UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ și CALCULATOARE
DOMENIUL: CALCULATOARE și TEHNOLOGII INFORMAȚIONALE
SPECIALIZAREA: CALCULATOARE

Emulator pentru Nintendo Entertainment System

LUCRARE DE LICENȚĂ

Coordonator științific:
Conf.dr.ing Andrei Stan

Absolvent:
Rotaru Cristian

Iași 2019

DECLARAŢIE DE ASUMARE A AUTENTICITĂŢII
LUCRĂRII DE LICENȚĂ

Subsemnatul _____,

legitimat cu ____ seria _____ nr. _____ , CNP_____

autorul lucrării _____

_____

_____

elaborată în vederea susținerii examenului de finalizare a studiilor de _____ organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea _____ a anului universitar _____, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data Semnătura

Cuprins

Introducere

Scopul acestui proiect este crearea unui program (emulator) care să permită rularea jocurilor create pentru consola de jocuri **NINTENDO ENTERTAINMENT SYSTEM** (NES).

NINTENDO ENTERTAINMENT SYSTEM este o consolă de jocuri bazată pe procesorul **MOS6502**. Aceasta a fost lansată pentru prima dată în anul 1983 în Japonia, urmând să fie lansată ulterior în SUA și Europa. Consola **NES** a avut un succes foarte mare, fiind, până în prezent, una dintre cele mai bine vândute console din lume, cu peste 60 de milioane de unități vândute.

Pentru această platformă au fost create peste 700 de jocuri (lansate oficial), unele dintre ele fiind considerate legendare: „Super Mario Bros", „Donkey Kong", „Legend of Zelda", „Tetris", „PAC-MAN", etc.

Deși consola nu mai este disponibilă pe piață, unii oameni ar dori să aibă posibilitatea de a juca jocurile create pentru această platformă. Această problemă poate fi soluționată prin crearea unui emulator, care să simuleze comportamentul hardware al consolei, permițând astfel, rularea jocurilor pe alte platforme.

Programul creat în cadrul acestui proiect suportă un număr mare de jocuri create pentru consola **NES**.

Abrevieri

AD – Audio Device
ALU – Unitate Aritmetico-Logică
APU – Audio Processing Unit
ASIC – Application Specific Integrated Circuit
BCD – Binary-Coded Decimal
CIC – Checking Integrated Circuit
CISC – Complex Instruction-Set Computing
CHR – Character
CPU – Central Processing Unit
CR – Cartridge Reader
CS – Chip Select
CVBS – Composite Video Baseband Signal
DMC – Delta Modulation Channel
GC – Game Controller
MB – Memory Bus
MM – Memory Mapper
NES – Nintendo Entertainment System
NTSC – National Television System Committe
OAM – Object Attribute Memory
OPC – Operation Code
PAL – Phase Alternating Line
PPU – Picture Processing Unit
PRG – Program
RAM – Random Access Memory
R/W – Read / Write
RO – Read Only
RW – Rendering Window
WO – Write Only

Capitolul 1: Structura hardware NES

Componentele principale ale unui sistem **NES** sunt: unitatea de procesare centrală (**CPU**) și audio (**APU**) încorporate în cipul **Ricoh RP2A03** (sau **RP2A07**, în dependență de regiune), unitatea de procesare grafică (**PPU**), memoria **RAM**, memoria **V-RAM** și cipul de securitate (**CIC**). Alte componente importante sunt plasate pe cartridge: memoria de program (**PRG**), memoria de elemente grafice (**CHR**), **RAM** extern (opțional), mapper de memorie (opțional) și un cip de securitate (**CIC**).

La consolă trebuie conectate 1 sau 2 controlere și un televizor ce suportă intrări **CVBS** și audio.

Schema funcțională a sistemului este descrisă în *figura 1.1*.

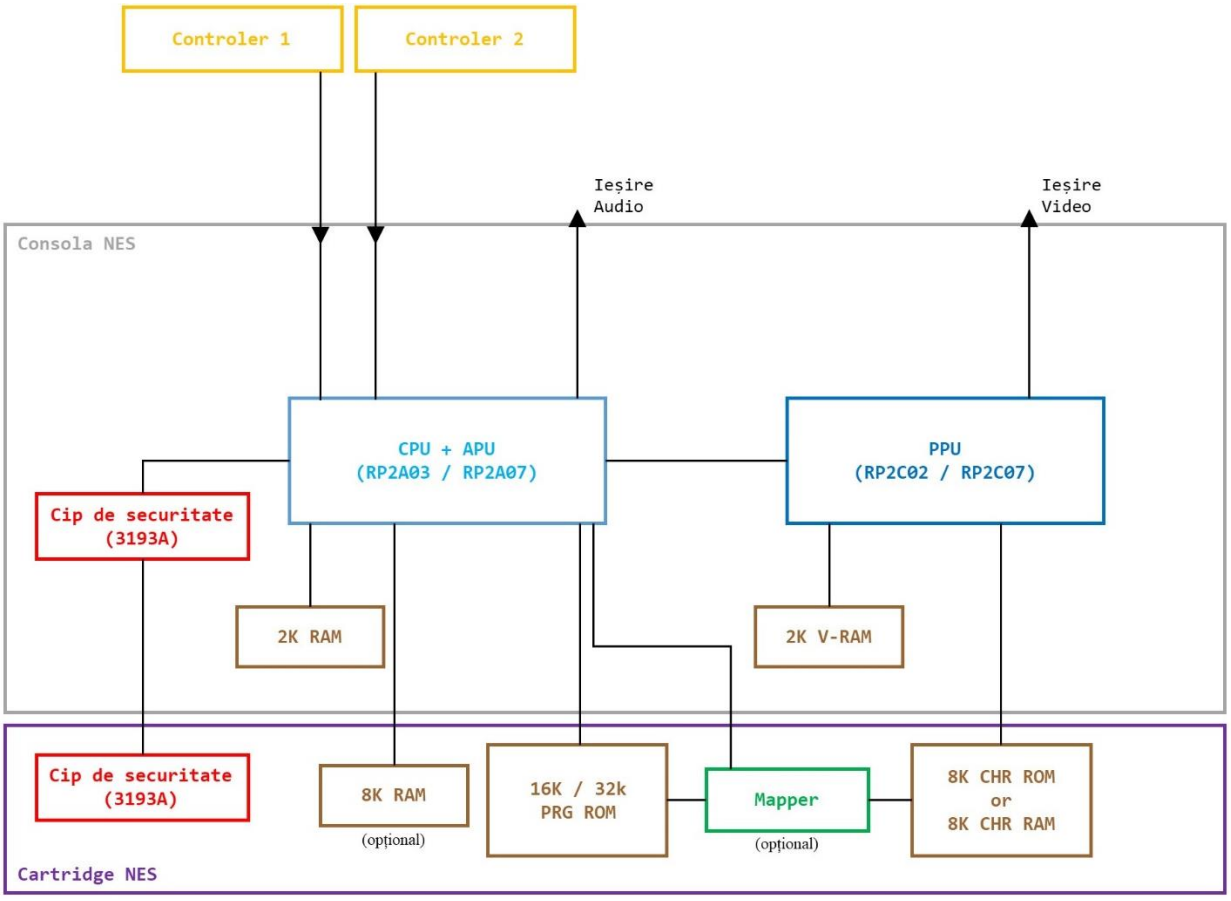


figura 1.1 (Schema funcțională a sistemului NES)

§1.1 Unitatea centrală de procesare (CPU)

Nucleul de procesare a consolei **NES** este bazat pe procesorul **MOS6502**, diferența dintre acestea fiind lipsa modului **BCD** la instrucțiunile de adunare și scădere. **MOS6502** este un microprocesor de tip **CISC** pe 8 biți cu un bus de adrese pe 16 biți, capabil să lucreze la frecvențe de până la 3MHz.

Unitatea de procesare este integrată în capsula **Ricoh RP2A03** (**RP2A07**), care conține și procesorul audio.

§1.1.1 Descrierea pinout a procesorului

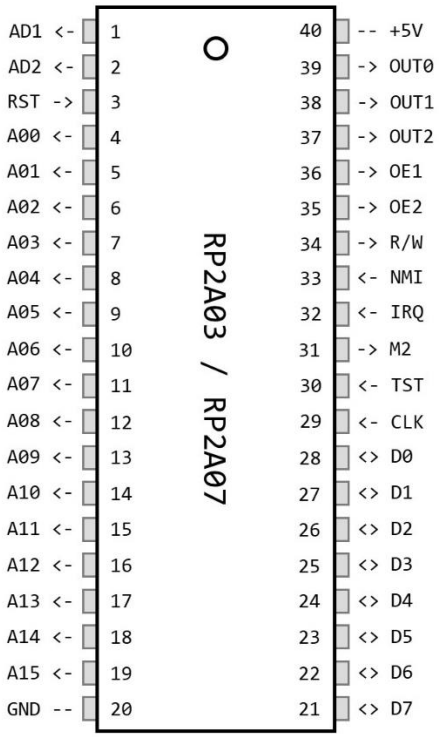


figura 1.2 (RP2A03 / RP2A07 pinout)

[AD1 / AD2] – Ieșiri audio (vezi *§1.3*)

RST – Pin de reset activ pe 0

[A00:A15] – Bus de adrese. Menține adresa țintă pe parcursul unui ciclu de citire/scriere.

[D0:D7] – Bus de date. La citire, datele sunt citite de pe acești pini. La scriere, datele sunt plasate pe acești pini.

CLK – Intrare de clock (vezi *§1.6*)

TST – Mod de test activ pe 1. La activare, regiștrii de test devin accesibili programelor ce rulează.

M2 – „signals ready”. Semnalează perifericelor că datele sunt disponibile pe **[D0:D7]**.

IRQ – Pin de întrerupere activ pe palierul negativ. (vezi *§1.1.5*)

NMI – Pin de întrerupere nemascabilă activ pe frontul negativ. (vezi *§1.1.5*)

R/W – Indică dacă operația este de scriere sau de citire. 1 = citire, 0 = scriere.

[OE1 / OE2] – Activează/dezactivează semnalele de output pentru controlerele de joc.

[OUT0:OUT2] – Portul de expansiune al procesorului (registrul $4016).

§1.1.2 Regiștrii procesorului

Nucleul de procesare conține un registru acumulator (**A**), 2 regiștri de indexare (**X**, **Y**), un program counter (**PC**), un stack pointer (**SP**) și un registru de flag-uri (**P**). Regiștrii **A**, **X**, **Y**, **SP** și **P** sunt pe 8 biți, iar registrul **PC** este pe 16 biți.

Registrul acumulator este folosit de unitatea aritmetico-logică pentru stocarea rezultatelor operațiilor în modul de adresare **accumulator** (vezi *§1.1.3*).

Regiștrii **X** și **Y** sunt folosiți în anumite moduri de adresare ale instrucțiunilor. Aceștia pot fi folosiți pentru numărarea iterațiilor în bucle sau la accesul la blocuri de memorie alocate continuu.

Registrul **SP** este folosit pentru urmărirea vârfului stivei la operațiunile **push** și **pop**. Adresa din memorie a vârfului stivei este calculat astfel: `$(SP) | 0x0100`.

Registrul **PC** indică adresa instrucțiunii curente ce trebuie executată.

Registrul de flag-uri conține informații despre starea procesorului. Acesta poate fi modificat de unitatea aritmetico-logică și de sistemul de întreruperi. Valorile anumitor biți din acest registru determină rezultatul instrucțiunilor de salt condiționat (vezi *§1.1.4*).

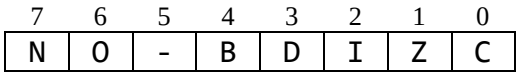


figura 1.3 (Registrul de flag-uri)

C(carry) – este setat dacă ultima operațiune **ALU** a efectuat transport către exterior.
Z(zero) – este setat dacă ultima operațiune **ALU** a returnat 0.
I(inhibit) – este setat de fiecare dată când are loc o întrerupere. Dacă este setat, întreruperile **IRQ** sunt inhibate (vezi *§1.1.5*).
D(decimal) – dacă este setat, operațiile de adunare și scădere rulează în modul **BCD**. (nu se aplică în cazul procesorului din consola **NES**)
B(break) – este setat la sfârșitul unei întreruperi de tip **BRK** (vezi *§1.1.5*).
O(overflow) – este setat de **ALU** în cazul unei erori de overflow.
N(negative) – este setat dacă ultima operațiune **ALU** a returnat un număr negativ.

§1.1.3 Moduri de adresare

Procesorul **MOS6502** are un spațiu de adrese pe 16 biți (`$0000-$FFFF`). Maparea memorie este descrisă în *tabelul 1.1*.

Adresa de început	Adresa de sfârșit	Dimensiunea	Descriere
`$0000`	`$07FF`	`2KB`	**RAM** intern
`$0800`	`$1FFF`	`6KB`	Pointează spre spațiul `$0000-$07FF` (se repetă la fiecare 2KB)
`$2000`	`$2007`	`8B`	Regiștri **PPU**
`$2008`	`$3FFF`	`8KB - 8B`	Pointează spre regiștrii **PPU** (se repetă la fiecare 8B)
`$4000`	`$401F`	`32B`	Regiștri **APU** și **I/O**
`$4020`	`$5FFF`	`8160B`	Spațiu nefolosit
`$6000`	`$7FFF`	`8K`	**RAM** extern
`$8000`	`$FFFF`	`32K`	**PRG ROM**

tabelul 1.1 (Spațiul de adrese al procesorului)

Vectorii de întreruperi și de reset se găsesc la adresele: (vezi *§1.1.5*)

- `$FFFA-$FFFB` – vectorul **NMI**
- `$FFFC-$FFFD` – vectorul de reset
- `$FFFE-$FFFF` – vectorul **BRK** / **IRQ**

Spațiul de memorie este împărți în pagini de 256B (mapabile pe 8 biți). Dacă în timpul unei operațiuni de citire pentru încărcarea unui registru de 16 biți (vezi *§1.1.2*) este necesară schimbarea paginii, operațiunea va dura cu un ciclu mai mult.

Instrucțiunile procesorului pot accesa memoria prin intermediul a 13 moduri de adresare descrise în *tabelul 1.2*.

Denumire	Asamblare	Descriere
Implicit	`OPC`	Instrucțiunea nu are operand. Destinația rezultatului este implicită.
Accumulator	`OPC A`	Operația este executată asupra registrului **A**.
Immediate	`OPC #$BB`	Operandul sursă este o valoare imediată pe 8 biți. Rezultatul se scrie în unul din registrele procesorului în dependență de instrucțiune.
Zero Page	`OPC $LL`	Operandul este luat din pagina 0 a memoriei **RAM** de la o adresă dată pe 8 biți.
Absolute	`OPC $LLHH`	Operandul este luat de la o adresă dată pe 16 biți.
Relative	`OPC $BB`	Este folosit de instrucțiunile de salt. Operandul reprezintă un offset, valoare pe 8 biți cu semn.
Indirect	`OPC ($LLHH)`	Este folosit de o instrucțiune de salt necondiționat. Operandul reprezintă o adresă pe 16 biți la care se găsește valoarea ce urmează a fi încărcată în registrul **PC**.
Zero Page indexed by X	`OPC $LL,X`	Operandul se află la adresa `($LL + X) & 0xFF`.
Zero Page indexed by Y	`OPC $LL,Y`	Operandul se află la adresa `($LL + Y) & 0xFF`.
Absolute indexed by X	`OPC $LLHH,X`	Operandul se află la adresa `$LLHH + X`.
Absolute indexed by Y	`OPC $LLHH,Y`	Operandul se află la adresa `$LLHH + Y`.

Indexed indirect	`OPC ($LL,X)`	Operandul se obține în 3 pași:
		`baseAddr=read($LL)+X`
		`addr=read(baseAddr)|read((baseAddr+1)&0xFF)<<8`
		`operand=read(addr)`

Indirect indexed	`OPC ($LL),Y`	Operandul se obține în 3 pași:
		`baseAddr=read($LL)`
		`addr=(read(baseAddr)|(read(baseAddr+1)&0xFF)<<8)+Y`
		`operand=read(addr)`

tabelul 1.2 (Moduri de adresare)

§1.1.4 Setul de instrucțiuni

Setul oficial de instrucțiuni al procesorului **MOS6502** este constituit din 151 de instrucțiuni codificate pe 8 biți. Fiecare instrucțiune poate fi urmată de un parametru pe 8 sau 16 biți. Întreg setul este descris în *tabelul 1.3*.

Opcode	Adresare	Asamblare	Cod	Descriere
`BRK`	Implicit	`BRK`	`00`	Cauzează o întrerupere de tip **BRK**. (vezi *§1.1.5*)
`PHP`	Implicit	`PHP`	`08`	Pune pe stivă registrul de flag-uri.
`PLP`	Implicit	`PLP`	`28`	Ia de pe stivă registrul de flag-uri.
`PHA`	Implicit	`PHA`	`48`	Pune pe stivă registrul acumulator (**A**).
`PLA`	Implicit	`PLA`	`68`	Ia de pe stivă registrul acumulator (**A**).

