PPU rendering

From Nesdev wiki

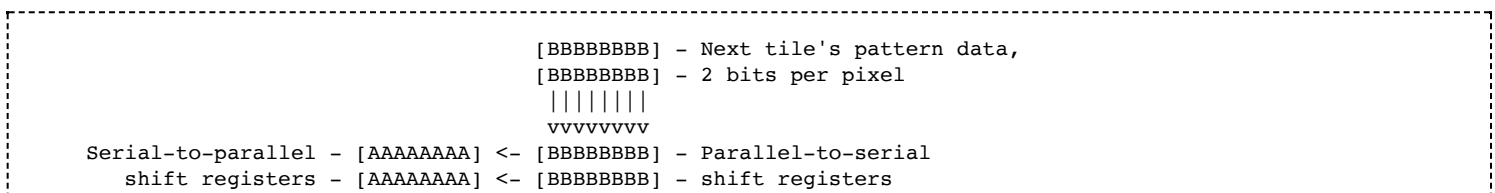
Contents

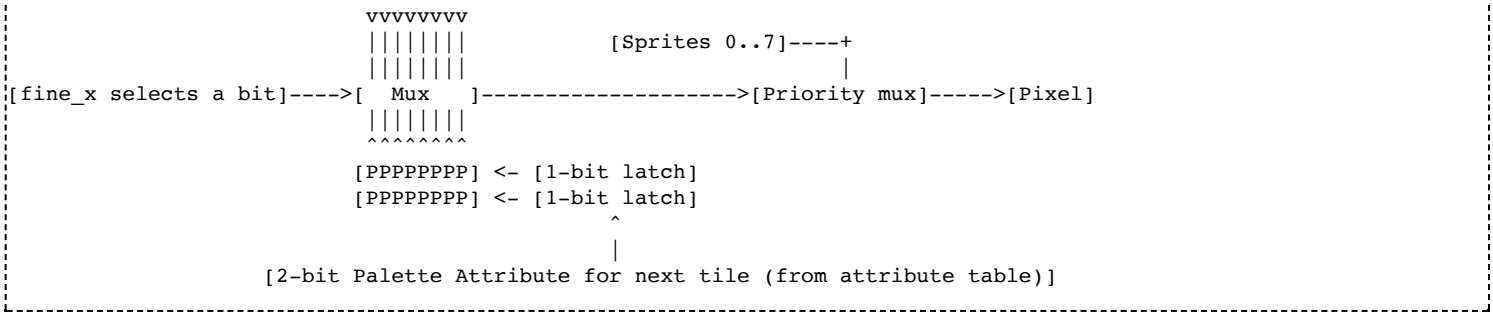
- 1 Preface
- 2 Line-by-line timing
  - 2.1 Pre-render scanline (-1, 261)
  - 2.2 Visible scanlines (0-239)
    - 2.2.1 Cycle 0
    - 2.2.2 Cycles 1-256
    - 2.2.3 Cycles 257-320
    - 2.2.4 Cycles 321-336
    - 2.2.5 Cycles 337-340
  - 2.3 Post-render scanline (240)
  - 2.4 Vertical blanking lines (241-260)
- 3 PPU address bus contents
- 4 Frame timing diagram
- 5 See also
- 6 References

Preface

The PPU contains the following:

- Background
  - VRAM address, temporary VRAM address, fine X scroll, and first/second write toggle - This controls the addresses that the PPU reads during background rendering. See PPU scrolling.
  - 2 16-bit shift registers - These contain the bitmap data for two tiles. Every 8 cycles, the bitmap data for the next tile is loaded into the upper 8 bits of this shift register. Meanwhile, the pixel to render is fetched from one of the lower 8 bits.
  - 2 8-bit shift registers - These contain the palette attributes for the lower 8 pixels of the 16-bit shift register. These registers are fed by a latch which contains the palette attribute for the next tile. Every 8 cycles, the latch is loaded with the palette attribute for the next tile.
- Sprites
  - Primary OAM (holds 64 sprites for the frame)
  - Secondary OAM (holds 8 sprites for the current scanline)
  - 8 pairs of 8-bit shift registers - These contain the bitmap data for up to 8 sprites, to be rendered on the current scanline. Unused sprites are loaded with an all-transparent bitmap.
  - 8 latches - These contain the attribute bytes for up to 8 sprites.
  - 8 counters - These contain the X positions for up to 8 sprites.





Every cycle, a bit is fetched from the 4 background shift registers in order to create a pixel on screen. Exactly which bit is fetched depends on the fine X scroll, set by $2005 (this is how fine X scrolling is possible). Afterwards, the shift registers are shifted once, to the data for the next pixel.

Every 8 cycles/shifts, new data is loaded into these registers.

Every cycle, the 8 x-position counters for the sprites are decremented by one. For each sprite, if the counter is still nonzero, nothing else happens. If the counter is zero, the sprite becomes "active", and the respective pair of shift registers for the sprite is shifted once every cycle. This output accompanies the data in the sprite's latch, to form a pixel. The current pixel for each "active" sprite is checked (from highest to lowest priority), and the first non-transparent pixel moves on to a multiplexer, where it joins the BG pixel.

