NES FPGA Emulator

UCR CS 179J

SUMMER 2014

- Sergio Morales
- Hector Dominguez
- Omar Torres
- Randy Truong
- Kevin Mitton

Who are We?

- Computer Science and Engineering Students trying to implement an NES emulator on an FPGA.

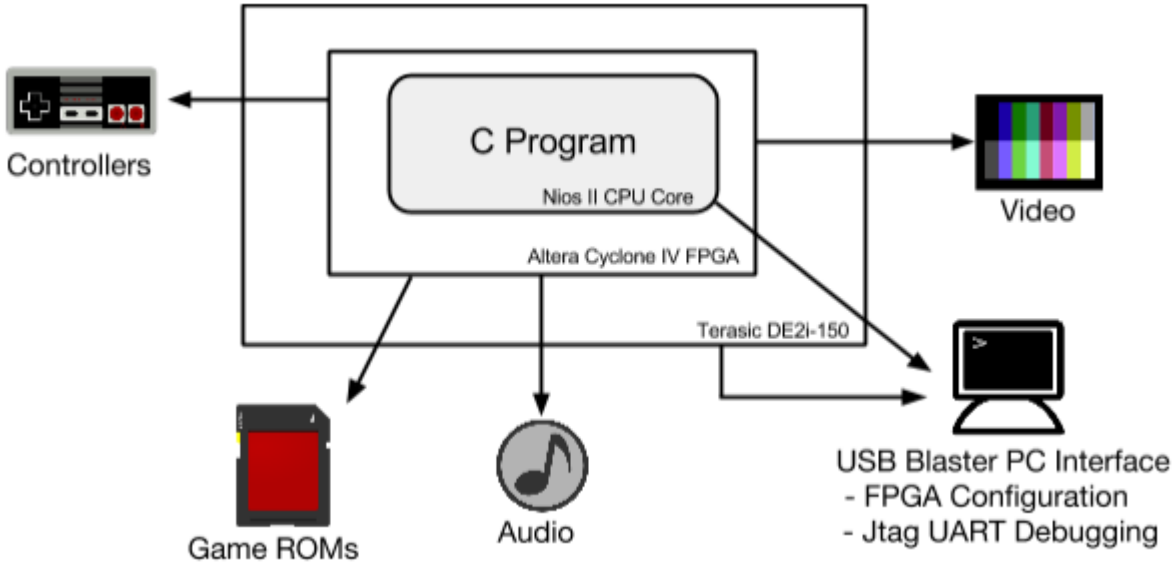
Outline

- Introduction
- The NES Central Processing Unit
- The NES Picture Processing Unit
- Hardware Implementations

  - VGA Display

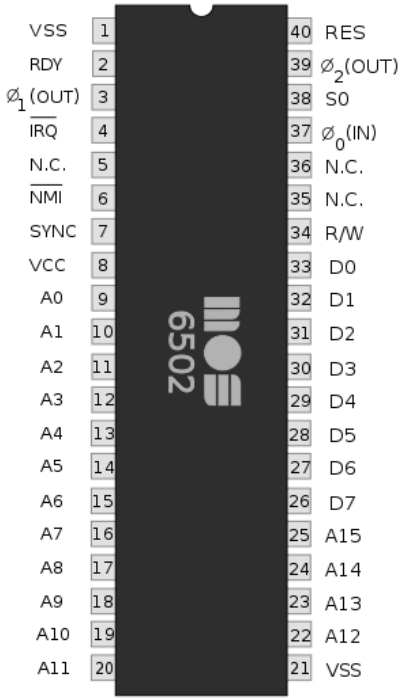
  -Controllers
- The NES pseudo-Audio Processing Unit

Introduction



The NES CPU - RP2A03

- Stock NMOS 6502
  - Lacks BCD mode
- Memory Mapped I/O
- pseudo-Audio Processing
- 1.79 Mhz Clock