CLC	Implicit	CLC	18	Setează flag-ul *carry* (**C**) pe 0.
SEC	Implicit	SEC	38	Setează flag-ul *carry* (**C**) pe 1.
CLI	Implicit	CLI	58	Setează flag-ul *inhibit* (**I**) pe 0.
SEI	Implicit	SEI	78	Setează flag-ul *inhibit* (**I**) pe 1.
CLD	Implicit	CLD	D8	Setează flag-ul *decimal* (**D**) pe 0.
SED	Implicit	SED	F8	Setează flag-ul *decimal* (**D**) pe 1.
CLV	Implicit	CLV	B8	Setează flag-ul *overflow* (**O**) pe 0.
TAX	Implicit	TAX	AA	Copie valoarea registrului **A** în registrul **X**. Flag-uri afectate: **N**, **Z**
TAY	Implicit	TAY	A8	Copie valoarea registrului **A** în registrul **Y**. Flag-uri afectate: **N**, **Z**
TSX	Implicit	TSX	BA	Copie valoarea registrului **SP** în registrul **X**. Flag-uri afectate: **N**, **Z**
TXA	Implicit	TXA	8A	Copie valoarea registrului **X** în registrul **A**. Flag-uri afectate: **N**, **Z**
TYA	Implicit	TYA	98	Copie valoarea registrului **Y** în registrul **A**. Flag-uri afectate: **N**, **Z**
TXS	Implicit	TXS	9A	Copie valoarea registrului **X** în registrul **SP**.
NOP	Implicit	NOP	EA	Nici o operațiune.
INX	Implicit	INX	E8	Incrementează valoarea registrului **X**.
INY	Implicit	INY	C8	Incrementează valoarea registrului **Y**.
DEX	Implicit	DEX	CA	Decrementează valoarea registrului **X**.
DEY	Implicit	DEY	88	Decrementează valoarea registrului **Y**.
JSR	Absolute	JSR $*LLHH*	20	Salt la subrutină. Salvează pe stivă valoarea registrului **PC**+1 și sare la adresa $*LLHH*.
RTS	Implicit	RTS	60	Revenire din subrutină. Citește de pe stivă valoarea registrului **PC**.
RTI	Implicit	RTI	40	Revenire din întrerupere. Citește de pe stivă valoarea registrului **PC** și a registrului de flag-uri.
JMP	Indirect	JMP ($*LLHH*)	6C	Salt necondiționat la o nouă locație.
	Absolute	JMP $*LLHH*	4C	
LDA	Immediate	LDA #$*BB*	A9	Încarcă o valoare în registrul **A**. Flag-uri afectate: **N**, **Z**
	Zero Page	LDA $*LL*	A5	
	Zero Page indexed by X	LDA $*LL*,X	B5	
	Absolute	LDA $*LLHH*	AD	
	Absolute indexed by X	LDA $*LLHH*,X	BD	
	Absolute indexed by Y	LDA $*LLHH*,Y	B9	
	Indexed indirect	LDA ($*LLHH*,X)	A1	
	Indirect indexed	LDA ($*LLHH*),Y	B1	
LDX	Immediate	LDX #$*BB*	A2	Încarcă o valoare în registrul **X**.

	Zero Page	LDX $*LL*	A6	Flag-uri afectate: **N**, **Z**
	Zero Page indexed by Y	LDX $*LL*,Y	B6	
	Absolute	LDX $*LLHH*	AE	
	Absolute indexed by Y	LDX $*LLHH*,Y	BE	
LDY	Immediate	LDY #$*BB*	A0	Încarcă o valoare în registrul **Y**. Flag-uri afectate: **N**, **Z**
	Zero Page	LDY $*LL*	A4	
	Zero Page indexed by X	LDY $*LL*,X	B4	
	Absolute	LDY $*LLHH*	AC	
	Absolute indexed by X	LDY $*LLHH*,X	BC	
STA	Zero Page	STA $*LL*	85	Stochează valoarea registrului **A**.
	Zero Page indexed by X	STA $*LL*,X	95	
	Absolute	STA $*LLHH*	8D	
	Absolute indexed by X	STA $*LLHH*,X	9D	
	Absolute indexed by Y	STA $*LLHH*,Y	99	
	Indexed indirect	STA ($*LLHH*,X)	81	
	Indirect indexed	STA ($*LLHH*),Y	91	
STX	Zero Page	STX $*LL*	86	Stochează valoarea registrului **X**.
	Zero Page indexed by Y	STX $*LL*,Y	96	
	Absolute	STX $*LLHH*	8E	
STY	Zero Page	STY $*LL*	84	Stochează valoarea registrului **Y**.
	Zero Page indexed by X	STY $*LL*,X	94	
	Absolute	STY $*LLHH*	8C	
CMP	Immediate	CMP #$*BB*	C9	Compară o valoare din memorie cu valoarea acumulatorului (**A**). Flag-uri afectate: **N**, **Z**, **C**
	Zero Page	CMP $*LL*	C5	
	Zero Page indexed by X	CMP $*LL*,X	D5	
	Absolute	CMP $*LLHH*	CD	
	Absolute indexed by X	CMP $*LLHH*,X	DD	
	Absolute indexed by Y	CMP $*LLHH*,Y	D9	
	Indexed indirect	CMP ($*LLHH*,X)	C1	
	Indirect indexed	CMP ($*LLHH*),Y	D1	
CPX	Immediate	CPX #$*BB*	E0	Compară o valoare din memorie cu valoarea registrului **X**. Flag-uri afectate: **N**, **Z**, **C**
	Zero Page	CPX $*LL*	E4	
	Absolute	CPX $*LLHH*	EC	
CPY	Immediate	CPY #$*BB*	C0	

	Zero Page	CPY $*LL*	C4	Compară o valoare din memorie cu valoarea registrului **Y**. Flag-uri afectate: **N**, **Z**, **C**
	Absolute	CPY $*LLHH*	CC	
BIT	Zero Page	BIT $*LL*	24	Biții 7 și 6 ai operandului sunt copiați în biții 7 și 6 ai registrului de flag-uri. Dacă operandul este 0, se setează flag-ul **N**.
	Absolute	BIT $*LLHH*	2C	
ADC	Immediate	ADC #$*BB*	69	Adună valoarea acumulatorului (**A**) cu o valoare din memorie și cu bitul de *carry* (**C**) și stochează rezultatul în acumulator. Flag-uri afectate: **O**, **N**, **Z**, **C**
	Zero Page	ADC $*LL*	65	
	Zero Page indexed by X	ADC $*LL*,X	75	
	Absolute	ADC $*LLHH*	6D	
	Absolute indexed by X	ADC $*LLHH*,X	7D	
	Absolute indexed by Y	ADC $*LLHH*,Y	79	
	Indexed indirect	ADC ($*LLHH*,X)	61	
	Indirect indexed	ADC ($*LLHH*),Y	71	
SBC	Immediate	SBC #$*BB*	E9	Din valoarea acumulatorului (**A**) scade o valoare din memorie și bitul de *carry* (**C**). Flag-uri afectate: **O**, **N**, **Z**, **C**
	Zero Page	SBC $*LL*	E5	
	Zero Page indexed by X	SBC $*LL*,X	F5	
	Absolute	SBC $*LLHH*	ED	
	Absolute indexed by X	SBC $*LLHH*,X	FD	
	Absolute indexed by Y	SBC $*LLHH*,Y	F9	
	Indexed indirect	SBC ($*LLHH*,X)	E1	
	Indirect indexed	SBC ($*LLHH*),Y	F1	
ASL	Accumulator	ASL A	0A	Se realizează o deplasare la stânga. Bitul 7 al operandului este copiat în *carry* (**C**). Flag-uri afectate: **N**, **Z**, **C**
	Zero Page	ASL $*LL*	06	
	Zero Page indexed by X	ASL $*LL*,X	16	
	Absolute	ASL $*LLHH*	0E	
	Absolute indexed by X	ASL $*LLHH*,X	1E	
LSR	Accumulator	LSR A	4A	Se realizează o deplasare aritmetică la dreapta. Bitul 0 al operandului este copiat în *carry* (**C**). Flag-uri afectate: **Z**, **C**
	Zero Page	LSR $*LL*	46	
	Zero Page indexed by X	LSR $*LL*,X	56	
	Absolute	LSR $*LLHH*	4E	
	Absolute indexed by X	LSR $*LLHH*,X	5E	
ROL	Accumulator	ROL A	2A	Se realizează o rotire la stânga cu *carry* (**C**).
	Zero Page	ROL $*LL*	26	

	Zero Page indexed by X	ROL $*LL*,X	36	Flag-uri afectate: **N**, **Z**, **C**
	Absolute	ROL $*LLHH*	2E	
	Absolute indexed by X	ROL $*LLHH*,X	3E	
ROR	Accumulator	ROR A	6A	Se realizează o rotire la dreapta cu *carry* (**C**). Flag-uri afectate: **N**, **Z**, **C**
	Zero Page	ROR $*LL*	66	
	Zero Page indexed by X	ROR $*LL*,X	76	
	Absolute	ROR $*LLHH*	6E	
	Absolute indexed by X	ROR $*LLHH*,X	7E	
ORA	Immediate	ORA #$*BB*	09	Realizează operația *sau logic* între acumulator și operand. Flag-uri afectate: **N**, **Z**
	Zero Page	ORA $*LL*	05	
	Zero Page indexed by X	ORA $*LL*,X	15	
	Absolute	ORA $*LLHH*	0D	
	Absolute indexed by X	ORA $*LLHH*,X	1D	
	Absolute indexed by Y	ORA $*LLHH*,Y	19	
	Indexed indirect	ORA ($*LLHH*,X)	01	
	Indirect indexed	ORA ($*LLHH*),Y	11	
AND	Immediate	AND #$*BB*	29	Realizează operația *și logic* între acumulator și operand. Flag-uri afectate: **N**, **Z**
	Zero Page	AND $*LL*	25	
	Zero Page indexed by X	AND $*LL*,X	35	
	Absolute	AND $*LLHH*	2D	
	Absolute indexed by X	AND $*LLHH*,X	3D	
	Absolute indexed by Y	AND $*LLHH*,Y	39	
	Indexed indirect	AND ($*LLHH*,X)	21	
	Indirect indexed	AND ($*LLHH*),Y	31	
EOR	Immediate	EOR #$*BB*	49	Realizează operația *sau exclusiv* între acumulator și operand. Flag-uri afectate: **N**, **Z**
	Zero Page	EOR $*LL*	45	
	Zero Page indexed by X	EOR $*LL*,X	55	
	Absolute	EOR $*LLHH*	4D	
	Absolute indexed by X	EOR $*LLHH*,X	5D	
	Absolute indexed by Y	EOR $*LLHH*,Y	59	
	Indexed indirect	EOR ($*LLHH*,X)	41	

	Indirect indexed	`EOR ($LLHH),Y`	`51`	
`INC`	Zero Page	`INC $LL`	`E6`	Incrementează cu 1 valoarea operandului. Flag-uri afectate: **N**, **Z**
	Zero Page indexed by X	`INC $LL,X`	`F6`	
	Absolute	`INC $LLHH`	`EE`	
	Absolute indexed by X	`INC $LLHH,X`	`FE`	
`DEC`	Zero Page	`DEC $LL`	`C6`	Decrementează cu 1 valoarea operandului. Flag-uri afectate: **N**, **Z**
	Zero Page indexed by X	`DEC $LL,X`	`D6`	
	Absolute	`DEC $LLHH`	`CE`	
	Absolute indexed by X	`DEC $LLHH,X`	`DE`	
`BEQ`	Relative	`BEQ $BB`	`F0`	Salt condiționat de **Z**=1.
`BNE`	Relative	`BNE $BB`	`D0`	Salt condiționat de **Z**=0.
`BMI`	Relative	`BMI $BB`	`30`	Salt condiționat de **N**=1.
`BPL`	Relative	`BPL $BB`	`10`	Salt condiționat de **N**=0.
`BCS`	Relative	`BCS $BB`	`B0`	Salt condiționat de **C**=1.
`BCC`	Relative	`BCC $BB`	`90`	Salt condiționat de **C**=0.
`BVS`	Relative	`BVS $BB`	`70`	Salt condiționat de **O**=1.
`BVC`	Relative	`BVC $BB`	`50`	Salt condiționat de **O**=0.

tabelul 1.3 (Instrucțiunile procesorului 6502)

§1.1.5 Sistemul de întreruperi

Nucleul de **MOS6502** suportă 3 tipuri de întreruperi: **BRK**, **NMI** și **IRQ**.

Întreruperea **IRQ** are loc pe palierul negativ al pinului **IRQ** (vezi *§1.1.1*). La fiecare ciclu **CPU** se verifică nivelul pinului de întrerupere și se activează întreruperea pe nivelul logic 0, dacă flag-ul *inhibit* (**I**) nu este setat. Întreruperea **IRQ** este cea mai puțin prioritară.

Întreruperea **NMI** este cauzată de detecția unui front negativ pe pinul **NMI**. Această întrerupere nu ține cont de valoare flag-ului *inhibit* și poate întrerupe procesorul în timpul execuției unei alte rutine de tratare a întreruperii.

Întreruperea de tip **BRK** (break) este cauzată de instrucțiunea cu același nume (**BRK**, vezi *§1.1.4*). Procesorul își cauzează singur o întrerupere. La fel ca întreruperea **NMI** aceasta se execută chiar dacă flag-ul *inhibit* este setat.

BRK și **NMI** au aceeași prioritate și se pot întrerupe una pe cealaltă.

Vectorii de întreruperi și reset sunt situați la sfârșitul memoriei de program la adresele:

- `$FFFA-$FFFB` – vectorul **NMI**
- `$FFFC-$FFFD` – vectorul de reset
- `$FFFE-$FFFF` – vectorul **BRK** / **IRQ**

La aceste adrese se găsesc adresele de început a rutinelor de tratare a întreruperilor și, în cazul vectorului reset, adresa de început a programului. Întreruperile **BRK** și **IRQ** folosesc același vector. Pentru ca procesorul să poată determina care întrerupere a avut loc, la activarea întreruperii **BRK** se setează flagul **B** (*break*).

§1.2 Unitatea de procesare video (PPU)

Unitatea de procesare grafică folosită în consola **NES** este **Ricoh RP2C02** (**RP2C07**). Acesta generează un semnal video **CVSB** cu o rezoluție de 240 de linii a câte 256 de pixeli. Frecvența de cadre este de aproximativ 60fps pentru varianta **NTSC** și aproximativ 50fps pentru varianta **PAL**.

§1.2.1 Descrierea pinout a procesorului grafic

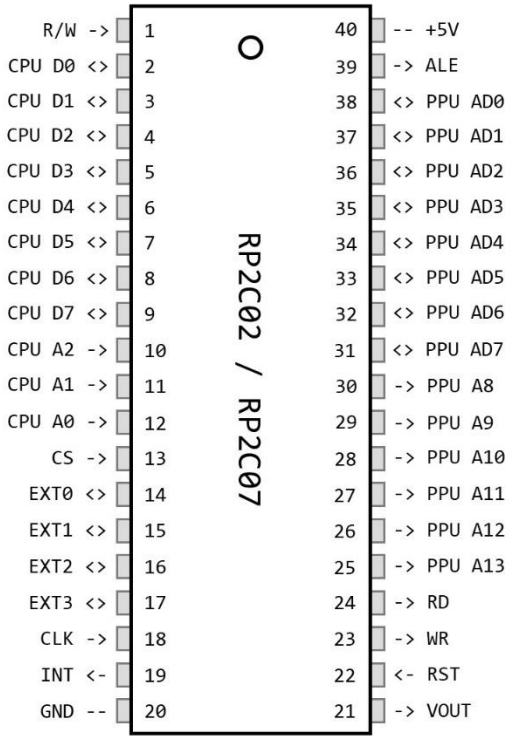


figura 1.4 (RP2C02 / RP2C07 pinout)

R/W – Semnal de la procesor care determină dacă următorul ciclu va fi de citire sau de scriere în memoria registrelor.

[CPU D0:D7] – Semnalele bus-ului de date prin care se comunică cu procesorul central.

[CPU A0:A2] – Semnale care selectează adresa la care va avea loc operațiunea de citire/scriere.

CS – Semnal generat de un decodor de adrese pentru spațiul `$2000-$3FFF`.

[EXT0:EXT3] – Pot fi folosiți ca mod alternativ de selecție a culorii de fundal (vezi *§1.2.5*).

CLK – Intrare de clock (vezi *§1.6*).

INT – Ieșire de întrerupere. Este conectată la pinul **NMI** al procesorului (vezi *§1.1.5*).

RST – Pin de reset activ pe 0.
WR – Semnal de scriere în **V-RAM**.
RD – Semnal de citire din **V-RAM**.
[PPU A8:A13] – Biții cei mai semnificativi ai bus-ului de adrese.
[PPU AD0:AD7] – Bus-ul de date multiplexat cu cei mai puțin semnificativi biți ai bus-ului de adrese.
ALE *(address latch enable)* – Când este activ, menține cei mai puțin semnificativi 8 biți din bus-ul de adrese pentru a permite pinilor **[PPU AD0:AD7]** să fie folosiți pentru transferul de date.

1.2.2 Spațiul de adresare

Procesorul grafic are un spațiu de adrese pe 14 biți (`$0000-$3FFF`), care este separat de bus-ul de memorie al procesorului central. Acesta poate fi accesat direct de către **PPU** sau de către **CPU**, prin intermediul regiștrilor procesorului grafic (vezi *§1.1.3*).

Adresa de început	Adresa de sfârșit	Dimensiunea	Descriere
`$0000`	`$0FFF`	`4KB`	Tabela de șabloane 0. Este situată în prima jumătate a memoriei **CHR** de pe cartridge.
`$1000`	`$1FFF`	`4KB`	Tabela de șabloane 1. Este situată în a doua jumătate a memoriei **CHR** de pe cartridge.
`$2000`	`$23FF`	`1KB`	Tabela de afișare (Nametable) 0. Situată în **V-RAM**.
`$2400`	`$27FF`	`1KB`	Tabela de afișare (Nametable) 1. Situată în **V-RAM**.
`$2800`	`$2BFF`	`1KB`	Tabela de afișare (Nametable) 2. Situată în **V-RAM**.
`$2C00`	`$2FFF`	`1KB`	Tabela de afișare (Nametable) 3. Situată în **V-RAM**.
`$3000`	`$3EFF`	`3840B`	Pointează spre spațiul `$2000-$2EFF`.
`$3F00`	`$3F1F`	`32B`	Memorie **RAM** pentru indecșii culorilor (Palette **RAM**).
`$3F20`	`$3FFF`	`224B`	Pointează spre spațiul `$3F00-$3F1F` repetat de 7 ori.

tabelul 1.4 (Spațiul de adresare PPU)

Procesorul grafic are o memorie **V-RAM** de 2KB în care se stochează tabelele de afișare. Odată ce, evident, nu există suficient spațiu pentru 4 tabele, acestea sunt suprapuse (vezi *§1.2.6*).

Pe lângă acest spațiu de adresare, modulul **PPU** mai are o memorie de 256 octeți pentru a stoca sprite-urile.

