
34 end
35
36 always @ (posedge clk)
37 begin
38     ph1_rising <= clkgen_cnt_en & (clkgen_cnt == 4'd0);
39     ph1_falling <= clkgen_cnt_en & (clkgen_cnt == 4'd5);
40     ph2_rising <= clkgen_cnt_en & (clkgen_cnt == 4'd6);
41     ph2_falling <= clkgen_cnt_en & (clkgen_cnt == 4'd11);
42 end

```

**6.3. hardverleírás részlet.** 6502  $\phi_1$  és  $\phi_2$  vezérlő jelek előállítása

Ezekből az órajelekből kell előállítani a NES hardver számára a megfelelő kiválasztó jeleket. Az NTSC NES-ben található MOS 6502 1.789773 MHz-es bemeneti  $\phi_1$  órajellel rendelkezik. Ebből hozza létre a fázis tolt  $\phi_1$  és  $\phi_2$  külső időzítési órajelet. A  $\phi_2$  órajel lefutó élére történik a periféria adtok olvasása.

A projekt VGA rendrelését úgy implementáltam, hogy amíg a PPU képgenerálást kiegészíti a hardver VGA jellé, addig a PPU és vele együtt az egész NES hardver egy alvó állapotban várakozik. Ezt a  $\phi_1$  és  $\phi_2$  jelek megfelelő időben történő generálásával (késleltetésével) tudtam kialakítani. Ezt láthatjuk a 6.3 hardverleírásban. A Verilog kód első felében azt láthatjuk, hogy a PPU aktív kép generálása során az óra jel generálásáért felelős számláló engedélyezve van, viszont a képalkotás végeztével (VGA blanking terület) a számlálót tiltjuk. Legfelül pedig az látható, hogy az eredeti PPU megkülönböztette minden második (páratlan) kép generálását úgy, hogy specifikus pixeleket kihagyott a generálásból, ezt is figyelembe kell venni a hardver tervezése során (clkgen\_cnt\_en\_clr jel). Ezt követően pedig kissámolható, hogy a 25 MHz-es órajelünkre működtetett számlálót, ha 11-re nullába állítjuk, akkor az első hat óra jel alatt aktív a  $\phi_1$  és a második hat alatt pedig a  $\phi_2$ . Ha ehhez még hozzá számoljuk a számláló megfelelő engedélyezését, 1.8 MHz körüli értéket kapunk. Ezen órajelek éldetektálása adja az általam fejlesztett NES engedélyező jeleit 6.2.



**6.2. ábra.** MOS 6502 órajel generálása (FPGA sim)

### 6.3. Picture Process Unit

A PPU időzítésének és működésének alappillérét két számláló, egy horizontális és vertikális képzi. Ezek végigkövetik a teljes eredeti képgenerálást és ezzel együtt a új HDMI-n keresztül történő képgenerálást is, szinkronban. A 4.1 fejezetben bemutatott két módszer közül, az egyszerűbb megoldást választottam a képgenerálás megalkotására. Tehát minden egyes NES pixelt felnagyítok 2 x 2-es pixelekké, ezzel egy pixelesebb, de élesebb képet kapva végeredményként (amely jobban hasonlít az eredeti konzol képére).

Az NTSC PPU képalkotását az F.4 függelékben láthatjuk. Az eredeti hardver, a 0-239-ig tartó sorokban, az 1. pixeltől kedve a 256. pixelig minden egyes órajelben elő állít egy pixelt (kompozit jelként). Ha ezt a képalkotást szeretnénk szinkronba hozni a 640 x 480 pixeles VGA jel elő állításával, ami 800 pixelt jelent horizontálisan és 525 sort vertikálisan (blanking pixelekkel együtt). Akkor ehhez egy 1600-as horizontális és egy 261-ig tartó vertikális számlálóra van szükség, ezáltal egy PPU sor ideje alatt, két sor VGA képet küldünk ki. Természetesen így egy puffereléssel el kell csúsztatnunk a VGA képalkotást a PPU-hoz képest és az 1600-as PPU sort, minden második rendszer órajelre kell mintavételeznünk. Ennek függvényében az F.4 függelékben látható állapotgépek állapotaiban töltött időt, is meg kell hosszabbítani, még hozzá úgy, hogy a képen látható horizontális pixel órajel, minden pixele az FPGA képalkotásban négy rendszer órajelnek feleljen meg.

#### 6.3.1. CPU által elérhető regiszterek és CPU adatbusz

Először is tekintsük meg a CPU adatbusz címének dekódolását és az ebből elő állított vezérlő jeleket a PPU számára 6.4. Ezek a jelek vezérlik a chip belső hét regiszterének elérését. A regiszterek írását a  $\phi_2\_falling$  jelre szinkronizálom (inaktív RNW jel mellett).

Az olvasás során pedig az adatokat a *phi2\_rising* jel hatására (aktív RNW jel mellett) helyezem el a CPU buszinterfész adatkimenetén. Ezt az adat kimenetet a *phi2\_falling* jelre nullába állítom kell, az adatbusz helyes működése érdekében.

```

1 //register write enable signals
2 //CTRL register write #2000
3 wire control_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b000);
4 //MASK register write #2001
5 wire render_mask_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b001);
6 //OAM read/write address #2003
7 wire oam_addr_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b011);
8 //OAM data read/write #2004
9 wire oam_data_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b100);
10 //fine scroll position (two writes: X scroll, Y scroll) #2005
11 wire scrolling_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b101);
12 //PPU read/write address (two writes: most significant byte, least significant byte) #2006
13 wire vram_addr_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b110);
14 //PPU data read/write #2007
15 wire vram_data_wr = ph2_falling & slv_mem_cs & ~slv_mem_rnw & (slv_mem_addr == 3'b111);
16 //Register read enable signals
17 wire status_rd = slv_mem_cs & slv_mem_rnw & (slv_mem_addr == 3'b010);
18 wire oam_data_rd = slv_mem_cs & slv_mem_rnw & (slv_mem_addr == 3'b100);
19 wire vram_data_rd = slv_mem_cs & slv_mem_rnw & (slv_mem_addr == 3'b111);
```

#### 6.4. hardverleírás részlet. A CPU adatbusz feldolgozása (PPU vezérlő jelek)

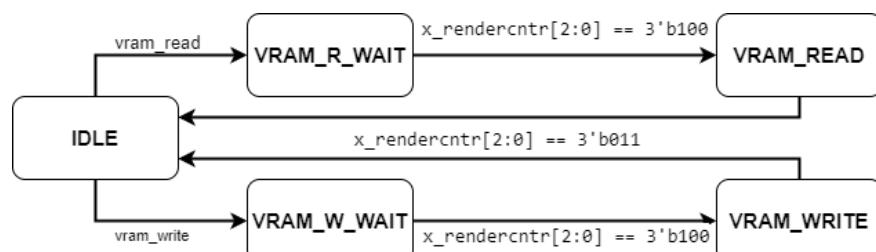
A CPU által írt PPU regiszterek értékét, a 6.4-ben látható vezérlőjelek segítségével mentem regiszterekbe. Majd ezeknek a regisztereknek a megfelelő bitjeit wire-ök útján szétbontom, a könnyebb feldolgozás érdekében. Viszont a PPU rendelkezik 16 bites belső regiszterekkel is, ezeket a regisztereket készeri írással tudja elérni a CPU 8 bites adatbusza. Ilyen regiszter a görgetési pozíció és a VRAM adat és cím regisztere. Az FPGA projektben a 6.5-ben leírt *second\_write* segítségével vagyunk képesek a második írások detektálására. A második 8 bites adat mentésé pedig, ennek a jelnek és a regiszter írását jelző jel hatására, történik.

```

1 reg second_write;
2 always @ (posedge clk)
3 begin
4     if (rst || (status_rd && ph2_falling))
5         second_write <= 1'b0;
6     else
7         if (scrolling_wr || vram_addr_wr)
8             second_write <= ~second_write;
9 end
```

#### 6.5. hardverleírás részlet. Cím és görgetés regiszterek második írásának figyelése

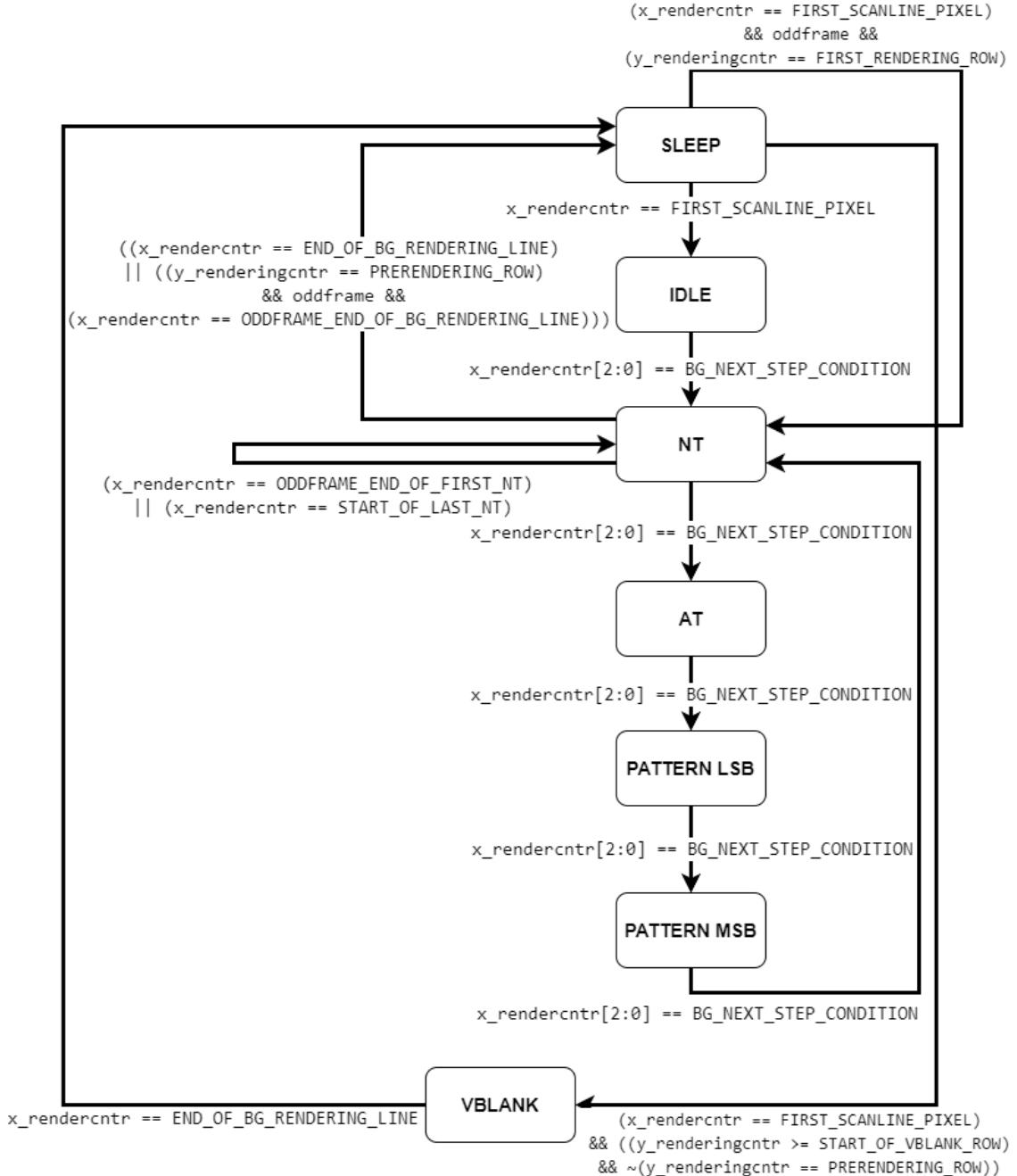
A PPU regiszter elérésének vannak olyan részei, amelyek időzítés kritikusak (ha ezekre nem figyelünk a játékaink kép generálásába különböző hibákat figyelhetünk meg). Ilyen a VRAM írása és olvasása, ezeket a jelek megfelelő módon késleltetnünk kell, hogy szinkronban legyen a háttér generálási állapotgép memória eléréseivel. Erre a legkönnyebb mód egy állapotgép, ezt a 6.3 ábrán láthatjuk.



**6.3. ábra.** CPU-ból ékrekző VRAM írási és olvasási jelek PPU időzítéséhez igazító állapotgép

### 6.3.2. Háttér képalkotást és memória olvasást vezérlő állapot gép

A 3.2 fejezetben olvasottak alapján eredeti PPU kép generálása, két nagyobb egységre bonthatjuk a háttér és a mozgó csempe elemek sprite-ok generálására. Ebben a fejezetben először az FPGA-ban implementált háttér generálási megoldásokat olvashatjuk.



**6.4. ábra.** Háttér képalkotás és memória olvasást vezérlő állapotgép

Az FPGA projekt háttér kép alkotását, a 6.4 ábrán látható állapotgép határozza meg. Az állapot átmenetek képzéséhez a chip horizontális és vertikális számlálóiit használtam, illetve az eredeti háttér alkotási állapotgépet vettem alapul (F.4). A PPU ismertetése során, már olvashattuk, hogy az eredeti állapotgépben található állapotokban négyszer annyi időt kell tölteni a hardvernek, a VGA jel kiadásának szinkronizálása miatt. Az

állapotgépünk hat az eredeti PPU-ban is megtalálható állapotra, és egy SLEEP, blokkoló állapotra bonthatjuk. A SLEEP állapot során a PPU és vele együtt az egész NES hardver egy blokkolt állapotba kerül. Ezalatt az idő alatt a CPU órajel képzését is blokkoljuk, így csak a pixel kétszerezés és a TMDS jelek kiadása zajlik.

A 6.4 ábrán látható hat alap állapot, és működése hasonlít az eredeti PPU állapotokhoz (F.4):

1. *Idle*: minden képgenerálási sor első (nulladik) órajele egy *Idle* állapot, itt a PPU nem végez memória elérést, csak cím regisztereit állít. minden pártatlan teljes kép generálása során, ez az állapot kihagyásra kerül (helyette az utolsó NT olvasás zajlik le).
2. *NT, névtáblák olvasása*: Ebben az állapotban kerülnek kiolvasásra a névtábla adatok. A fentebb látható állapotgépben még azt láthatjuk, hogy az F.4-ben is látható módon, a 261. sor és a látható képtartomány (0-239 sor) végén, a chip még kétszer NT állapotba kerül.
3. *AT, csempe attribútumok olvasása*: Ebben az állapotban a névtábla memóriából olvassuk ki, a csempehez tartozó attribútum adatokat. A későbbiekben ez határozza meg a csempe színeit.
4. *Pattern LSB (least significant byte)*: A névtábla olvasás során megszerzett Pattern csempe cím használatával, egy csempe sor (8 pixel) adatainak, alsó 8 bitjének olvasása történik ebben az állapotban. (a 3.7 képen látható csempe egy sorának alsó bitjei).
5. *Pattern MSB (most significant byte)*: A negyedik állapotban leírt olvasás, befejező lépése történik ebben az állapotban, tehát a felső 8 bit kiolvasása és mentése. Illetve ebben az állapotnak történik még a vetikális és horizontális VRAM cím számlálók növelése, az F.4 függelékben látható helyeken (vörös négyzetek).
6. *VBLANK*: Ez az állapot az eredeti PPU vertikális kioltása, a legtöbb játék ezalatt az idő alatt végezte a PPU memória műveleteket. Itt hozom létre a PPU státusz regiszter, VBLANK (IRQ) jelét. Ez jelez a NES CPU számára (NMI), hogy a VBLANK periódus elkezdődött (szabadon elkezdheti a memória műveleteket a PPU regisztereiken keresztül).

A Pattern olvasási állapotok nem csupán a háttér olvasását hanem a Sprite adatok kiolvasását is végzik. Az F.4 függelékben láthatjuk is, hogy a 257. órajeltől a 320. órajelig történik a mozgó csempe adatok olvasása (következő aktív sorra), a sor többi részen pedig a háttér adatok olvasása történik.

A PPU memória területeinek olvasását (NT, AT, PATTERN) úgy időzítem, hogy az adott állapot utolsó órajelére, mentésre kerüljön az éppen aktuális adat (felhasználható legyen). Ennek pontos időzítését a 6.6 hardverleírásban olvashatjuk.

```

1 assign nametable_read = (bgrender_state == NT) & (x_rendercntr[2:0] == 3'b011);
2 assign attribute_read = (bgrender_state == AT) & (x_rendercntr[2:0] == 3'b011);
3 assign bg_lsb_read = (bgrender_state == BG_LSB) & (x_rendercntr[2:0] == 3'b011);
4 assign bg_msb_read = (bgrender_state == BG_MSB) & (x_rendercntr[2:0] == 3'b011);

```

## 6.6. hardverleírás részlet.

Az állapotgép alapján az adatok mentét vezérlő jelek

Az F.4 függelékben a 241. sorban látható VBLANK jel elő állítása során különösen figyelni kellet az időzítésre. Mivel ennek a jelnek a hatására képződik a 6502-es CPU felé az NMI (non maskable interrupt) megszakítás (a 6.7 IRQ), amelynek pontos lefutása ellengedhetetlen a játékok emulálása szempontjából. Ez a megszakítás tartalmazza általában

azt a programkódot, amely a név (NT) memória területek frissítéséért felelős. A pontos időzítést az FPGA projektben, egy két bites léptető regiszter használatával állítom be, ezt és a vezérlő jelek képzését a 6.7 hardverleírásban láthatjuk.