The NES CPU - Memory Map

- Program Space, AKA PRG ROM
- SRAM, for save games
- Expansion ROM
- I/O Registers
- RAM
- Stack
- Zero Page

The NES CPU - ISA

- Registers
  - 3 special purpose registers
    - PC, P, and S
  - 3 general purpose
    - A, X and Y
- P, or Status Register is crucial for most instructions

The NES CPU - ISA

Bit	Flag	Description
0	C - Carry	This flag is set to ‘1’ if the last instruction, most likely arithmetic operations such as ADC caused a carry.
1	Z - Zero	Set to ‘1’ if last instruction caused a result of 0.
2	I - Interrupt Disable	Set to ‘1’ for IRQ interrupts to be disabled. See ‘**Power-up State and Interrupt Handling**’ for more info on interrupts.
3	D - Enable BCD	Not used in NES. Normally this would cause add and subtract instructions to operate in BCD mode.
4	B - Break flag	This flag only exists in BRK operations, when the flags are pushed to the stack.
5	Unused	
6	V - Overflow	Set to ‘1’ if last instruction resulted in an overflow.
7	N- Negative	Set to ‘1’ if last instruction resulted in bit 7 of the result to be 1.

The NES CPU - ISA

- Addressing Modes
  - 13 different addressing modes
- Around 50 instructions
- Combining unique instructions…
  - Over 150 total instructions

The NES CPU - Interrupts

- Interrupt Service Request (IRQ)
- Non-maskable Interrupt (NMI)
- Reset (RES)

Interrupt	Location
NMI	$FFFA-$FFFB
Reset	$FFFC-$FFFD
IRQ	$FFFE-$FFFF

The NES CPU - Implementation

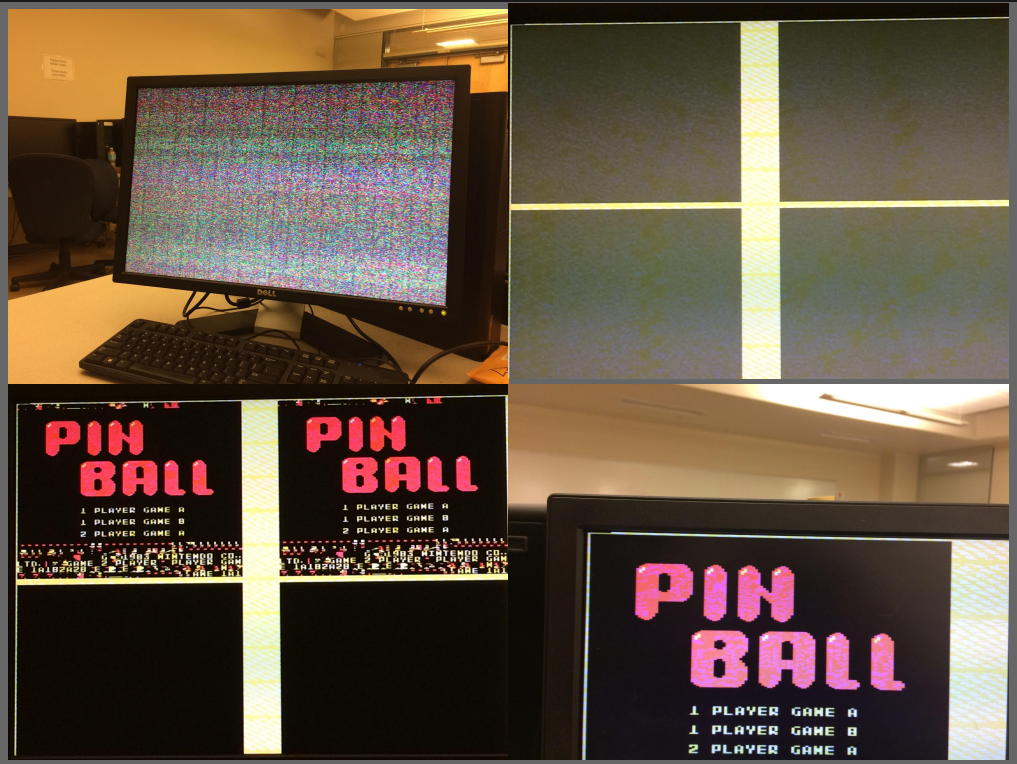
- Case statements for opcodes
  - AKA Jump table
- Check for interrupts in main loop
- Hard code instruction count + page boundary
- Memory mapping checks on writes and reads

