Nintendo Entertainment System Architecture

version 2.6 (01/24/2005)
by Marat Fayzullin

The following document describes the hardware architecture of Nintendo Entertainment System videogame console, also known as Famicom in the East (Korea, Japan), and Dandy in Europe (Russia, etc.). Note that this document is mainly based on experimentation and may be incomplete and incorrect in many places. *"Nintendo Entertainment System"*, *"Famicom"*, and (most likely) *"Dandy"* are registered trademarks of Nintendo.

This document has also been translated to Japanese (by Bero), French (by Guillaume Tuloup), and Portugese (by Daniel Loureiro).

I would like to thank following people for their help in obtaining this information and writing a NES emulator, as well as moral support from some of them:

```
(sorted alphabetically)
Pascal Felber      Patrick Lesaard      Tink
Goroh              Pan of Anthrox       Bas Vijfwinkel
Kawasedo           Paul Robson
Marcel de Kogel    Serge Skorobogatov
Alex Krasivsky     John Stiles
```

Contents

- What Is Covered And What Not?
- General Architecture
- CPU Memory Map
- PPU Memory Map
- I/O Ports
- Interrupts
- Joysticks
- PPU Memory
  - Accessing PPU Memory
  - Name Tables
  - Pattern Tables
  - Attribute Tables
  - Palettes
  - Sprites
- PPU Details
  - Mirroring
  - VBlank Flag
  - Hit Flag

    - Accessing VRAM During Refresh
    - Accessing PPU Status Register
    - Accessing PPU Data Register
- Sprites
- Memory Mappers
    - Sequential
    - Konami
    - VROM Switch
    - 5202 Chip
    - Others
- Sound
- Famicom Disk System
    - Memory Map
    - IRQ Counter
    - Sound Hardware
    - Disk Format
    - Disk Access
    - Disk Errors
- .NES File Format

What Is Covered And What Not?

This document covers NES memory layout, I/O ports, video controller (PPU), sound hardware, and the most videly used memory switching circuits (mappers). It also describes the .NES file format.

The 6502 CPU used in NES is not covered in this document though, as there is enough information on it in other literature. You are assumed to know how 6502 works.

Also not covered are some more exotic memory mappers, Famicom DiskSystem, and some other esoteric pieces of hardware, as I don't have enough information about them. If you have any information on NES which is not in this manual, feel free to write to marat at server komkon dot org. Your help will be appreciated.

In this manual, you may encounter notation of the form *"Nth bit"* (5th, 3rd, etc.). Bits are counted from the least significant (0th) to the most significant (7th).

All hexadecimal numbers are prepended with a dollar sign ($2002, $4016, etc.) which is a common notation used in 6502 CPU assembler.

General Architecture

NES is based on the 6502 CPU and a custom video controller known as PPU (Picture Processing Unit). The PPU's video memory is separated from the main CPU memory and can be read/written via special ports.

Cartridges may contain both ROM appearing in the CPU address space at $8000-$FFFF, and VROM or VRAM appearing in the PPU address space at $0000-$1FFF.

In smaller cartridges, which only have 16kB ROM, it takes place at $C000-$FFFF leaving $8000-$BFFF area unused. In cartridges with >32kB ROM, it is paged into address space with special circuitry (see "Mappers"). Some cartridges also have RAM at $6000-$7FFF which may or may not be battery-backed.

Cartridge VROM (VRAM) is used for Pattern Tables (aka Tile Tables, Character Generators, etc.). The usual amount is 8kB which contain 2 Pattern Tables. In cartridges with >8kB VROM (VRAM), it is paged into address space with special circuitry (see "Mappers").

Internal NES VRAM is located at addresses $2000-$3FFF in the PPU memory and used to store Name Tables (aka Screen Buffers, etc.). Although PPU supports 4 Name Tables, there is only enough memory for two of them. Other two mirror the first two (see "PPU Details").