```

1 //Driving the interrupt flag
2 reg interrupt_flag;
3 wire interrupt_clr;
4 always @ (posedge clk)
5 begin
6     if (rst || interrupt_clr || (status_rd && ph2_falling))
7         interrupt_flag <= 1'b0;
8     else
9         if (vblank_set[2] && vblank[1])
10             interrupt_flag <= 1'b1;
11 end
12 //NMI towards cpu
13 always @ (posedge clk)
14 begin
15     if (rst)
16         irq <= 1'b0;
17     else
18         irq <= interrupt_flag & vblank_irq_enable;
19 end
20
21 // vblank set signals
22 assign vblank_set[0] = (x_rendercntr == (FIRST_SCANLINE_PIXEL + 0)) & (y_renderingcntr ==
23 V_BLANK_BEGIN);
24 assign vblank_set[1] = (x_rendercntr == (FIRST_SCANLINE_PIXEL + 4)) & (y_renderingcntr ==
25 V_BLANK_BEGIN);
26 assign vblank_set[2] = (x_rendercntr == (FIRST_SCANLINE_PIXEL + 12)) & (y_renderingcntr ==
27 V_BLANK_BEGIN);
28
29 //Indicates the end of the VBLANK period.
30 assign vblank_clr = (x_rendercntr == FIRST_SCANLINE_PIXEL + 4) & (y_renderingcntr ==
31 PRERENDERING_ROW);
32 assign interrupt_clr = (x_rendercntr == FIRST_SCANLINE_PIXEL + 12) & (y_renderingcntr ==
33 PRERENDERING_ROW);

```

**6.7. hardverleírás részlet.** VBLANK vezérlőjelek képzése és az NMI (IRQ) jel előállítása

### 6.3.3. Mozgó csempe elemek "Sprite" képalkotás

A PPU-ban található képalkotás második rétege a mozgó csempe elemek kiolvasása és feldolgozása a OAM memóriákból. Az eredeti hardver felépítéséről részletesen olvashatunk a 3.2.4 fejezetben, az itt olvasottak alapján alakítottam ki a OAM memóriákat is az FPGA modulban.

Az elsődleges és a másodlagos OAM memóriát is Distributed memória ként hoztam létre, a lehető leggyorsabb olvasási idő miatt. A Distributed memória az FPGA LUT-jából közvetlenül leképeződő memória, általában olyan helyeken szokták alkalmazni ahol kevés adatot kell tárolni, és gyors elérésre van szükségünk (azonnal elérhető). A mi felhasználásunk során elengedetlen a késleltetés mentes adat kiolvasás, mert a háttér állapotgépet és a sprite állapotgépet szinkronizálni kell a Pattern adatok olvasása miatt.

Az Sprite képalkotásban fontos szerepe van a DMA-ból érkező adat időben történő mentésének és ezzel párhuzamosan egy olyan elsődleges OAM RAM címzésnek, amely jól lekezeli a DMA-ból érkező adatokat és a sprite állapotgépben a másodlagos OAM memória feltöltését is.

A címképzésnek két fő módja van. Az első mód során, az elsődleges RAM címzését két mozgó számlálóval címezük meg, m és n. Az m a négy bájtos sprite adatokon halad végig, amíg az n az elsődleges memóriában található 64 sprite-ot címzi meg, ezeknek a számlálóknak az időzítését a sprite állapotgéphez kötöttem. Ezt a címzési módot használom a másodlagos OAM memória töltése során, tehát a következő sorban aktív spritok

keresésére. A második mód pedig a DMA működése során aktív cím, ez egy egyszerű fölfelé számláló, amely a CPU-tól kapja kezdő címét, az OAM címmegszámlálását során.

Az így létrejövő két címet megfelelően multiplexálva kapjuk meg, az elsődleges OAM memória címzését. A cím képzését és az elsődleges OAM írását és olvasását a 6.8 hardverleírásban olvashatjuk részletesen.

```

1 (* ram_style = "distributed" *)
2 reg [7:0] pri_oam [255:0];
3 reg [7:0] pri_oam_addr;
4 wire [1:0] pri_oam_addr_sel;
5 wire [7:0] pri_oam_dout = pri_oam[pri_oam_addr];
6
7 always @(*)
8 begin
9     case (pri_oam_addr_sel)
10        2'b01: pri_oam_addr <= {n_cnt, m_cnt};
11        2'b11: pri_oam_addr <= {pri_oam_addr_cnt[7:3], n_cnt[0], m_cnt};
12        default: pri_oam_addr <= pri_oam_addr_cnt;
13    endcase
14 end
15
16 always @(posedge clk)
17 begin
18     if (oam_data_wr)
19         pri_oam[pri_oam_addr] <= slv_mem_din;
20 end

```

#### 6.8. hardverleírás részlet. Az elsődleges OAM RAM írása és olvasása

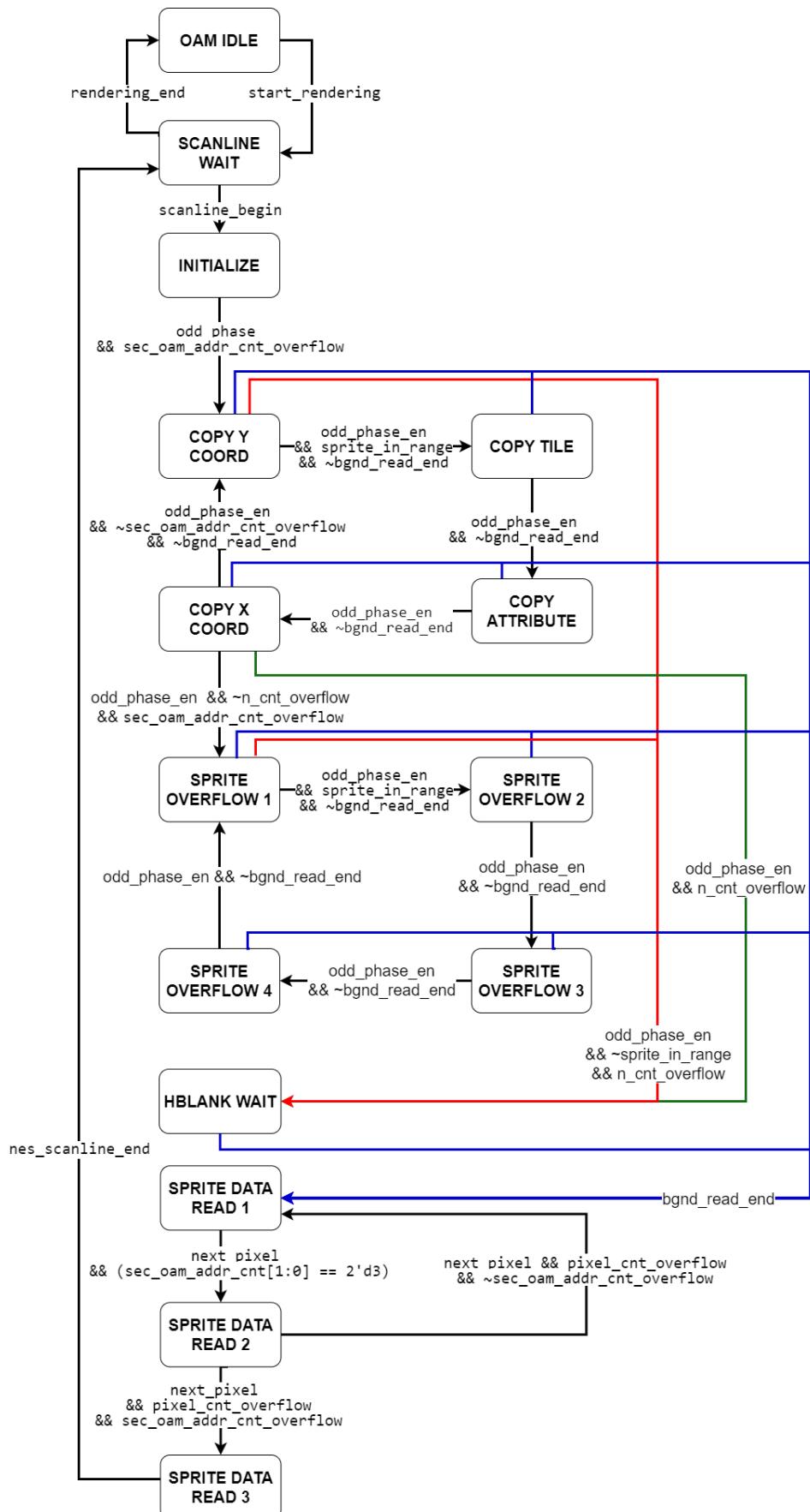
Az sprite képgenerálás állapotgépének fontos eleme még, a következő sorban aktív sprite-ok detektálásáért felelős hardveres egység, ezt a 6.9 hardver leírásban láthatjuk. Ennek alapja, hogy az állapotgép alapján képzett vezérlőjellel (oam\_temp\_reg\_wr) elmentjük az éppen aktív OAM sprite vertikális pozíció értékét és az éppen aktív sor értékből kivonva egy vertikális különbség értéket képzünk. Ezt vizsgálva a 8 x 8 és 8 x 16-os spritok esetére, detektálhatjuk, hogy az éppen aktív sprite a következő sorban megjelenik-e. Abban az esetben ha találtunk egy megjelenítendő sprite-ot akkor az állapotgép alapján (6.5) továbblépve ezt mentjük a másodlagos memóriába, vagy még egyel növeljük a túlcordulás jelzőt.

```

1 reg [8:0] y_diff_reg1;
2 wire sprite_in_range = (sprite_size) ? (y_diff_reg1[8:4] == 5'd0) : (y_diff_reg1[8:3] == 6'd0);
3
4
5 // if I subtract from scanline cnt - tile_addr then check the upper bits of y_diff_reg to be zero
6 // then we have a sprite in range
7 always @(posedge clk)
8 begin
9     if (rst)
10         y_diff_reg1 <= 9'd0;
11     else
12         if (oam_temp_reg_wr)
13             y_diff_reg1 <= scanline_cnt - {1'b0, pri_oam_dout};

```

#### 6.9. hardverleírás részlet. A következő kép generálási sorban aktív sprite-ok jelzése



6.5. ábra. Sprite képkotás és memória olvasást vezérlő állapotgép

Az állapotgép teljes körű megértéséhez elengedetlen, hogy az eredeti hardvert megértsük. Az eredeti sprite feldolgozás menete négy fő részre bontható, ez az F.4 függelékben is látható.

1. A nyolc előző sorban mentet sprite törlése (1-64 órajel) A 6.5 állapotgépben ez az IDLE állapot.
2. Az elsődleges OAM RAM, végig olvasása aktív sprite-okat keresve. minden egyes páratlan órajel során olvassa az adatokat, a párosok alatt pedig írja őket a memória területre. Ha aktív sprite-ot találunk a négy bájtját mentjük a másodlagos OAM RAM-ba. (65-256 órajel) A 6.5 állapotgépben ez sorrendben az Y koordináta másolása, csempe cím másolása, csempe attribútum másolása és X koordináta másolása állapotoknak felel meg.
3. Ha megtalált nyolc aktív sprite-ot, akkor tovább keres a sorban hátha túlcordulást érzékel. Itt a memória olvasásban egy hardveres hiba található, mivel ha talált egy extra aktív spriteot is akkor a következő sprite-ok Y koordinátája helyett a sorban következő bájtot (csempe cím) vizsgálja a rendszer. Ha ezt követően is talál még egy aktív sprite-ot, akkor pedig az ezt követő sprite bájtokat kezdi el vizsgálni (csempe attribútum). (65-256 órajel) A 6.5 állapotgépben ez a négy túlcordulási állapotnak felel meg.
4. Végül az utolsó lépésben az aktív sprite adatok kiolvasása és mentése következik. (257-320 órajel) A 6.5 állapotgépben ez a három sprite adat olvasási állapotnak felel meg.

A 6.5 ábrán látható állapotgép két extra állapottal bővül ki az FPGA rendszer szinkronizációja miatt. Az első két állapot (OAM IDLE és SCANLINE WAIT) a VGA jel generálás alatti IDLE állapotot reprezentálja (hasonlóan a 6.4 SLEEP állapotához). Illetve a HBLANK WAIT állapot amely a túlcordulás figyelést követően szinkronizálja, az állapot gép működését.

Az állapotgép negyedik szakasza teljes mértékben szinkron működik a háttér képalkotást vezérlő állapotgép Pattern adat olvasásával. Az innen kiolvasott két bájtos csempe adatokat egy-egy pufferbe mentjük, ha kevesebb mint 8 aktív sprite van egy sorban, akkor a fenntartható puffer területek nullás értéket kapnak. A pufferek léptető regiszterként tárolják a csempe adatokat és figyelik a képalkotás horizontális számlálóját, ha az aktuális sprite X koordinátája megegyezik a sor indexel, akkor elkezdődik a 8 pixelnyi képadat léptetése és kiadása. Ha a *sprite\_clipping* jel aktív akkor az első csempe oszlop idő tartalma alatt nem történhet mozgó csempe megjelenítés, tehát ezeknek a sprite-oknak az értéke nullázódik. Ennek segítségével tudunk szép folyamatos bal oldali görgetést létrehozni.

Végül a pufferekből érkező sprite, pixeladatai között meghatározzuk a prioritási sorrendet és a nagyobb prioritással rendelkező pixel adatait adjuk ki a sprite képalkotási modulból. A prioritást az elsődleges OAM memóriában való elhelyezkedés határozza meg, a nulladik sprite-nak van a legnagyobb prioritása és a 63.-nak pedig a legkisebb (tehát minél előbb kerül a másodlagos OAM-ba a sprite annál nagyobb a prioritása).

A játékok hibátlan emulálása szempontjából elengedhetetlen még a *sprite0\_hit* vezérlő jel pontos elő állítása. Ha ezt nem pontosan hozzuk létre egyes játékok, többek között a Super Mario Bros. a kezdő képernyőn ragad és nem kezd el futni a teljes szoftver (valószínűleg egy várakozó állapotban ragad a program). A jel egyik komponensét a *sprite0\_visible* jelet, a sprite képalkotást vezérlő modulon belül, a 6.10 hardverleírásban látható módon hozom létre. A végső *sprite0\_hit* jel pedig, a logikai és kapcsolata, a *sprite0\_visible*,

*visible\_bg\_pixel* és a *sprite0\_check* jeleknek. A látható háttér pixel, a nem átlátszó háttér pixeleket vizsgája, a *sprite0\_check* pedig azt, hogy a horizontális képalkotás elérte a 255. pixelig.