The NES CPU - Implementation

- Sync Picture Processing Unit with CPU
  - Keep track of CPU cycles versus PPU cycles
- Debug, debug, debug!

Picture Processing Unit (PPU)

- Memory Map
- Pattern Tables
- Nametable/Attribute Tables
- Color Palettes
- Object Attribute Memory (OAM)
- Registers
- Timing



PPU Memory Map

PPU Memory Map		
Address	Size	Description
$0000	$1000	Pattern Table 0
$1000	$1000	Pattern Table 1
$2000	$3C0	Name Table 0
$23C0	$40	Attribute Table 0
$2400	$3C0	Name Table 1
$27C0	$40	Attribute Table 1
$2800	$3C0	Name Table 2
$2BC0	$40	Attribute Table 2
$2C00	$3C0	Name Table 3
$2FC0	$40	Attribute Table 3
$3000	$F00	Mirror of 2000h-2EFFh
$3F00	$10	BG Palette
$3F10	$10	Sprite Palette
$3F20	$E0	Mirror of 3F00h-3F1Fh

Object Attribute Memory (OAM)

Address	*Size*	*Description*
0x0000	*256 Bytes*	*Sprite Data*

Pattern Tables

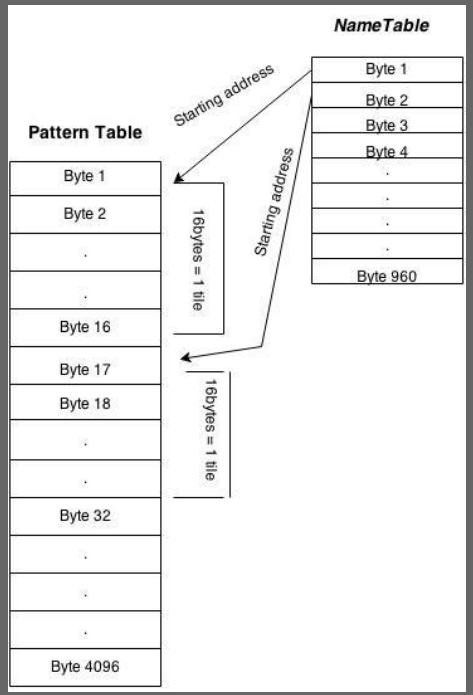
PPU Memory Map		
Address	Size	Description
$0000	$1000	Pattern Table 0
$1000	$1000	Pattern Table 1
$2000	$3C0	Name Table 0
$23C0	$40	Attribute Table 0
$2400	$3C0	Name Table 1
$27C0	$40	Attribute Table 1
$2800	$3C0	Name Table 2
$2BC0	$40	Attribute Table 2
$2C00	$3C0	Name Table 3
$2FC0	$40	Attribute Table 3
$3000	$F00	Mirror of 2000h-2EFFh
$3F00	$10	BG Palette
$3F10	$10	Sprite Palette
$3F20	$E0	Mirror of 3F00h-3F1Fh

Address	Value	Address	Value
$0000	00010000	$0008	00000000
$0001	00000000	$0009	00101000
$0002	01000100	$000A	01000100
$0003	00000000	$000B	10000010
$0004	11111110	$000C	00000000
$0005	00000000	$000D	10000010
$0006	10000010	$000E	10000010
$0007	00000000	$000F	00000000