De menționat că există un silicon-bug care cauzează ca orice încercare de a scrie la adresa `$3F10` să rezulte într-o scriere la adresa `$3F00`. Adresa `$3F10` poate fi scrisă prin intermediul adreselor din spațiul `$3F20-$3FFF` care pointează spre aceeași zonă de memorie. Bug-ul nu se manifestă la citire.

§1.2.3 Registrele PPU

Unitatea de procesare grafică expune 8 registre de 8 biți în bus-ul de memorie al procesorului central. Acestea apar în spațiul de adresare al procesorului la adresele `$2000-$2007`, dar pot fi accesate și prin intermediul altor adrese (vezi *§1.1.3*).

Procesorul grafic are un bus intern care este folosit la comunicarea cu procesorul. Acesta se comportă ca un latch din cauza proprietăților capacitive ale firelor din acest bus. Încercările de citire a regiștrilor **WO** sau a biților nefolosiți din regiștri rezultă în citirea unei valori ce a fost plasată pe bus la o operațiune anterioară.

Registrele **PPU** sunt descrise în *tabelul 1.5*.

Denumire	Adresă	Acces	Descriere
PPUCTRL	$2000	**WO**	Registru de control.
PPUMASK	$2001	**WO**	Controlează ce elemente vor fi randate și modul de randare.
STATUS	$2002	**RO**	Descrie starea curentă (din momentul citirii) a modulului **PPU**.
OAMADDR	$2003	**R/W**	Adresa de citire/scriere a memoriei de sprite-uri.
OAMDATA	$2004	**R/W**	Permite citirea/scrierea memoriei de sprite-uri.
PPUSCROLL	$2005	**WO**	Poziția de derulare (scroll) orizontal și vertical. Sunt necesare două scrieri.
PPUADDR	$2006	**WO**	Aici se scrie adresa din bus-ul **PPU** pe care procesorul central vrea să o acceseze. Sunt necesare două scrieri.
PPUDATA	$2007	**R/W**	Folosit pentru transferul de date dintre **CPU** și bus-ul de memorie **PPU**.

tabelul 1.5 (Registrele PPU)

Registrul **PPUCTRL**:

7	6	5	4	3	2	1	0
V	P	H	B	S	I	N1	N0

figura 1.5 (Registrul PPUCTRL)

[N1:N0] – Adresa de început pentru tabela de afișare (0=$2000; 1=$2400; 2=$2800; 3=$2C00).

I – Setează cu cât va fi incrementată adresa din bus-ul **PPU** după fiecare operațiune de citire sau scriere realizată de **CPU**. (0=incrementare cu 1; 1=incrementare cu 32).

S – Selectează tabela de șabloane pentru sprite-urile de dimensiune 8x8. Pentru sprite-urile 8x16 acest bit este ignorat. (0=tabela 0; 1=tabela 1) (vezi *§1.2.8*).

B – Selectează tabela de șabloane pentru fundal. (0=tabela 0; 1=tabela 1) (vezi *§1.2.6*).

H – Setează dimensiunea de afișare a sprite-urilor. (0=8x8; 1=8x16) (vezi *§1.2.8*).

P – Setează modul de funcționare a pinilor **EXT**. (0=citește indexul paletei de pe **EXT**; 1=scrie indexul paletei pe **EXT**).

V – Activează/dezactivează întreruperea **NMI** de la sfârșitul fiecărui cadru (la intrarea în *vertical blank*) (0=**NMI** dezactivat; 1=**NMI** activat).

Setarea bitului **V** atunci când modulul **PPU** se află în starea *vertical blank* va cauza generarea imediată a unei întreruperi **NMI**. Acest lucru poate cauza randarea greșită a următorului cadru.

Registrul **PPUMASK**:

7	6	5	4	3	2	1	0
B	G	R	s	b	M	m	g

figura 1.6 (Registrul PPUMASK)

g – Dacă este setat, la ecran vor fi afișate doar tonuri de gri.

m – Setează dacă pe banda de 8 pixeli de la marginea ecranului va fi afișat fundalul. (0=ascunde; 1=afișează).

M – Setează dacă pe banda de 8 pixeli de la marginea ecranului vor fi afișate sprite-urile. (0=ascunde; 1=afișează).

b – Afișare fundal. Dacă nu este setat se va afișa negru în locul pixelilor de fundal.

s – Afișare sprite-uri.

R – Dacă este setat vor fi evidențiate culorile roșii.

G – Dacă este setat vor fi evidențiate culorile verzi.

B – Dacă este setat vor fi evidențiate culorile albastre.

Registrul **STATUS**:

7	6	5	4	3	2	1	0
V	S	O	-	-	-	-	-

figura 1.7 (Registrul STATUS)

O – (*sprite everflow*) Se setează dacă a se încearcă afișarea a mai mult de 8 sprite-uri pe o singură linie a ecranului.

S – Se activează după afișarea unui sprite ce are indexul 0 în tabela de șabloane.

V – Se setează în momentul în care procesorul grafic trece în starea *vertical blank*. Se resetează doar la citirea registrului.

Citirea acestui registru cauzează resetarea bitului **V** și a numărătoarelor de scrieri pentru registrele **PPUSCROLL** și **PPUADDR**.

Comportamentul bitului **O** este imprevizibil din cauza unor bug-uri. Poate raporta atât falsuri pozitive cât și negative.

Registrul **OAMADDR**:

În acest registru se scrie o adresă de 8 biți din memoria de sprite-uri care se dorește o fi accesată. Pentru copierea datelor despre sprite-uri, majoritatea jocurilor scriu în acest registru `$00` și activează modulul **DMA** (vezi *§1.2.4*).

Registrul **OAMDATA**:

Acest registru este folosit pentru interacționarea (scrierea/citirea) cu zona de memorie din memoria de sprite-uri selectată de registrul **OAMADDR**.

Registrul **PPUSCROLL**:

Acest registru este folosit pentru setarea poziției de derulare a imaginii de pe ecran (vezi *§1.2.7*). Prima scriere în acest registru va seta derularea orizontală (pe axa X), iar a doua scriere va seta derularea verticală (pe axa Y).

Registrul **PPUADDR**:

În acest registru va fi scrisă adresa din bus-ul **PPU** pe care procesorul central vrea să o acceseze. Odată ce adresa țintă este pe 14 biți, iar registrul pe 8, vor fi nevoie de două scrieri pentru a introduce adresa completă. La prima scriere se vor copia cei mai semnificativi 6 biți ai adresei,

iar la a doua ce mai puțin semnificativi 8. Accesarea zonei de memorie se realizează prin intermediul registrului **PPUDATA**. Fiecare operațiune cu registrul **PPUDATA** va cauza incrementarea adresei cu 1 sau cu 32, în dependență de valoarea bitului **I** din registrul **PPUCTRL**.

Registrul **PPUDATA**:

Prin intermediul acestui registru procesorul poate accesa bus-ul **PPU**. La fiecare operațiune realizată cu acest registru (scriere/citire) are loc incrementarea adresei din bus.

§1.2.4 Modulul DMA

Pentru copierea rapidă a datelor despre sprite-uri din bus-ul procesorului în memoria de sprite-uri a unității grafice este folosit un **DMA**. Acesta are un singur registru situat la adresa `$4014` (**OAMDMA**) în spațiul de adresare al procesorului central.

Modulul se activează la scrierea unei valori de 8 biți în registrul **OAMDMA**. Valoarea scrisă reprezintă pagina din bus-ul de memorie al procesorului de unde se vor copia datele. Odată pornit, **DMA**-ul copie 256 de octeți (adică toată pagina) în memoria de sprite-uri a procesorului grafic începând cu adresa aflată în registrul **OAMADDR**.

Copierea durează 514 cicluri de ceas (256 de citire + 256 de scriere + 2 de inițializare și finalizare), timp în care procesorul nu execută instrucțiuni.

§1.2.5 Paleta de culori

Procesorul grafic **NES** poate genera 64 de culori, indexate de la 0 la 63.

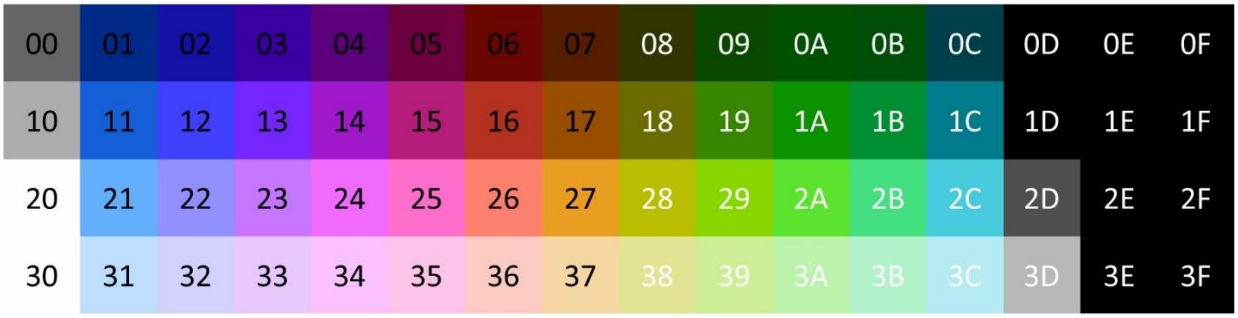


figura 1.8 (Paleta de culori NES)

Culorile generate pot avea anumite variații în dependență de versiunea consolei și de televizorul (monitorul) utilizat.

Pe ecran pot fi afișate, la un moment dat, maxim 25 de culori dintre cele afișate în *figura 1.8*. Indecșii culorilor ce se doresc a fi utilizate se scriu în memoria Palette **RAM** (vezi *§1.2.2*) situată la adresele `$3F00-$3F1F`. Indecșii pentru culorile de fundal sunt situate la adresele `$3F00-$3F0F`, iar cei pentru culorile sprite-urilor la adresele `$3F10-$3F1F`.

Însemnătatea fiecărei adrese este descrisă în *tabelul 1.6*.

Adrese	Însemnătate
$3F00	Culoare universală de fundal.
$3F01-$3F03	Paleta de fundal 0.
$3F04	Neutilizată. Considerată drept transparent (se afișează $3F00).
$3F05-$3F07	Paleta de fundal 1.
$3F08	Neutilizată. Considerată drept transparent (se afișează $3F00).
$3F09-$3F0B	Paleta de fundal 2.
$3F0C	Neutilizată. Considerată drept transparent (se afișează $3F00).
$3F0D-$3F0F	Paleta de fundal 3.
$3F10	Considerată a fi partea transparentă a sprite-ului.
$3F11-$3F13	Paleta de sprite-uri 0.
$3F14	Considerată a fi partea transparentă a sprite-ului.
$3F15-$3F17	Paleta de sprite-uri 1.
$3F18	Considerată a fi partea transparentă a sprite-ului.
$3F19-$3F1B	Paleta de sprite-uri 2.
$3F1C	Considerată a fi partea transparentă a sprite-ului.
$3F1D-$3F1F	Paleta de sprite-uri 3.

tabelul 1.6 (Tabela indecșilor de culori)

§1.2.6 Tabelele de afișare

Tabela de afișare este o zonă de memorie de 1024 de octeți folosită de **PPU** pentru randarea pixelilor de fundal. Procesorul grafic poate accesa 4 tabele virtuale de afișare. Fizic, memoria în care se află aceste tabele, este de doar 2048 de octeți, ceea ce înseamnă că pot exista maxim două tabele de afișare.

Cele 4 tabele pot fi accesate în bus-ul de adresare **PPU** în spațiul $2000-$2FFF (vezi *§1.2.2*).

Tabelele de afișare virtuale sunt create prin suprapunerea adreselor. Nativ (fără *memory mapper*, vezi *§1.5.1*), consola **NES** suportă 4 tipuri de suprapunere: *horizontal mirroring*, *vertical mirroring*, *one-screen higher*, *one-screen lower*.

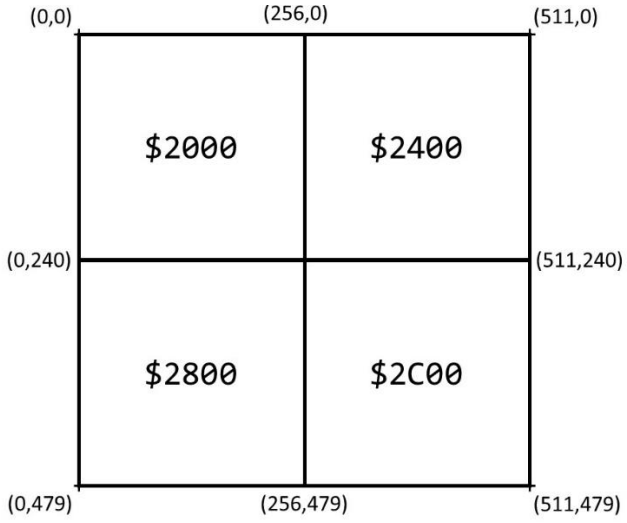


figura 1.9 (Tabelele de afișare)

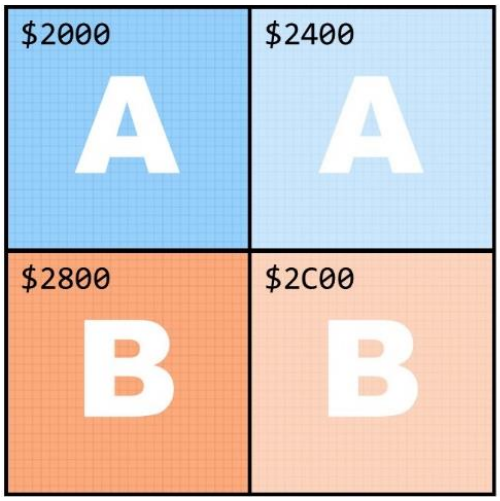


figura 1.9a (horizontal mirroring)

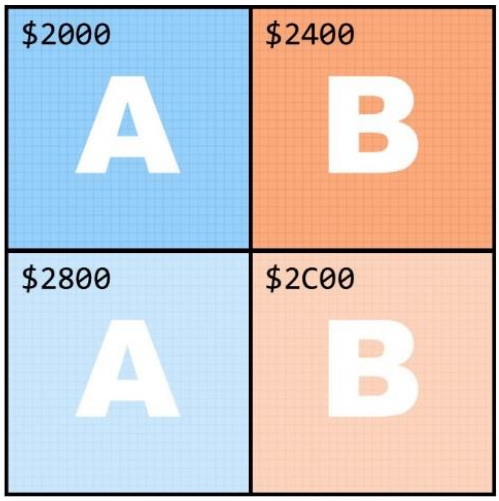


figura 1.9b (vertical mirroring)

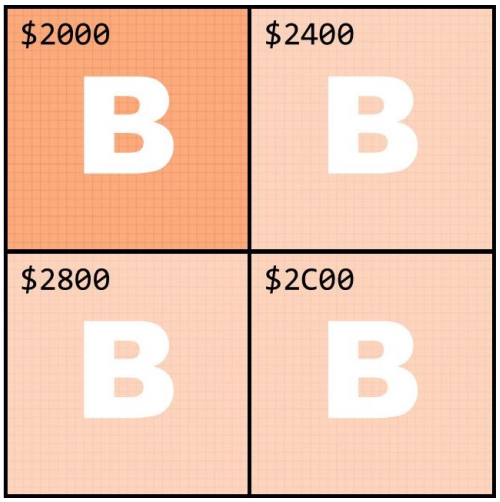


figura 1.9c (one-screen higher)

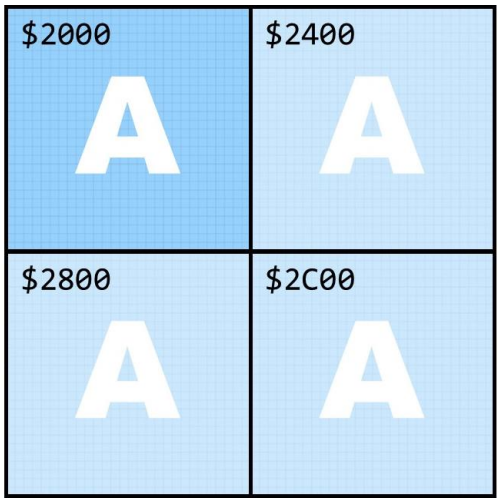


figura 1.9d (one-screen lower)

Tabele de căutare conțin indecșii blocurilor ce trebuie afișate. Blocurile au dimensiunea de 8x8 pixeli și pe ecran încap 30 rânduri a câte 32 blocuri. Fiecare tabelă de afișare conține o matrice cu 30 rânduri și 32 coloane de indecși pe 8 biți. La sfârșitul tabelei de afișare se află un vector de 64 de octeți numit tabela de atribute. Fiecare octet din tabela de atribute conține 4 indecși (de 2 biți) ai paletei de fundal ce trebuie folosită pentru fiecare bloc (vezi *§1.2.5*).

§1.2.7 Derularea fundalului

Derulare reprezintă mișcarea ecranului (viewport-ului) peste tabelele de afișare. Acest lucru este realizat pentru a afișa secvențe din hărțile de joc care au dimensiuni mai mari decât mărimea ecranului.

Poziția și direcția de derulare este controlată de procesor prin intermediul registrului **PPUSCROLL** (vezi *§1.2.3*). Modificarea poziției de derulare necesită două scrieri în acest registru: prima scriere modifică derularea orizontală (axa X), iar a doua scrierea pe cea verticală (axa Y). Valoarea de o biți scrisă în registru reprezintă ofsetul în pixeli peste care trebuie realizată derularea.

Pentru derularea continuă, procesorul rescrie în permanență partea din tabelele de afișare care nu se vede pe ecran. Efectul este demonstrat în *figura 1.10*.

În partea de sus a imaginilor este reprezentat conținutul tabelelor de afișare. Barele cafenii indică ce porțiune din tabelele de afișare sunt vizibile pe ecran. În partea de jos a imaginilor este reprezentat ce vede utilizatorul pe ecran.