```

1 reg [1:0] sprite0_in_range;
2 assign sprite0_visible = sprite0_in_range[1] & !sprite_pixel[1:0] & ~(~sprite_enabled | (
3   first_column & ~no_sprite_clip));
4
5 always @(posedge clk)
6 begin
7   if (rst)
8     sprite0_in_range <= 2'd0;
9   else
10     if (scanline_begin)
11       sprite0_in_range <= {sprite0_in_range[0], 1'd0};
12     else
13       if (odd_phase_en && (oam_state == OAM_COPY_X_COORD) && (n_cnt == 6'd0))
14         sprite0_in_range <= {sprite0_in_range[1], 1'b1};
end

```

**6.10. hardverleírás részlet.** A *sprite0\_visible* vezérlő jel elő állítása

### 6.3.4. PPU adatbusz és memória elérése

A PPU működéséhez elengedetlen a VRAM cím regiszterének képzése is, ennek felépítése két rétegből tevődik össze (az eredeti PPU mintájára). Az első réteg a 6.3.1 fejezetben leírtak alapján, a CPU adatbuszról érkező regiszter írások hatására, regiszterekbe menti a VRAM címrészleteket, ennek a pontos változásai a 6.6 ábrán láthatók (t regiszterként). A második réteg, pedig a képgenerálás során képződő számlálást és a soron kívüli regiszter töltésekét és fölfelé számolásokat foglalja magába (a 6.6 ábrán v regiszternek felel meg).

A cím számláló réteg különleges olyan szempontból, hogy megkülönböztet két fajta fölfelé számolási sorrendet. Az első a standard számlálás, amikor a cím két része a horizontális és a vertikális külön-külön számolnak fölfelé, mint két különálló regiszter. Ezt a számlálási módöt használja a PPU az F.4 függelékben látható vörös pixelek megvalósítása során, tehát az általános képgeneráláskor. A második számlálási mód pedig, amikor a VRAM írása vagy olvasása hatására egy fölfelé számlálás történik, itt a teljes VRAM címregiszter számol fölfelé 1-et vagy 32-t a (a számlálási mód függvényében).

Action	Before				Instructions	After				Notes
	t	v	x	w		t	v	x	w	
\$2000 write	.....	.....	...	.	LDA #\$00 (%00000000) STA \$2000	...00	.....	....	..	
\$2002 read	...00...	.....	...	.	LDA \$2002	...00...	.....	....	0	Resets paired write latch w to 0.
\$2005 write 1	...00...	.....	...	0	LDA #\$7D (%01111101) STA \$2005	...00... 01111	.....	101	1	
\$2005 write 2	...00... 01111	.....	101	1	LDA #\$5E (%01011110) STA \$2005	1100001 01101111	.....	101	0	
\$2006 write 1	1100001 01101111	.....	101	0	LDA #\$3D (%00111101) STA \$2006	0111101 01101111	.....	101	1	Bit 14 (15th bit) of t gets set to zero
\$2006 write 2	0111101 01101111	.....	101	1	LDA #\$F0 (%11110000) STA \$2006	0111101 01110000	0111101 11110000	101	0	After t is updated, contents of t copied into v

**6.6. ábra.** VRAM cím regiszter értékkadása (eredeti PPU)

A 3.2.3 fejezetben leírtak alapján, a VRAM címszámláló regiszterekből hozom létre, a név tábla és attribútum tábla címeit. Majd az NT állapotban kiolasott Pattern cím adatok segítségével, létrehozom a háttér csempeadatok olvasási címét. A sprite állapot gépből érkező adatok alapján a *sprite\_size*-nak megfelelően állítom elő sprite Pattern címeket, a 8 aktív csempeadat kiolasása számára. A pontos címképzéseket a 6.11 hardverleírásban olvashatjuk.

```

1 wire [13:0] ppu_nt_addr = {2'b10, v_cnt, h_cnt, vt_cnt, ht_cnt};
2 wire [13:0] ppu_at_addr = {2'b10, v_cnt, h_cnt, 4'b1111, vt_cnt[4:2], ht_cnt[4:2]};
3 wire [13:0] bg_lsb_addr = {1'b0, background_pattern_sel, tile_index_reg, 1'b0, fv_cnt};
4 wire [13:0] bg_msb_addr = {1'b0, background_pattern_sel, tile_index_reg, 1'b1, fv_cnt};
5
6 assign sprite_lsb_addr = (~sprite_size) ? ({1'b0, sprite_pattern_sel, sprite_tile_index, 1'b0,
    sprite_range[2:0]}) : ({1'b0, sprite_tile_index[0], sprite_tile_index[7:1], sprite_range[3],
    1'b0, sprite_range[2:0]});
7 assign sprite_msb_addr = (~sprite_size) ? ({1'b0, sprite_pattern_sel, sprite_tile_index, 1'b1,
    sprite_range[2:0]}) : ({1'b0, sprite_tile_index[0], sprite_tile_index[7:1], sprite_range[3],
    1'b1, sprite_range[2:0]});

```

**6.11. hardverleírás részlet.** VRAM cím képzése a számlálókból és az állapotgépek alapján

A PPU (mester) adatbusz cím regiszterének meghajtása során, figyelembe kell vennünk azt, hogy ha a chipen bármilyen VRAM elérést hajtanak végre, akkor a címregisztert ne a VRAM cím számláló rétegeből határozzuk meg, hanem egy extra regiszteren a *vram\_addr\_reg*-en keresztül adjuk ki, ezzel késleltetve a címünket egy órajellel. Ez az eredeti NES PPU-ban is, egy hardveres hiba volt, amit játékok fel is használtak (például a Super Mario Bros.). Abban az esetben is, ha a háttér vagy a sprite képalkotás nincs kiválasztva tehát *ppu\_enable* inaktív, automatikusan a késleltetett VRAM címét kell kiadni.

A végső cím kimenetet még szinkronizálnunk kell a chip többi részéhez, erre szolgál az *addr\_reg\_ld* jel. Az adatbusz címregiszterének képzését a 6.12 hardverleírásban láthatjuk részletesebben.

```

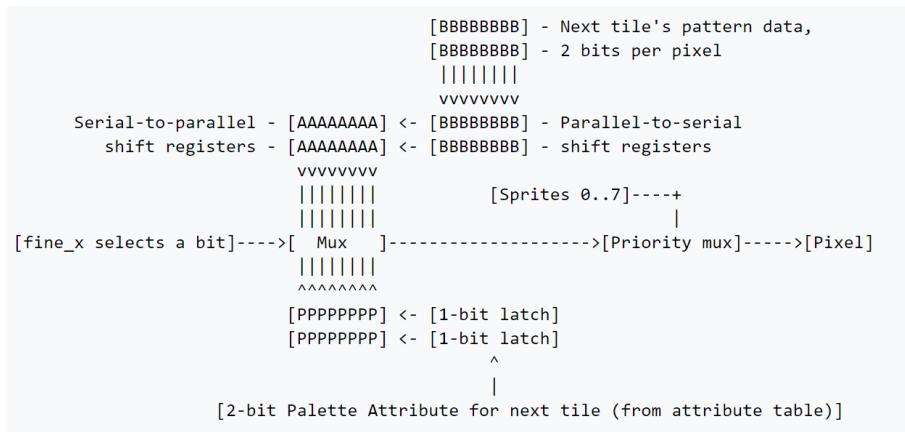
1 wire vram_address_sel = (vram_state_machine_sel == VRAM_RD_WAIT) || (vram_state_machine_sel ==
    VRAM_WR_WAIT);
2 wire [7:0] ppu_address_sel;
3 wire addr_reg_ld = (x_rendercntr[2:0] == 2'b100);
4
5 assign ppu_address_sel [0] = vram_address_sel;
6 assign ppu_address_sel [1] = (bgrender_state == NT);
7 assign ppu_address_sel [2] = (bgrender_state == AT);
8 assign ppu_address_sel [3] = (bgrender_state == BG_LSB) & sprite_read;
9 assign ppu_address_sel [4] = (bgrender_state == BG_MSB) & sprite_read;
10 assign ppu_address_sel [5] = (bgrender_state == BG_LSB) & ~sprite_read;
11 assign ppu_address_sel [6] = (bgrender_state == BG_MSB) & ~sprite_read;
12 assign ppu_address_sel [7] = ppu_enable;
13
14 always @(posedge clk)
15 begin
16     if (rst)
17         ppu_mem_addr <= 14'd0;
18     else
19         if (addr_reg_ld)
20             case (ppu_address_sel)
21                 8'b10000010 : ppu_mem_addr <= ppu_nt_addr;
22                 8'b10000100 : ppu_mem_addr <= ppu_at_addr;
23                 8'b10001000 : ppu_mem_addr <= sprite_lsb_addr;
24                 8'b10010000 : ppu_mem_addr <= sprite_msb_addr;
25                 8'b10100000 : ppu_mem_addr <= bg_lsb_addr;
26                 8'b11000000 : ppu_mem_addr <= bg_msb_addr;
27                 default      : ppu_mem_addr <= vram_addr_reg;
28             endcase
29     end

```

**6.12. hardverleírás részlet.** PPU master adatbusz címregiszterének meghajtása és időzítése

### 6.3.5. A PPU részéről történő kép generálás működése

A háttér állapot gép futása során kiolvasott csempe adatok feldolgozása hasonlít a sprite pixel feldolgozásra, azzal a különbséggel, hogy ennek folytonosan futnia kell a háttér képalkotás ideje alatt. Ezt az FPGA modulon belül egy különleges hardveres elemmel egy LUT-ban szintetizálódó léptető regiszterrel SRL-el (shift register LUT) oldottam meg. Ennek az elemnek a különbsége, hogy a teljes regiszter töltése nem engedélyezett, csak a regiszterek bitszélességével történő léptetéses töltés. Ennek előnye hogy könnyen össze szinkronizálható a háttér csempék két bájt-ja (mivel az LSB és MSB bájtok bit párai alkotják a háttér egy pixelét) és kimenetként pedig bármelyik bit kiválasztható. A kimenet választás azért fontos, mert a görgetés során egy csempét pixelekként szeretnénk ki görgetni a megjelenített kép tartományból és ennek segítségével az összes párhuzamosan működő regiszter aktív kimeneti bitje, egyszerre módosítható. Ilyen SRL-t használtam a csempe attribútumának megjelenítésére is. Ezt az adatfeldolgozó lánctot olvashatjuk a 6.13 hardverleíró részletben.



**6.7. ábra.** A NES PPU-jának pixel feldolgozó egysége

Az állapotgépek pixel feldolgozó egysége és működése nagyban hasonlít az eredeti hardver felépítésére, ezt a 6.7 ábrán láthatjuk. A PPU két állapot gépe során kapott pixel adatok összefűzésének és feldolgozásának három fő fázisa van.

1. *Prioritás vizsgálat, végső pixel meghatározása:* A sprite képalkotás kétféle képen történhet, a mozgó csempe elem kerülhet a háttér elő, illetve a háttér mögé. Erről a csempe prioritása dönt(ha: 1 akkor a háttér mögé kerül a mozgó elem), ennek segítségével alkották meg a játékfejlesztők például a Mario húsevő növényeit. A pixel kiválasztás során még figyelembe kell venni hogy a háttér pixel átlátszó-e, ebben az esetben mindenképpen a sprite képkockáját válasszuk. Ezt a kiválasztási folyamatot láthatjuk a 6.13 hardver leíró részlet alján.
2. *Paletta lookup táblázat:* Az így kiválasztott pixel egy paletta címet képez, a NES CPU-ja által töltött Paletta lookup táblázatban. Ennek működéséről olvashatunk a 3.2.2 fejezet során. Az FPGA-ban történő megvalósítás során Distributed memóriát használtam, a késleltetés mentes olvasás miatt. A helyes emulálás során különösen figyelnünk kell a memória területek helyes tükrözésére, ha ezt nem jól implementáljuk egyes játékok színei nagyban eltérhetnek az eredetitől, például a Mario háttere kék helyett fekete lesz. Ennek helyes implementálását olvashatjuk a 6.14 hardver leírásban. Ha a *BG\_clipping* jel aktív és az első csempe oszlopban vagyunk, vagy a PPU várakozó állapotban van (6.4 SLEEP állapota), akkor automatikusan egy fekete pixel értékével helyettesítjük a palettából olvasott címet.

3. *RGB blokk ROM*: Az FPGA implementálás során a paletta memóriából kiolvasott színértéket, egy címként értemeztem. Ennek feldolgozásához készítettem egy RGB adatokkal előre feltöltött ROM-ot, amely a címhez a megfelelő 24 bites RGB értéket rendeli. Itt még figyelembe vettetem a NES képalkotást befolyásoló két funkciót. Az előző a monokróm mód, amely fekete fehér képalkotást tesz lehetővé. Ezt úgy implementáltam, hogy a palettából kiolvasott cím, HUE értékének megfelelő biteket nullával helyettesítem. A második ilyen mód pedig, a szín kiemelés (a PPU képes volt, adott színek kiemelésére és a többi szín elsötétítésére). Ezt a tulajdonságot úgy implementáltam, hogy a Paletta ROM címébe beletartoznak az *emphesize\_r/ - \_g/ - \_b* jelek, így egy kicsit nagyobb blokk ROM-ot képezve a hardverben.

```

1 //LSB of background pixel
2 reg [7:0] bg_lsb_buff_reg;
3 always @(posedge clk)
4 begin
5   if (rst)
6     bg_lsb_buff_reg <= 8'd0;
7   else
8     if (bg_msb_read)
9       bg_lsb_buff_reg <= bg_lsb_reg; // shift reg
10    else if ((x_rendercntr[1:0] == 2'b11) && (x_rendercntr > START_OF_SHIFT) && (x_rendercntr <=
11      END_OF_SHIFT) && ~(bgrender_state == VBLANK))
12      bg_lsb_buff_reg <= {bg_lsb_buff_reg[6:0], 1'b0};
13 end
14 reg [7:0] shr_lsb_render = 8'd0;
15 wire bg_lsb_out;
16 always @(posedge clk)
17   if ((x_rendercntr[1:0] == 2'b11) && (x_rendercntr > START_OF_SHIFT) && (x_rendercntr <=
18      END_OF_SHIFT) && ~(bgrender_state == VBLANK))
19     shr_lsb_render <= {shr_lsb_render[6:0], bg_lsb_buff_reg[7]};
20 assign bg_lsb_out = shr_lsb_render[~fh_reg];
21 //MSB of background pixel
22 reg [7:0] shr_msb_render = 8'd0;
23 wire bg_msb_out;
24 always @(posedge clk)
25   if ((x_rendercntr[1:0] == 2'b11) && (x_rendercntr > START_OF_SHIFT) && (x_rendercntr <=
26      END_OF_SHIFT) && ~(bgrender_state == VBLANK))
27     shr_msb_render <= {shr_msb_render[6:0], bg_msb_reg[7]};
28 assign bg_msb_out = shr_msb_render[~fh_reg];
29
30 //0 bit of attribute
31 reg [7:0] shr_attr0_render = 8'd0;
32 always @(posedge clk)
33   if ((x_rendercntr[1:0] == 2'b11) && (x_rendercntr > START_OF_SHIFT) && (x_rendercntr <=
34      END_OF_SHIFT) && ~(bgrender_state == VBLANK))
35     shr_attr0_render <= {shr_attr0_render[6:0], tile_attr_reg_saved[0]};
36 wire shr_attr0_out = shr_attr0_render[~fh_reg];
37 //1 bit of attribute
38 reg [7:0] shr_attr1_render = 8'd0;
39 always @(posedge clk)
40   if ((x_rendercntr[1:0] == 2'b11) && (x_rendercntr > START_OF_SHIFT) && (x_rendercntr <=
41      END_OF_SHIFT) && ~(bgrender_state == VBLANK))
42     shr_attr1_render <= {shr_attr1_render[6:0], tile_attr_reg_saved[1]};
43 wire shr_attr1_out = shr_attr1_render[~fh_reg];
44
45 //pixel output
46 reg [3:0] bg_pixel;
47 always @(posedge clk)
48 begin
49   if (rst)
50     bg_pixel <= 4'b0;
51   else
52     bg_pixel <= {shr_attr1_out, shr_attr0_out, bg_msb_out, bg_lsb_out};
53 end
54 wire [4:0] palette_addr = (sprite_priority & visible_bg_pixel) ? ({1'b0, bg_pixel}) : ({1'b1,
55   sprite_pixel});

```

### 6.13. hardverleírás részlet. Pixel adat feldolgozó SRL léptető regiszterek