- Each 4096 Bytes long
- hold lower two color bits

Result
00010000
00202000
03000300
20000020
11111110
30000020
30000030
00000000

Nametable/Attribute Tables



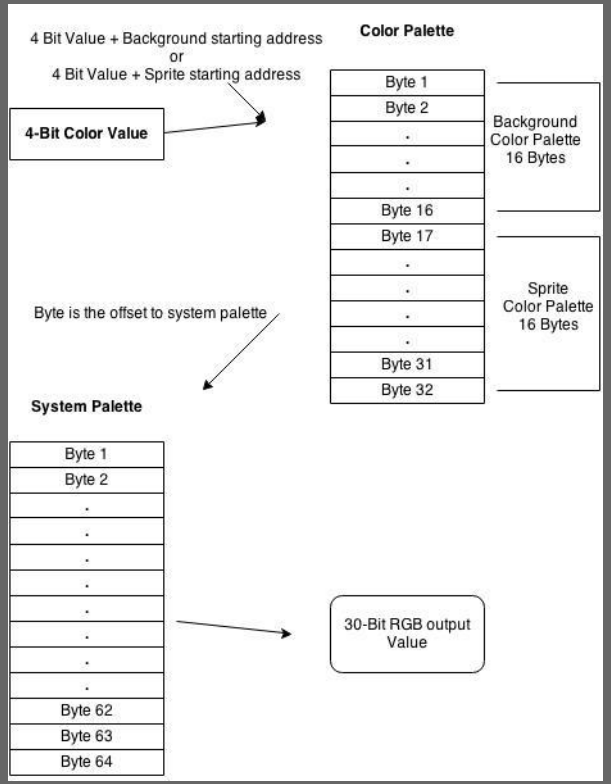
1	2	3	4	5	6	7	8	11	12	.	.	30	30	31	32
33	34	35	36	37	38	39	40	43	44	.	.	61	62	63	64
65	66	67	68	69	70	71	72	75	76	.	.	93	94	95	96
97	98	99	100	101	102	103	104	105	106	.	.	125	126	127	128
129	130	131	132	133	134	135	136	137	138	.	.	157	158	159	160
161	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	928
929	930	931	932	933	934	935	936	.	.	.	.	.	.	.	960

1	2	3	4
33	34	35	36
65	66	67	68
97	98	99	100

1	2	3	4
33	34	35	36
65	66	67	68
97	98	99	100

Color from example above	Bits from Attribute example above	Tile Numbers
RED	0 0	1, 2 , 33, 34
GREEN	0 1	3, 4. 35, 36
PURPLE	1 0	65, 66, 97, 98
BLAKCK	1 1	67, 68, 99, 100

Color Palettes



- The PPU has two color palettes one for the background tiles, and for the sprite tiles.
- The lowest two bits were obtained from the pattern table and the upper two bits were obtained from the attribute table.
- The NES does not output in RGB format, instead it outputs in NTSC format.
- We created a converter NTSC to RGB.

Object Attribute Memory (OAM)

Object Attribute Memory (OAM)		
Address	*Size*	*Description*
0x0000	*256 Bytes*	*Sprite Data*