CPU Memory Map

```
--------------------------------------- $10000
 Upper Bank of Cartridge ROM
--------------------------------------- $C000
 Lower Bank of Cartridge ROM
--------------------------------------- $8000
 Cartridge RAM (may be battery-backed)
--------------------------------------- $6000
 Expansion Modules
--------------------------------------- $5000
 Input/Output
--------------------------------------- $2000
 2kB Internal RAM, mirrored 4 times
--------------------------------------- $0000
```

PPU Memory Map

```
--------------------------------------- $4000
 Empty
--------------------------------------- $3F20
 Sprite Palette
--------------------------------------- $3F10
 Image Palette
--------------------------------------- $3F00
 Empty
--------------------------------------- $3000
 Attribute Table 3
--------------------------------------- $2FC0
 Name Table 3 (32x30 tiles)
--------------------------------------- $2C00
 Attribute Table 2
--------------------------------------- $2BC0
 Name Table 2 (32x30 tiles)
--------------------------------------- $2800
 Attribute Table 1
--------------------------------------- $27C0
 Name Table 1 (32x30 tiles)
--------------------------------------- $2400
 Attribute Table 0
--------------------------------------- $23C0
 Name Table 0 (32x30 tiles)
--------------------------------------- $2000
 Pattern Table 1 (256x2x8, may be VROM)
--------------------------------------- $1000
 Pattern Table 0 (256x2x8, may be VROM)
--------------------------------------- $0000
```

I/O Ports

NES internal I/O ports are mapped into the areas of $2000-$2007 and $4000-$4017. Some ports' usage is unknown or unclear and any information about them is appreciated.

```
------+-----+---------------------------------------------------------------
$2000 | RW  | PPU Control Register 1
      | 0-1 | Name Table Address:
      |     |
      |     |           +-----------+-----------+
      |     |           | 2 ($2800) | 3 ($2C00) |
      |     |           +-----------+-----------+
      |     |           | 0 ($2000) | 1 ($2400) |
      |     |           +-----------+-----------+
      |     |
      |     | Remember that because of the mirroring there are only 2
      |     | real Name Tables, not 4. Also, PPU will automatically
      |     | switch to another Name Table when running off the current
      |     | Name Table during scroll (see picture above).
      |   2 | Vertical Write, 1 = PPU memory address increments by 32:
      |     |
      |     |    Name Table, VW=0          Name Table, VW=1
      |     |   +----------------+        +----------------+
      |     |   |----> write     |        | | write        |
      |     |   |                |        | V              |
      |     |
      |   3 | Sprite Pattern Table Address, 1 = $1000, 0 = $0000.
      |   4 | Screen Pattern Table Address, 1 = $1000, 0 = $0000.
      |   5 | Sprite Size, 1 = 8x16, 0 = 8x8.
      |   6 | PPU Master/Slave Mode, not used in NES.
      |   7 | VBlank Enable, 1 = generate interrupts on VBlank.
------+-----+---------------------------------------------------------------
$2001 | RW  | PPU Control Register 2
      |   0 | Unknown (???)
      |   1 | Image Mask, 0 = don't show left 8 columns of the screen.
      |   2 | Sprite Mask, 0 = don't show sprites in left 8 columns.
      |   3 | Screen Enable, 1 = show picture, 0 = blank screen.
      |   4 | Sprites Enable, 1 = show sprites, 0 = hide sprites.
      | 5-7 | Background Color, 0 = black, 1 = blue, 2 = green, 4 = red.
      |     | Do not use any other numbers as you may damage PPU hardware.
------+-----+---------------------------------------------------------------
$2002 | R   | PPU Status Register
      | 0-5 | Unknown (???)
      |   6 | Hit Flag, 1 = Sprite refresh has hit sprite #0.
      |     | This flag resets to 0 when screen refresh starts
      |     | (see "PPU Details").
      |   7 | VBlank Flag, 1 = PPU is in VBlank state.
      |     | This flag resets to 0 when VBlank ends or CPU reads $2002
      |     | (see "PPU Details").
------+-----+---------------------------------------------------------------
$2003 | W   | Sprite Memory Address
      |     | Used to set the address of the 256-byte Sprite Memory to be
      |     | accessed via $2004. This address will increment by 1 after
      |     | each access to $2004. Sprite Memory contains coordinates,
      |     | colors, and other sprite attributes (see "Sprites").
------+-----+---------------------------------------------------------------
$2004 | RW  | Sprite Memory Data
      |     | Used to read/write the Sprite Memory. The address is set via
      |     | $2003 and increments by 1 after each access. Sprite Memory
      |     | contains coordinates, colors, and other sprite attributes
      |     | sprites (see "Sprites").
------+-----+---------------------------------------------------------------
$2005 | W   | Screen Scroll Offsets
      |     | There are two scroll registers, vertical and horizontal,
      |     | which are both written via this port. The first value written
      |     | will go into the Vertical Scroll Register (unless it is >239,
      |     | then it will be ignored). The second value will appear in the
      |     | Horizontal Scroll Register. Name Tables are assumed to be
      |     | arranged in the following way:
      |     |
      |     |           +-----------+-----------+
      |     |           | 2 ($2800) | 3 ($2C00) |
      |     |          +-----------+-----------+
      |     |          | 0 ($2000) | 1 ($2400) |
      |     |          +-----------+-----------+
      |     |
      |     | When scrolled, the picture may span over several Name Tables.
      |     | Remember that because of the mirroring there are only 2 real
      |     | Name Tables, not 4.
------+-----+---------------------------------------------------------------
$2006 | W   | PPU Memory Address
      |     | Used to set the address of PPU Memory to be accessed via
      |     | $2007. The first write to this register will set 8 lower
      |     | address bits. The second write will set 6 upper bits. The
      |     | address will increment either by 1 or by 32 after each
      |     | access to $2007 (see "PPU Memory").
------+-----+---------------------------------------------------------------
$2007 | RW  | PPU Memory Data
      |     | Used to read/write the PPU Memory. The address is set via
      |     | $2006 and increments after each access, either by 1 or by 32
      |     | (see "PPU Memory").
------+-----+---------------------------------------------------------------
$4000-$4013 | Sound Registers
            | See "Sound".
------+-----+---------------------------------------------------------------
$4014 | W   | DMA Access to the Sprite Memory
      |     | Writing a value N into this port causes an area of CPU memory
      |     | at address $100*N to be transferred into the Sprite Memory.
------+-----+---------------------------------------------------------------
$4015 | W   | Sound Channel Switch
      |   0 | Channel 1, 1 = enable sound.
      |   1 | Channel 2, 1 = enable sound.
      |   2 | Channel 3, 1 = enable sound.
      |   3 | Channel 4, 1 = enable sound.
      |   4 | Channel 5, 1 = enable sound.
      | 5-7 | Unused (???)
------+-----+---------------------------------------------------------------
$4016 | RW  | Joystick1 + Strobe
      |   0 | Joystick1 Data (see "Joysticks).
      |   1 | Joystick1 Presence, 0 = connected.
      | 2-5 | Unused, set to 000 (???)
      | 6-7 | Unknown, set to 10 (???)
------+-----+---------------------------------------------------------------
$4017 | RW  | Joystick2 + Strobe
      |   0 | Joystick2 Data (see "Joysticks).
      |   1 | Joystick2 Presence, 0 = connected.
      | 2-5 | Unused, set to 000 (???)
      | 6-7 | Unknown, set to 10 (???)
------+-----+---------------------------------------------------------------
```