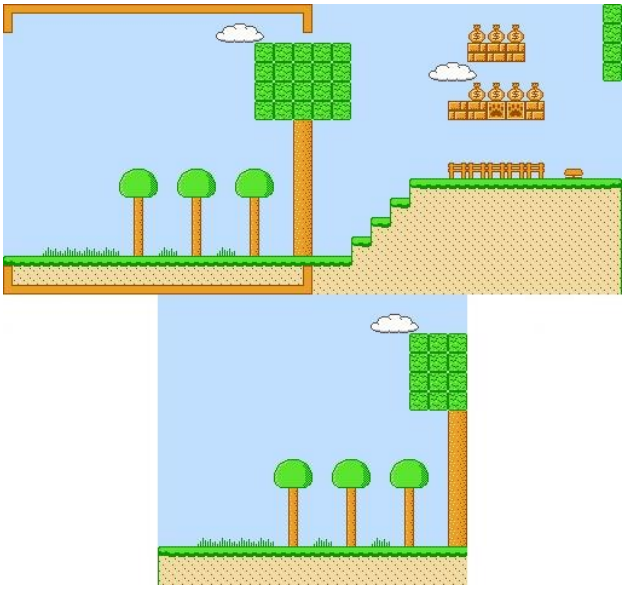


figura 1.10a (Derularea fundalului)

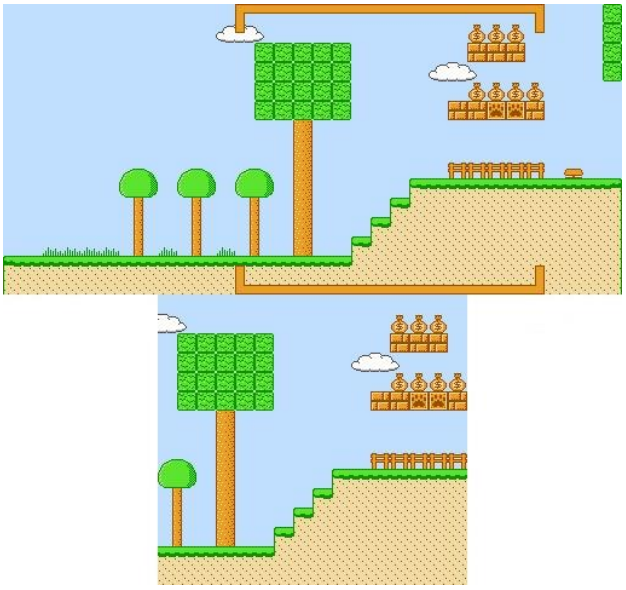


figura 1.10b (Derularea fundalului)

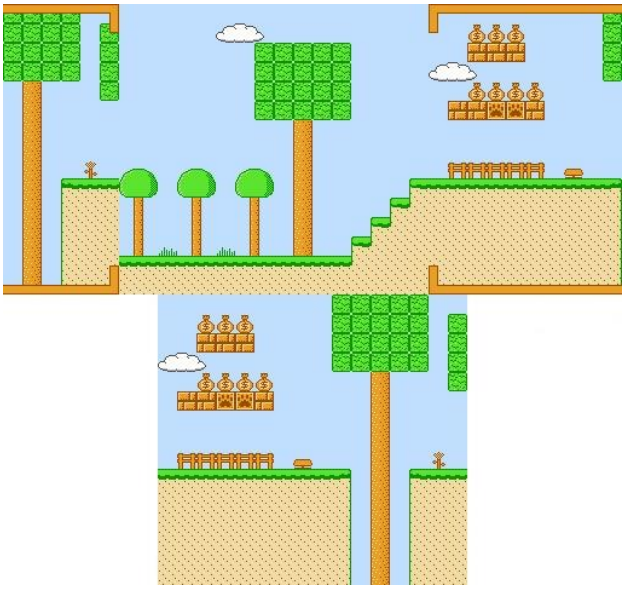


figura 1.10c (Derularea fundalului)

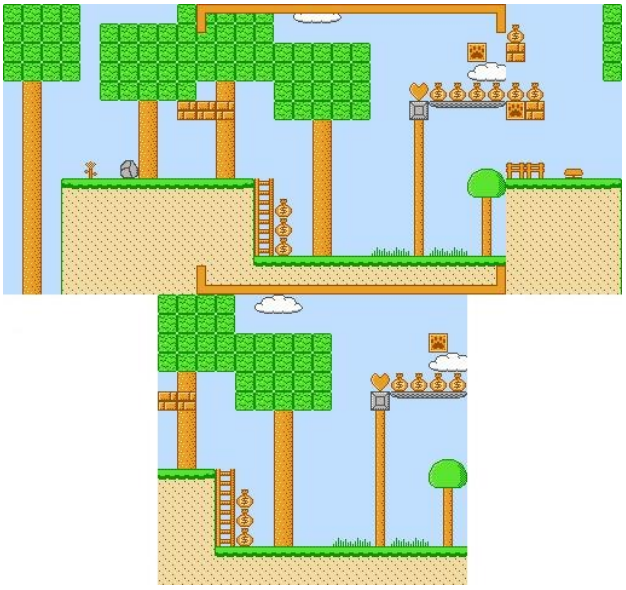


figura 1.10d (Derularea fundalului)

Din perspectiva utilizatorului harta este derulată continuu spre dreapta. În memoria consolei, totuși, are loc modificarea permanentă a datelor dintr-un „buffer” circular.

Datele din tabelele de afișare și pozițiile de derulare sunt modificate atunci când procesorul grafic este în starea *vertical blank*. Procesorul este informat de trecerea în *vertical blank* prin intermediul unei întreruperi **NMI** (vezi *§1.1.5*).

§1.2.8 Procesarea sprite-urilor

Datele despre sprite-uri sunt stocate într-o memorie internă a procesorului grafic de 256 de octeți. Acolo încap maxim 64 de sprite-uri și fiecare ocupă câte 4 octeți.

Octetul 0 conține poziția sprite-ului pe axa Y (ofsetul în pixeli de la partea de sus a ecranului la partea de sus a sprite-ului). Odată ce viewport-ul are o înălțime de 240 de pixeli, setarea acestui atribut pe oricare valoare între 240 și 255 va rezulta în ascunderea sprite-ului.

Octetul 1 reprezintă indexul șablonului ce se va afișa. Pentru sprite-urile de 8x8 pixeli se va citi un singur șablon din memoria **CHR**, iar pentru sprite-urile 8x16 pixeli sunt citite două șabloane consecutive începând cu o adresă pară (bitul 0 este ignorat).

Octetul 2 conține atributele pentru randarea sprite-ului. Semnificația biților este descrisă în *figura 1.11*.

7	6	5	4	3	2	1	0
FV	FH	P	-	-	-	C1	C0

figura 1.11 (Atributele sprite-ului)

[C1:C0] – Indexul paletei care se va folosi la colorarea sprite-ului.
P – Determină dacă sprite-ul va fi randat în fața sau în spatele fundalului (0=în față; 1=în spate).
FH – (*flip horizontally*) Dacă bitul este setat, sprite-ul va fi afișat ca în oglindă.
FV – (*flip vertically*) Dacă bitul este setat, sprite-ul va fi afișat cu susul în jos.

Oglindirea sprite-urilor nu modifică poziția catetei de încadrarea a acestora.

Octetul 3 conține poziția sprite-ului pe axa X (ofsetul în pixeli de la partea din stânga a ecranului la partea din stânga a sprite-ului).

Atunci când două sau mai multe sprite-uri se suprapun, prioritate are sprite-ul cu adresa cea mai mică în memoria **OAM**.

În timpul randării, în momentul în care se afișează primul pixel opac al sprite-ului cu indexul 0, se setează bitul **S** din registrul **STATUS** (vezi *§1.2.3*) pentru a raporta procesorului la ce etapă din randare se află cadrul curent.

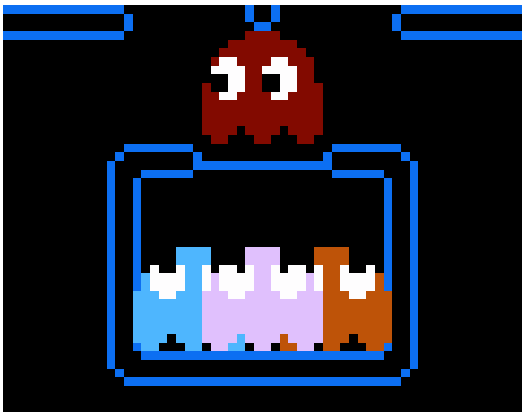


figura 1.12 (Suprapunerea sprite-urilor)

§1.2.9 Randarea

Unitatea de procesare grafică randează cadre cu rezoluția de 256x240 pixeli la o frecvență de cadre de 60fps sau 50fps, pentru modelele **NTSC** și respectiv **PAL**. La fiecare ciclu de ceas, se scoate la ieșire culoarea unui pixel. Durata *horizontal blank* este de 84 de cicli, iar durata *vertical blank* este de 21 de linii. Sincronizarea cadrelor este descrisă în *figura 1.13*.

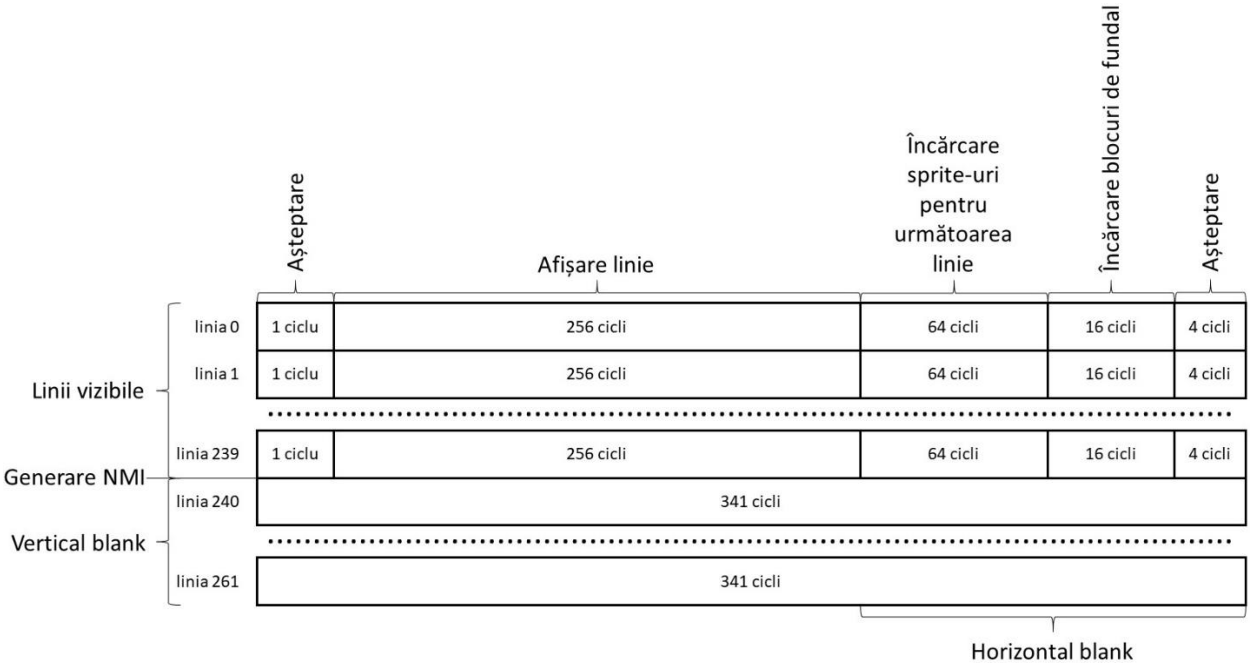


figura 1.13 (Sincronizarea cadrelor)

Randarea propriu-zisă a cadrului se realizează pe 4 straturi (vezi *figura 1.14*).

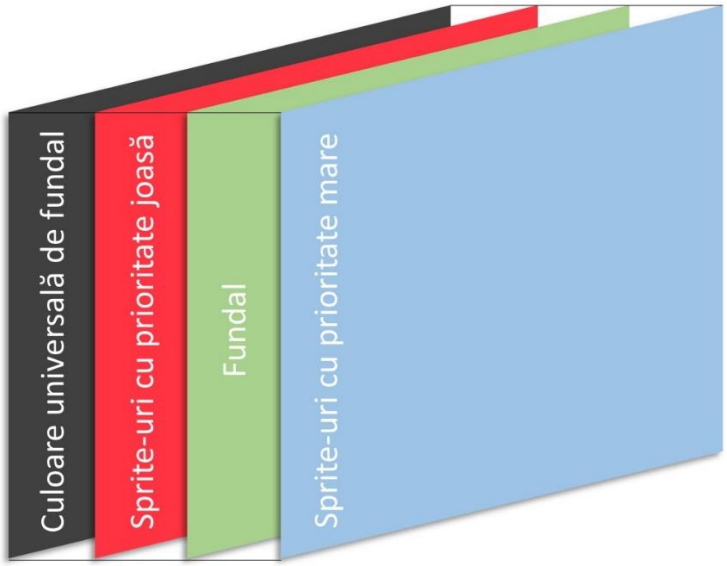


figura 1.14 (Straturile vizibile)

Stratul din spate reprezintă o culoare solidă luată de la adresa `$3F00` (vezi *§1.2.5*). Stratul cu verde reprezintă fundalul jocului, iar cele cu roșu și albastru sprite-urile. Aceste 3 straturi pot

avea pixeli transparenți. Culoarea de ieșire este calculată, la fiecare moment de timp cu două multiplexoare de prioritate și unul de transparență (vezi *figura 1.15*).

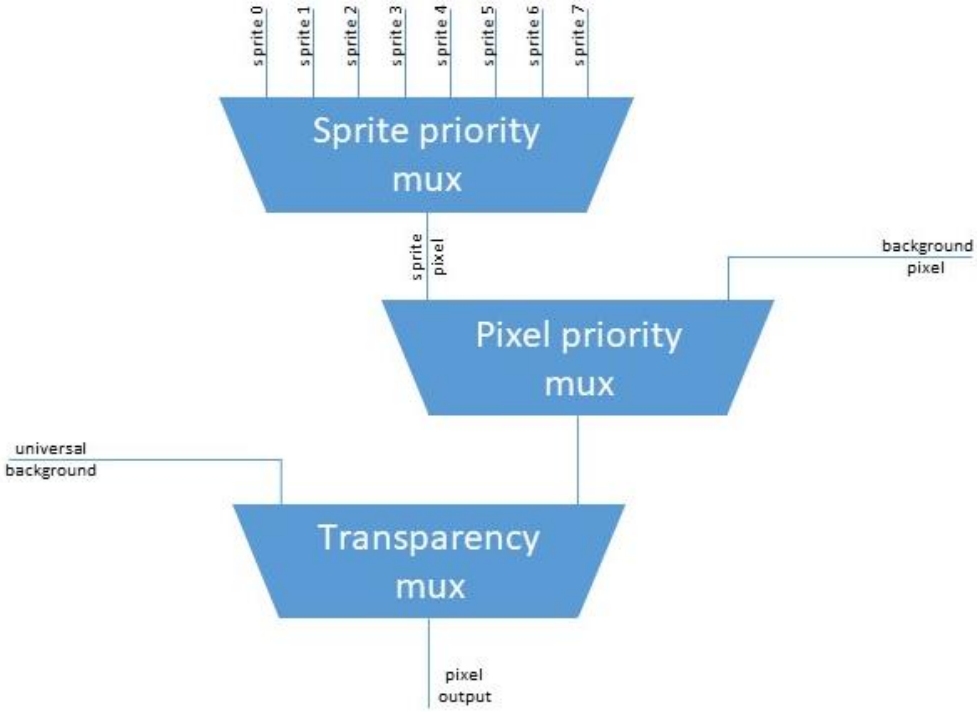


figura 1.15 (Randarea)

Din cauza modului în care funcționează această logică de prioritate, se poate întâmpla ca unii pixeli din sprite-urile din stratul de suprafață să nu se vadă. Acest fenomen apare atunci când în spatele acestui sprite se află un alt sprite cu index mai mic în memoria **OAM** dar are prioritate mai mică decât fundalul (vezi *figura 1.16*).

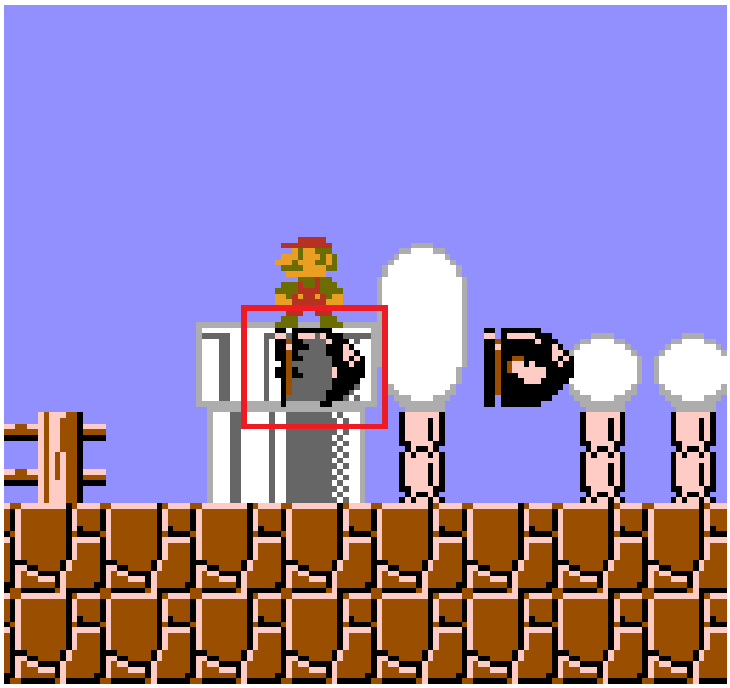


figura 1.16 (Greșeală de prioritate)

§1.3 Unitatea de procesare audio (APU)

Unitatea de procesare audio generează sunete pentru jocurile **NES**. Această unitate este încorporată împreună cu nucleul de procesare în pastila **RP2A03/RP2A07**. Cipul are doi pini de ieșire pentru semnalele audio (vezi *§1.1.1*): **AD1** și **AD2**. Semnalele de la cele două ieșiri sunt adunate de un circuit extern și redate la un difuzor (vezi *§1.3.7*).