```

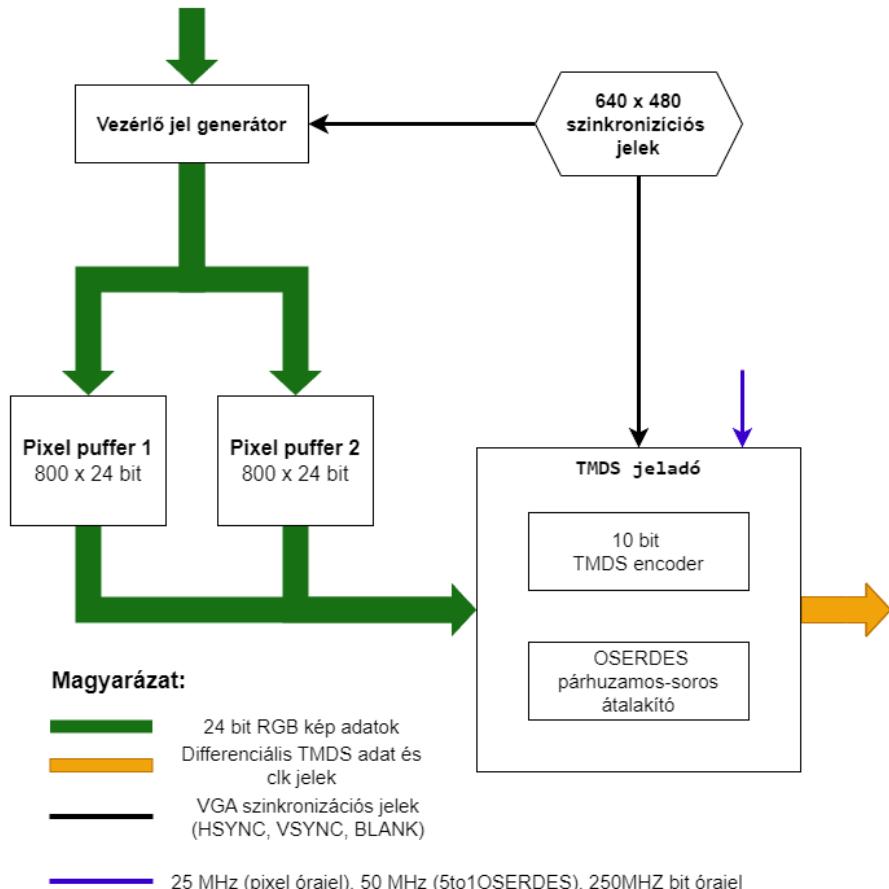
1 wire [4:0] palette_ram_nt_addr_mirrored = {vram_addr_cnt[4] & ~vram_addr_cnt[1:0], vram_addr_cnt[3:0]};
2 wire [4:0] palette_ram_addr_mirrored = {palette_addr[4] & ~palette_addr[1:0], palette_addr[3:0]};

```

#### 6.14. hardverleírás részlet. A paletta címek tükrözése

##### 6.3.6. A VGA képalkotás

A PPU véső hardveres eleme, az RGB adatokat tároló és megjelenítő VGA modul, ennek felépítését láthatjuk a 6.8 ábrán.



6.8. ábra. VGA top modul blokkvázlata

A PPU bemutatása során (a 6.3) már bemutattam a chip áttervezésének, alap ötletét, miszerint a PPU egy képalkotási sora alatt 1600 órajel, a pixeleket minden második órajelben mintavételezve megkapom a 800 pixel széles VGA képem egy sorát. Majd a következő kép sor mintavételezése alatt, az előző sort kétszer kiadva hozzuk létre az eredeti NES pixelek nagyítását és kapjuk meg a 640 x 480-as VGA képalkotás két sorát. Így a VGA kép egy sor késésben lesz a PPU képalkotáshoz képest, viszont helyes indulási időzítések mellett, ez nem észrevehető. Puffereket egy-egy kilobájtnyi blokk RAM-ként alakítottam ki a VGA modulon belül, mivel az így kiolvasható adatok egy órajelnyi késleltetése nem okoz semmilyen extra csúszást a VGA időzítésben. Ez a dupla pufferelés látható a 6.8 ábrán is, természetesen ezek közül egyszerre csak az egyik puffer kimenet aktív és a két pufferkimenetei dinamikusan váltják egymást a PPU-ban történő képsorok számának növekedésével.

A TMDS jeladó és a pufferek töltésének kezdetét vezérlő jelek időzítése különösen fontos, hiszen ezek segítségével vagyunk képesek a PPU aktív képtartományát pozicionálni

a VGA pixel sor közepére. A pixel kétszerezést követően a PPU aktív képtartománya 512 pixel órajelig tart. Tehát A helyes időzítéshez 64 pixel órajelig fekete pixelt kell a TMDS adónak venni, majd következik az 512 pixelnyi aktív képkocka, végül pedig 224 képkockányi fekete pixel, ez a tartomány tartalmazza a horizontális kioltási területeket is. Ezt a képkocka arányt határozza meg a 6.4 háttér állapotgében, a SLEEP állapotból történő kilépés időzítése. Illetve a puffer töltés kezdetének meghatározásával, még finom hangolható a kép pontos beállítása. Ahhoz, hogy a kezdetektől fogva helyes kép jelenjen meg a monitorunkon, a VGA szinkronizálási jelek generálását is be kellet késleltetnem a pre-rendering sor, és az első puffer feltöltésének idő tartalmával. Ezt az időzítést egy reset jelre nullázódó számláló segítségével oldottam meg, amely 3201 pixel órajelet (két teljes PPU sor + 1 pixel órajel) követően engedélyezi a generálást. A szinkronizációs jelek pontos időzítése a pixel órajelhez képest, a 6.15 hardverleírásban látható.

```

1 localparam H_BLANK_BEGIN      = 10'd639;
2 localparam H_SYNC_BEGIN       = 10'd655;
3 localparam H_SYNC_END         = 10'd751;
4 localparam H_BLANK_END        = 10'd799;
5
6 localparam V_BLANK_BEGIN      = 10'd479;
7 localparam V_SYNC_BEGIN       = 10'd489;
8 localparam V_SYNC_END         = 10'd491;
9 localparam V_BLANK_END        = 10'd523;
```

### 6.15. hardverleírás részlet. A képgenerálás időzítési paraméterei

TMDS adók két fő elemből épülnek fel. minden színcsatorna tartalmaz egy 10 bites TMDS kódolót, amely az RGB adatokat, csatornánként (8 bit) egy 10 bites kiadható jelle kódolja. Ezek közül egyedül a kék színcsatorna tartalmazza a hsync, vsync jeleket. Majd ezt követően, ezeket a 10 bites adatokat egy-egy öt az egyhez sorosító (OSERDES) alakítja át a differenciális p és n jelekké. Az OSERDES vezérlő strobe-ja 50 MHz-en működik, a kiadott bitek pedig 250 MHz-el jönnek. A VGA jel szinkronizációs jeleiről és a TMDS adók működéséről részletesebben a [22] szakirodalomban olvashatunk, ennek alapján hoztam létre én is a TMDS jeladót.

## 6.4. NES memória felépítése FPGA-ban

A memória menedzser modul fejlesztése, során a 6 bevezetésben leírt alapelveket követve, az egyszerűséget és a gyors, könnyű tesztelhetőséget előtérbe helyezve, a játék kazettákon található karakter és program ROM-okat is a modulon belül blokk ROM-ként helyeztem el. Az 5 FPGA kártya ismertetése során kitértem a Spartan-6-ban található blokk RAM-ok számára, ez 32 db 8 kilobites blokk RAM (64 kilobájt). A NES FPGA hardverhez, legnagyobb mapper chip nélküli játék esetén a következő memória területek szükségesek:

- 32 kilobájt program ROM (játék kártya),
- 8 kilobájt CH ROM (játék kártya),
- 2 kilobájt cpu work RAM (alaplap),
- 2 kilobájt név tábla memória (alaplap),
- 2 kilobájt vga képalkotás pufferek (PPU),
- 512 \* 3 bájt paletta RAM (PPU).

A fentebbiekből kiszámolható, hogy a kezdeti teszteléshez nem szükséges, hogy a játék kártyák memória területét SRAM-ban helyezzem el, mivel a Super Mario Bros. esetén is

(32 kilobájt program ROM), még marad 16 kilobájt blokk RAM a hardverben. Ennek az egyszerűsítésnek a segítségével a teszteléshez nem kell lefejleszteni a 6502-es processzorhoz egy UART interfést (és az ehhez szükséges szoftvereket), hogy ezen keresztül töltük fel a SRAM memória területeit, a NES hardver indítása előtt.

A játék kártyákon található ROM területek implementálása során, figyelembe vettet, hogy a blokk RAM-jaim elérési ideje ugyanakkora legyen mintha az SRAM-ot érnénk el, így a jövőben könnyen kiegészíthetővé és cserélhetővé válik ez a hardveres modul. Az egységesítés miatt az NT RAM elérési idejét is ugyanennyire állítottam be, így a PPU memória eléréseinek időzítése során csupán egyfélé időzítésre kellet figyelnem. Ez a gyakorlatban azt jelentette, hogy a memória olvasások két órajel késleltetéssel jelennek meg a kimeneten, az írások pedig egy órajel időtartam alatt valósulnak meg (ez megfelel az SRAM 10 ns-umos elérési idejének). Az olvasás késleltetését egy regiszter pufferrel valósítottam meg.

A memória menedzser teljes adatbusz interfésekkel implementál, ez azt jelenti, hogy a teljes címet megkapja a CPU (16 bit) és a PPU (14 bit) felől a modul. A különböző memóriák kiválasztása és címének meghatározása a 3.1 és 3.2 táblázatok alapján, a 6.16 hardver leírásban olvasható.

```

1 wire [10:0] cpu_work_ram_addr = cpu_addr[10:0]; //cpu inner memory access
2 wire      cpu_work_ram_sel  = (cpu_addr[15:13] == 3'b000); //0x0000-0x07FF and the mirrors until
3          Ox1FFF
4
5 wire [14:0] cpu_prog_rom_addr = cpu_addr[14:0];
6 wire      cpu_prog_rom_sel  = cpu_addr[15];
7
8 wire [11:0] ppu_name_table_addr = {ppu_name_table_addr_h, ppu_addr[9:0]};
9 wire      ppu_name_table_sel = ppu_addr[13];
10
11 wire [12:0] ppu_ch_rom_address = ppu_addr[12:0];
12 wire      ppu_ch_rom_sel = ~ppu_addr[13];

```

### 6.16. hardverleírás részlet.

A memória menedzserben található memória területek címzése

A jövőbeli könnyebb kibővíthetőség érdekében, a PPU névtábla tükrözései közül az összeset implementáltam, illetve fenntartottam egy extra helyet különlegesebb mapper-ek tükrözései számára. Jelenleg a tükrözést a modulban egy paramétereként állíthatjuk, de a jövőben ezt a mapper chip, illetve a játék program fogja állítani. A tükrözések működéséről és felhasználásáról részletesebben olvashatunk a 3.2.3 fejezetben, ezek implementálását pedig, a 6.17 hardverleírásban láthatjuk.

```

1 reg [1:0] ppu_name_table_addr_h;
2 reg [1:0] mapper_mirroring; //in the future if we have a mapper
3
4 always @ (*)
5 begin
6   case (NT_MIRRORING)
7     // Horizontal Mirroring
8     2'b00: ppu_name_table_addr_h <= {1'b0, ppu_addr[11]};
9     // Verical Mirroring
10    2'b01: ppu_name_table_addr_h <= {1'b0, ppu_addr[10]};
11    // Mappers Mirroring
12    2'b10: ppu_name_table_addr_h <= mapper_mirroring;
13    // Four-screen mirroring
14    2'b11: ppu_name_table_addr_h <= {ppu_addr[11:10]};
15   endcase
16 end

```

### 6.17. hardverleírás részlet.

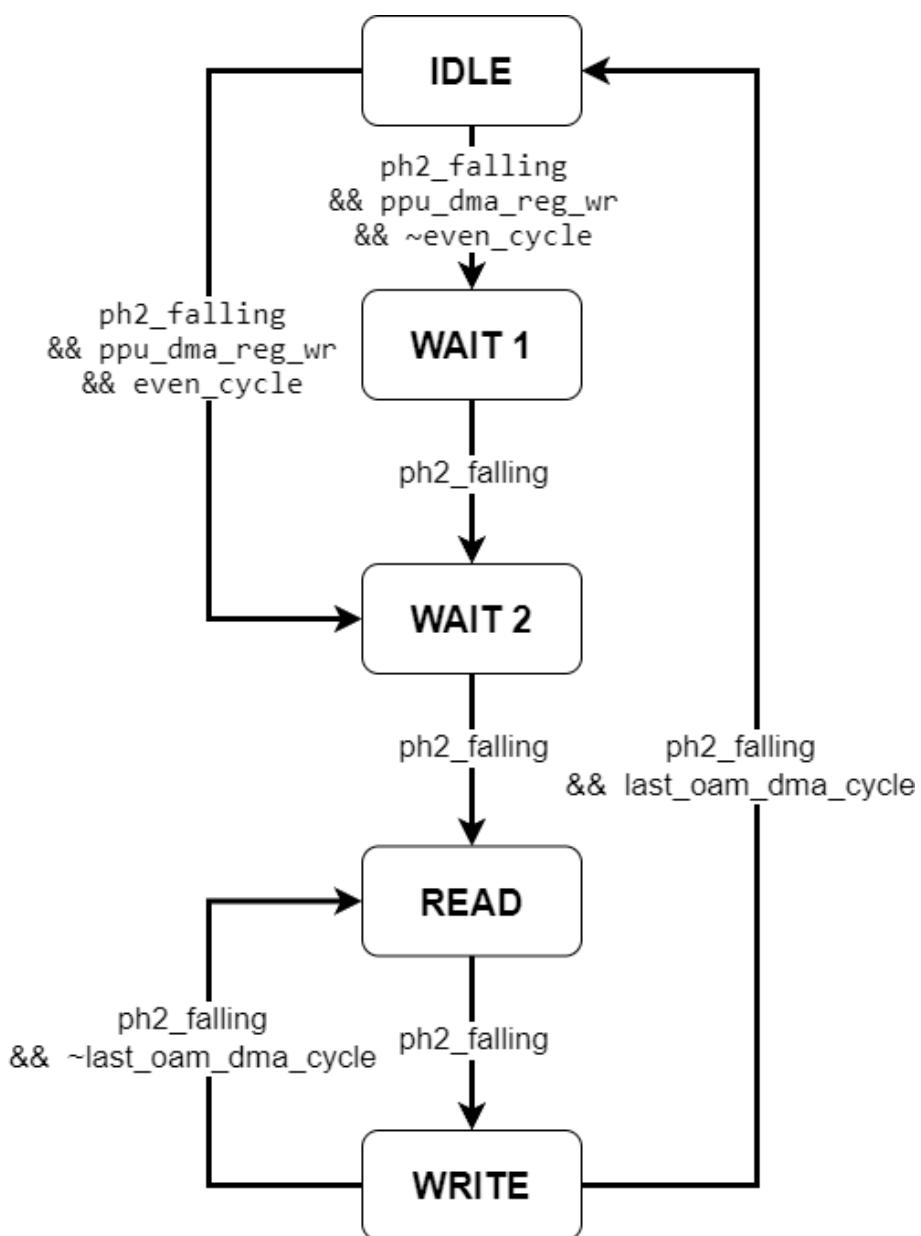
A PPU görgetéshez használt névtábla tükrözés

A CPU memória elérésének csupán a blokk RAM egy órajeles késleltetése van, viszont az adatbusz meghajtása során figyelni kell, hogy a *ph2\_falling* jel hatására nullába kell állítanunk modul kimenetét (a top modulban az adatbuszok összefűzésének érdekében).

## 6.5. DMA - FPGA megvalósítása

Az eredeti DMA működéséről a 3.3.3 fejezetben olvashatunk. Az DMA modul létrehozása során a legnagyobb kihívást a pontos időzítése jelentette. Mivel a 6.1 ábrán is látható módon, ha a DMA-t eléri a CPU, akkor ezt követően a DMA átveszi a CPU adatbuszát, viszont ennek szinkronban kell végbemennie a többi hardveres elem miatt. Ennek megoldása érdekében a modul egy speciális bemenettel rendelkezik, amely a CPU órajel ciklusának paritását vizsgálja. A 6.9 állapotgépen látható is, hogy ez az órajel illesztést egy speciális késleltetéssel oldottam meg, ha páratlan ciklusban érkezett a CPU kérése akkor két teljes CPU órajelet várunk, ha párosban akkor csak egyet. Ezzel pontosan annyi ciklus időt hagyva a processzor számára, hogy befejezze a jelenlegi feladatait.

A CPU adatbuszét pedig a processzor ready jelének elvételével, és a DMA *valid\_address* vezérlő jel egybe állításával veszi el. A többi része az adatbusznak egyszerű logikai vagy műveletek segítségével összevonható 6.1.

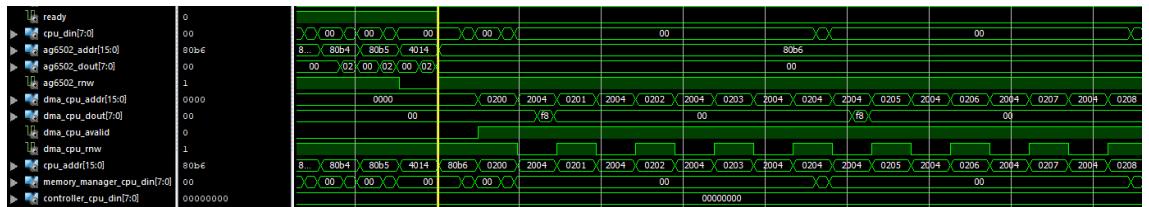


6.9. ábra. OAM DMA működési állapotgépe

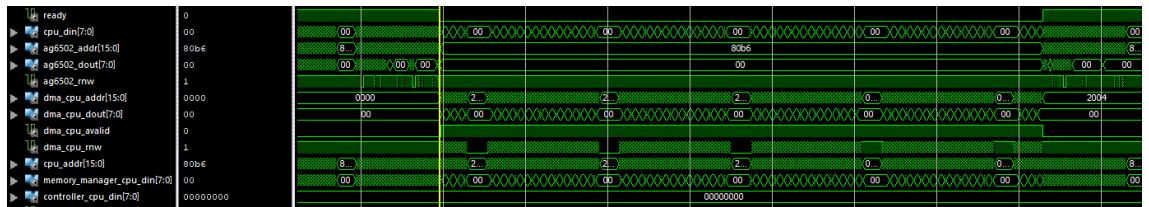
Ezt a szinkronizációs problémát leszámítva, a hardver működése egyszerű. A CPU a DMA elérése során egy címet ad az eszköznek, amely a CPU munka memóriájának egy területére mutat, általában \$0200. Majd innen az olvasási és írási állapotok váltakozásával  $64 * 4$  bájt memória mennyiséget olvas és ír (a teljes elsődleges OAM memóriát). Az írás a PPU \$2004-es OAM data regiszterére történik, ennek feldolgozását a 6.3.3 fejezetben olvashatjuk.

A memória terület másolását követően az eszköz, vissza adja a CPU címbuszát és master adatbusz kimeneteit vissza állítom nullára, ezzel elengedve a CPU adatbuszát.

Egy DMA ciklus teljes szimulációját a 6.11 képen láthatjuk, már itt is megfigyelhető, hogy a ready jel elvételét követően a DMA címbusza elkezd adni és ezek az értékek kerülnek ki a CPU összesített címbuszára. A 6.10 szimulációs képen pedig egy DMA ciklus kezdetét láthatjuk. Itt nyomon követhetjük a címbuszok pontos tartalmát. Tehát láthatjuk, hogy a DMA \$0200 címről kezd el olvasni (ezt a címet ciklusonként növeli) és a \$2004-es címre pedig ír. Az írás és olvasás váltakozását, pedig az RNW jel váltakozása mutatja.



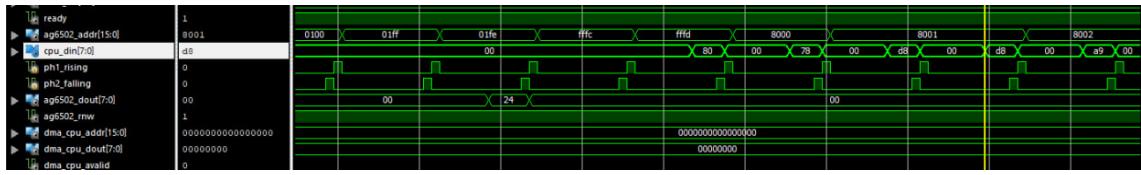
**6.10. ábra.** A DMA ciklus kezdete szimulációban



**6.11. ábra.** Teljes DMA ciklus szimuláció

## 6.6. 6502 processzor működése az FPGA rendszerben

A PPU és az eddig elkészített hardveres elemek tesztelhetősége véget, a projekt ezen fázisában inkább különböző nyílt forráskódú (Open Source) processzorokat próbáltam a rendszerbe illeszteni. Itt nagy segítségemre volt az OpenCores nevű weboldal [18], amely különböző nyílt forráskódú processzorok és hardveres eszközök hardverleírását tartalmazta. Illetve a szintén nyílt forráskódú Mister NES projekt [11], amely rendelkezik egy VHDL nyelven íródott processzorral. Ezek rendszerbe illesztése viszont nagyon idő igényesnek érződött, ezért konzulensemhez fordultam segítségért. Szerencsére Raikovitch Tamás el tudott látni, egy általa készített és a karbantartott 6502-es processzor spartan-6-osra szintetizált és imlementált processzor binárisával a tesztelésekhez. Ez egy régebbi projektje során használt megbízható processzor implementációjának bizonyult, amelyet könnyen az eddig implementált rendszerembe tudtam illeszteni. A 6.12 szimulációs képen jól látható, hogy a rendszer helyesen olvassa be a Super Mario első utasítását, ez körülbelül a kép közepén a *cpu\_din* csatornáján, 78-as (hexa) assembly kód.



**6.12. ábra.** A CPU szimulációja működés közben, Super Mario Bros. első assembly kódjának olvasása

A CPU-nak köszönhetően, a tesztelések során biztos lehettem abban, hogy csak az általam létrehozott rendszerben van hiba és a processzor teljes mértékben jól működik. Ezt követően könnyebb lesz egy nyílt forráskódú 6502 illesztése a NES-embe mivel biztos lehetek, hogy az általam készített hardver helyesen, az eredeti NES-nek megfelelően működik (így könnyebb a CPU hibáinak megtalálása).

## 6.7. NES kontrollerek kezelése

Az eredeti NES kontrollerek működését a 3.4 fejezet során már ismertettem. Az FPGA implementáció során a modul feladata a kontrollerek párhuzamos-soros léptető regisztereit felé a vezérlő jelek küldése, tehát az olvasási szándék jelzése *controller\_out\_latch*, illetve a bitek olvasását követő órajelek küldése *controller<sub>1-2</sub>\_out\_clk*. A kontrollerekből érkező adatok mintavételezésére, egy felfutó él érzékelést készítettem a CPU \$4016-as (kontroller 1) és \$4017-as (kontroller 2) regiszterek olvasásának detektálására annak érdekében, hogy minden stabil adatot olvassunk (az olvasás egyszer történjen meg). Az érzékelést egy-egy kétbites léptetőregiszterrel implementáltam, amelyekbe egyes értéket léptetek a jelek érzékelésekor.

A kontrollerekből érkező egybites adatokat, a CPU egy bájt ként olvassa ki (az adat a 0. helyi értékű biten szerepel). A CPU adatbusz kezelése során itt is figyeltem az interfész helyes implementációjára, tehát a *ph2\_falling* jel hatására nullás értékre állítom a CPU adat kimeneteket. A modulban minden kontroller, írását és olvasását elhelyeztem.

## 7. fejezet

# Az FPGA NES tesztelése

Az FPGA-s projektek tesztelésének általában három nagy fázisa van. Az első a fejlesztői környezet, szimulációs rendszerében futtatott tesztek. A második, pedig ezt követően az éles hardverrel futtatott tesztek. Végül a teszteket végeztével az eredmények kiértékelése a harmadik fázis. Ezeket a teszt folyamatokat fogom bemutatni a diplomaterv projektemen, ebben a fejezetben.

### 7.1. Rendszer szimuláció

A legtöbb esetben egy FPGA-s fejlesztés során a tesztelések két csoportra bonthatók, a modul tesztekre, és végül a teljes rendszer tesztelése. A NES projekt esetén viszont olyan komplex változásokat és inputokat kellet volna időzítéspontosan megadni a különálló modul tesztekhez (teljes CPU ciklusok processzor oldali műveletei), hogy inkább közvetlenül a teljes rendszer tesztelését kezdtem el.

A rendszer teszteket az ISE-n bellül található, ISim szimulátorában végeztem, ezzel képesek vagyunk szintetizálható hardverek működési szimulációjára, és implementálás utáni, úgy nevezet post-implemented rendszerek szimulálására is. A szintetizálható rendszerek esetben Verilog test fixture készítésével tudjuk meghatározni a szimulált jeleinket és szimulációs órajeleinket. A NES test programjának írása során, a VGA modult és a hozzá tartozó elemeket kivettem a rendszerből, mivel ez csak extra számítást vett volna el a szimulátortól, illetve a rendszer időzítéseinek szinkronizálását is nehezebbé tette volna (50 MHz és 250 MHz-es órajelek arányainak megtartása a rendszer órajellel).

Ahhoz hogy a teljes NES hardveremet szimulálhassam, szimulálhatóvá kellet tennem a konzulensemétől kapott 6502 futtatható binárisát. Ezt a feladatot az UNISIM konzol alkalmazás segítségével tudtam megoldani. Ez az eszköz azt teszi lehetővé, hogy implementált modulokból post-implemented szimulációs fájlt készít. Majd ehhez egy általunk értelmezhető Verilog input és output portokkal ellátott modult illeszt, így képesek vagyunk az így kapott szimulációs fájt beilleszteni egy Verilog test fixture-be. A program futtatásához a 7.1 ábrán látható beállításokat alkalmaztam.