Interrupts

NES uses non-maskable interrupts (NMIs) generated by PPU in the end of each frame (so-called VBlank interrupts). The VBlank interrupts can be enabled/disabled by writing 1/0 into 7th bit of $2000.

When a VBlank interrupt occurs, CPU pushes return address and the status register on stack, then jumps to the address stored at location $FFFA (ROM in NES).

The interrupt handler is supposed to finish its execution with RTI command which returns CPU to the main program execution. More information on the interrupt handling can be found in a decent book on 6502 CPU.

Maskable interrupts (aka IRQs) can be generated by external circuitry on the cart (see "Mappers"), but most carts do not generate them.

Joysticks

Two NES joysticks are accessed via locations $4016 and $4017 accordingly. To reset joysticks, write first $01 then $00 into $4016. This will generate a strobe in the joysticks' circuitry. Then read either from $4016 (joystick #0) or from $4017 (joystick #1). Each consequent read will return the status of a single button in the 0th bit (1 if pressed, 0 otherwise):

```
Read # |    1      2      3      4      5      6      7      8
-------+---------------------------------------------------------
Button |    A      B   SELECT   START   UP    DOWN   LEFT  RIGHT
```

1st bit indicates whether joystick is connected to the port or not. It is set to 0 if a joystick is connected, 1 otherwise. 6th and 7th bits of $4016/$4017 also seem to have some significance. The rest of bits appear to return zeroes. Some games expect to get **exactly** $41 from $4016/$4017 if a button is pressed.