Modulul **APU** conține 5 canale audio: două generatoare de unde dreptunghiulare, un generator de unde triunghiulare, un generator de zgomot și un demodulator delta-sigma. Acesta rulează la jumătate din frecvența procesorului.

§1.3.1 Registrele APU

Registrele procesorului de sunet sunt mapate în spațiul de adresare al procesorului.

Denumire	Adresă	Acces	Descriere
SQ1DLVOL	$4000	**WO**	Setează factorul de umplere, volumul sunetului și dacă volumul este constant pentru primul generator de semnal dreptunghiular.
SQ1SWEEP	$4001	**WO**	Setează parametrii pentru modulatorul de frecvență a primului generator de semnal dreptunghiular.
SQ1TMR	$4002	**WO**	Setează biții mai puțin semnificativi ai perioadei numărătorului pentru primul generator de semnal dreptunghiular.
SQ1LNGH	$4003	**WO**	Setează durata sunetului și biții cei mai semnificativi ai perioadei numărătorului primul generator de semnal dreptunghiular.
SQ2DLVOL	$4004	**WO**	Setează factorul de umplere, volumul sunetului și dacă volumul este constant pentru al doilea generator de semnal dreptunghiular.
SQ2SWEEP	$4005	**WO**	Setează parametrii pentru modulatorul de frecvență al celui de-al doilea generator de semnal dreptunghiular.
SQ2TMR	$4006	**WO**	Setează biții mai puțin semnificativi ai perioadei numărătorului pentru al doilea generator de semnal dreptunghiular.
SQ2LNGH	$4007	**WO**	Setează durata sunetului și biții cei mai semnificativi ai perioadei numărătorului al doilea generator de semnal dreptunghiular.
TRLIN	$4008	**WO**	Setează perioada semnalului pentru generatorul de semnal triunghiular.
TRTMR	$400A	**WO**	Setează biții mai puțin semnificativi ai perioadei numărătorului pentru generatorul de semnal triunghiular.
TRLNGH	$400B	**WO**	Setează durata sunetului și biții mai semnificativi ai perioadei numărătorului pentru generatorul de semnal triunghiular.
NZLVOL	$400C	**WO**	Setează volumul sunetului și dacă sunetul este constant pentru generatorul de zgomot.
NZPER	$400E	**WO**	Setează perioada semnalului și dacă sunetul se va repeta în buclă.
NZLNGH	$400F	**WO**	Setează durata sunetului produs de generatorul de zgomot.
DMCFRQ	$4010	**WO**	Setează rata de eșantioane pe secundă și activează întreruperea de final.
DMCLOAD	$4011	**WO**	În acest registru se scrie valoarea unui eșantion care se dorește a fi scos la ieșire.
DMCADDR	$4012	**WO**	Setează adresa de unde o să înceapă citirea eșantioanelor.
DMCLNGH	$4013	**WO**	Setează lungimea șirului de eșantioane.

CTRL/STATUS	$4015	**R/W**	Scrierea setează care canale sunt active. Citirea returnează care canale au terminat redare și care a fost ultima sursă de întrerupere (numărătorul de cadre sau DMC).
FRMCNT	$4017	**WO**	Setează lungimea cadrului și activează întreruperea de final de cadru.

tabelul 1.7 (Registrele APU)

Registrul CTRL/STATUS este cel mai important registru **APU**, întrucât de acesta depinde care canale audio rulează și care nu.

7	6	5	4	3	2	1	0
ID	IF	-	D	N	T	P2	P1

figura 1.17 (Registrul CTRL/STATUS)

P1/P2 – La scriere setează dacă canalele generatoare de unde dreptunghiulare sunt active sau nu. La citire returnează 0 dacă numărătoarele de lungime pentru canalele respective au terminat numărarea.

T – La scriere setează dacă canalul generator de unde triunghiulare este activ sau nu. La citire returnează 0 dacă numărătorul de lungime al acestui canal a terminat numărarea.

N – La scriere setează dacă canalul generator de zgomot este activ sau nu. La citire returnează 0 dacă numărătorul de lungime al acestui canal a terminat numărarea.

D – La scriere setează dacă modulatorul sigma-delta este activ sau nu. La citire returnează starea de activitate a canalului (aceeași valoare care a fost scrisă la ultima scriere).

IF – La citire returnează 1 dacă a avut loc o cerere de întrerupere de la numărătorul de cadre.

ID – La citire returnează 1 dacă a avut loc o cerere de întrerupere de la modulatorul sigma-delta.

§1.3.2 Generatoarele de unde dreptunghiulare

Unitatea de procesare audio conține două generatoare de unde dreptunghiulare independente unul de celălalt. Acestea sunt denumite *pulse channels* și pot genera sunete cu frecvențe cuprinse între 55Hz și 12.4kHz.

Fiecare canal de puls are câte 4 registre: SQxDLVOL, SQxSWEEP, SQxTMR, SQxLNGH.

7	6	5	4	3	2	1	0
D1	D0	L	C	V3	V2	V1	V0

figura 1.18 (Registrele SQxDLVOL)

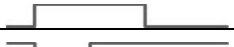
[D1:D0] – Setează factorul de umplere al semnalului generat.

L – Dacă este setat, modulatorul de volum va funcționa în buclă.

C – Dacă este setat, volumul de la ieșirea canalului este constant și este reprezentat de valoare biților **V3:V0**. Dacă nu este setat, volumul de la ieșire variază în dependență de funcționarea modulatorului de volum.

[V3:V0] – Dacă volumul este constant, acești biți setează amplitudinea semnalului. Dacă volumul este variabil, aceștia setează viteza de variație a volumului.

Factorul de umplere al semnalului generat poate fi de 12,5% (D=00), 25% (D=01), 50% (D=10) sau 75% (25% inversat; D=11). Factorul de umplere este generat cu un vector de secvență (*tabelul 1.8*) care este constituit din 8 pași și un numărător pe 3 biți. La pornire, numărătorul este inițializat cu valoarea 0, dar acesta numără descrescător.

D	Vector de secvență	Ieșire numărător	Forma de undă
00	0 0 0 0 0 0 0 1	0 1 0 0 0 0 0 0	
01	0 0 0 0 0 0 1 1	0 1 1 0 0 0 0 0	
10	0 0 0 0 1 1 1 1	0 1 1 1 1 0 0 0	
11	1 1 1 1 1 1 0 0	1 0 0 1 1 1 1 1	

tabelul 1.8 (Secvența generatorului)

7	6	5	4	3	2	1	0
E	P2	P1	P0	N	S2	S1	S0

figura 1.19 (Registrele SQxSWEEP)

E – Activează / dezactivează modulatorul de frecvență.

[P2:P0] – Setează perioada dintre două modificări succesive ale frecvenței semnalului.

N – Setează direcția modulării (0=modulează spre frecvență mai joasă; 1=modulează spre frecvență mai înaltă).

[S2:S0] – Setează cu cât va fi modificată perioada la fiecare pas.

Modulatorul de frecvență oprește automat generatorul de semnal dreptunghiular atunci când frecvența iese din intervalul 55Hz-12.4KHz.

Registrele SQxTMR conțin biții mai puțin semnificativi ai valorii pe 11 biți până la care numărătoarele generatoarelor vor număra. De fiecare dată când numărătorul depășește această valoare, generatorul trece la următoarea valoare din vectorul de secvență.

7	6	5	4	3	2	1	0
L4	L3	L2	L1	L0	T10	T9	T8

figura 1.20 (Registrele SQxLNGH)

[T10:T8] – Biții cei mai semnificativi ai valorii pe 11 biți până la care numărătoarele generatoarelor vor număra.

[L4:L0] – Valoarea respectivă reprezintă durata dorită a sunetului.

Atunci când numărătorul de durată ajunge la 0, se setează bitul Px din registrul CTRL/STATUS.

§1.3.3 Generatorul de unde triunghiulare

Generatorul de unde triunghiulare generează o undă pseudo-triunghiulară cuantizată pe 4 biți. Acest modul nu are control de volum. Spre deosebire de celelalte generatoare de sunet, care rulează la jumătate din frecvența procesorului central, acesta rulează la frecvența procesorului. Generatorul de unde triunghiulare are 3 registre de configurare și control.

7	6	5	4	3	2	1	0
C	L6	L5	L4	L3	L2	L1	L0

figura 1.21 (Registrul TRLIN)

C – Dacă acest bit este setat, generatorul se nu va opri la expirarea numărătorului liniar.

[L6:L0] – Acești biți conțin valoarea inițială a numărătorului liniar. Acesta numără descrescător. Scopul numărătorului este de a controla durata sunetului generat. Dacă bitul **C** nu este setat, atunci generatorul se va opri atunci când numărătorul atinge valoarea 0.

7	6	5	4	3	2	1	0
T7	T6	T5	T4	T3	T2	T1	T0

figura 1.22 (Registrul TRTMR)

[T7:T0] – Biții mai puțin semnificativi ai perioadei numărătorului generatorului. Această valoare setează frecvența undei generate.

7	6	5	4	3	2	1	0
L4	L3	L2	L1	L0	T10	T9	T8

figura 1.23 (Registrul TRLNGH)

[T10:T8] – Biții mai semnificativi a perioadei numărătorului generatorului.

[L4:L0] – Acești biți setează durata până când va fi activat bitul **T** din registrul CTRL/STATUS.

Spre deosebire de generatoarele de unde dreptunghiulare, nu este setată vreo limitare a valorilor valide a perioadei numărătorului la generatorul de unde triunghiulare. Acesta poate genera frecvențe ce depășesc cu mult limita audibilă a oamenilor și capacitatea plăcilor de sunet ale multor calculatoare. Gama de frecvențe generabile este 25.3Hz – 55.9KHz.

§1.3.4 Generatorul de zgomot

Generatorul de zgomot folosește un generator de biți pseudo-aleatorii. Ieșirea generatorului este pe un bit. Pentru configurarea acestuia se folosesc 3 registre.

7	6	5	4	3	2	1	0
-	-	L	C	V3	V2	V1	V0

figura 1.24 (Registrul NZLVOL)

L – Dacă acest bit este setat, modulatorul de volum va funcționa în buclă.

C – Dacă este setat, volumul de la ieșire va fi constant, iar dacă nu, volumul va fi variat de către modulatorul de volum.

[V3:V0] – Dacă bitul **C** este setat, acești biți setează volumul de la ieșirea generatorului, iar în caz contrat setează viteza de variație a volumului.

7	6	5	4	3	2	1	0
M	-	-	-	P3	P2	P1	P0

figura 1.25 (Registrul NZPER)

M – Acest bit setează modul de generare a biților pseudo-aleatorii.

[P3:P0] – Acești biți reprezintă indexul perioadei numărătorului pentru generatorul pseudo-aleatoriu. Valoarea perioadei este luat dintr-un tabel de căutare cu 16 poziții.

7	6	5	4	3	2	1	0
L4	L3	L2	L1	L0	-	-	-

figura 1.26 (Registrul NZLNGH)

[L4:L0] – Acești biți setează durata până când va fi setat bitul **N** din registrul CTRL/STATUS.

§1.3.5 Demodulatorul sigma-delta

Canalul de demodulare (**DMC**) poate scoate la ieșire eșantioane audio digitale codificate pe un bit delta. Dacă eșantionul curent este 1, atunci valoarea de tensiune de la ieșire va crește, iar dacă eșantionul este 0, valoarea tensiunii de ieșire va scădea. Tensiunea de ieșire este controlată de un numărător pe 7 biți. Modulul **DMC** permite și încărcarea directă a unei valori pe 7 biți în numărător. Acest lucru permite o precizie mai mare a sunetului, dar consumă mult timp **CPU**.

Modulul **DMC** are 4 regiștri de configurare și control:

7	6	5	4	3	2	1	0
I	L	-	-	R3	R2	R1	R0

figura 1.27 (Registrul DMCFRQ)

I – Dacă este setat, modulul **DMC** va genera o întrerupere **IRQ** (vezi *§1.1.5*) după terminarea redării eșantioanelor din buffer.

L – Dacă este setat buffer-ul de eșantioane va fi redat în buclă.

[R3:R0] – Setează rata de eșantionare a demodulatorului. Rata de eșantionare este luată dintr-un tabel de căutare.

Rata de eșantionare poate varia de la 4182 la 33148 eșantioane pe secundă, în dependență de valoarea **[R3:R0]** aleasă.

7	6	5	4	3	2	1	0
-	D6	D5	D4	D3	D2	D1	D0

figura 1.28 (Registrul DMCLOAD)

[D6:D0] – Reprezintă valoarea pe 7 biți ce se încarcă direct în numărător la scrierea acestui registru. Acest lucru cauzează schimbarea imediată a ieșirii modulului.

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

figura 1.29 (Registrul DMCADDR)

[A7:A0] – Setează adresa din memorie unde începe buffer-ul audio. Adresa efectivă este calculată după formula: `0xC000 + ([A7:A0] << 6)`.

7	6	5	4	3	2	1	0
L7	L6	L5	L4	L3	L2	L1	L0

figura 1.30 (Registrul DMCLNGH)

[L7:L0] – Setează lungimea a buffer-ului. Lungimea efectivă în octeți este calculată după formula: `([L7:L0] << 4) + 1`. Fiecare octet conține 8 eșantioane codificate pe câte un bit.

Modulul **DMC** mai poate fi folosit și pentru sincronizarea programului cu unitatea de procesare video. Pentru acest lucru se pornește demodulatorul la cea mai mare rată de eșantionare și se activează întreruperea (vezi *figura 1.27*). Cu aceste setări modulul **DMC** va genera câte o întrerupere la fiecare 972 pixeli (uneori 973) afișați de unitatea **PPU**. Aceasta este cea mai precisă metodă de sincronizare a procesorului central cu procesorul grafic în absența unui sistem avansat de mapare a memoriei (vezi *§1.5.1*).

§1.3.6 Numărătorul de cadre

Numărătorul de cadre controlează modulatoarele de volum, modulatoarele de frecvență, numărătoarele de durată și sistemul de întreruperi. Acesta funcționează într-o secvență de 4 sau 5 pași, în dependență de modul de funcționare, fiecare având durata de 3728 (sau 3729) de cicli. Configurarea numărătorului se face printr-un singur registru.

7	6	5	4	3	2	1	0
M	I	-	-	-	-	-	-

figura 1.31 (Registrul FRMCNT)

M – Modul de funcționare (0=secvență de 4 pași; 1=secvență de 5 pași).
I – Dacă bitul este setat, numărătorul nu va genera întreruperi.

La fiecare pas din secvență au loc diferite evenimente care fie produc efecte sonore (modulare volum și frecvență), fie actualizează starea canalelor și generează întreruperi. Comportamentul este descris în *tabelele 1.9* și *1.10*.

Pas	Nr. cicli	Evenimente
1	3728	• Executare modulare de volum (vezi *§1.3.2*, *§1.3.4*) • Activare numărător liniar (vezi *§1.3.3*)
2	7456	• Executare modulare de volum • Activare numărător liniar • Activare numărătoare de durată și actualizare a biților de stare • Executare modulare de frecvență (vezi *§1.3.2*)
3	11185	• Executare modulare de volum • Activare numărător liniar
4	14914	• Executare modulare de volum • Activare numărător liniar • Activare numărătoare de durată și actualizare a biților de stare • Executare modulare de frecvență • Generare **IRQ**

tabelul 1.9 (Secvența numărătorului în 4 pași)

Cu aceste setări, modulatoarele de volum și numărătorul liniar sunt activate de aproximativ 240 de ori pe secundă, modulatoarele de frecvență și numărătoarele de durată de aproximativ 120 de ori pe secundă, iar întreruperea este generată de aproximativ 60 de ori pe secundă.

Chiar dacă frecvența de cadre audio pare să corespundă cu frecvența de cadre video, diferența de timp dintre întreruperea **NMI** generată de **PPU** și întreruperea **IRQ** generată de **APU** nu reprezintă o metodă bună de sincronizare. Motivul este existența unei diferențe de câțiva cicli între durata unui cadru audio și a unui cadru video.

Aceste valori sunt valabile doar pentru sistemele **NTSC**, sistemele **PAL** rulând la frecvențe mai mici ale procesorului (vezi *§1.6*).

Pas	Nr. cicli	Evenimente
1	3728	• Executare modulare de volum • Activare numărător liniar
2	7456	• Executare modulare de volum • Activare numărător liniar • Activare numărătoare de durată și actualizare a biților de stare • Executare modulare de frecvență
3	11185	• Executare modulare de volum • Activare numărător liniar
4	14914	
5	18640	• Executare modulare de volum • Activare numărător liniar • Activare numărătoare de durată și actualizare a biților de stare • Executare modulare de frecvență

tabelul 1.10 (Secvența numărătorului în 5 pași)

Dacă numărătorul de cadre funcționează în secvența de 5 pași, acesta nu generează întreruperi. În acest caz, singurul mod în care procesorul poate să determine starea în care se află modulul **APU** este citirea în continuu a registrului CTRL/STATUS. În acest mod de funcționare modulatoarele de volum și numărătorul liniar sunt activate de aproximativ 192 de ori pe secundă, iar modulatoarele de frecvență și numărătoarele de durată de aproximativ 96 de ori pe secundă.

§1.3.7 Mixarea ieșirilor generatoarelor

Nivelele de ieșire ale generatoarelor de unde dreptunghiulare, a generatorului de unde triunghiulare și a generatorului de zgomot sunt codificate pe 4 biți. Ieșirea **DMC** este codificată pe 7 biți. Dacă considerăm că ieșirea audio poate lua valori în intervalul [0, 1], atunci aceasta poate fi calculată după formula:

$$out = \frac{95.88}{\left(8128 \div (\boldsymbol{P1} + \boldsymbol{P2})\right) + 100} + \frac{159.79}{\frac{1}{(\boldsymbol{T} \div 8227) + (\boldsymbol{N} \div 12241) + (\boldsymbol{DMC} \div 22638)} + 100}$$

P1, P2 – Ieșirile pe 4 biți ale generatoarelor de unde dreptunghiulare.
T – Ieșirea pe 4 biți a generatorului de unde triunghiulare.
N – Ieșirea pe 4 biți a generatorului de zgomot.
DMC – Ieșirea pe 7 biți a demodulatorului sigma-delta.