```

Command Line: netgen -ofmt verilog -sim -ism -tb -w -dir
D:\NESDev\NES_hw\NES_fpga_hw\src\cpu
D:\NESDev\NES_hw\NES_fpga_hw\src\cpu\nes_cpu6502.ngc nes_cpu_test.v

Reading design 'D:/NESDev/NES_hw/NES_fpga_hw/src/cpu/nes_cpu6502.ngc' ...
Flattening design ...
Processing design ...
    Preping design's networks ...
    Preping design's macros ...
Writing Verilog test file 'D:\NESDev\NES_hw\NES_fpga_hw\src\cpu\nes_cpu_test.tv'
...
Writing Verilog netlist file
'D:\NESDev\NES_hw\NES_fpga_hw\src\cpu\nes_cpu_test.v' ...
INFO:NetListWriters:633 - The generated Verilog netlist contains Xilinx UNISIM
    simulation primitives and has to be used with UNISIM simulation library for
    correct compilation and simulation.
Number of warnings: 0
Number of info messages: 1
Total memory usage is 4395368 kilobytes

Created netgen log file 'D:\NESDev\NES_hw\NES_fpga_hw\src\cpu\nes_cpu_test.nlf'.

D:\Xilinx\14.7\ISE_DS>_

```

**7.1. ábra.** Az UNISIM konzol alkalmazás beállításai

A szimulációk és későbbiekben a hardveres tesztekhez is szükségünk van arra, hogy a CH és Program ROM-jaink helyes adatokat tartalmazzanak. A NES játék kártyák tartalmát általában egy .nes bináris fájl ként tudjuk kinyerni a kártyákból. Ez tartalmaz egy 16 bájtos fejlécet majd a program ROM-ot végül pedig a karakter ROM-ot. A fejléc írja le, hogy az adott játék ROM-jai mekkora területet foglalnak, illetve a játék kártya minden mapper-t tartalmaz. Ezt egy egyszerű program segítségével könnyen felbonthatjuk a két ROM területre, majd egy egyszerű shell script segítségével, szöveges fájlba írhatjuk a kapott binárisaink tartalmát. Majd az így kapott szöveges fájlok tartalmat a \$readmemh parancs segítségével tudunk blokk RAM-okba tölteni. A szimulációk helyes működéséhez érdemes még az összes használt memória területet 0-val feltölteni. Ezeknek az inicializációknak egy részét a 7.1 hardverleírásban olvashatjuk.

```

1 //Character ROM initialization
2 initial begin
3     $readmemh( "src/games/DK_chr_rom_hex.txt" , ch_rom); //Super_Mario_Bros_chr_rom.txt
4 end
5
6 //NT RAM initialization
7 reg [7:0]      name_table_ram [2047:0];
8 integer x;
9 initial begin for (x=0; x<2048; x=x+1) name_table_ram[x] = 8'b0;
10 end

