# PPU frame timing

From Nesdev wiki

The following behavior is tested by the ppu_vbl_nmi_timing test ROMs. Only the NTSC PPU is covered, though most probably applies to the PAL PPU.

## Contents



## Even/Odd Frames

- The PPU has an even/odd flag that is toggled every frame, regardless of whether rendering is enabled or disabled.
- With rendering disabled (background and sprites disabled in PPUMASK ($2001)), each PPU frame is 341*262=89342 PPU clocks long. There is no skipped clock every other frame.
- With rendering enabled, each odd PPU frame is one PPU clock shorter than normal. This is done by skipping the first idle tick on the first visible scanline (by jumping directly from (339,261) on the pre-render scanline to (0,0) on the first visible scanline and doing the last cycle of the last dummy nametable fetch there instead; see this diagram).
- By keeping rendering disabled until after the time when the clock is skipped on odd frames, you can get a different color dot crawl pattern than normal (it looks more like that of interlace, where colors flicker between two states rather than the normal three). Presumably Battletoads (and others) encounter this, since it keeps the BG disabled until well after this time each frame.

## CPU-PPU Clock Alignment

The NTSC PPU runs at 3 times the CPU clock rate, so *for a given power-up* PPU events can occur on one of three relative alignments with the CPU clock they fall within. Since the PPU divides the master clock by four, there are actually more than just three alignments possible: The beginning of a CPU tick could be offset by 0-3 master clock ticks from the nearest following PPU tick. The results below only cover one particular set of alignments, namely the one which gives the fewest number of special cases, where a read will see a change to a flag if and only if it starts at or after the PPU tick where the flag changes. (Other alignments might cause the change to be visible 1 PPU tick earlier or later; see this thread (http://forums.nesdev.com/viewtopic.php?p=62253).)

### Synchronizing the CPU and PPU clocks

If rendering is off, each frame will be 341*262/3 = 29780 2/3 CPU clocks long. If the CPU checks the VBL flag in a loop every 29781 clocks, the read will occur one PPU tick later relative to the start of the frame each frame, until at some point the CPU "catches up" to the location where the flag gets set. At this point, the CPU and PPU synchronization is known down the PPU tick.

```
During frame 5 below, the CPU will read the VBL flag as set, and the loop will stop.

Frame 1: ...-C---V-...
Frame 2: ...--C--V-...
Frame 3: ...---C-V-...
Frame 4: ...----CV-...
Frame 5: ...-----*-...

-: PPU tick
C: Location where the CPU starts reading $2002
V: Location where the VBL flag is set in $2002
*: Beginning of $2002 read synched with VBL flag setting

(This assumes the alignment with the fewest number of special cases as mentioned above.)
```

# VBL Flag Timing

*See also: NMI*

- Reading $2002 within a few PPU clocks of when VBL is set results in special-case behavior. Reading one PPU clock before reads it as clear and never sets the flag or generates NMI for that frame. Reading on the same PPU clock or one later reads it as set, clears it, and suppresses the NMI for that frame. Reading two or more PPU clocks before/after it's set behaves normally (reads flag's value, clears it, and doesn't affect NMI operation). This suppression behavior is due to the $2002 read pulling the NMI line back up too quickly after it drops (NMI is active low) for the CPU to see it. (CPU inputs like NMI are sampled each clock.)
- On an NTSC machine, the VBL flag is cleared 6820 PPU clocks, or exactly 20 scanlines, after it is set. In other words, it's cleared at the start of the pre-render scanline. (*TO DO: confirmation on PAL NES and common PAL famiclone*)

# See Also

- PPU rendering



---