Valoarea ieșirii poate fi calculată și cu o aproximație liniară:

$$out = 0.00752 * (\boldsymbol{P1} + \boldsymbol{P2}) + 0.00851 * \boldsymbol{T} + 0.00494 * \boldsymbol{N} + 0.00335 * \boldsymbol{DMC}$$

§1.4 Controlerele de joc

Fiecare unitate **NES** include cel puțin un controler de joc (vezi *figura 1.32*). Fără acesta utilizatorul nu ar avea cum să folosească consola. Controlerul standard conține un D-pad, și 4 butoane: Start, Select, A și B.

figura 1.32 (Controlerul NES standard)

Controlerul are la bază un registru de deplasare pe 8 biți (câte un bit pentru fiecare buton) și funcționează în baza unui semnal de strobare. Cât timp semnalul de strobare este activ, în registru se încarcă starea butoanelor. La dezactivarea semnalului registrul păstrează ultima stare în care se aflau butoanele. Biții corespunzători fiecărui buton în registrul de deplasare sunt reprezentați în *figura 1.33*.

7	6	5	4	3	2	1	0
→	←	↓	↑	st	sel	B	A

figura 1.33 (Registrul de deplasare a controlerului)

Semnalul de strobare al ambelor controlere este controlat prin intermediul registrului `$4016`.

7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	S

figura 1.34 (Registrul JOYSTROBE)

S – Semnal de strobare activ pe 1.

Citirea controlerelor se face prin registre diferite: `$4016` și `$4017`. La fiecare citire se preia câte un singur bit din registrul de deplasare.

7	6	5	4	3	2	1	0
-	-	-	-	C	-	-	B

figura 1.35 (Registrele JOYx)

C – Returnează starea de conexiune a controlerului respectiv (0=neconectat; 1=conectat).
B – Returnează starea bitului 0 din registrul de deplasare a controlerului.

După fiecare citire a registrului JOYx, are loc o deplasare la dreapta cu o poziție a registrului de deplasare din controlerul respectiv.

§1.5 Cartridge-ul NES

Rolul principal al unui cartridge **NES** este stocarea memoriei de instrucțiuni și a memoriei cu elemente grafice, care sunt accesibile procesorului central sau procesorului grafic (vezi *§1.1.3*, *§1.2.2*). Capacitatea standard a unui cartridge este de 32KB de memorie de instrucțiuni și 8KB de memorie grafică. Totuși, unele cartridge-uri au integrate sisteme de mapare a memoriei care permit accesarea unor memorii mai mari (vezi *§1.5.1*), sau au alte funcții avansate: generare de sunete, numărare de linii afișate pe ecran, etc.

§1.5.1 Sistemele de mapare a memoriei

Sistemele de mapare au fost create pentru a permite producătorilor de jocuri să facă jocuri mai mare de 40KB. Acestea funcționează prin împărțirea memoriei de program în blocuri de câte 16KB, 8KB sau 4KB și a celei grafice în blocuri de câte 4KB, 2KB sau 1KB. Apoi, în baza valorilor unor regiștri interni, acestea leagă unele blocuri la anumite adrese din spațiul de memorie al **CPU** sau **PPU** (vezi *figura 1.36*). Astfel, chiar dacă procesorul este limitat la a vedea 32KB de memorie de instrucțiuni, la diferite momente de timp, la aceleași adrese, va vedea blocuri diferite de memorie.

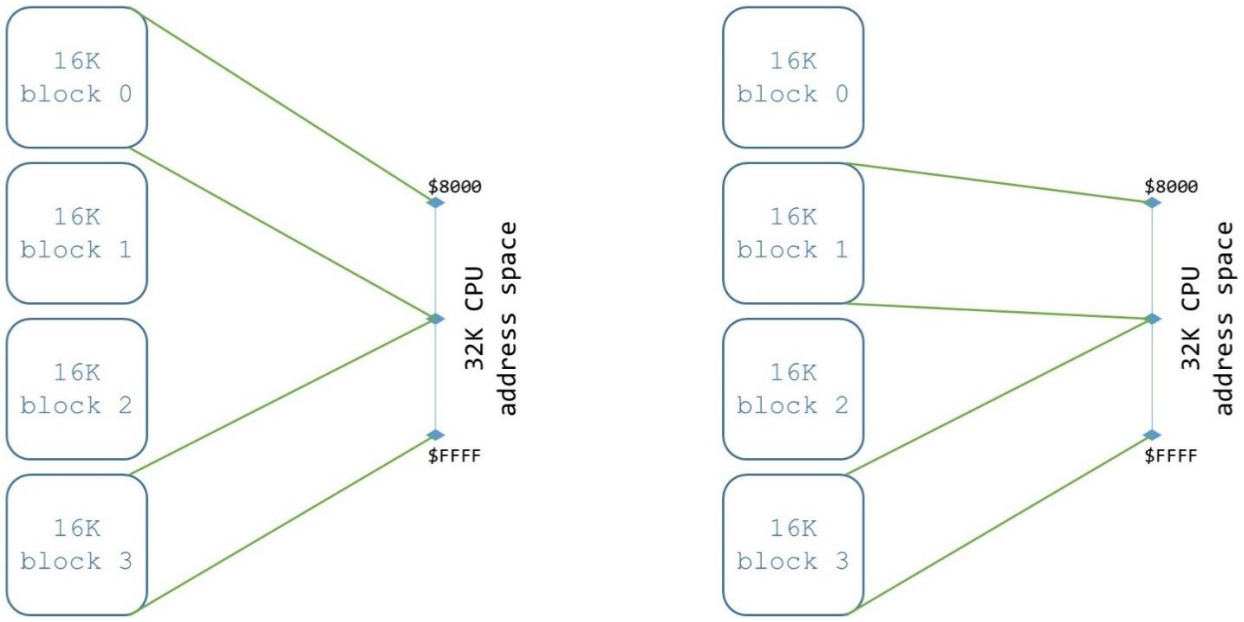


figura 1.36 (Maparea memoriei de program)

Pentru a schimba blocurile, procesorul scrie în anumite registre de pe cartridge. Trebuie menționat faptul că vectorii de reset și întreruperi ai procesorului se află în memoria de program (vezi *§1.1.3*). În acest caz, pe fiecare bloc trebuie să existe câte o copie a acestora, sau blocul pe care se află vectorii să fie fixat.

În aceeași manieră este realizată și maparea memoriei de elemente grafice (**CHR**), doar că în cazul ei nu există nici o constrângere legată de zone critice de memorie cum ar fi vectorii de întrerupere și reset.

§1.5.2 Cipul de securitate

Pe fiecare consolă și cartridge există câte un cip **3193A**. Acesta avea două scopuri: asigurarea compatibilității versiunii de cartridge cu versiunea consolei (vezi *§1.6*) și prevenirea lansării jocurilor nelicențiate de Nintendo. Acest integrat era distribuit producătorilor de jocuri doar în baza unui contract. În acest fel Nintendo putea asigura că jocurile lansate pentru platforma **NES** sunt de calitate.

Cipul funcționează prin intermediul unei chei de autentificare. La pornire, integratul **3193A** de pe consolă cere o cheie de autentificare de la integratul de pe cartridge. Dacă cheia nu este furnizată sau dacă este greșită, cipul menține procesorul în reset, prevenind astfel execuția codului de pe cartridge.

§1.6 Diferențele dintre versiunile NES

Consola **NES** a fost lansată în două versiuni: **NTSC** pentru Statele Unite și Japonia și **PAL** pentru Europa. Diferențele dintre cele două constau în procesorul central, procesorul grafic pe care le foloseau și frecvențele la care acestea rulau. O reprezentare mai detaliată se găsește în *tabelul 1.11*.

Metrică	**NTSC**	**PAL**
Unitate de procesare centrală	RP2A03	RP2A07
Unitate de procesare grafică	RP2C02	RP2C07
Frecvență master	21.477 MHz	26.601 MHz
Divizor frecvență **CPU**	12	16
Frecvență **CPU**	1.789 MHz	1.662 MHz
Divizor frecvență **PPU**	4	5
Frecvență **PPU**	5.361 MHz	5.320 MHz
Vertical blank	20 de linii	70 de linii
Cadre pe secundă	60 fps	50 fps

tabelul 1.11 (Diferențele dintre versiunile NTSC și PAL)

Capitolul 2: Programul de emulare

Programul de emulare este constituit din mai multe module care fie emulează o componentă hardware, fie creează o interfață cu utilizatorul, pentru ca acesta să poată interacționa cu jocurile **NES**. Modulele sunt legate între ele conform schemei din *figura 2.1*. Săgețile groase din schemă reprezintă direcția de transmitere a datelor între module, iar săgețile subțiri reprezintă semnalele de întrerupere ce pot fi generate de anumite module.

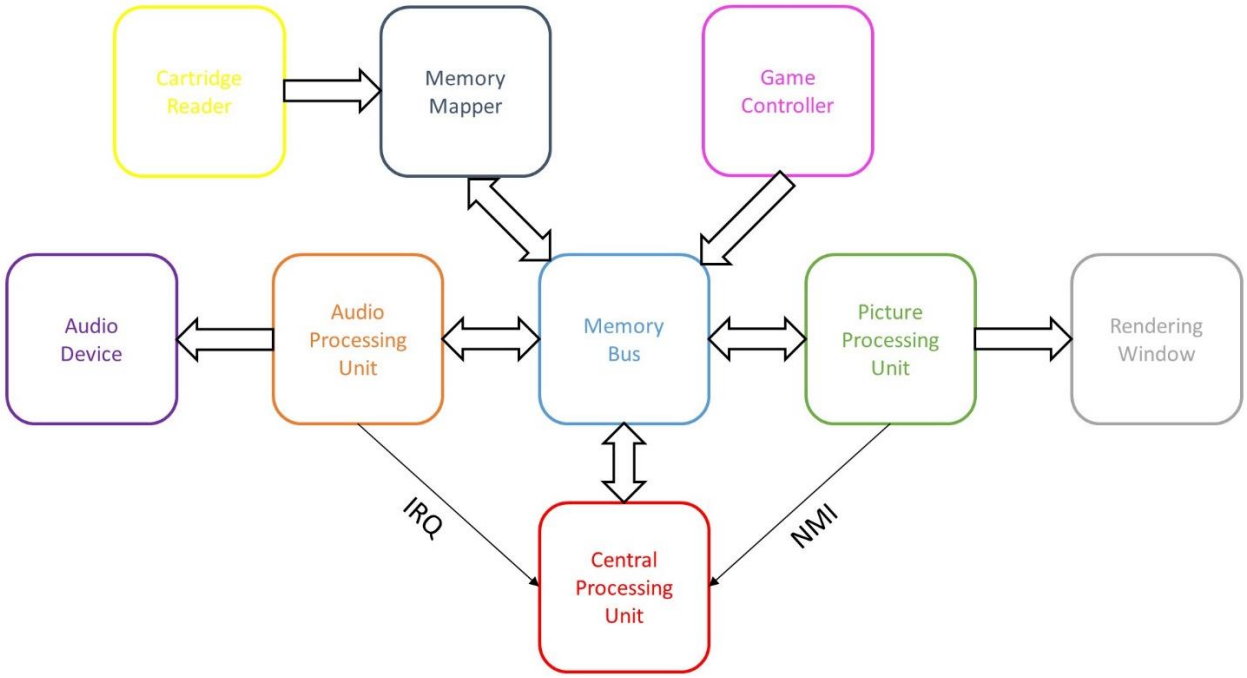


figura 2.1 (Structura emulatorului)

Programul de emulare a fost creat în limbajul de programare `C++` și folosește bibliotecile **SFML 2** și **SDL 1.2.15**. Programul este exportat cu denumirea `NES_emulator.exe` și are ca dependențe 4 fișiere `dll` (vezi *figura 2.2*). Biblioteca **SFML** este folosită pentru crearea unei ferestre grafice și randare grafică, iar **SDL** este folosită pentru a administra redarea audio a emulatorului.

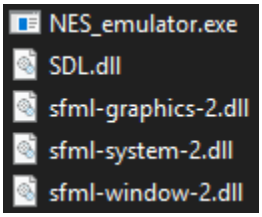


figura 2.2 (Fișierele necesare emulatorului)

Jocurile pot fi deschise cu emulatorul dându-le ca parametru în linia de comandă sau folosind funcția de *drag and drop* din Windows. Dacă jocul este suportat, se va deschide o fereastră în care jocul va fi randat. Formatele de jocuri suportate sunt `iNES 0.7` și `iNES` (extensia .nes). În implementarea curentă, emulatorul suportă aproximativ 60% din totalitatea jocurilor lansate oficial pentru platforma **NES**.

§2.1 Modulul ‘Cartridge reader’

Rolul acestui modul este citirea, decodificarea și organizarea datelor din fișierele de joc. Acesta trebuie să determine tipul cartridge-ului pe care funcționează jocul pentru a determina ce mapper de memorie trebuie folosit, dimensiunea memoriei de program și de grafică, existența unei memorii **RAM** externe și alte caracteristici ale jocului.

Modulul ‘Cartridge reader’ folosește 10 funcții pentru interfațarea cu celelalte module și cu programul principal (vezi *figura 2.3*).

```
namespace CR
{
    uint8_t loadFile(std::string file_name);
    uint8_t* getROM();
    uint8_t* getVideoROM();
    uint8_t getROMBankCount();
    uint8_t getVROMBankCount();
    uint8_t getNameTableMirroring();
    uint8_t getMapperType();
    uint8_t getSystemType();
    bool getBatteryBackedRAMAvailability();
    void clean();
}
```

figura 2.3 (Interfața modulului ‘Cartridge reader’)

Funcția `loadFile` este apelată la pornirea programului. Aceasta deschide și analizează conținutul fișierului de joc primit ca parametru. După analiză, funcția va returna starea de succes sau un cod de eroare dacă fișierul nu a putut fi deschis sau dacă fișierul nu este în formatul `iNES`. Codurile de eroare sunt descrise în *tabelul 2.1*.

Cod de eroare	Semnificație
`SUCCESS`	Fișierul a fost încărcat și procesat fără erori.
`FILE_PROTECTED_OR_NONEXISTENT`	Fișierul pe care încercați să-l deschideți nu există, sau programul nu are permisiunile necesare pentru a-l citi.
`UNABLE_TO_READ_FROM_FILE`	Fișierul este prea scurt sau a apărut o eroare în sistemul de fișiere.
`WRONG_FILE_FORMAT`	Fișierul nu este în format `iNES`.
`NO_PROGRAM_ROM_BANKS`	Câmpul ce descrie numărul de blocuri de program este 0 (vezi *§2.1.1*).
`ROM_HAS_TRAINER`	Jocul are atașat un program ‘trainer’ care nu este suportat de emulator.
`MEMORY_ALLOCATION_FAILURE`	Nu a putut fi alocată destulă memorie pentru încărcarea memoriei de program și de grafică.

tabelul 2.1 (Codurile de eroare ale funcției `loadFile`)

Funcția `getROM` returnează un pointer către începutul zonei de memorie unde au fost încărcate blocurile de memorie de instrucțiuni.

Funcția `getVideoROM` returnează un pointer către începutul zonei de memorie unde au fost încărcate blocurile de memorie grafică, dacă acestea există. În caz contrar funcția returnează `nullptr`.

Funcția `getROMBankCount` returnează numărul de blocuri de 16KB de memorie de program.

Funcția `getVROMBankCount` returnează numărul de blocuri de 8KB de memorie grafică.

Funcția `getNameTableMirroring` returnează modul de organizare a tabelelor de afișare pentru magistrala procesorului grafic.

Funcția `getMapperType` returnează identificatorul sistemului de mapare a memoriei necesar pentru rularea jocului respectiv.

Funcția `getSystemType` returnează tipul sistemului pentru care a fost creat jocul (**NTSC** sau **PAL**).

Funcția `getBatteryBackedRAMAvailability` returnează `true` dacă cartridge-ul are memorie **RAM** externă menținută de o baterie.

Funcția `clean` eliberează memoria alocată în timpul încărcării datelor din fișier. Această funcție este apelată la ieșirea din program.

§2.1.1 Formatul de fișier iNES

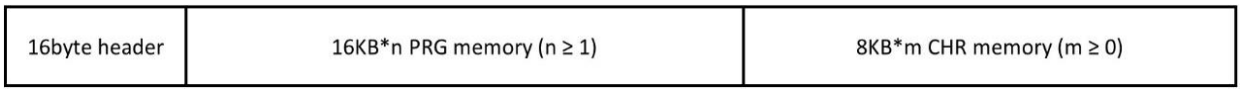


figura 2.4 (Formatul `iNES`)

Header-ul fișierului conține identificatorul de format și informațiile necesare încărcării și emulării jocului.

	Valoare	Semnificație
octetul 0	`N`	Acești 4 octeți sunt folosiți pentru identificarea formatului fișierului.
octetul 1	`E`	
octetul 2	`S`	
octetul 3	`0x1A`	
octetul 4	`1-255`	Conține numărul de blocuri de 16KB de memorie de program (**PRG**). Acest număr trebuie să fie diferit de 0.
octetul 5	`0-255`	Conține numărul de blocuri de 8KB de memorie grafică (**CHR**). Poate fi 0. În acest caz se consideră că există 8KB de **RAM** grafic pe cartridge.
octetul 6	`m|m|m|m|n|t|r|n`	mmmm – Biții mai puțin semnificativi ai identificatorului mapperului necesar emulării jocului. n--n – Modul de aranjare a tabelelor de afișare. t – Dacă este setat, fișierul respectiv are 'trainer'. r – Dacă este setat, cartridge-ul conține **RAM** extern menținut de baterie, sau un alt tip de memorie persistentă.

octetul 7	m|m|m|m|-|-|-|-	mmmm – Biții mai semnificativi ai identificatorului mapperului necesar emulării jocului.
octetul 8		Nefolosit.
octetul 9	-|-|-|-|-|-|-|s	s – Tipul sistemului (0 = **NTSC**; 1 = **PAL**)
octetul 10		Nerelevant pentru mapperele suportate de emulator.
octeții 11-15		Nefolosiți.

tabelul 2.2 (Descrierea header-ului `iNES`)

§2.2 Modulul 'Central processing unit'

Modulul 'Central processing unit' emulează funcționarea procesorului central al consolei **NES** bazat pe nucleul **MOS6502** (vezi *§1.1*).

```
namespace CPU
{
    void reset();
    void step();
    void pullInterruptPin(uint8_t source);
    void releaseInterruptPin(uint8_t source);
    void causeInterrupt(uint8_t interrupt);
    void skipCyclesForDMA();
    void skipCyclesForDMCFetch();
}
```

figura 2.5 (Interfața modulului 'Central processing unit')

Funcția `reset` aduce modulul în starea de reset și încarcă în registrul `PC` valoarea vectorului de reset. Înainte de apelarea acestei funcții modulul 'Memory bus' trebuie să fie deja inițializat.