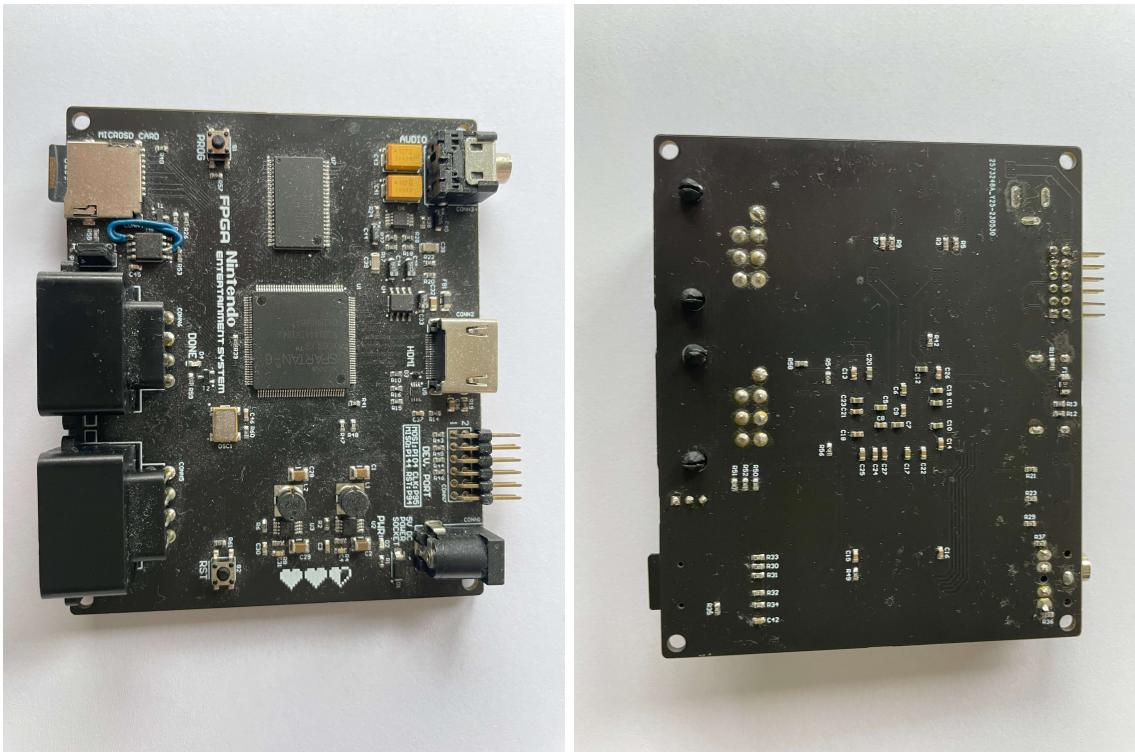
```

**7.1. hardverleírás részlet.** A memória területek inicializálása a szimulációs és hardveres tesztekhez

Sajnos viszont azt tapasztaltam, hogy nagyobb és összetettebb rendszerek szimulációja során az ISim, hosszabb időtartalmú szimulációkra nem alkalmas (nem konzisztensek a számításai). Ezáltal csak egy adott pontig tudtam a szimulációkra hagyatkozni, viszont ez pont elégnek volt ahhoz, hogy könnyen elkezdhessem a teljes rendszer hardveres tesztelését.

## 7.2. Hardveres tesztek

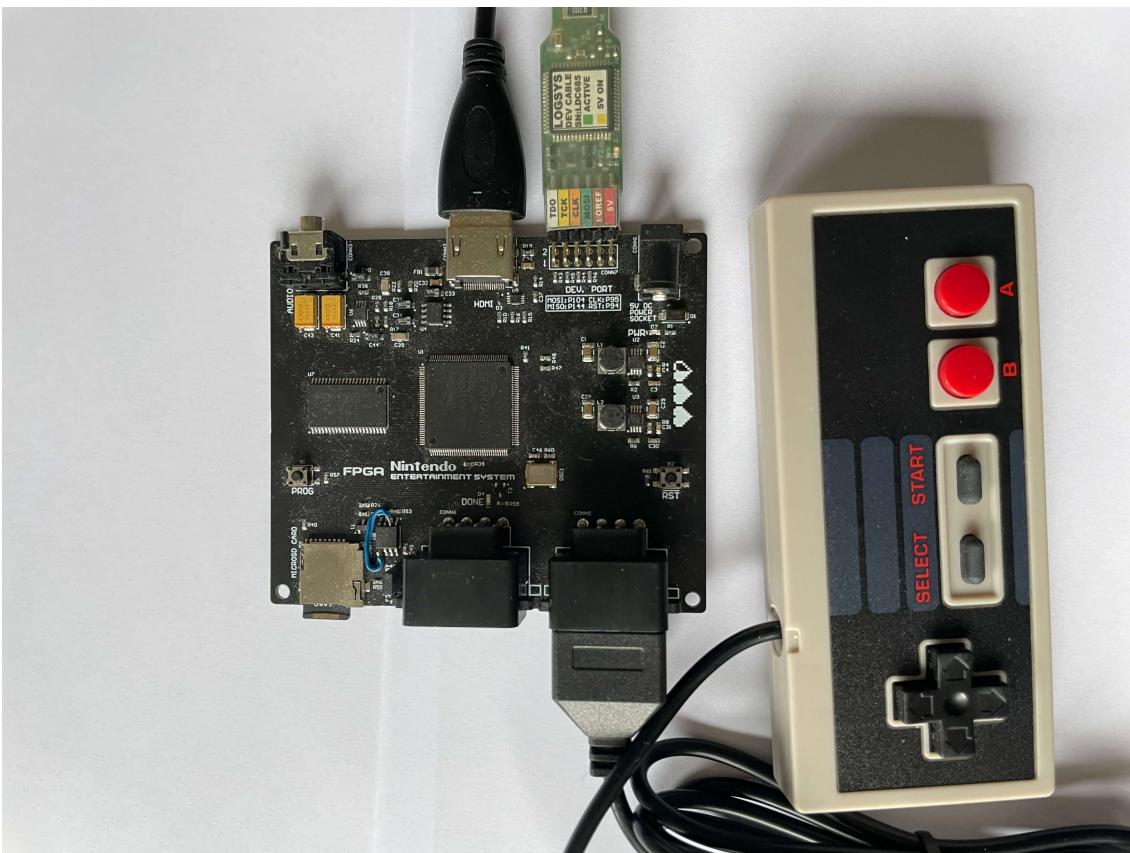
A félév során sikerült elkészíteni, konzulensem segítségével az első verzióját az FPGA NES kártyának, ez a 7.2 képen látható. Az 5 fejezet során viszont már egy javított második verziót mutatok be (ezért láthatunk némi különbséget a két kártya elrendezése között). A második verzióra javítottam a FLASH chip FPGA-be huzalozásán és két extra hangerő szabályzó gombot adtam a nyákhöz.



7.2. ábra. A kész FPGA NES kártya

A hardveres tesztelés során, a hardver binárisának feltöltésére a MIT-en fejlesztett, Logsys FLASH programozót használtam, ennek implementálását már a fentebbiekben olvashattuk (5.10). Az FPGA NES kártya mérési és tesztelési elrendezését a 7.3 ábrán láthatjuk.

A NES rendszer tesztelésére két klasszikus játékot választottam. Az első a Donkey Kong, amely egy könnyebben emulálható 16 kilobájtos program ROM-al rendelkező játék. A második pedig a NES egyik legnehezebben emulálható játéka, a Super Mario Bros. . Ehhez a játékhöz hibátlanul implementált időzítésre és helyes kialakított PPU és CPU memória címekre van szükség. Ezt a játékot szokták a legtöbbször végső teszt kent használni emulált PPU-k és CPU-k tesztelésére. A Donkey Kong tesztelését és futását a 7.4 és 7.5 képeken láthatjuk. A Super Mario Bros.-t pedig futás közben a 7.6 és 7.7 ábrán tekinthetjük meg.



7.3. ábra. FPGA NES kártya mérési és tesztelési elrendezésé



7.4. ábra. A Donkey Kong kezdő képernyője



7.5. ábra. Donkey Kong ikonikus első pálya futás közben



7.6. ábra. A Super Mario Bros. kezdő képernyője



7.7. ábra. A Super Mario Bros. ötödik világának első pályája

### 7.3. A tesztek eredményeinek kiértékelése

A szimulációs tesztek során sikeresen feltudtam a NES rendszerét, olyan szintre, hogy elkezdhettettem a hardveres teszteket. Illetve a későbbiek során is voltak olyan hardveres modulok amelyekben hiba keresés során, többször is vissza nyúltam a szimulátorhoz. Sajnos a ISim hibás működése miatt a teljes rendszert nem tudtam nyomon követni, viszont a szimulációk során megfigyelt hardveres működéseket, a 6 fejezetben több helyen is láthatjuk.

A hardveres tesztelések eredménye is sikeres lett, a két játék teljes mértékben játszató állapotban van. A hosszabb tesztelések során sem akadtam olyan hibára, amely a programok futását megakadályozná és ezzel a játékokat játszhatatlanná tenné. A képalkotásért felelős chip a PPU időzítései is pontosak, hiszen még az idő kritikus játékok Mario is játszható a konzolon ez a 7.7 ábrán is látható, itt még jól megfigyelhető az is, hogy a PPU háttér és sprite alkotási része is összhangban van (egy sűrűbb mozgó elemekkel teli képen sem hibásodik meg a képalkotás). Az általam használt szín paletták, szépen vissza adják az eredeti NES játékok színvilágát, ezzel egy nagyon jó minőségű emulálást lehetővé téve.

Úgy érzem az FPGA-s hardverfejlesztések teszteléseinek széles körű tárházát ismertem meg, és ezen eszközök segítségével sikeresen működésre bírtam az általam fejlesztett konzol hardverét.

## 8. fejezet

# Összefoglalás, jövőbeli tervezek

A diplomatervezés során, sikerült elkészíteni Nintendo Entertainment System (NES) 8 bites videojáték konzol újratervezett alaplapját. Ezt a nyomtatott áramköri lemezt sikerült a félévek során beültetni és tesztelni működés és funkciók szempontjából. A NES kártyából a második félév során készült egy javított verzió, amely kiküszöbölte az első verzió hibáit.

Az FPGA-ba implementált NES hardveréből sikerült elkészíteni a képalkotásért felelős chipet a PPU-t, illetve a működéshez elengedhetetlen kisebb hardveres elemeket, DMA, kontrollervezérlő és memória menedzser. A rendszer processzorát konzulensemtől kaptam a tesztelésekhez. Az általam tesztelt hardveres elemek képesek az eredeti NES-nek megfelelő működésre és a mapper nélküli játékok hibátlan futtatására.

Sajnos az időszűke miatt, nem tudtam a két félév alatt lefejleszteni a saját 6502-es processzoromat és a NES hardverének utolsó chipjét a hang kiadásért felelős APU-t. Viszont, ezzel jövőbeli fejlesztési célokat hagytam a projekt számára.