Priority multiplexer decision table

BG pixel	Sprite pixel	Priority	Output
0	0	X	BG ($3F00)
0	1-3	X	Sprite
1-3	0	X	BG
1-3	1-3	0	Sprite
1-3	1-3	1	BG

- If the sprite has foreground priority or the BG pixel is zero, the sprite pixel is output.
- If the sprite has background priority and the BG pixel is nonzero, the BG pixel is output. (**Note:** Even though the sprite is "behind the background", it was *still* the the highest priority sprite to have a non-transparent pixel, and thus the only sprite to be looked at. Therefore, the BG pixel is output even if another foreground priority sprite is present at this pixel. This is where the sprite priority quirk comes from.)

Line-by-line timing

The PPU renders 262 scanlines per frame. Each scanline lasts for 341 PPU clock cycles (113.667 CPU clock cycles; 1 CPU cycle = 3 PPU cycles), with each clock cycle producing one pixel. The line numbers given here correspond to how the internal PPU frame counters count lines.

The information in this section is summarized in the diagram in the next section.

The timing below is for NTSC PPUs. PPUs for 50 Hz TV systems differ: PAL NES PPUs render 70 vblank scanlines instead of 20, and Dendy PPUs render 51 post-render scanlines instead of 1.

Pre-render scanline (-1, 261)

This is a dummy scanline, whose sole purpose is to fill the shift registers with the data for the first two tiles of the next scanline. Although no pixels are rendered for this scanline, the PPU *still* makes the same memory accesses it would for a regular scanline.

This scanline varies in length, depending on whether an even or an odd frame is being rendered. For odd frames, the cycle at the end of the scanline is skipped (this is done internally by jumping directly from (339,261) to (0,0), replacing the idle tick at the beginning of the first visible scanline with the last tick of the last dummy nametable fetch). For even frames, the last cycle occurs normally. This is done to compensate for some shortcomings with the way the PPU physically outputs its video signal, the end result being a crisper image when the screen isn't scrolling. However, this behavior can be bypassed by keeping rendering disabled until after this scanline has passed, which results in an image that looks more like a traditionally interlaced picture.

During pixels 280 through 304 of this scanline, the vertical scroll bits are reloaded if rendering is enabled.

Visible scanlines (0-239)

These are the visible scanlines, which contain the graphics to be displayed on the screen. This includes the rendering of both the background and the sprites. During these scanlines, the PPU is busy fetching data, so the program should *not* access PPU memory during this time, unless rendering is turned off.

Cycle 0

This is an idle cycle. The value on the PPU address bus during this cycle appears to be the same CHR address that is later used to fetch the low background tile byte starting at dot 5 (possibly calculated during the two unused NT fetches at the end of the previous scanline).

Cycles 1-256

The data for each tile is fetched during this phase. Each memory access takes 2 PPU cycles to complete, and 4 must be performed per tile:

1. Nametable byte
2. Attribute table byte
3. Tile bitmap low
4. Tile bitmap high (+8 bytes from tile bitmap low)

The data fetched from these accesses is placed into internal latches, and then fed to the appropriate shift registers when it's time to do so (every 8 cycles). Because the PPU can only fetch an attribute byte every 8 cycles, each sequential string of 8 pixels is forced to have the same palette attribute.

Sprite zero hits act as if the image starts at cycle 2 (which is the same cycle that the shifters shift for the first time), so the sprite zero flag will be raised at this point at the earliest. Actual pixel output is delayed further due to internal render pipelining, and the first pixel is output during cycle 4.

The shifters are reloaded during ticks 9, 17, 25, ..., 257.

Note: At the beginning of each scanline, the data for the first two tiles is already loaded into the shift registers (and ready to be rendered), so the first tile that gets fetched is Tile 3.

While all of this is going on, sprite evaluation for the *next* scanline is taking place as a seperate process, independent to what's happening here.

Cycles 257-320

The tile data for the sprites on the *next* scanline are fetched here. Again, each memory access takes 2 PPU cycles to complete, and 4 are performed for each of the 8 sprites:

1. Garbage nametable byte
2. Garbage nametable byte
3. Tile bitmap low
4. Tile bitmap high (+8 bytes from tile bitmap low)

The garbage fetches occur so that the same circuitry that performs the BG tile fetches could be reused for the sprite tile fetches.

If there are less than 8 sprites on the next scanline, then dummy fetches to tile $FF occur for the left-over sprites, because of the dummy sprite data in the secondary OAM (see sprite evaluation). This data is then discarded, and the sprites are loaded with a transparent bitmap instead.

In addition to this, the X positions and attributes for each sprite are loaded from the secondary OAM into their respective counters/latches. This happens during the second garbage nametable fetch, with the attribute byte loaded during the first tick and the X coordinate during the second.