PPU Memory

1. Accessing PPU Memory

In a real NES, reading/writing PPU memory should only be attempted during VBlank period. Attempts to access PPU memory during screen refresh will corrupt refresh address registers in a certain way which is often used to implement "split screen" effects (see "PPU Details").

Many smaller ROMs have read-only memory (VROM) for the Pattern Tables. In this case, you won't be able to write into this part of PPU address space, only read from it.

```
Writing to PPU memory:
a) Write upper address byte into $2006
b) Write lower address byte into $2006
c) Write data into $2007. After each write, the
   address will increment either by 1 (bit 2 of
   $2000 is 0) or by 32 (bit 2 of $2000 is 1).

Reading from PPU memory:
a) Write upper address byte into $2006
b) Write lower address byte into $2006
c) Read data from $2007. The first byte read from
   $2007 will be invalid (see "PPU Details"). Then,
   the address will increment by 1 after each read.
```

2. Name Tables

PPU supports 4 Name Tables at addresses $2000, $2400, $2800, and $2C00. A Name Table is very similar to the text mode screen buffer which contains codes of characters to display on the screen. It contains tile (character) codes organized into 30 rows of 32 bytes each. Tiles are 8x8 pixels each. Thus, the whole Name Table is 32x30 tiles or 256x240 pixels. On NTSC TVs, upper and lower 8 pixels are usually shown off screen leaving you with 256x224 pixels. In the PAL standard, the screen is whole 256x240 pixels.

It is necessary to say that although PPU supports 4 Name Tables, NES only has enough internal VRAM for 2 Name Tables. Two others are mirrored.

3. Pattern Tables

PPU supports 2 Pattern Tables at addresses $0000 and $1000. A Pattern Table contains tile images (patterns) in the following format:

```
Character   Colors      Contents of Pattern Table
...*....    00010000    00010000 $10  +-> 00000000 $00
..O.O...    00202000    00000000 $00  |   00101000 $28
.#...#..    03000300    01000100 $44  |   01000100 $44
O.....O.    20000020    00000000 $00  |   10000010 $82
*******. -> 11111110 -> 11111110 $FE  |   00000000 $00
O.....O.    20000020    00000000 $00  |   10000010 $82
#.....#.    30000030    10000010 $82  |   10000010 $82
........    00000000    00000000 $00  |   00000000 $00
                            +---------+
```

Note that only two bits for each pixel of each tile are stored in the Pattern Table. Other two are taken from the Attribute Table. Thus, the total number of simultaneous colors on the NES screen is 16, although each tile has only 4 colors.

4. Attribute Tables

Each Name Table has its own Attribute Table. A byte in this table represents a group of 4x4 tiles on screen which makes an 8x8 Attribute Table. Each 4x4 tile group is subdivided into four 2x2 squares as follows:

```
(0,0)  (1,0) 0|  (2,0)  (3,0) 1
(0,1)  (1,1)  |  (2,1)  (3,1)
--------------+----------------
(0,2)  (1,2) 2|  (2,2)  (3,2) 3
(0,3)  (1,3)  |  (2,3)  (3,3)
```

The attribute byte contains upper two bits of the color number for each 2x2 square (the lower two bits are stored in the Pattern Table):

```
Bits   Function                        Tiles
--------------------------------------------------------------
7,6    Upper color bits for square 3   (2,2),(3,2),(2,3),(3,3)
5,4    Upper color bits for square 2   (0,2),(1,2),(0,3),(1,3)
3,2    Upper color bits for square 1   (2,0),(3,0),(2,1),(3,1)
1,0    Upper color bits for square 0   (0,0),(1,0),(0,1),(1,1)
```

5. Palettes

There are two 16-byte Palettes. The one at $3F00 used for the picture. Another one, at $3F10, contains sprite colors. The $3F00 and $3F10 locations in VRAM mirror each other (i.e. it is the same memory cell) and define the background color of the picture.

PPU Details

1. Mirroring

As you may have noticed, PPU supports 4 Name Tables with corresponding Attribute Tables. Nevertheless, there is only enough internal VRAM for 2 Name Tables. Other 2 tables are going to be mirrors of the first 2.