A rövidtávú fejlesztések közé tartozik, egy nyílt forráskódú 6502-es processzor, illesztése a NES-rendszeréhez, majd az audió kiadásáért felelős chip lefejlesztése az eszközökhez. Ezt követően a NES kártya SRAM-ját fogom a rendszerhez illeszteni, ezzel lehetővé téve a későbbiekbén nagyobb játékok játszását is.

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

Úgy gondolom, hogy a diplomatervezés két féléve alatt rengeteget fejlődtem a nyák tervezés, FPGA fejlesztés és tesztelés területén. Az így megszerzett tudás korszerű és a jövőben széles körben felhasználható. Örülök, hogy egy izgalmas hosszútávú projekt alap pilléreit hozhattam létre a tantárgy keretében.

# Köszönetnyilvánítás

A diplomaterv készítése során nagyban támaszkodhattam mind a tanszéki konzulensem Raikovich Tamás, mind a NESDev wiki közösség tanácsaira. Köszönnettel tartozom a konzulensemnek a támogatásáért, ötleteiért és azért, mert az ő processzora nélkül nem jutott volna el ilyen szintre a projekt. Úgy érzem, hogy nekik köszönhetően tudtam a tőlem telhető maximumot megtenni, a diplomaterv projekt elkészítése során.

# Ábrák jegyzéke

3.1.	Nintendo Entertainment System [5] . . . . .	3
3.2.	Nintendo Entertainment System NTSC alaplap [16] . . . . .	4
3.3.	Nintendo Entertainment System játék kazetta [14] . . . . .	5
3.4.	Komponensek közti kapcsolat (fent játék kazetta) [16] . . . . .	6
3.5.	Katódsugárcsöves TV-k működése [3] . . . . .	8
3.6.	Super Mario Bros. Pattern táblái [2] . . . . .	9
3.7.	Gumba (Mario egyik ellenfele) bal felső csempe elem szín adatai [15] . . . . .	9
3.8.	A MOS 6502 egyedi nyolcbites architektúrája [23] . . . . .	13
3.9.	A MOS 6502 processzor státusz regisztere (P) [4] . . . . .	14
3.10.	A 2A03-ban található 6502 teljes utasítás készlete [12] . . . . .	15
3.11.	Az audió feldolgozó egység hang sávjai [9] . . . . .	15
4.1.	A NES játék konzol dimenziói [6] . . . . .	20
5.1.	NES kártya blokkdiagramja . . . . .	21
5.2.	NES audió jelért felelős áramkörök . . . . .	24
5.3.	aljzat anya [10] . . . . .	25
5.4.	NES kontroller csatlakozó [20] . . . . .	26
5.5.	Az FPGA NES kártya réteg beállításai . . . . .	28
5.6.	A top layer alkatrész elhelyezési terve . . . . .	28
5.7.	Dual Strip Coplanar Waweguide Grounded felépítése [8] . . . . .	30
5.8.	Coplanar differential pair routings Altium beállításai . . . . .	30
5.9.	A HDMI adatvonalainak bekötése . . . . .	31
5.10.	FPGA tápellátásának kialakítása . . . . .	31
5.11.	3D PCB rajzolat . . . . .	32
6.1.	Megvalósult FPGA projekt (nes_top) működési diagramja . . . . .	33
6.2.	MOS 6502 órajel generálása (FPGA sim) . . . . .	36
6.3.	CPU-ból ékrekző VRAM írási és olvasási jelek PPU időzítéséhez igazító állapotgép . . . . .	37
6.4.	Háttér képalkotás és memória olvasást vezérlő állapotgép . . . . .	38
6.5.	Sprite képalkotás és memória olvasást vezérlő állapotgép . . . . .	42
6.6.	VRAM cím regiszter értékadása (eredeti PPU) . . . . .	44
6.7.	A NES PPU-jának pixel feldolgozó egysége . . . . .	46
6.8.	VGA top modul blokkvázlata . . . . .	48
6.9.	OAM DMA működési állapotgépe . . . . .	51
6.10.	A DMA ciklus kezdete szimulációban . . . . .	52
6.11.	Teljes DMA ciklus szimuláció . . . . .	52
6.12.	A CPU szimulációja működés közben, Super Mario Bros. első assembly kódjának olvasása . . . . .	53
7.1.	Az UNISIM konzol alkalmazás beállításai . . . . .	55

7.2.	A kész FPGA NES kártya	56
7.3.	FPGA NES kártya mérési és tesztelési elrendezésé	57
7.4.	A Donkey Kong kezdő képernyője	57
7.5.	Donkey Kong ikonikus első pálya futás közben	58
7.6.	A Super Mario Bros. kezdő képernyője	58
7.7.	A Super Mario Bros. ötödik világának első pályája	59

# Táblázatok jegyzéke

3.1. A PPU memória kezelése (14 bit címek) mirroring jelenséggel . . . . .	10
3.2. A CPU memória címtartománya (16 bit címek) mirroring jelenséggel . . . . .	15
5.1. HDMI lánckiosztás . . . . .	25
5.2. Fejlesztői port bekötése . . . . .	27

# Irodalomjegyzék

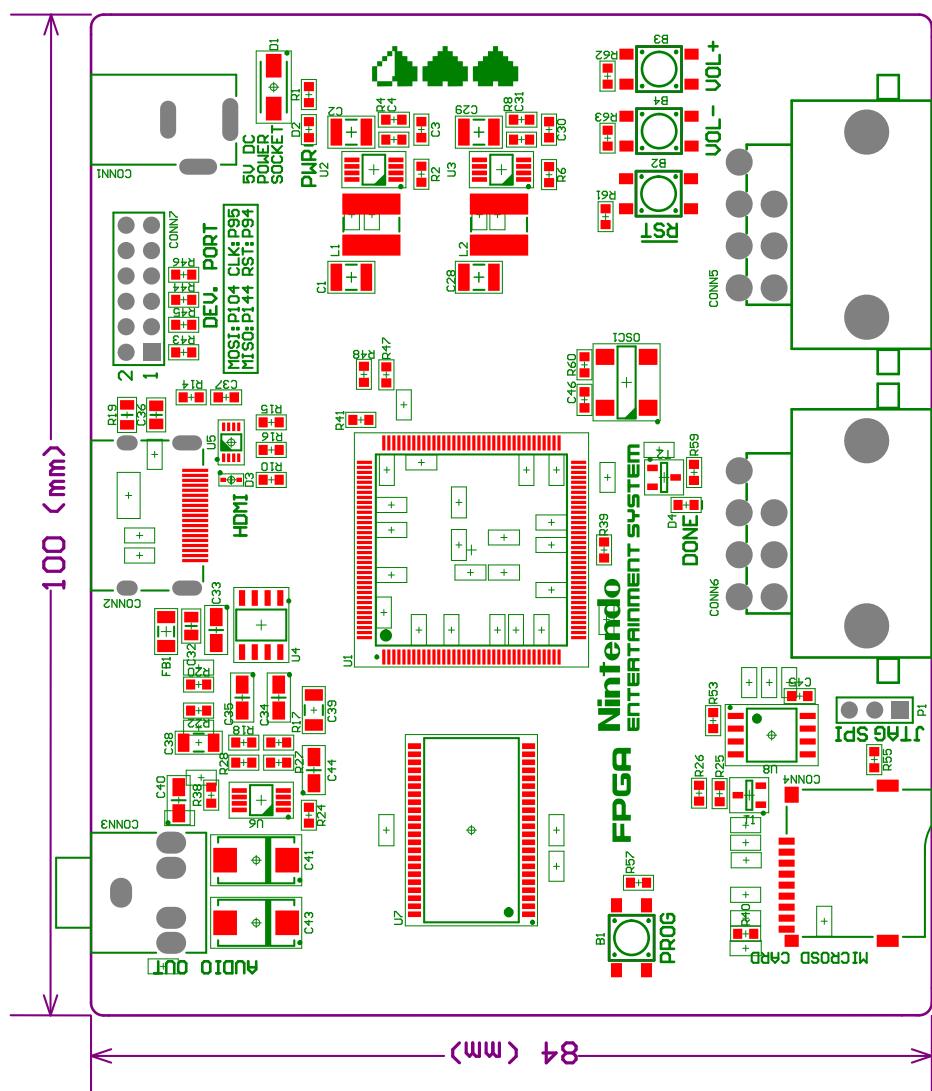
- [1] Nathan Altice: *I Am Error - The Nintendo Family Computer / Entertainment System Platform*. September 8, 2017, The MIT Press.
- [2] beta knowledge: The pattern tables of super mario brothers (2023. december 13.). <https://betaknowledge.nearfuturelaboratory.com/post/33657221281/the-pattern-tables-of-super-mario-brothers-this>.
- [3] Britannica: Cathode-ray tube (2023. december 13.). <https://www.britannica.com/technology/cathode-ray-tube>.
- [4] Christiaan008: 27c3: Reverse engineering the mos 6502 cpu (en) (2023. december 13.). [https://www.youtube.com/watch?v=fWqBmmPQP40&t=1777s&ab\\_channel=Christiaan008](https://www.youtube.com/watch?v=fWqBmmPQP40&t=1777s&ab_channel=Christiaan008).