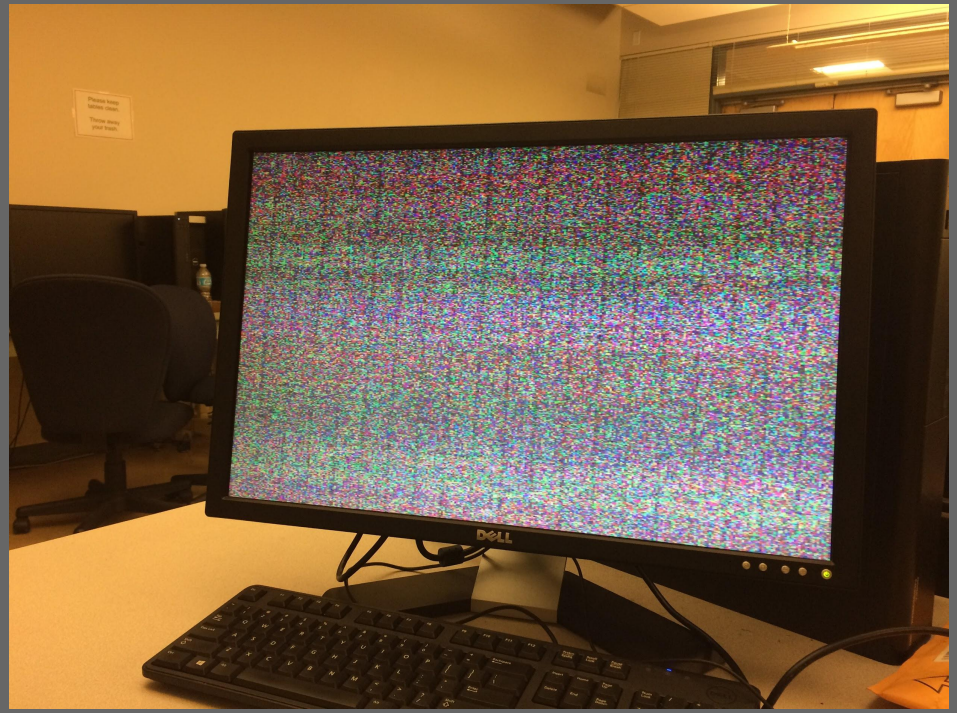
- Extra memory block outside the PPU and CPU memory maps
- contains all of the sprite information to be displayed at that exact frame to be rendered.
- The sprite data is stored on the pattern tables while the sprite attributes are stored on this extra memory block.
- stores a total of up to 64 sprites
  - Each sprite in the OAM is composed of 4 bytes
    - Byte 0: Y Position of the top left of sprite minus one
    - Byte 1: The Tile index number within the pattern table
    - Byte 2: Attributes of the sprite (palette, priority, flipping)
    - Byte 3: X Position of left side of the sprite to be rendered

Registers

1. PPUCTRL $2000
    a. Holds Flags to control PPU Operations.
        i. Base nametable address, sprite sze, generate NMI.
2. PPUMASK $2001
    a. controls screen enable, masking.
        i. Show Background
3. STATUS $2002
    a. reflects the state of various functions inside the PPU.
        i. Vblank
4. OAMADDR $2003
    a. Address of where the OAM data will go to.
5. OAMDATA $2004
    a. Sprite data to be written on the address specified on register $2003
6. PPUSCROLL $2005
    a. This register is used to change the scroll position
7. PPUADDR $2006
    a. This register writes the address of where background data will go to.
8. PPUDATA $2007
    a. Data that will be stored on the address specified by register $2006

Timing

- The PPU renders 262 scan lines per frame
  - 240 Visible scanlines
  - 20 Fetching data (Called Vblank)
  - 2 dummy scanlines
- only allowed to write one pixel for every PPU cycle
  - Takes 341 PPU cycles per scanline
    - 256 for rendering and remaining to fetch the data from nametables