Which pages are mirrored depends on the cartiridge circuitry. Each cartridge has control of the PPU address bits A10 and A11. It may set them in one of four possible ways:

```
A11 A10 Effect
----------------------------------------------------------
 0   0  All four screen buffers are mapped to the same
        area of memory which repeats at $2000, $2400,
        $2800, and $2C00.
 0   x  "Upper" and "lower" screen buffers are mapped to
        separate areas of memory at $2000, $2400 and
        $2800, $2C00.
 x   0  "Left" and "right" screen buffers are mapped to
        separate areas of memory at $2000, $2800 and
        $2400,$2C00.
 x   x  All four screen buffers are mapped to separate
        areas of memory. In this case, the cartridge
        must contain 2kB of additional VRAM.
```

2. VBlank Flag

The VBlank Flag is contained in the 7th bit of the PPU Status Register ($2002). It indicates whether PPU is refreshing the screen or generating a Vertical Blanking Impulse. It goes up in the end of each frame and stays on until a next screen refresh starts. When VBlank Flag is up, you can access PPU memory via $2006/$2007. The program can reset VBlank Flag prematurely by reading from the PPU Status Register ($2002).

3. Hit Flag

The Hit Flag is contained in the 6th bit of the PPU Status Register ($2002). It goes up when PPU starts displaying sprite #0 and its first non-transparent pixel coincides with the first non-transparent pixel of the background. For example, if the screen is filled with the non-transparent color (>0), sprite #0's coordinates are [12,34], and it only has pixels in the 4th line from the top, then the Hit Flag will be set when screen refresh reaches location [12,37].

The Hit Flag allows to implement both horizontal and vertical screen splits, as well as variety of other interesting effects. It is **not** reset by reading from the PPU Status Register. Instead, it appears to go down on its own every time PPU starts refreshing screen.

4. Accessing VRAM During Refresh

As was said before, accessing VRAM address ($2006) and data ($2007) registers during screen refresh is officially considered illegal. A lot of programs access these registers to produce various scrolling effects. For example, if some game has a scrolling playfield and a stationary status bar at the bottom of the screen, it may write to $2006 at the first line of the status bar to reset screen scroll.

The idea behind this trick is that PPU uses the VRAM address register to store current address during screen refresh. By writing into $2006 you effectively modify this address and make PPU continue screen refresh from a different location. Details on how $2007 affects screen refresh are unknown.

While it is not clear **how** exactly values written into $2006 relate to the screen locations, they seem to follow this diagram:

```
Address Written into $2006
xxYYSSYYYYYXXXXX
   | |  |    |
   | |  |    +---- Horizontal scroll in tiles (i.e. 1 = 8 pixels)
   | |  +--------- Vertical scroll in tiles (i.e. 1 = 8 pixels)
   | +------------ Number of Name Table ($2000,$2400,$2800,$2C00)
   +-------------- Additional vertical scroll in pixels (0..3)
```

5. Accessing PPU Status Register

To be written

6. Accessing PPU Data Register

To be written

Sprites

There are 64 sprites which can be either 8x8 or 8x16 pixels. Sprite patterns are stored in one of the Pattern Tables in the PPU Memory. Sprite attributes are stored in a special Sprite Memory of 256 bytes which is not a part of neither CPU nor PPU address space. The entire contents of Sprite Memory can be written via DMA transfer using location $4014 (see above). Sprite Memory can also be accessed byte-by-byte by putting the starting address into $2003 and then writing/reading $2004 (the address will be incremented after each access). The format of sprite attributes is as follows:

```
Sprite Attribute RAM:
| Sprite#0 | Sprite#1 | ... | Sprite#62 | Sprite#63 |
     |          |
     +---- 4 bytes: 0: Y position of the left-top corner - 1
                    1: Sprite pattern number
                    2: Color and attributes:
                       bits 1,0: two upper bits of color
                       bits 2,3,4: Unknown (???)
                       bit 5: if 1, display sprite behind background
                       bit 6: if 1, flip sprite horizontally
                       bit 7: if 1, flip sprite vertically
                    3: X position of the left-top corner
```

Sprite patterns are fetched in the exactly same way as the tile patterns for the background picture. The only difference occurs in 8x16 sprites: the top half of a sprite is taken from the Sprite Pattern Table at $0000 while the bottom part is taken from the same location of $1000 Sprite Pattern Table.