- [5] Despota: Nintendo entertainment system (nes) (2023. december 13.). [https://retrofilia.blog.hu/2015/02/14/nintendo\\_entertainment\\_system\\_nes](https://retrofilia.blog.hu/2015/02/14/nintendo_entertainment_system_nes).
- [6] Dimensions.com: Nintendo entertainment system (nes) (2023. december 13.). <https://www.dimensions.com/element/nintendo-entertainment-system-nes>.
- [7] JLCPBC: *PCB Manufacturing and Assembly Capabilities*. JLCPBC, 2023. 06. <https://jlcpbc.com/capabilities/pcb-capabilities>.
- [8] Kyle Handfield: What is coplanar waveguide pcb fabrication (2023. december 13.). <https://wellerpcb.com/coplanar-waveguide-pcb/>.
- [9] KYLXBN (Kyle): How i created the perfect nes sound chip (2023. december 13.). [https://www.youtube.com/watch?v=8RrQrATnXXY&t=49s&ab\\_channel=KYLXBN%28Kyle%29](https://www.youtube.com/watch?v=8RrQrATnXXY&t=49s&ab_channel=KYLXBN%28Kyle%29).
- [10] madpcb's admin: Hdmi (2023. december 13.). <https://madpcb.com/glossary/hdmi/>.
- [11] MiSTer: Nintendo entertainment system for mister platform (2023. december 07.). [https://github.com/MiSTER-devel/NES\\_MiSTER/tree/master/rtl/t65](https://github.com/MiSTER-devel/NES_MiSTER/tree/master/rtl/t65).
- [12] NESDev wiki community: Cpu unofficial opcodes (2023. december 13.). [https://www.nesdev.org/wiki/CPU\\_unofficial\\_opcodes](https://www.nesdev.org/wiki/CPU_unofficial_opcodes).
- [13] NESDev wiki community: Nes reference guide (2023. november 26.). [https://www.nesdev.org/wiki/NES\\_reference\\_guide](https://www.nesdev.org/wiki/NES_reference_guide).
- [14] NesHacker youtube csatorna: Nes carts explained (2023. október 21.). <https://www.youtube.com/watch?v=GssRNEaKoPw>.

- [15] NesHacker youtube csatorna: Nes graphics explained (2023. október 21.). [https://www.youtube.com/watch?v=7Co\\_8dC2zb8&ab\\_channel=NesHacker](https://www.youtube.com/watch?v=7Co_8dC2zb8&ab_channel=NesHacker).
- [16] NesHacker youtube csatorna: Nes hardware explained (2023. október 21.). <https://www.youtube.com/watch?v=mMq4FFUnBPC>.
- [17] NesHacker youtube csatorna: Neshacker (2023. november 26.). <https://www.youtube.com/@NesHacker>.
- [18] OpenCore weboldal: Nyílt forráskódú hardverleírások (2023. december 07.). <https://opencores.org/projects>.
- [19] 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](https://datasheet.lcsc.com/lcsc/1804240720_Cypress-Semicon-CY7C1041DV33-10ZSXI_C32338.pdf).
- [20] raphnet-tech.com: Nes controller connector (2023. december 13.). [https://www.raphnet-tech.com/products/nes\\_controller\\_connector/index.php](https://www.raphnet-tech.com/products/nes_controller_connector/index.php).
- [21] Raikovich Tamás: *DIGITÁLIS VIDEO INTERFÉSZ MEGVALÓSÍTÁSA A LOGSYS KINTEX-7 FPGA KÁRTYÁVAL.* Budapesti Műszaki és Gazdaságtudományi Egyetem, 2023. 12. [https://home.mit.bme.hu/~rtamas/Logsys/Dokumentumok/Kintex7/LOGSYS\\_Kintex7\\_HDMI\\_ado.pdf](https://home.mit.bme.hu/~rtamas/Logsys/Dokumentumok/Kintex7/LOGSYS_Kintex7_HDMI_ado.pdf).
- [22] Raikovich Tamás: *LOGSYS SPARTAN-6 FPGA KÁRTYA (V2.1) FELHASZNÁLÓI ÚTMUTATÓ.* Budapesti Műszaki és Gazdaságtudományi Egyetem, 2023. 06. [http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS\\_SP6\\_FPGA\\_Board.pdf](http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_SP6_FPGA_Board.pdf).
- [23] Xor: 6502 colorized block diagram (2023. december 13.). <http://forum.6502.org/viewtopic.php?t=1744>.

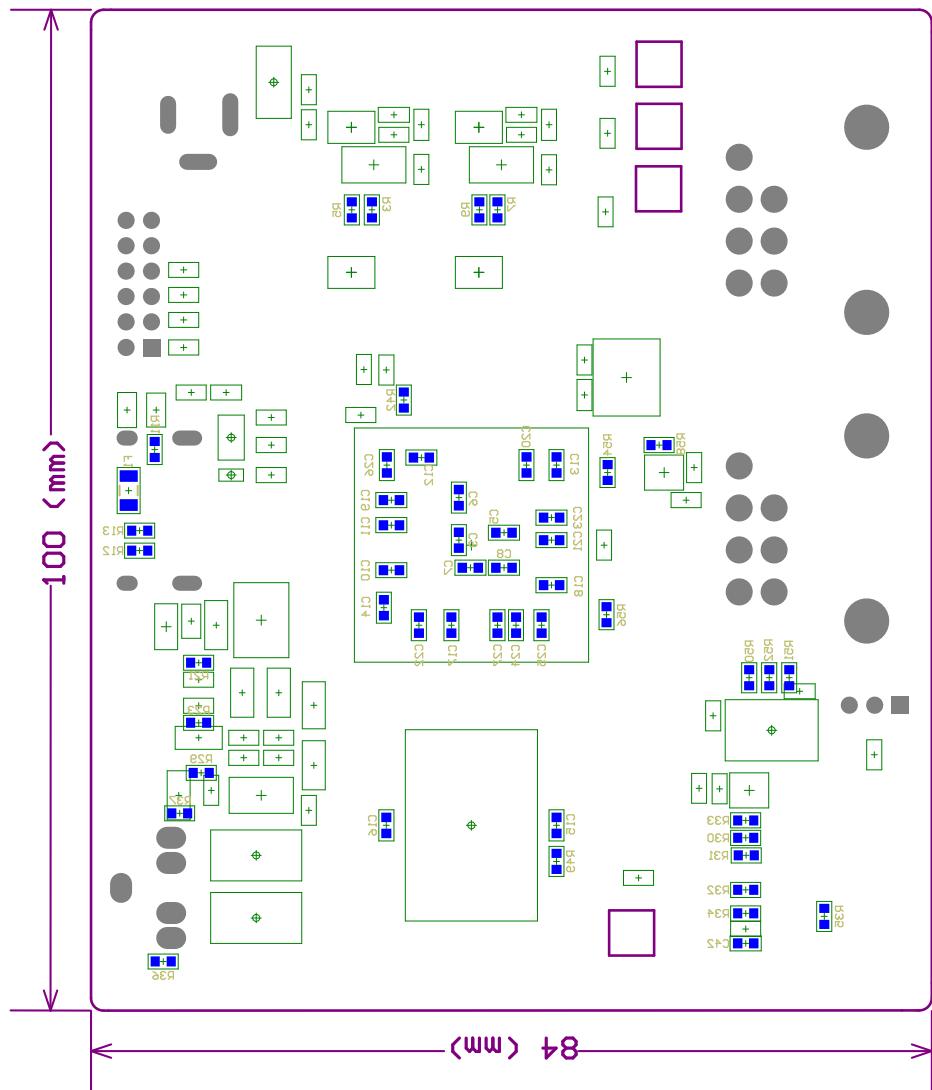
# Függelék

## F.1. NES kártya alkatrész elhelyezési terve

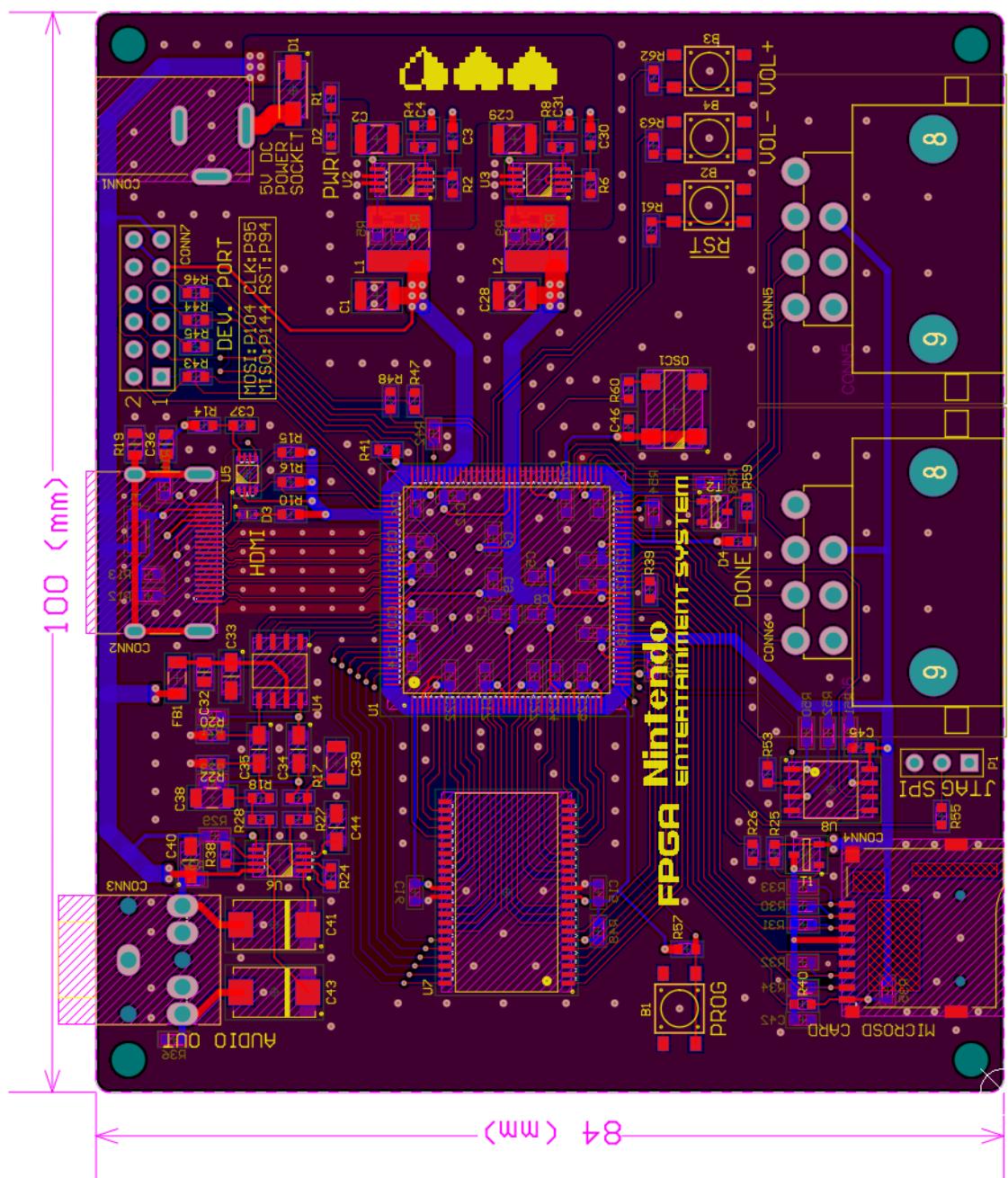
#### F.1.1. Top



### F.1.2. Bottom

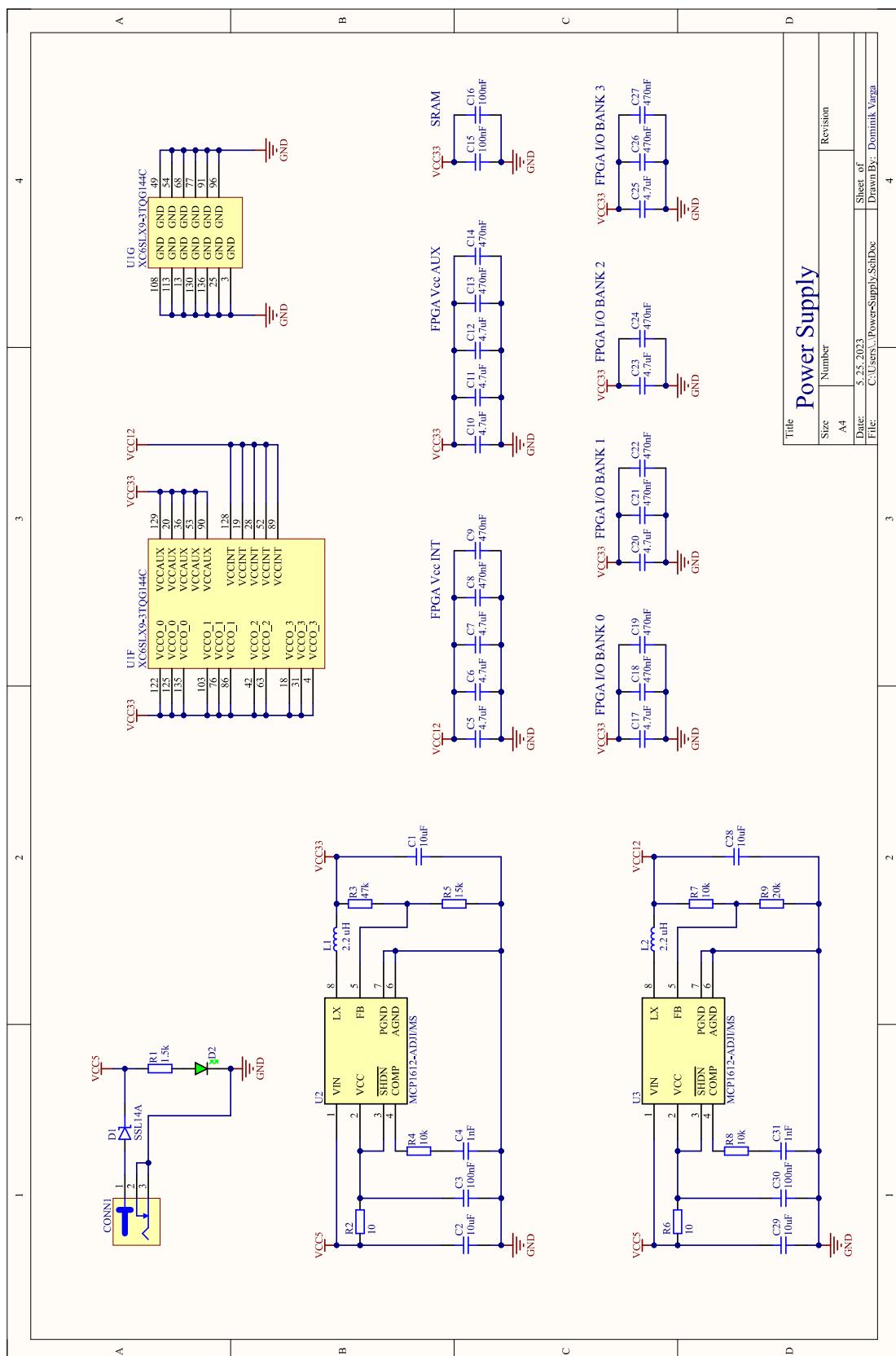


F.2. Nyomtatott áramköri terve (2D transparent)

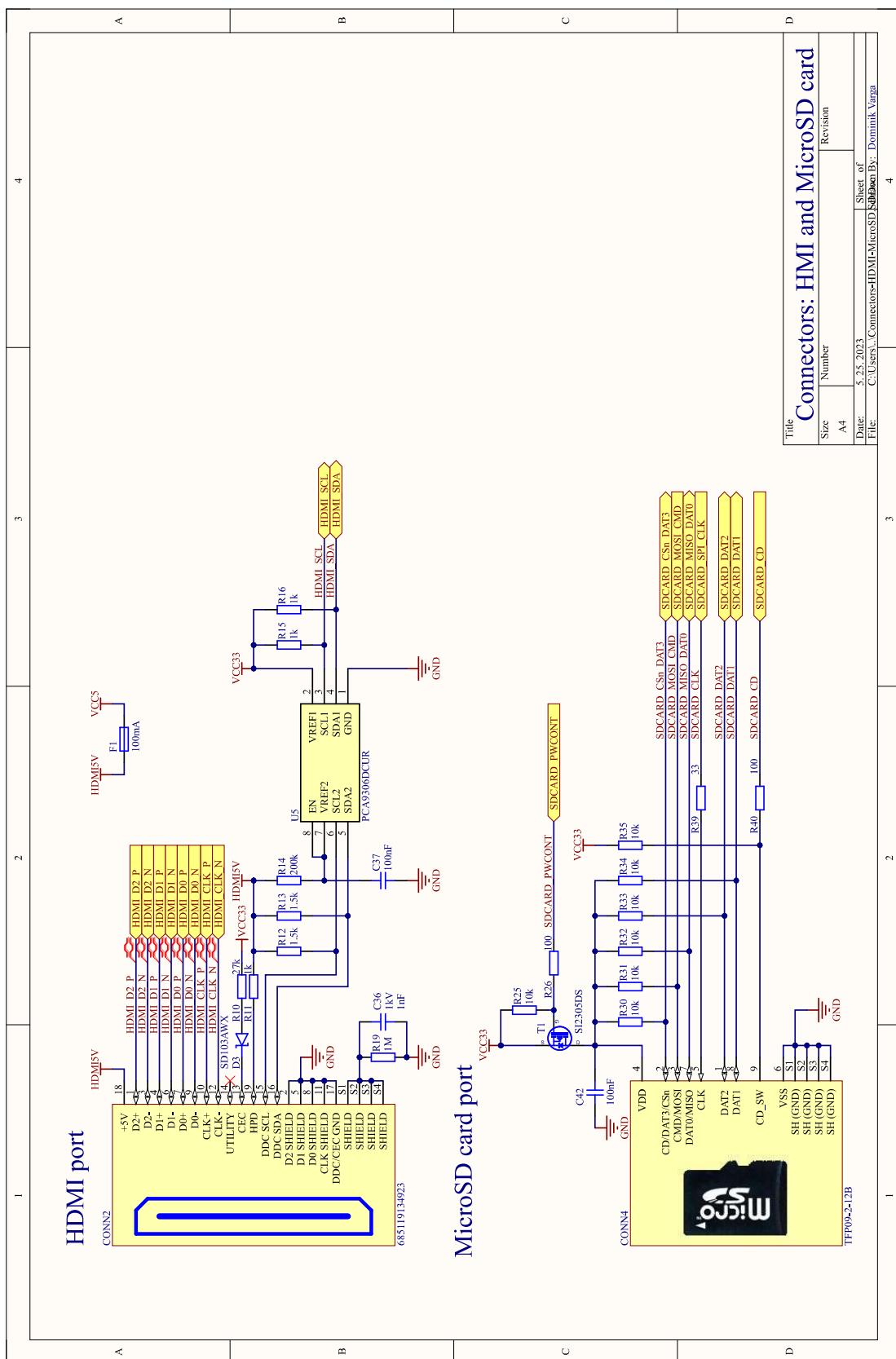


## F.3. FPGA NES kártya kapcsolási rajza

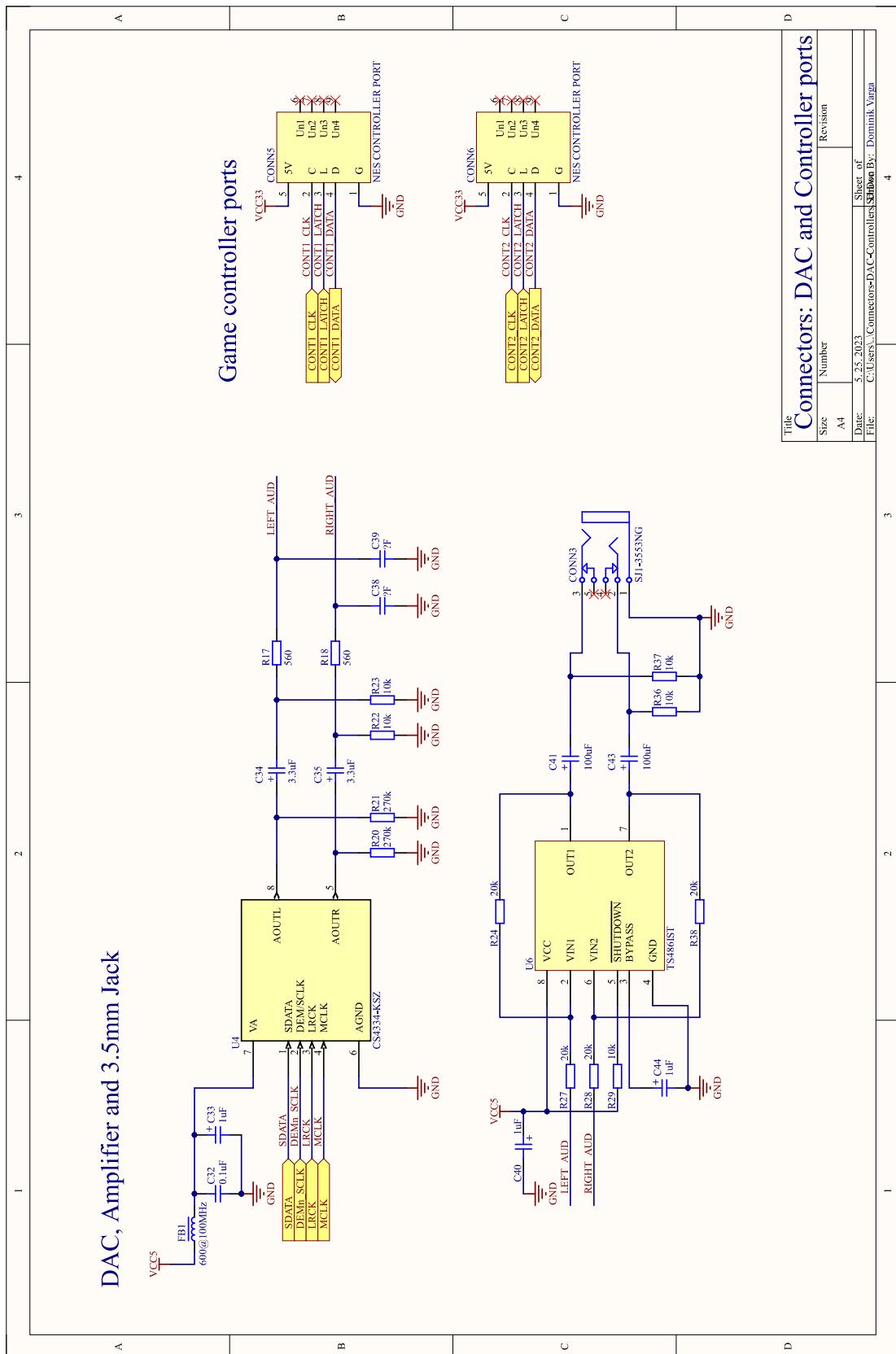
### F.3.1. Tápegység



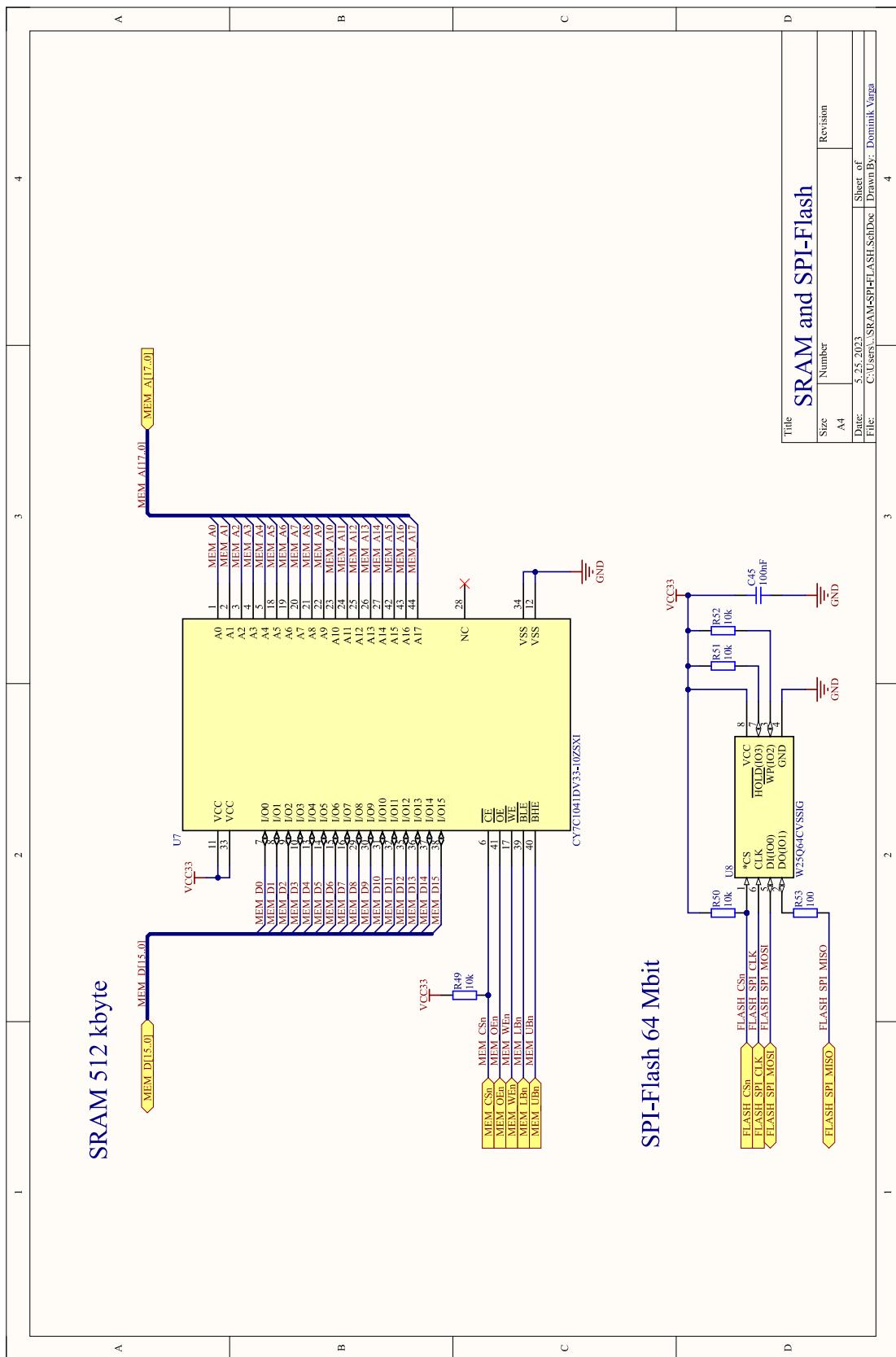
### F.3.2. HDMI és MicroSD kártya csatlakozó



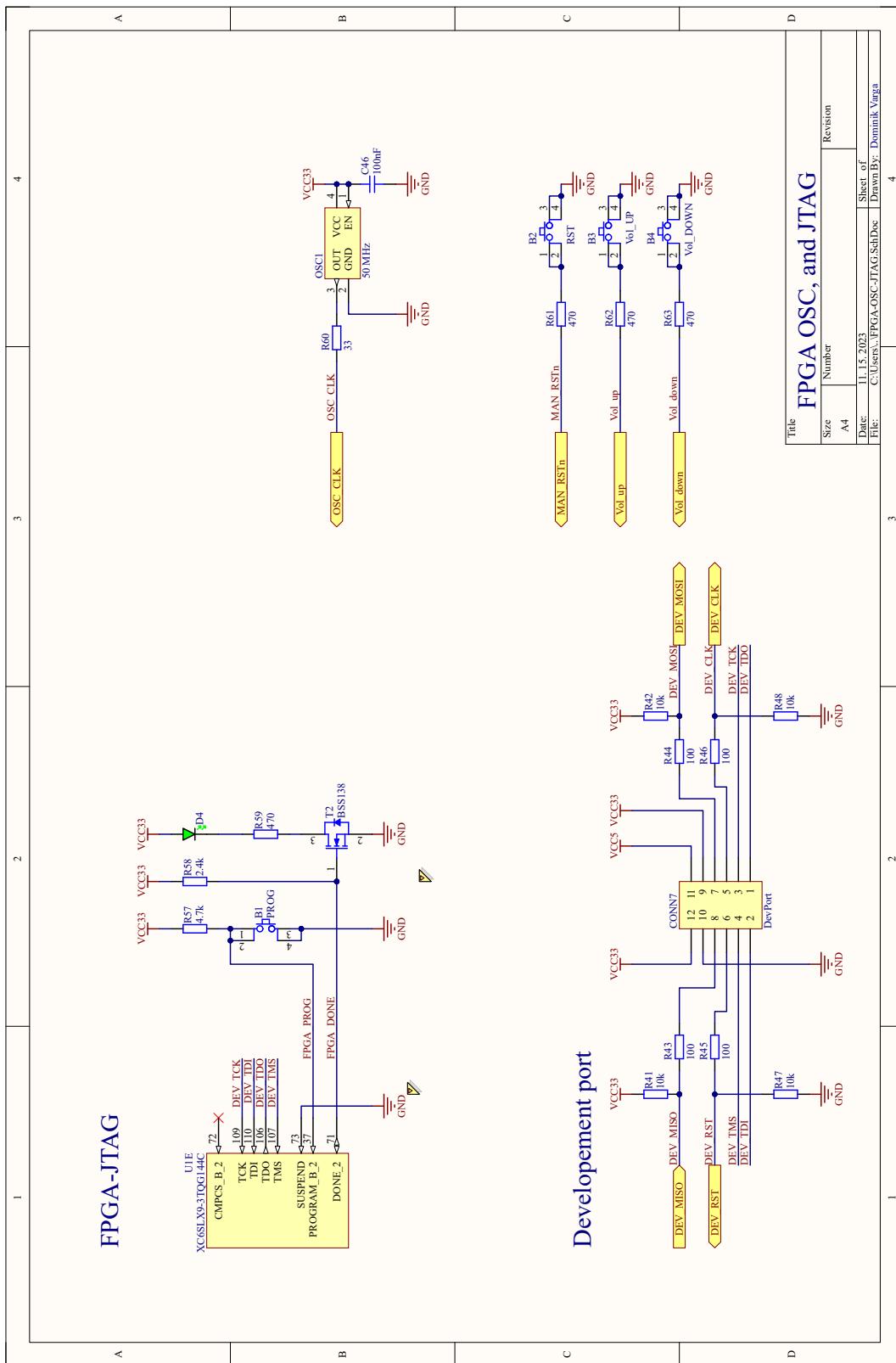
### F.3.3. DAC, erősítő és kontroller áramkörök



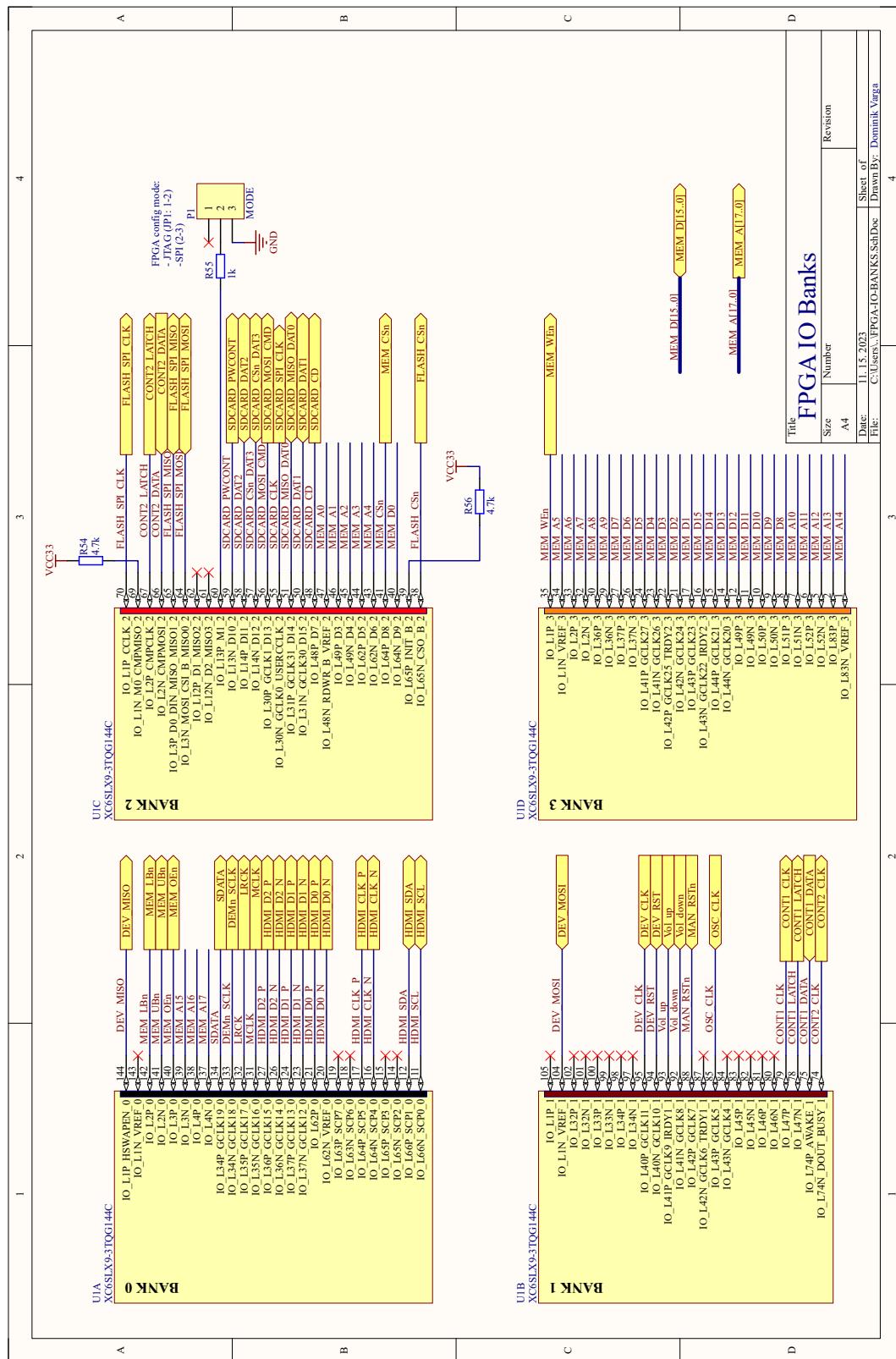
### F.3.4. SRAM és SPI-Flash



### F.3.5. FPGA OSC és JTAG



### F.3.6. FPGA IO bankok



## F.4. NTSC PPU eredeti képalkotás

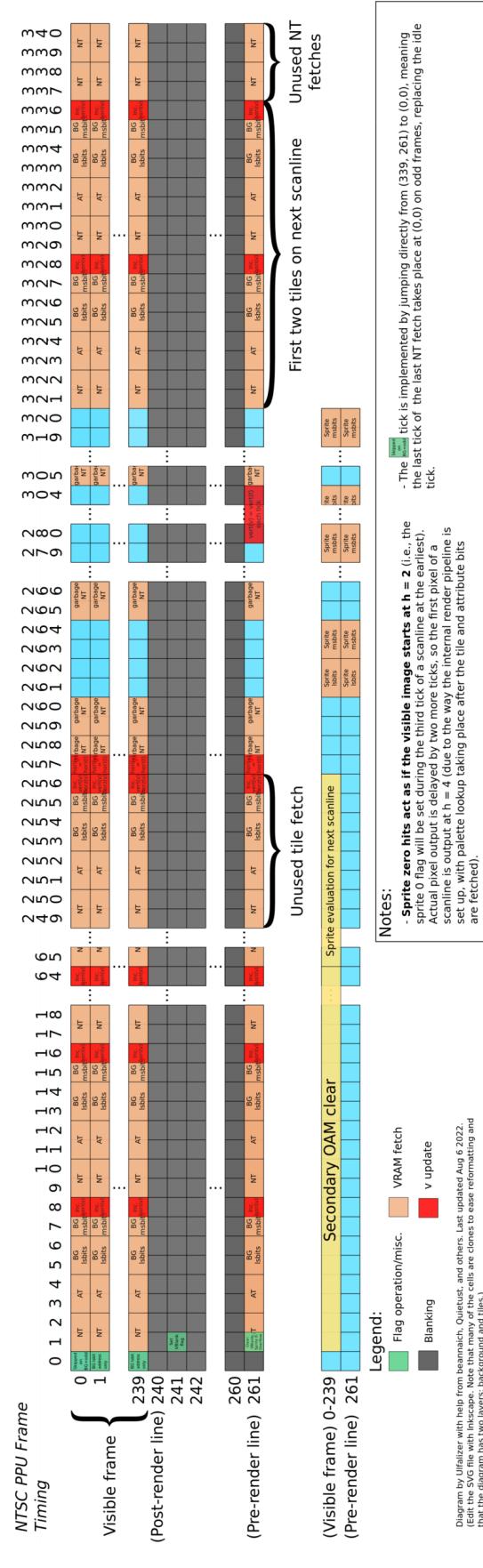


Diagram by Ulfalizer with help from beamnach, Quietut, and others. Last updated Aug 6 2022.  
(edit the SVG file with Inkscape. Note that many of the cells are clones to ease reformatting and that the diagram has two layers: background and tiles.)