Picture Processing Unit

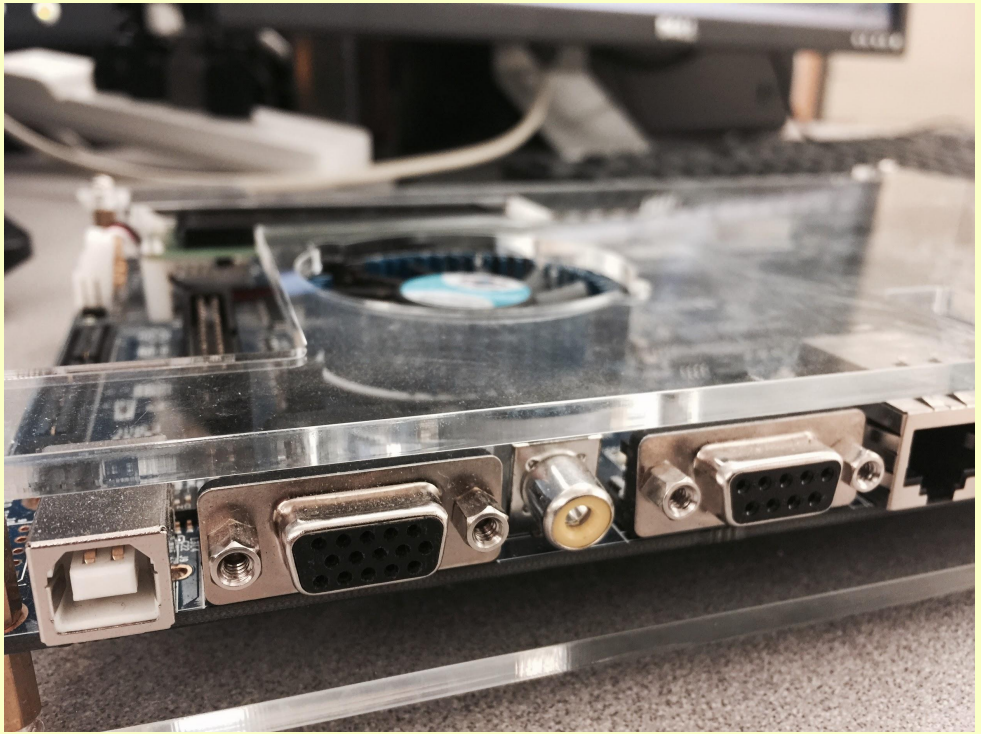
- What we could not do

Hardware

- VGA
- Controllers

VGA

- NES graphics via FPGA VGA ports
- Began with VGA Controller on Nios II Processor

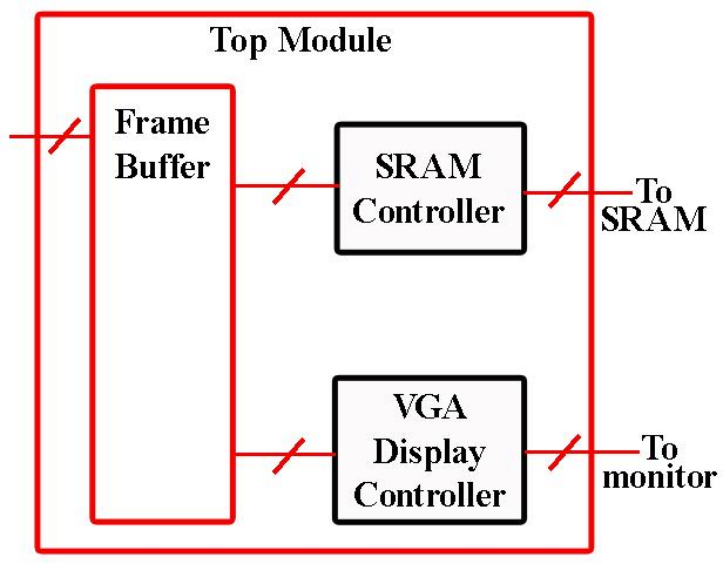


VGA

- Emulator supports 256x240 resolution
- NTSC is not supported by vga controller
- Used RGB instead