Memory Mappers

There are many different memory mappers (aka MMCs) used in the NES cartridges. They are used to switch ROM and VROM pages and for some other tasks. I will try to describe all known MMCs. Any new information on these and other MMCs is highly appreciated. MMC numbers are given in terms of the "Mapper Type" field in .NES files (see ".NES File Format").

To be written

Sound

To be written

Famicom Disk System

Famicom Disk System (FDS) is a Famicom extension unit which was produced by Nintendo and only sold in Asian countries. It consists of a disk drive accepting 2.5" floppies, 32kB of RAM (instead of ROM) to load programs into, 8kB of VRAM (instead of VROM), and some other hardware described below.

1. Memory Map

With the Famicom Disk System, the address space is laid out in a following way:

```
--------------------------------------- $10000
 8kB FDS BIOS ROM
--------------------------------------- $E000
 32kB Program RAM
--------------------------------------- $6000
 Expansion Modules
--------------------------------------- $5000
 Input/Output
--------------------------------------- $2000
 2kB Internal RAM, mirrored 4 times
--------------------------------------- $0000
```

The Famicom Disk System also adds some I/O ports into $4000-$40FF range to control the disk drive, the sound hardware, and the IRQ counter built into FDS:

```
------+-----+-------------------------------------------------------------
$4020 | W   | Lower Byte of IRQ Counter
------+-----+-------------------------------------------------------------
$4021 | W   | Upper Byte of IRQ Counter
------+-----+-------------------------------------------------------------
$4022 | W   | Enable/Disable IRQs
      |   2 | \ = Stop IRQ counter and reset its interrupt request.
      |     | / = Load IRQ counter with a value from $4020-$4021 and
      |     |     start it.
------+-----+-------------------------------------------------------------
$4024 | W   | Data Write Register
      |     | To write data to the disk, output it into this register.
------+-----+-------------------------------------------------------------
$4025 | W   | Control Register
      |   0 | Drive Motor, 0 = on, 1 = off.
      |   1 | \ = Set drive head to the start of the first track.
      |   2 | Disk Write, 0 = enabled, 1 = disabled.
      |   3 | Screen Mirroring, 0 = vertical, 1 = horizontal.
      | 4-6 | Unknown (???)
      |   7 | Disk IRQs, 0 = disabled, 1 = enabled.
------+-----+-------------------------------------------------------------
$4026 | W   | ExPort Output (???)
------+-----+-------------------------------------------------------------
$4030 | R   | Disk Status Register 0
      |   4 | Unknown (???)
      |   6 | Unknown (???)
------+-----+-------------------------------------------------------------
$4031 | R   | Data Read Register
      |     | To read data from the disk, input it from this register.
------+-----+-------------------------------------------------------------
$4032 | R   | Disk Status Register 1
      |   0 | Disk Presence, 0 = inserted, 1 = not inserted.
      |   1 | Disk Ready, 0 = ready, 1 = not ready.
      |   2 | Write Protection, 0 = unprotected, 1 = protected.
------+-----+-------------------------------------------------------------
$4033 | R   | ExPort Input
      |   7 | Battery Status, 0 = ok, 1 = low.
------+-----+-------------------------------------------------------------
```

2. IRQ Counter

FDS provides a 16bit IRQ counter connected to the CPU clock generator. This counter starts from a value preset in $4020-$4021 when you write into $4021. It will then decrement by one on each CPU clock cycle. When the counter reaches zero, it reloads from the preset value and continues counting. An IRQ is generated at this moment. It may occur during VBlank as well as during screen refresh. The 2nd bit of $4022 allows to control IRQ counter. When it is set to 0, the counter stops and a pending IRQ request is reset (if there is such request). The counting can then be resumed by setting 2nd bit of $4022 to 1.

