NES

Zach Schuermann, Jeff Jaquith, Minghao Li
Nintendo Entertainment System
NES Subsystems

- CPU (6502)
- Memory (RAM/ROM)
- PPU (picture processing unit)
  - Background rendering
  - Sprite rendering
- APU (audio processing unit)
- Controllers
NES Subsystems
ultraNES Subsystems

Controller — CPU — APU — PPU

CPU bus

RAM — ROM

PPU bus

PPU bus

VRAM — ROM

NTSC out
ultraNES Subsystems
ultraNES Subsystems

CPU

PPU

VRAM

VGA

CPU bus

RAM

ROM

ROM

VGA out
ultraNES Subsystems

CPU

PPU

VRAM

ROM

RAM

CPU bus

VGA out

VGA

Avalon Bus

Linux Memory-Mapped Device
Goals

Main goal: PPU
- CPU integration
- ROM loading / user interface
- Stretch: controller support
Contributions

1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
ultraNES Subsystems

- CPU
- PPU
- RAM
- VRAM
- CPU bus
- VGA out
- ROM
- Linux Memory-Mapped Device
- Avalon Bus
1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
PPU

- Mainly comprised of:
  - Tile rendering
  - Sprite rendering
  - Internal state/communication with CPU

- 32x30 tiles for background rendering
  - 8x8 pixels per tile

- 64 sprites for a given frame
  - 8 sprites per scanline

- Priority mux for tile and sprite pixel output

- Internal VRAM which is modified (indirectly) by CPU
PPU Subsystems

CPU data, address signals

PPU FSM + Registers

Tile Renderer

Priority MUX

Sprite Renderer

VRAM

ROM

Palette

hsync vsync

Pixel out

PPU
<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPUCTRL</td>
<td>$2000</td>
<td>NMI enable (V), PPU master/slave (P), sprite height (H), background tile select (B), sprite tile select (S), increment mode (I), nametable select (NN)</td>
</tr>
<tr>
<td>PPUMASK</td>
<td>$2001</td>
<td>color emphasis (BGR), sprite enable (s), background enable (b), sprite left column enable (M), background left column enable (m), greyscale (G)</td>
</tr>
<tr>
<td>PPUSTATUS</td>
<td>$2002</td>
<td>vblank (V), sprite 0 hit (S), sprite overflow (O); read resets write pair for $2005/$2006</td>
</tr>
<tr>
<td>OAMADDR</td>
<td>$2003</td>
<td>OAM read/write address</td>
</tr>
<tr>
<td>OAMDATA</td>
<td>$2004</td>
<td>OAM data read/write</td>
</tr>
<tr>
<td>PPUSCROLL</td>
<td>$2005</td>
<td>fine scroll position (two writes: X scroll, Y scroll)</td>
</tr>
<tr>
<td>PPUADDR</td>
<td>$2006</td>
<td>PPU read/write address (two writes: most significant byte, least significant byte)</td>
</tr>
<tr>
<td>PPUDATA</td>
<td>$2007</td>
<td>PPU data read/write</td>
</tr>
<tr>
<td>OAMDMA</td>
<td>$4014</td>
<td>OAM DMA high address</td>
</tr>
</tbody>
</table>
PPU State Machine

- Model PPU’s state as FSM.

- Controls PPU current state
  - VRAM fetch state
    - nametable
    - attribute
    - low and high byte in pattern table
  - Set control flags
PPU Background

- **Nametable**
  - 8x8 pixel tiles a total of 32x30 tiles. Each tile is a single byte: index into the pattern table

- **Pattern table**
  - Each index has 16 bytes, low and high combined to form a pattern table

- **Attribute table**
  - Contains index into the palette RAM
  - Each block has four tiles, and each block is a single byte in attribute table

- **Palette RAM**
  - 8 palettes and each sub-palette has 4 colors
  - 0-3 for background and 4-7 for sprites
PPU Sprite

- 64 sprites in any given frame and 8 sprites per scanline
- A sprite has 4 bytes that can be accessed in the OAM
  - x pos, y pos, tile and attribute index
- In-position sprites are stored in a secondary OAM (8 sprites)
  - Then loaded to 8 shift registers to be displayed
  - Counter will count down to 0 to load the next scanline
Donkey Kong Tile Rendering Example
Donkey Kong Tile Rendering Example
Mario Sprite Rendering Example

OAM

Combining Pattern Table

[Table and diagrams]
PPU Rendering Figures

- PPU renders 262 scan lines per frame
  - 240 visible scan lines
  - 20 fetching data (vblank)
  - 2 dummy
- Only can write one pixel per PPU cycle
  - Takes 341 PPU cycles per scanline
  - 256 for rendering; remaining are used to fetch data from nametables, etc.
  - (2 clock cycles per pfetch, PPU multiplexes bottom 8 VRAM Address pins to also use as data pins)
- For each frame:
  - -1 scanline: prefetch tile info for first two tiles
  - 0-239 scanline: render background and sprite
  - 240 scanline: idle
  - 241-260 scanline: vblank lines, CPU can access VRAM
- For each visible scanline:
  - 0 cycle: idle
  - 1-256 cycle: visible pixels
    - Output pixel based on VRAM
    - Prefetch next tiles
    - Sprite evaluation for next scanline
  - 257-340: prefetch tile data for next line’s first two tiles
VGA

1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
VGA

- Scanbuffer hold 2 full scanlines
- Dual clocking
- Renders two VGA scanlines for every PPU scanline
- VGA runs 4x the speed and ‘renders’ 4x the pixels
  - Doubled horizontal resolution
  - Doubled vertical resolution
VGA

PPU data, counters → scanbuf → LUT → RGB

- 256x2 array
- vga_counter

hsync vsync → VGA
CPU/RAM

1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
CPU

- Pre-existing 6502 core implemented in Verilog
- 8-bit data bus and 16-bit address bus
- Communicates with the PPU through memory-mapped registers into CPU address space
- Tested using functional regression tests (Klaus Dormann’s)
- Simulated with Verilator + tested on FPGA
RAM/ROM

- SystemVerilog implementations reliant on Quartus software to infer RAM blocks.
- Utilize dual-port RAM to ease multiple access
- Combine RAM+ROM in many cases
- Avalon Bus writes to ROM
Memory Map: Memory and Nametable Mirroring

- Full address is not fully decoded to reduce hardware space
  - Same byte being accessed at multiple addresses
- Vertical and Horizontal Mirroring for scrolling and rendering off screen at distance.

<table>
<thead>
<tr>
<th>Program ROM</th>
<th>Program ROM Upper Bank</th>
<th>$6000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Program ROM Lower Bank</td>
<td>$4000</td>
</tr>
<tr>
<td>SRAM</td>
<td>SRAM</td>
<td>$4000</td>
</tr>
<tr>
<td>Expansion ROM</td>
<td>Expansion ROM</td>
<td>$2008</td>
</tr>
<tr>
<td>I/O Registers</td>
<td>I/O Registers</td>
<td>$2000</td>
</tr>
<tr>
<td>RAM</td>
<td>RAM</td>
<td>$1000</td>
</tr>
<tr>
<td>Zero Page</td>
<td></td>
<td>$0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mirrors</th>
<th>Mirrors $0000-$3FF</th>
<th>$10000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$3F00-$3FF</td>
<td>$4000</td>
</tr>
<tr>
<td>Palettes</td>
<td>Sprite Palette</td>
<td>$3F20</td>
</tr>
<tr>
<td></td>
<td>Image Palette</td>
<td>$3F10</td>
</tr>
<tr>
<td>Name Tables</td>
<td>Attribute Table 1</td>
<td>$3F00</td>
</tr>
<tr>
<td></td>
<td>$2000-$21FF</td>
<td>$1000</td>
</tr>
<tr>
<td></td>
<td>Name Table 1</td>
<td>$2FC0</td>
</tr>
<tr>
<td></td>
<td>Name Table 2</td>
<td>$2FC0</td>
</tr>
<tr>
<td></td>
<td>Attribute Table 2</td>
<td>$2BC0</td>
</tr>
<tr>
<td></td>
<td>$2000-$7FF</td>
<td>$1000</td>
</tr>
<tr>
<td>Pattern Tables</td>
<td>Name Table 0</td>
<td>$2000</td>
</tr>
<tr>
<td></td>
<td>Pattern Table 0</td>
<td>$0000</td>
</tr>
<tr>
<td></td>
<td>$0000-$3FF</td>
<td>$0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mirrors</th>
<th>Mirrors $0000-$3FF</th>
<th>$10000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$3F00-$3FF</td>
<td>$4000</td>
</tr>
<tr>
<td>Palettes</td>
<td>Sprite Palette</td>
<td>$3F20</td>
</tr>
<tr>
<td></td>
<td>Image Palette</td>
<td>$3F10</td>
</tr>
<tr>
<td>Name Tables</td>
<td>Attribute Table 1</td>
<td>$3F00</td>
</tr>
<tr>
<td></td>
<td>$2000-$21FF</td>
<td>$1000</td>
</tr>
<tr>
<td></td>
<td>Name Table 1</td>
<td>$2FC0</td>
</tr>
<tr>
<td></td>
<td>Name Table 2</td>
<td>$2FC0</td>
</tr>
<tr>
<td></td>
<td>Attribute Table 2</td>
<td>$2BC0</td>
</tr>
<tr>
<td></td>
<td>$2000-$7FF</td>
<td>$1000</td>
</tr>
<tr>
<td>Pattern Tables</td>
<td>Name Table 0</td>
<td>$2000</td>
</tr>
<tr>
<td></td>
<td>Pattern Table 0</td>
<td>$0000</td>
</tr>
<tr>
<td></td>
<td>$0000-$3FF</td>
<td>$0000</td>
</tr>
</tbody>
</table>
Integration

1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
Timing Figures

- PPU is 4 times slower than the VGA
  - Each PPU frame will take 89,342 PPU cycles
  - Each VGA frame will take 357,368 VGA cycles
- 50 MHz global clock
- 25 MHz VGA clock (50/2)
- 6.25 MHz PPU clock (50/8)
- 2.083 MHz CPU clock (50/24)
Timing Figures

- Facilitated via global clock + clock enables
- Each clocked module requires 50MHz clock and subsystem-specific clock enable
Linux Userspace Utilities

1. PPU (incomplete)
2. VGA subsystem
3. Integration with CPU/RAM
4. Linux userspace utilities
Linux Userspace Utilities

Three main components:

1. Avalon bus interface to FPGA
2. Linux device driver for memory-mapped access to Avalon bus
3. Userspace utility to issue IOCTL’s to modify RAM/ROM onboard FPGA
Linux Userspace Utilities

- Installer script to build device driver, install kernel module, and install pre-compiled userspace utility
- User interface: `ultranes` binary

```
$ ultranes            # print current value on address bus
$ ultranes reset 1    # set CPU reset high
$ ultranes reset 0    # set CPU reset low
$ ultranes load cpu bin/test.hex # load test.hex ROM into CPU RAM
$ ultranes load ppu bin/vtest.hex # load vtest.hex ROM into PPU RAM
$ ultranes load dk bin/dk.nes # load dk.nes ROM into CPU+PPU RAM
$ ultranes load mario bin/mario.nes # load mario.nes ROM into CPU+PPU RAM
$ ultranes write 55 128 # write value 55 to address 128
$ ultranes help       # call in backup
```
Linux Userspace Utilities

root@de1-soc:~# ultranes --help
ultranes 0.1.0
Zack Schuermann

**ULTRANES**

**USAGE:**
ultranes [SUBCOMMAND]

**FLAGS:**
- **h**, **--help**  Prints help information
- **V**, **--version**  Prints version information

**SUBCOMMANDS:**
- **help**  Prints this message or the help of the given subcommand(s)
- **load**  load ROM
- **reset**  set reset high/low
- **write**  write value to address
Current Status

- CPU integrated
- Device driver + userspace program
- PPU framework
- Background rendering
- Memory + ROM’s
- Clocking regression
- Sprites unimplemented
Planned

- PPU background testing + debug
- PPU sprite rendering

Future work:

- Controller interfacing
- [non-goal] audio/APU
Lessons Learned

- Test + integrate ASAP
- Clocking + synchronization
- Differences between systemverilog semantics and inferred hardware
- Subsystem division
- Test/debug via hardware
- Validate early and often
- Workflow for easy compilation/programming is essential
Thank you!