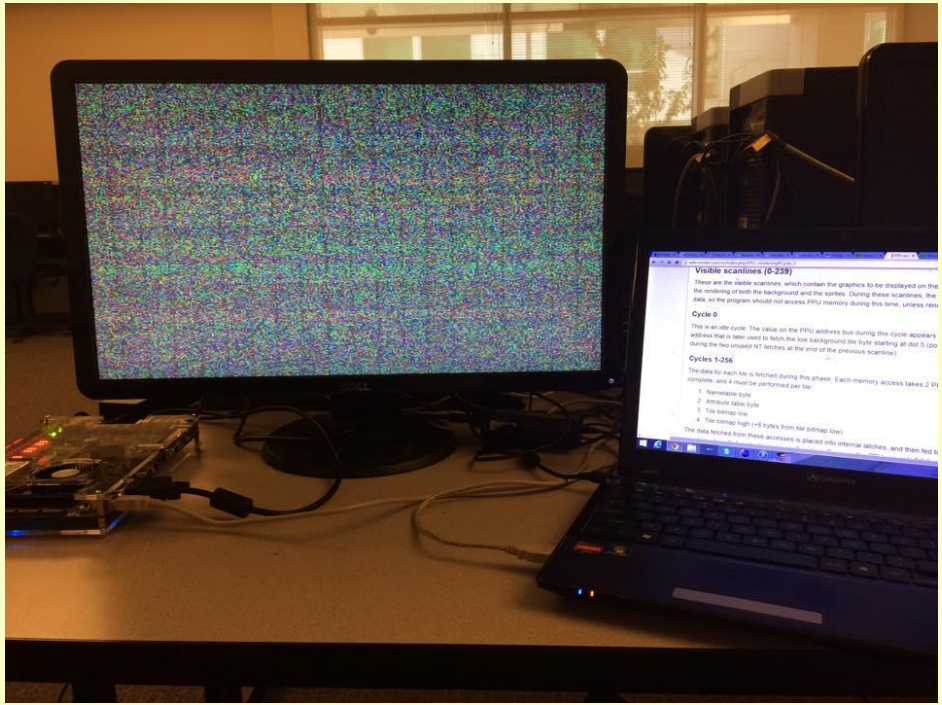
VGA

- Frame buffer used for reading and writing at a fast rate.



VGA

Testing Vga controller with random sram data.



VGA

- Nios running to slow with computer interface
- Need to check other options

VGA

- Verilog!
- First time working with Verilog.

  -Error message “No inputs detected”
  - Black screens no RGB Signals
  - Timing issues

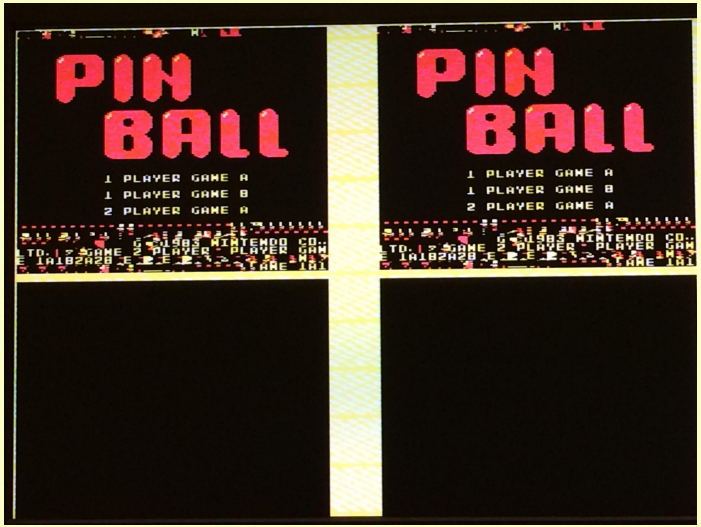
VGA

- Using incorrect clock.
- Fixed timing issues.

  -Finally Correct RGB output to the screen.

VGA

- Decided to use verilog, because it would write to the screen at a faster rate.