3. Sound Hardware

To be written

4. Disk Format

Each disk has two sides, *A* and *B*, each of them containing exactly 65000 bytes of data. To change an active side, the disk has to be removed, flipped, and inserted back into the drive. The data on each side is stored in several types of blocks in a sequence `[1,2,3,4,3,4,...3,4]` where

```
-------------------------------------------------------
1. Side Header Block (56 bytes)
       0 $01
    1-14 "*NINTENDO-HVC*"
      15 Maker ID
   16-19 Side Name
      20 Version Number
      21 Side Number
         $00 = Side A
         $01 = Side B
   22-24 Additional Disk Data
      25 $08
   26-56 Reserved Space (not used by BIOS)
-------------------------------------------------------
2. File Number Block (2 bytes)
       0 $02
       1 Number of Files on this side
-------------------------------------------------------
3. File Header Block (16 bytes)
       0 $03
     1-2 File Number (not used by BIOS?)
    3-10 File Name (not used by BIOS?)
   11-12 Starting Address in memory
   13-14 File Size
      15 File Type
         $00 = Program Data
         $01 = Character Data
         $02 = Screen Data
-------------------------------------------------------
4. File Data Block (variable length)
       0 $04
   1-... Data (see File Header Block for size)
-------------------------------------------------------
```

All two-byte fields are written with the most significant byte first. Side names may be different for different sides of the same game floppy, but they are usually the same. Starting addresses are actual addresses in RAM or VRAM where files are loaded.

5. Disk Access

Following diagrams show how FDS performs reads and writes from a floppy disk. These diagrams have been created by Goroh in his document on the FDS hardware. I am providing them here with only minor cosmetic changes, with all credit going to Goroh:

```
READ:
$4025| A |  B   | C | D ||    E    |    A) Initialization
```

```
7bit |___|______|___|---||------___|    B) Motor on
6bit |___|______|---|---||------___|    C) Read start mark
5bit |---|------|---|---||---------|    D) Enable IRQs
4bit |___|______|___|___||___---___|    E) Read end mark
2bit |---|------|---|---||---------|    1. Read data, ($4030)=[xx0xxxxx]
1bit |---|---___|___|___||_________|    2. Read data, ($4030)=[xxx0xxxx]
0bit |___|------|---|---||------___|
Note |   |      |   |   || 1  2    |

WRITE:
$4025| A |  B   |  C   | D ||    E    | A) Initialization
7bit |___|______|______|---||------___| B) Motor on
6bit |___|______|___---|---||------___| C) Write start mark
5bit |---|------|------|---||---------| D) Enable IRQs
4bit |___|______|______|___||___---___| E) Write end mark
2bit |---|------|______|___||______---| 1. Delay, write [00000000]
1bit |---|---___|______|___||_________| 2. Write [10000000]
0bit |___|------|------|---||------___| 3. Write data, ($4030)=[xx0xxxxx]
Note |   |      |  1   | 2 || 3  4    | 4. Delay
```

It is not clear from the diagrams how actual reading and writing is done. It is quite clear though that FDS generates an IRQ every time a byte is read from the disk. FDS IRQ handler (which is a part of the FDS BIOS) reads this byte from $4031 while the head is advanced to the next byte.

6. Disk Errors

FDS BIOS may display different errors when accessing a floppy disk. The list of errors given below has been taken from Goroh's document and verified on an FDS emulator to the best of my abilities. Errors whose meaning is not clear to me are marked with question marks.

```
ERR.01 No disk
ERR.02 No power
ERR.03 Broken prong on the disk
ERR.04 Wrong maker ID
ERR.05 Wrong game name
ERR.06 Wrong game version
ERR.07 Wrong side number (flip the disk)
ERR.08 Wrong disk #1
ERR.09 Wrong disk #2
ERR.10 Wrong disk #3
ERR.20 Allows it to recognize screen data differs (???)
ERR.21 Wrong Side Header Block ("*NINTENDO-HVC*")
ERR.22 Wrong Side Header Block ID ($01)
ERR.23 Wrong File Number Block ID ($02)
ERR.24 Wrong File Header Block ID ($03)
ERR.25 Wrong File Data Block ID ($04)
ERR.26 Error writing data to the disk
ERR.27 Block ends prematurely
ERR.28 The disk unit and the same period can't take it (???)
ERR.29 The disk unit and the same period can't take it (???)
ERR.30 Disk full
ERR.31 Data number of a disk doesn't match up (???)
```