Funcția `step` execută un ciclu de ceas **CPU**.

Apelul funcției `pullInterruptPin` este echivalent cu tragerea la masă a pinului `IRQ` de pe integrat (vezi *§1.1.1*). Odată ce pinul este conectat la mai multe dispozitive, sursa întreruperii este dată ca parametru acestei funcții.

Funcția `releaseInterruptPin` este folosită pentru a marca eliberarea pinului de întrerupere de către un anumit dispozitiv.

La apelul funcției `causeInterrupt` procesorul începe execuția rutinei de întrerupere asociate întreruperii primite ca parametru dacă toate condițiile necesare sunt îndeplinite. Această funcție este apelată atât din alte module (în cazul întreruperii `NMI`) cât și din modulul curent (în cazul întreruperilor `IRQ` și `BRK`). Funcția este construită în așa fel încât să respecte comportamentul pinilor de `IRQ` și `NMI` de pe integrat.

Funcțiile `skipCyclesForDMA` și `skipCyclesForDMCFetch` informează modulul despre faptul că magistrala de date a procesorului urmează să fie accesată de alte module externe și, din acest motiv, un anumit număr de cicluri de ceas trebuie ignorate pentru a permite modulelor respective accesul exclusiv la datele dorite.

§2.3 Modulul ‘Memory bus’

Acest modul emulează generatoarele de adrese ale modulelor **CPU**, **APU** și **PPU**, la fel și memoriile interne sau externe ale consolei: **RAM**, **V-RAM**, **RAM** extern. Toate operațiunile de scriere și citire generate de cele 3 module sunt executate prin intermediul modulului **MB**. În baza adresei la care modulele generează cereri de scriere/citire, **MB** știe dacă trebuie să acceseze o anumită unitate de memorie sau un registru al unei componente periferice (din alte module).

```
namespace MB
{
    void init();
    bool loadMapperInformation();
    void changeMirroring(uint8_t mirroring);
    void configureMemory(bool enable_external_RAM, bool protect_external_RAM);
    void writeMainBus(uint16_t address, uint8_t data);
    uint8_t readMainBus(uint16_t address);
    void writePictureBus(uint16_t address, uint8_t data);
    uint8_t readPictureBus(uint16_t address);
    void clean();
}
```

figura 2.6 (Interfața modulului ‘Memory bus’)

Funcția `init` inițializează variabilele modulului. Aceasta trebuie apelată înainte de a începe lucrul cu acest modul.

Funcția `loadMapperInformation` încarcă configurația memoriei pentru mapperul jocului deschis. Informațiile despre configurație sunt obținute din modulele ‘Cartridge reader’ și ‘Memory mapper’ (vezi *§2.1*, *§2.4*). Funcția returnează `false` dacă nu a putut fi alocată memorie sau dacă modul de organizare a memoriei nu este suportat de emulator în versiunea curentă.

Funcția `changeMirroring` modifică modul de aranjare a tabelelor de afișare (vezi *§1.2.6*) în baza parametrului primit.

Funcția `configureMemory` setează modul de funcționare a memoriei **RAM** externe în baza a 2 parametri, care determină dacă memoria va fi activată și respectiv, protejată. Dacă memoria este dezactivată, liniile de date ce duc spre aceasta se vor afla în starea de impedanță infinită. Dacă memoria este protejată, atunci toate cererile de scriere în aceasta vor fi respinse.

Funcțiile `writeMainBus` și `readMainBus` sunt folosite pentru accesarea magistralei de date a procesorului. În baza adresei primite ca parametru, modulul **MB** determină blocul de memorie sau registrul ce trebuie accesat și, dacă este cazul, calculează adresa efectivă la care va avea loc operațiunea de scriere/citire.

Funcțiile `writePictureBus` și `readPictureBus` sunt folosite pentru accesarea magistralei de date a procesorului grafic.

Funcția `clean` eliberează toată memoria alocată pe perioada funcționării modulului. Aceasta trebuie apelată în momentul închiderii programului.

§2.4 Modulul ‘Memory mapper’

Modulul **MM** emulează funcționalitatea anumitor tipuri de cartridge-uri de a configura dinamic memoria acestora (vezi *§1.5.1*). În standardul `iNES` (vezi *§2.1.1*) fiecărui mapper îi este asociat un index, de la 1 la 255, indexul 0 fiind rezervat pentru cartridge-urile fără sistem de mapare a memoriei. Indecșii au fost asociați mapperelor în așa fel încât acestea să fie sortate descrescător după numărul de jocuri suportate de ele (nu se aplică pentru mapperul 0).

Emulatorul suportă 4 tipuri de sisteme de mapare, acestea acoperind aproximativ 60% din totalul de jocuri lansate oficial pentru platforma **NES**.

Index	Denumire	Descriere
`0`	`NROM`	Cartridge fără sistem de mapare a memoriei.
`1`	`MMC1`	Sistem de mapare **ASIC** creat de Nintendo.
`2`	`UxROM`	Sistem de mapare de cost și complexitate redusă creat de Nintendo.
`3`	`CNROM`	Sistem de mapare de cost și complexitate redusă creat de Nintendo.

tabelul 2.3 (Sistemele de mapare suportate de emulator)

La pornirea emulatorului, modulul **MM** adoptă comportamentul unuia dintre mapperele suportate în dependență de jocul deschis.

```
namespace MM
{
    void init();
    bool setMapper(uint8_t mapper_type);
    void writePRG(uint16_t address, uint8_t data);
    uint8_t readPRG(uint16_t address);
    void writeCHR(uint16_t address, uint8_t data);
    uint8_t readCHR(uint16_t address);
    uint8_t getNameTableMirroring();
    void clean();
}
```

figura 2.7 (Interfața modulului ‘Memory mapper’)

Funcția `init` trebuie apelată înainte de a începe lucrul cu acest modul. Aceasta inițializează variabilele de stare ale modulului.

Funcția `setMapper` setează comportamentul modulului pentru a corespunde cu funcționalitatea hardware a mapperului de pe cartridge-ul jocului deschis. Funcția returnează `true` dacă mapperul este suportat și `false` în caz contrar.

Funcțiile `writePRG`, `readPRG`, `writeCHR` și `readCHR` sunt folosite pentru accesarea memoriilor **PRG** și **CHR** de pe cartridge. Comportamentul acestora este determinat de tipul de mapper activ la momentul apelului.

Funcția `getNameTableMirroring` returnează configurația curentă a tabelelor de afișare. Aceasta este apelată de modulul **MB** pentru autoconfigurare.

Funcția `clean` eliberează memoria alocată pe parcursul execuției.

§2.4.1 Cartridge-urile fără mapare de memorie (NROM)

Cartridge-urile `NROM` sunt cele mai simple după complexitate, dar sunt și cele mai limitate. Un astfel de cartridge poate conține maxim două blocuri de 16KB de memorie **PRG**, un bloc de 8KB de **CHR ROM** sau **CHR RAM** și până la 4KB de **RAM** extern. În ceea ce privește configurația tabelelor de afișare (vezi *§1.2.6*), acest tip de cartridge suportă doar suprapunerea verticală sau orizontală, și aceasta nu poate fi modificată dinamic în timpul rulării jocului.

§2.4.2 Mapperul MMC1

Mapperul `MMC1` permite maparea tuturor tipurilor de memorii de pe cartridge. Cartridge-urile ce folosesc acest mapper pot avea până la 512KB de memorie **PRG**, 32KB de memorie **RAM** externă și 128KB de memorie grafică. De asemenea, mapperul suportă suprapunerea verticală, orizontală și *one-screen* a tabelelor de afișare (vezi *§1.2.6*), și aceasta poate fi modificată dinamic.

Memoria **PRG** este împărțită în blocuri de câte 16KB. Blocul cu indexul cel mai mare este mereu fixat la spațiul de adresare `$C000-$FFFF`, iar celelalte pot fi conectate dinamic la spațiul de adresare `$8000-$BFFF`. Astfel doar blocul cu indexul cel mai mare trebuie să conțină vectorii de reset și întreruperi (vezi *§1.1.3*). Memoria **RAM** externă este împărțită în blocuri de câte 8KB și fiecare bloc poate fi conectat dinamic la spațiul de adresare `$6000-$7FFF`. Memoria grafică este împărțită în blocuri de câte 4KB și fiecare poate fi mapat în spațiul de adrese **PPU** la adresele `$0000-$0FFF` sau `$1000-$1FFF`.

Mapperul poate fi configurat prin intermediul a 4 regiștri.

Registru	Adrese	Descriere
Control	`$8000-$9FFF`	Configurează tabelele de afișare și modul de mapare a blocurilor **CHR**.
Blocul **CHR** 0	`$A000-$BFFF`	Alege blocul de memorie grafică care să fie mapat la spațiul de adresare `$0000-$0FFF`.
Blocul **CHR** 1	`$C000-$DFFF`	Alege blocul de memorie grafică care să fie mapat la spațiul de adresare `$1000-$1FFF`.
Blocul **PRG**	`$E000-$FFFF`	Alege blocul de memorie **PRG** care să fie mapat la spațiul de adresare `$8000-$BFFF`.

tabelul 2.4 (Registrele `MMC1`)

§2.4.3 Mapperul UxROM

Mapperul `UxROM` include două tipuri de cartridge-uri: `UNROM` și `UOROM`, diferența dintre ele fiind doar capacitatea de memorie de program. Acestea au o capacitate de memorie **PRG** de 256KB și respectiv 4MB. Ambele variante includ 8KB de memorie grafică și nu suportă **RAM** extern. În ceea ce privește configurația tabelelor de afișare, acest mapper se comportă exact ca `NROM` (vezi *§2.4.1*).

Odată ce capacitatea memorie grafice corespunde cu dimensiunea spațiului de adresare **PPU**, aceasta nu trebuie împărțită în blocuri care să fie conectate la magistrală în mod dinamic. În ceea

ce privește memoria de program, aceasta este împărțită în blocuri de 16KB. La fel ca în cazul mapperului `MMC1`, blocul cu indexul cel mai mare este fixat la spațiul de adresare `$C000-$FFFF`, iar celelalte pot fi multiplexate la spațiul `$8000-$BFFF`. Configurarea se face prin intermediul unui singur registru, care poate fi accesat scriind la orice adresă din spațiul `$8000-$FFFF`. În registru se scrie indexul blocului ce se dorește a fi conectat la magistrală.

§2.4.4 Mapperul CNROM

Mapperul `CNROM` are o capacitate foarte mare de memorie grafică: până la 2MB. Acesta este foarte bun pentru producătorii de jocuri care doresc să ofere o varietate foarte mare de elemente vizuale. Celelalte caracteristici sunt identice cu cele ale cartridge-ului `NROM` (vezi *§2.4.1*).

Memoria grafică este împărțită în blocuri de câte 8KB care pot fi conectate la magistrala **PPU** prin scrierea indexului blocului dorit în registrul mapperului. Acesta poate fi accesat la orice adresă din spațiul `$8000-$FFFF`.

§2.5 Modulul 'Picture processing unit'

Acest modul emulează funcționalitatea unității de procesare grafică **RP2C02/RP2C07** (vezi *§1.2*).

```
namespace PPU
{
    void reset();
    void registerScanlineCallback(void(*callback)(bool vblank));
    void step();
    void writeRegisterControl(uint8_t value);
    void writeRegisterMask(uint8_t value);
    void writeRegisterSpriteAddress(uint8_t value);
    void writeRegisterSpriteData(uint8_t value);
    void writeRegisterScroll(uint8_t value);
    void writeRegisterAddress(uint8_t value);
    void writeRegisterData(uint8_t value);
    uint8_t readRegisterControl();
    uint8_t readRegisterMask();
    uint8_t readRegisterStatus();
    uint8_t readRegisterSpriteData();
    uint8_t readRegisterData();
    void executeDMA(uint8_t *page_pointer);
}
```

figura 2.8 (Interfața modulului 'Picture processing unit')

Funcția `reset` aduce modulul în starea de reset și configurează parametrii modulului în dependență de tipul sistemului: **NTSC** sau **PAL**.

Funcția `registerScanlineCallback` activează raportarea sfârșitului de linie a ecranului prin apelarea unei funcții primite ca parametru.

Funcția `step` execută un ciclu de ceas **PPU**.

Funcțiile `writeRegister<…>` și `readRegister<…>` sunt folosite pentru interacțiunea cu registrele perifericului. Acestea sunt apelate din modulul **MB** dacă detectează că procesorul central încearcă să scrie/citească de la adresele corespunzătoare acestor regiștri.

Funcția `executeDMA` inițiază procesul de copiere a datelor **OAM** din spațiul de adresare al procesorului folosind funcționalitatea **DMA** a consolei (vezi *§1.2.4*).

§2.6 Modulul 'Rendering window'

Acest modul administrează o fereastră în care sunt afișate rezultatele procesării grafice realizată de **PPU**. Fereastra conține un viewport cu rezoluția de `768x720` pixeli, iar fiecare pixel virtual este reprezentat printr-o matrice de `3x3` pixeli fizici. De asemenea, modulul **RW** permite utilizatorului să închidă programul apăsând butonul `[×]`.

Afișarea în fereastră este realizată prin intermediul bibliotecii **SFML**. Aceasta nu permite accesul direct la pixelii viewport-ului, cea mai mică unitate grafică disponibilă fiind poligonul bidimensional din 3 vertice (triunghiul). Astfel, fiecare pixel virtual este reprezentat prin două triunghiuri, care acoperă împreună o matrice de `3x3` pixeli (vezi *figura 2.9*).

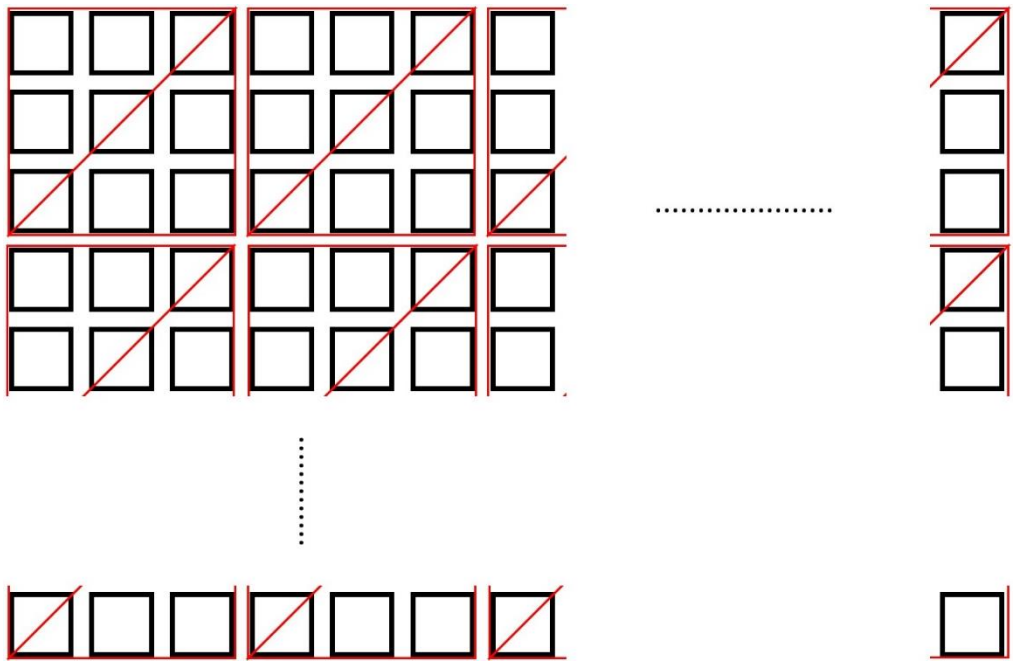


figura 2.9 (Reprezentarea pixelilor prin poligoane)

În *figura 2.9* pixelii fizici sunt reprezentați prin pătrate negre, iar poligoanele sunt desenate cu roșu.

Pentru a putea reacționa la apăsarea butonului `[×]`, fereastra trebuie să verifice starea acestuia în continuu (*polling*). Acest lucru, însă, durează mult timp. Pentru a putea menține o

performanță bună a emulatorului, toate procesele ce au legătură cu fereastra sunt executate pe un alt fir de execuție. Comunicarea cu celelalte module ale programului se face prin intermediul unei cozi de comenzi și a unei cozi de evenimente, accesul la fiecare dintre acestea fiind sincronizat prin intermediul a câte un obiect de tip `std::mutex`.

Interfața **RW** cu celelalte module este descrisă în *figura 2.10*.

```
namespace RW
{
    void init();
    void setPixel(size_t x_coordinate, size_t y_coordinate, uint32_t color);
    uint8_t pollWindowEvent();
    void redraw(bool forced = false);
    void dispose();
}
```

figura 2.10 (Interfața modulului 'Rendering window')

Funcția `init` creează fereastra, inițializează cozile de comenzi și de evenimente și începe procesul de verificare continuă a evenimentelor ferestrei și de execuție a comenzilor primite.

Funcția `setPixel` înregistrează o comandă de colorare a pixelului virtual de coordonate `(x,y)` în culoarea primită ca parametru.