Cycles 321-336

This is where the first two tiles for the *next* scanline are fetched, and loaded into the shift registers. Again, each memory access takes 2 PPU cycles to complete, and 4 are performed for the two tiles:

1. Nametable byte
2. Attribute table byte
3. Tile bitmap low
4. Tile bitmap high (+8 bytes from tile bitmap low)

Cycles 337-340

Two bytes are fetched, but the purpose for this is unknown. These fetches are 2 PPU cycles each.

1. Nametable byte
2. Nametable byte

Both of the bytes fetched here are the same nametable byte that will be fetched at the beginning of the next scanline (tile 3, in other words). At least one mapper is known to use this string of three consecutive nametable fetches to clock a scanline counter.

Post-render scanline (240)

The PPU just idles during this scanline. Even though accessing PPU memory from the program would be safe here, the VBlank flag isn't set until *after* this scanline.

Vertical blanking lines (241-260)

The VBlank flag of the PPU is set at tick 1 (the *second* tick) of scanline 241, where the VBlank NMI also occurs. The PPU makes no memory accesses during these scanlines, so PPU memory can be freely accessed by the program.

PPU address bus contents

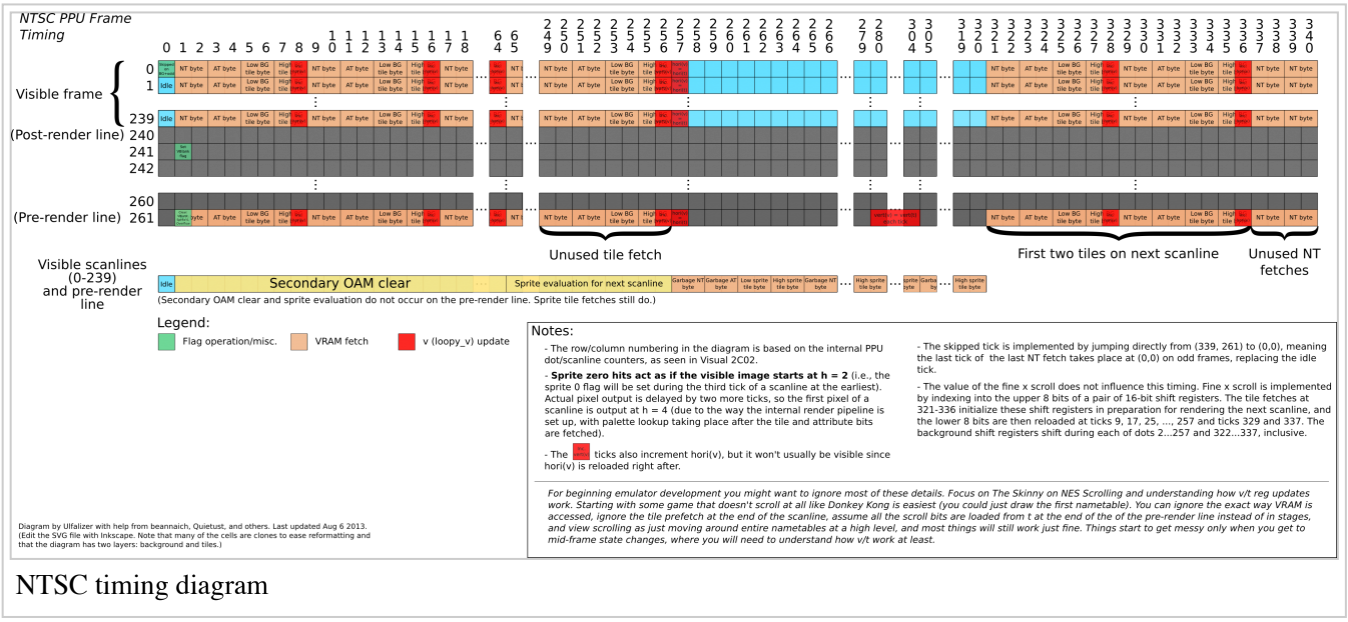
During frame rendering, provided rendering is enabled (i.e., when either background or sprite rendering is enabled in $2001:3-4), the value on the PPU address bus is as indicated in the descriptions above and in the frame timing diagram below. During VBlank and when rendering is disabled, the value on the PPU address bus is the current value of the v register.

To save pins, the PPU multiplexes the lower eight VRAM address pins, also using them as the VRAM data pins. This leads to each VRAM access taking two PPU cycles:

1. During the first cycle, the entire VRAM address is output on the PPU address pins and the lower eight bits stored in an external octal latch by asserting the ALE (Address Latch Enable) line. (The octal latch is the lower chip to the right of the PPU in this wiring diagram.)
2. During the second cycle, the PPU only outputs the upper six bits of the address, with the octal latch providing the lower eight bits (VRAM addresses are 14 bits long). During this cycle, the value is read from or written to the lower eight address pins.

As an example, the PPU VRAM address pins will have the value $2001 followed by the value $20AB for a read from VRAM address $2001 that returns the value $AB.

Frame timing diagram



NTSC timing diagram

(Source SVG file)

See also

- Nametable
- Attribute table
- NTSC video
- PPU frame timing

References

- NTSC 2C02 technical reference (http://nesdev.com/2C02%20technical%20reference.TXT)