.NES File Format

.NES files are used to store images of NES cartridges for emulators. The .NES format has been first implemented by me, Marat Fayzullin, in iNES and then adopted by other emulator authors. It has become a de-facto standard of storing NES ROM images. Following is the structure of a .NES file:

```
Byte     Contents
---------------------------------------------------------------------------
0-3      String "NES^Z" used to recognize .NES files.
4        Number of 16kB ROM banks.
5        Number of 8kB VROM banks.
6        bit 0     1 for vertical mirroring, 0 for horizontal mirroring.
         bit 1     1 for battery-backed RAM at $6000-$7FFF.
         bit 2     1 for a 512-byte trainer at $7000-$71FF.
         bit 3     1 for a four-screen VRAM layout.
         bit 4-7   Four lower bits of ROM Mapper Type.
7        bit 0     1 for VS-System cartridges.
         bit 1-3   Reserved, must be zeroes!
         bit 4-7   Four higher bits of ROM Mapper Type.
8        Number of 8kB RAM banks. For compatibility with the previous
         versions of the .NES format, assume 1x8kB RAM page when this
         byte is zero.
9        bit 0     1 for PAL cartridges, otherwise assume NTSC.
         bit 1-7   Reserved, must be zeroes!
10-15    Reserved, must be zeroes!
16-...   ROM banks, in ascending order. If a trainer is present, its
         512 bytes precede the ROM bank contents.
...-EOF  VROM banks, in ascending order.
---------------------------------------------------------------------------
```

Note that this format will most likely expand in future, therefore do not take it for something permanent. All new additions are guaranteed to be compatible with the older versions of a format though.

The 8 bits allocated for the mapper number give us a total of 256 possible mapper types. If you happen to find a new type of a memory mapper, please, email me its description and several sample ROMs and I will allocate and announce a new number for it. Do not pick the number yourself, as this will cause a general mess.

Following is a table of assigned mapper types. The ones with the "-" sign are not currently supported by iNES in a proper way.

Mapper#	Name	Examples
0	No mapper	All 32kB ROM + 8kB VROM games
1	Nintendo MMC1	Megaman2, Bomberman2, etc.
2	CNROM switch	Castlevania, LifeForce, etc.
3	UNROM switch	QBert, PipeDream, Cybernoid, many Japanese games
4	Nintendo MMC3	SilverSurfer, SuperContra, Immortal, etc.
5	Nintendo MMC5	Castlevania3
6	FFE F4xxx	F4xxx games off FFE CDROM
7	AOROM switch	WizardsAndWarriors, Solstice, etc.
8	FFE F3xxx	F3xxx games off FFE CDROM
9	Nintendo MMC2	Punchout
10	Nintendo MMC4	Punchout2
11	ColorDreams chip	CrystalMines, TaginDragon, etc.
12 -	FFE F6xxx	F6xxx games off FFE CDROM
13	CPROM switch	
15	100-in-1 switch	100-in-1 cartridge
16	Bandai chip	Japanese DragonBallZ series, etc.
17	FFE F8xxx	F8xxx games off FFE CDROM
18	Jaleco SS8806 chip	Japanese Baseball3, etc.
19	Namcot 106 chip	Japanese GhostHouse2, Baseball90, etc.
20	Nintendo DiskSystem	Reserved. Don't use this mapper!
21	Konami VRC4a	Japanese WaiWaiWorld2, etc.
22	Konami VRC2a	Japanese TwinBee3
23	Konami VRC2a	Japanese WaiWaiWorld, MoonWindLegend, etc.
24 -	Konami VRC6	
25	Konami VRC4b	
32	Irem G-101 chip	Japanese ImageFight, etc.
33	Taito TC0190/TC0350	Japanese PowerBlazer
34	Nina-1 board	ImpossibleMission2 and DeadlyTowers
64	Tengen RAMBO-1 chip	
65	Irem H-3001 chip	
66	GNROM switch	
67	SunSoft3 chip	
68	SunSoft4 chip	
69	SunSoft5 FME-7 chip	
71	Camerica chip	
78	Irem 74HC161/32-based	
79	AVE Nina-3 board	KrazyKreatures, DoubleStrike, etc.
81	AVE Nina-6 board	Deathbots, MermaidsOfAtlantis, etc.

```
91       Pirate HK-SF3 chip
---------------------------------------------------------------------------
```