Funcția `pollWindowEvent` returnează primul eveniment care a avut loc de la apelul precedent al acestei funcții.

Funcția `redraw` înregistrează o comandă de actualizare a cadrului afișat. Funcția primește ca parametru o variabilă care determină dacă actualizarea va fi sau nu forțată. Actualizarea forțată presupune ignorarea perioadei minime de afișare a unui cadru.

Funcția `dispose` închide fereastra și eliberează toate resursele folosite de aceasta pe parcursul execuției. Această funcție este apelată la închiderea programului.

§2.7 Modulul 'Audio processing unit'

Modulul **APU** emulează funcționalitatea audio a integratelor **RP2A03** și **RP2A07** (vezi *§1.3*). Modulul emulează fiecare canal audio în parte și apoi mixează ieșirile acestora conform aproximației liniare din *§1.3.7*. Ulterior fluxul de eșantioane este re-eșantionat printr-un filtru Gauss la rata de 48000 de eșantioane pe secundă pentru a fi compatibil cu plăcile audio standard ale calculatoarelor.

Interfața modulului este reprezentată în *figura 2.11*.

```
namespace APU
{
    void reset();
    void step();
    void writeRegisterSQ1Volume(uint8_t value);
    void writeRegisterSQ1Sweep(uint8_t value);
    void writeRegisterSQ1PeriodLow(uint8_t value);
    void writeRegisterSQ1PeriodHigh(uint8_t value);
    void writeRegisterSQ2Volume(uint8_t value);
    void writeRegisterSQ2Sweep(uint8_t value);
    void writeRegisterSQ2PeriodLow(uint8_t value);
    void writeRegisterSQ2PeriodHigh(uint8_t value);
    void writeRegisterTriangleLinearCounter(uint8_t value);
    void writeRegisterTriangleTimerLow(uint8_t value);
    void writeRegisterTriangleTimerHigh(uint8_t value);
    void writeRegisterNoiseVolume(uint8_t value);
    void writeRegisterNoiseMode(uint8_t value);
    void writeRegisterNoiseLength(uint8_t value);
    void writeRegisterDMCFrequency(uint8_t value);
    void writeRegisterDMCRaw(uint8_t value);
    void writeRegisterDMCAddress(uint8_t value);
    void writeRegisterDMCLength(uint8_t value);
    void writeRegisterChannels(uint8_t value);
    void writeRegisterFrameCounter(uint8_t value);
    uint8_t readRegisterStatus();
}
```

figura 2.11 (Interfața modulului 'Audio processing unit')

Funcția `reset` aduce modulul în starea de reset și configurează parametrii acestuia în dependență de tipul sistemului pe care rulează jocul: **NTSC** sau **PAL**.

Funcția `step` execută un ciclu de ceas **APU**. La fiecare astfel de ciclu este generat un eșantion audio.

Funcțiile `writeRegister<…>` și `readRegister<…>` sunt folosite pentru a interacționa cu registrele perifericului. Acestea sunt apelate din modulul **MB** dacă detectează că procesorul central încearcă să scrie/citească de la adresele corespunzătoare acestor regiștri.

§2.8 Modulul 'Audio device'

Modulul **AD** asigură redarea continuă la difuzor a sunetelor generate de modulul **APU**. Pentru administrarea dispozitivului audio este folosită biblioteca **SDL**. Modulul lucrează cu un buffer de 1024 de eșantioane. De fiecare dată când bufferul este golit, biblioteca apelează o funcție de callback în care bufferul trebuie rescris. Pentru a asigura transferul rapid de date către dispozitivul audio, este folosit încă un buffer temporar în care modulul **APU** scrie datele audio eșantion cu eșantion. În momentul apelului funcției de callback, are loc doar copierea datelor din

bufferul temporar în bufferul dispozitivului audio. Pentru a asigura că funcția de callback nu întrerupe lucrul emulatorului, administrarea dispozitivului audio se face pe un fir de execuție separat.

Odată ce timpul de redare a unui buffer este mereu același, modulul **AD** este folosit și pentru a ține evidența timpului, asigurând astfel viteza corectă de redare a jocului.

```
namespace AD
{
    void init();
    void queueSample(uint16_t sample);
    bool bufferFull();
    void dispose();
}
```

figura 2.12 (Interfața modulului 'Audio device')

Funcția `init` inițializează biblioteca **SDL**, activează dispozitivul audio și începe redarea.

Funcția `queueSample` scrie un singur eșantion în bufferul temporar. Dacă acesta este plin, cel mai vechi eșantion este șters.

Funcția `bufferFull` returnează `true` dacă bufferul temporar este plin. Această funcție este folosită pentru sincronizarea vitezei de redare a jocului. Cât timp bufferul nu este plin, se execută cicli de ceas **CPU**, **PPU** și **APU**, apoi se așteaptă apelul funcției de callback. După golirea bufferului începe din nou execuția ciclilor de ceas.

Funcția `dispose` oprește redarea audio și eliberează resursele ocupate pe parcursul execuției.

§2.9 Modulul 'Game controller'

Modulul **GC** emulează comportamentul controllerelor de joc și oferă utilizatorului posibilitatea de a controla jocul ce rulează pe emulator.

Fiecare buton de pe cele două controllere a fost mapat la câte un buton de pe tastatura calculatorului, conform *tabelului 2.5*.

Buton de pe controller	Mapare controller 1	Mapare controller 2
A	M	NUMPAD 9
B	N	NUMPAD 8
SELECT	RIGHT SHIFT	NUMPAD 3
START	RETURN	NUMPAD 6
D-PAD UP	W	↑
D-PAD DOWN	S	↓
D-PAD LEFT	A	←
D-PAD RIGHT	D	→

tabelul 2.5 (Maparea butoanelor de pe controllere)

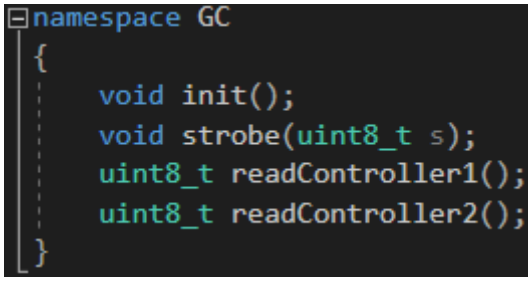


figura 2.13 (Interfața modulului 'Game controller')

Funcția `init` inițializează legăturile dintre butoanele controllerelor și butoanele de pe tastatură.

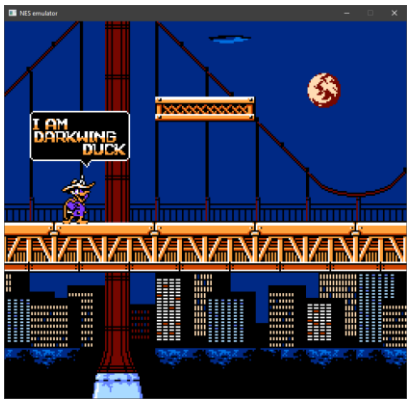
Funcția `strobe` activează sau dezactivează semnalele de strobare ale celor două controllere.

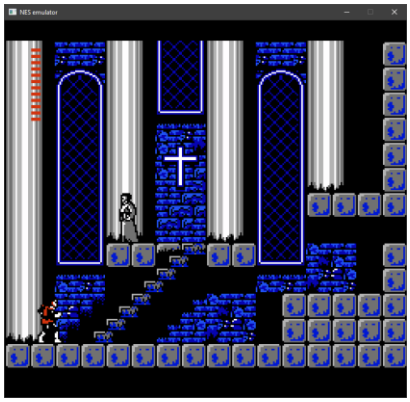
Funcțiile `readController1` și `readController2` returnează starea unui singur buton al controllerului, conform funcționării sale în baza unui registru de deplasare (vezi *§1.4*).

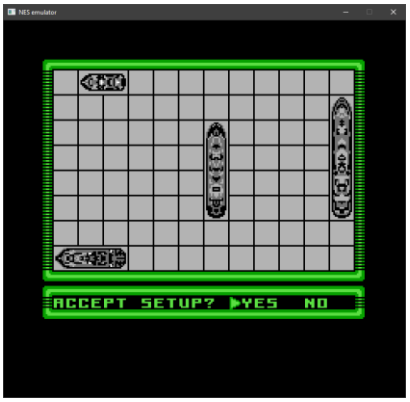
Anexa 1: Lista de jocuri testate

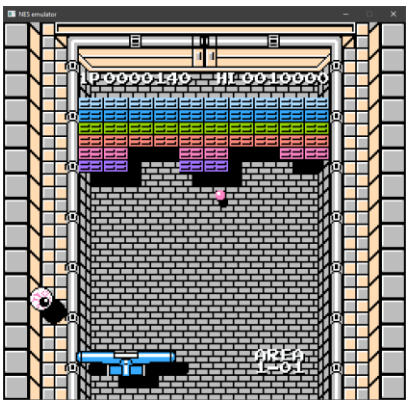
- 15
- 1942
- Adventure Island Classic
- Alpha Mission
- Aussie Rules Footy
- AV Poker
- Balloon Fight
- Battle City
- Battle of Olympus
- Battleship
- Binary Land
- Blaster Master
- Boulder Dash
- Bubble Bobble
- Castlevania
- Castlevania 2: Simon's Quest
- Chip 'n Dale Rescue Rangers
- Circus
- Contra
- Crackout
- Darkwing Duck
- Donkey Kong Classics
- Elite
- Excitebike
- Ghosts 'n Goblins
- Gradius
- Hook
- Ice Climber
- Jaws
- Jimmy Connor's Tennis
- Karate Kid
- Legend of Zelda
- Lunar Pool
- Metal Gear
- Micro Mages
- Monster Rally
- Ms Pac-Man
- Night Arrow
- Pac-Man
- Pinball
- Q-Bert
- Route-16 Turbo
- Rules to Driving a Car
- Side Winder
- Snake Rattle 'n Roll
- Spy Hunter
- Super Mario Bros.
- Terra Cresta
- Tetris
- Thunderbirds
- Top Gun
- Twin Eagle
- Wa Di Lei (Minesweeper)
- Warehouse No. 18
- Xevious

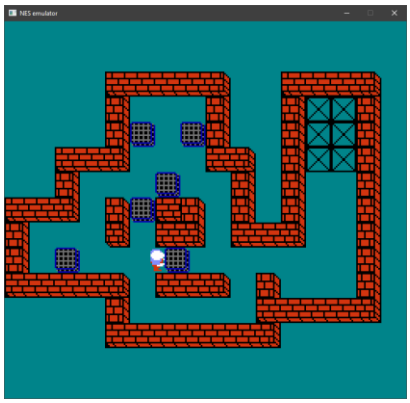
Anexa 2: Capturi de ecran





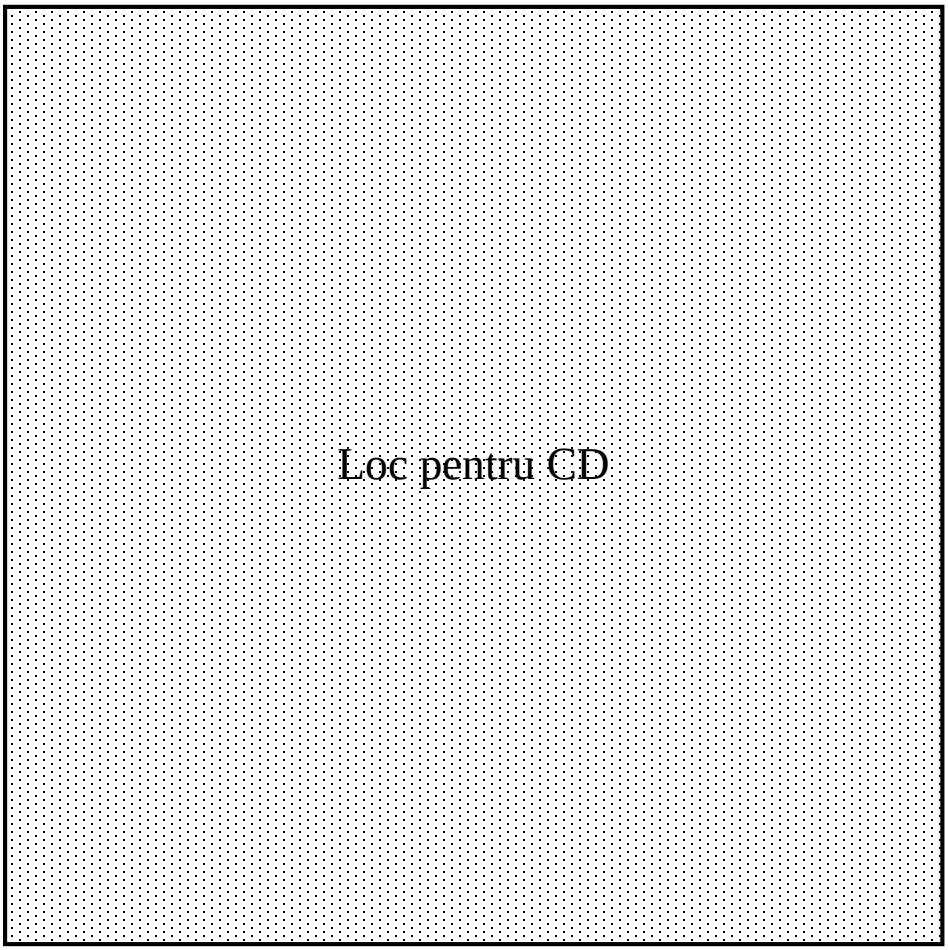






Anexa 3: CD cu conținutul proiectului

Acest CD conține codul sursă al proiectului, fișierul executabil, fișierele cu jocurile testate și versiunea .pdf al acestui document.



Bibliografie

§1.1.1 Descrierea pinout a procesorului:
http://wiki.nesdev.com/w/index.php/CPU_pin_out_and_signal_description

§1.1.2 Regiștrii procesorului:
http://wiki.nesdev.com/w/index.php/CPU_registers
https://www.masswerk.at/6502/6502_instruction_set.html

§1.1.3 Moduri de adresare:
http://wiki.nesdev.com/w/index.php/CPU_addressing_modes
https://www.masswerk.at/6502/6502_instruction_set.html

§1.1.4 Setul de instrucțiuni:
https://www.masswerk.at/6502/6502_instruction_set.html
http://6502.org/tutorials/6502opcodes.html

§1.1.5 Sistemul de întreruperi:
http://wiki.nesdev.com/w/index.php/CPU_interrupts

§1.2.1 Descrierea pinout a procesorului grafic:
http://wiki.nesdev.com/w/index.php/PPU_pin_out_and_signal_description

§1.2.2 Spațiul de adresare:
http://wiki.nesdev.com/w/index.php/PPU_memory_map

§1.2.3 Registrele PPU:
http://wiki.nesdev.com/w/index.php/PPU_registers

§1.2.4 Modulul DMA:
http://wiki.nesdev.com/w/index.php/PPU_OAM#DMA

§1.2.5 Paleta de culori:
http://wiki.nesdev.com/w/index.php/PPU_palettes

§1.2.6 Tabelele de afișare:
http://wiki.nesdev.com/w/index.php/PPU_nametables
http://wiki.nesdev.com/w/index.php/PPU_attribute_tables

§1.2.7 Derularea fundalului:
http://wiki.nesdev.com/w/index.php/PPU_scrolling

§1.2.8 Procesarea sprite-urilor:
http://wiki.nesdev.com/w/index.php/PPU_OAM

§1.2.9 Randarea:
http://wiki.nesdev.com/w/index.php/PPU_rendering
http://wiki.nesdev.com/w/index.php/PPU_sprite_evaluation
http://wiki.nesdev.com/w/index.php/PPU_sprite_priority

§1.3.1 Registrele APU:
http://wiki.nesdev.com/w/index.php/APU_registers

§1.3.2 Generatoarele de unde dreptunghiulare:
http://wiki.nesdev.com/w/index.php/APU_Pulse

§1.3.3 Generatorul de unde triunghiulare:
http://wiki.nesdev.com/w/index.php/APU_Triangle

§1.3.4 Generatorul de zgomot:
http://wiki.nesdev.com/w/index.php/APU_Noise

§1.3.5 Demodulatorul sigma-delta:
http://wiki.nesdev.com/w/index.php/APU_DMC

§1.3.6 Numărătorul de cadre:
http://wiki.nesdev.com/w/index.php/APU_Frame_Counter
http://wiki.nesdev.com/w/index.php/APU_Envelope

§1.3.7 Mixarea ieșirilor generatoarelor:
http://wiki.nesdev.com/w/index.php/APU_Mixer

§1.4 Controlerele de joc:
https://wiki.nesdev.com/w/index.php/Standard_controller

§1.5 Cartridge-ul NES, §1.5.1 Sistemele de mapare a memoriei:
https://wiki.nesdev.com/w/index.php/NROM

§1.5.2 Cipul de securitate:
https://en.wikipedia.org/wiki/CIC_(Nintendo)

§1.6 Diferențele dintre versiunile NES:
http://wiki.nesdev.com/w/index.php/Cycle_reference_chart

§2.1.1 Formatul de fișier iNES:
https://fms.komkon.org/EMUL8/NES.html#LABM
https://wiki.nesdev.com/w/index.php/INES

§2.4.1 Cartridge-urile fără mapare de memorie (NROM):
https://wiki.nesdev.com/w/index.php/NROM

§2.4.2 Mapperul MMC1:
https://wiki.nesdev.com/w/index.php/MMC1

§2.4.3 Mapperul UxROM:
https://wiki.nesdev.com/w/index.php/UxROM

§2.4.4 Mapperul CNROM:
https://wiki.nesdev.com/w/index.php/INES_Mapper_003

figura 1.10:
https://wiki.nesdev.com/w/index.php/PPU_scrolling#The_common_case

figura 1.16:
https://wiki.nesdev.com/w/index.php/PPU_sprite_priority

figura 1.32:
https://www.thecampussocialite.com/nintendo-is-discontinuing-the-nes-classic-heres-why-that-sucks/

Biblioteci utilizate:
SFML 2: https://www.sfml-dev.org
SDL 1.2.15: https://www.libsdl.org/index.php

Instrumente software utilizate: