6.111 Final Project
Jonathan Downey
Lauri Kauppila
Brian Myhre
Project Motivation

- Distinct Sub-systems
- Complex Behavior and Architecture
- FPGA Required
  - Not Possible with Microcontroller
  - Large Amount of Logic
- Entertainment Value
- Appreciated by Non-Technical People
NES Background

- 1983, NES Introduced in Japan
- 1985, NES released in US
- 1995, Discontinued Production of NES
- Nintendo sold over 62 million NES systems and 500 million games
- Currently the most widely emulated system with over thirty different emulators.
NES System Overview

- **CPU: Custom 8-bit 2A03**
  - Based on the 6502 core running at 1.79 MHz
  - Onboard Audio: 2 Square Waves, 1 Triangle, 1 Digital, 1 Noise
  - Input from Control Pads

- **Main Ram: 2 KB**

- **PPU: Picture Processing Unit**
  - Sprite and background generation
  - 2 KB Video Ram
  - 64 Colors
  - 256 by 240 pixel resolution

- **Removable Game Cartridges**
  - Program Data ROM: 32KB to 1MB
  - Graphical Data ROM: 8KB to 1MB
CPU: The Historic 6502

- Designed by MOS Technologies, a division of Commodore
- Released in 1975 for $25
- In the first PCs such as the Commodore 64, Apple I and Apple II.
- Landmark in Microprocessor design.
  - New “pipelining” led to no wasted memory cycles and amazing speed.
  - Fast Access to RAM, less registers
- Efficient design of the 6502 is said to have inspired the development of the ARM processors
6502 Specifics

- 8-bit Bi-directional Data Bus
- 16-bit Address Bus (64 KB)
- Complex Instruction Set with over 50 Instructions
- Eleven Addressing Modes
- Operation up to 2 MHz
CPU Control

- Reset Line
- IRQ Line
- NMI Line

- Reset
- Interrupt Controller
- Clock Generator
- Timing Control

- Instruction Decode
  - OPCODE
  - ADDRESSING

- Control Signals
- INSTR
- INSTR DECODE
- 21 MHz Clock
  - Clock 1
  - Clock 2
- Clock 1
- Clock 2
- Instruction Complete
- Cycle
- INSTR
- RESET_INT
- IRQ
- NMI

- Reset Clear
- INSTR
- DECODE
- OPCODE
- ADDRESSING

- Cycle
- Instruction Complete
- Clock 2
- Clock 1
- INSTR
- INSTR DECODE
- Control Signals

- Reset Line
- IRQ Line
- NMI Line

- Reset
- Interrupt Controller
- Clock Generator
- Timing Control

- Instruction Decode
  - OPCODE
  - ADDRESSING

- Control Signals
- INSTR
- INSTR DECODE
- 21 MHz Clock
  - Clock 1
  - Clock 2
- Clock 1
- Clock 2
- Instruction Complete
- Cycle
- INSTR
- RESET_INT
- IRQ
- NMI

- Reset Clear
- INSTR
- DECODE
- OPCODE
- ADDRESSING

- Cycle
- Instruction Complete
- Clock 2
- Clock 1
- INSTR
- INSTR DECODE
- Control Signals
ALU Functions:
0: A
1: B
2: Add
3: Subtract
4: AND
5: OR
6: XOR
7: Shift A Left
8: Shift A Right
9: Rotate A Left
10: Rotate A Right
11: Shift B Left
12: Shift B Right
13: Rotate B Left
14: Rotate B Right
Address Bus

[Diagram showing Address Bus with labels for ABH, PCH, Zero Page, Stack Page, Vector Page, ADH, BAH, PCL, Reset Vector (H), Interrupt Vector (H), NMI Vector (H), Reset Vector (L), Interrupt Vector (L), NMI Vector (L), Stack Pointer (S), ADL, BAL, ABH_SEL, ABL_SEL, and the concept of address selection.]
Data Bus

Input Data

LATCH_INPUT

Input Latch

INSTR_SEL

Instruction Buffer

INSTR

Input Data

NMI

 IRQ

INSTR_SEL

INSTR

Instruction Buffer

Input Data

INSTR_SEL

INSTR

Instruction Buffer

Input Data

LATCH_INPUT

Input Latch

R

Output Data Latch

R/W

Tri-state Driver

Data Bus

clk2

Out

Input Data

~R/W

Tri-state Driver

Input Data

Data Bus
Picture Processing Unit

- Works 3 times faster than CPU, a whopping 5.37 MHz
- 256 x 240 resolution
- 64 colors
- Contains:
  - ~300 bytes of internal RAM
  - 2kb of external RAM (inside NES)
  - 8kb or more of external ROM (inside cartridge)
Background Rendering

- 32 x 30 “tiles” in one screen
- Each tile contains 8x8 pixels
- Background rendered ~ in real time
- Four steps to picture creation:
  1. Fetch a pattern for the tile (no color information yet)
  2. Fetch an “attribute” for the tile (tells what range of colors exist)
  3. Fetch colors from a color table
  4. Output pixel
### Example

#### "NAME TABLE"

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

#### "PATTERN TABLE"

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Coloring

- 4 palettes for background. Each palette has 3 unique colors and a standard background color.

- Each 2 x 2 tile area uses the same palette. This “palette select” data is stored in an “attribute table”.

- 4-bit pattern data determines what color to choose inside a palette.
64 sprites (8x8 pixel tiles) can be shown per frame. 8 sprites per line (due to time constraints)

Sprite RAM (inside PPU) contains:
- 8-bit x-coordinate
- 8-bit y-coordinate
- Vertical/horizontal flip
- Pattern table address
- 2-bit palette selection
Example

“PATTERN TABLE”

Sprite memory
- coordinates
- pattern
- attributes (flip, color)
Final Output

- Background and sprite pixels are overlaid, with specific priority instructions
- Send CPU a “hit flag” for object collision, “more-than-8-sprites-on-one-line flag”
PPU Architecture
PPU Timing / Memory Access

**Background engine:**
128 memory fetches for real-time background rendering

**Sprite engine:**
- determines what sprites are in range for *next* scanline
- outputs previously determined sprites

**Background engine:**
Fetches information for first two tiles on *next* scanline

**Sprite engine:**
finds all relevant data for in-range sprites (includes external memory fetch), and prepares output phase

- h_blank goes high
- h_blank goes low
PPU Timing / Memory Access

**Background engine:**
- Fetches memory for first two tiles

**Sprite engine:**
- Determines what sprites are in range for first scanline

[do nothing for 1 scanline]

v_blank goes low

v_blank goes high

[repeating scanline processing]
Possible Extensions

- Audio Processing Unit
- Ability to Load Multiple Games
6.111 Final Project
Jonathan Downey
Lauri Kauppila
Brian Myhre