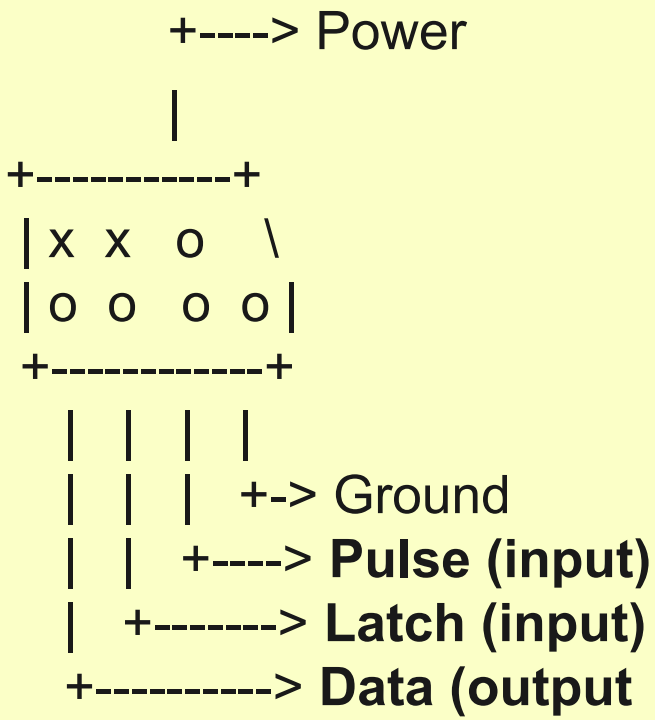


Controllers

What is it?

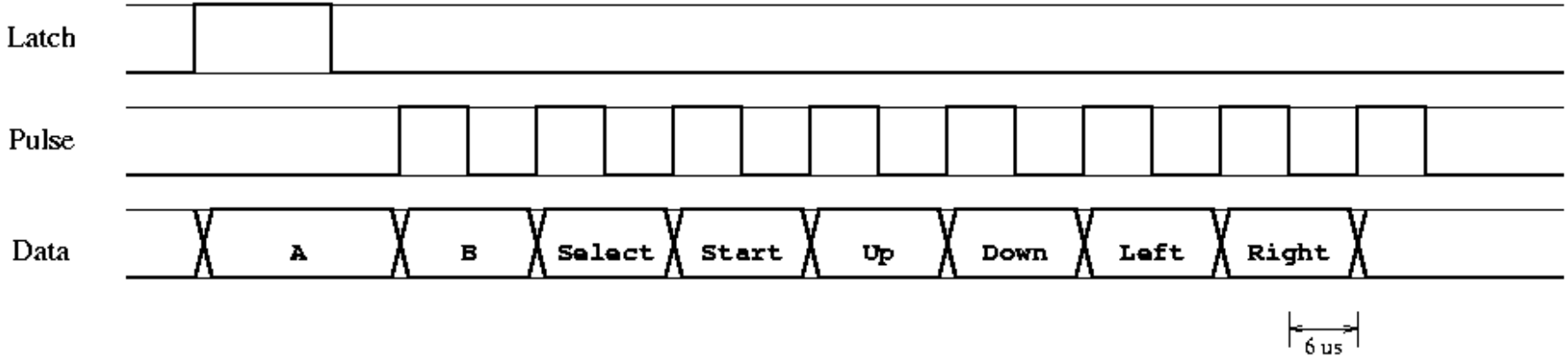
-8 buttons for inputs

Controllers



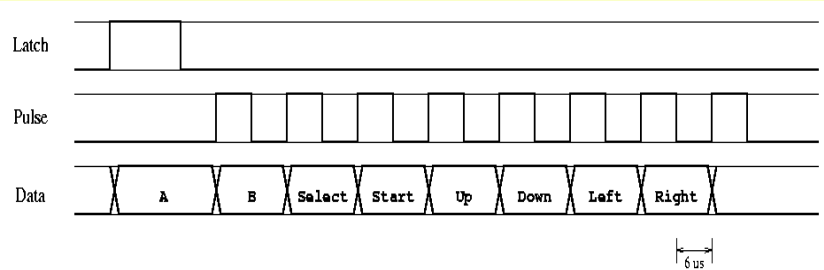
Controllers

- NES



Controllers

- NES States:
  17 States



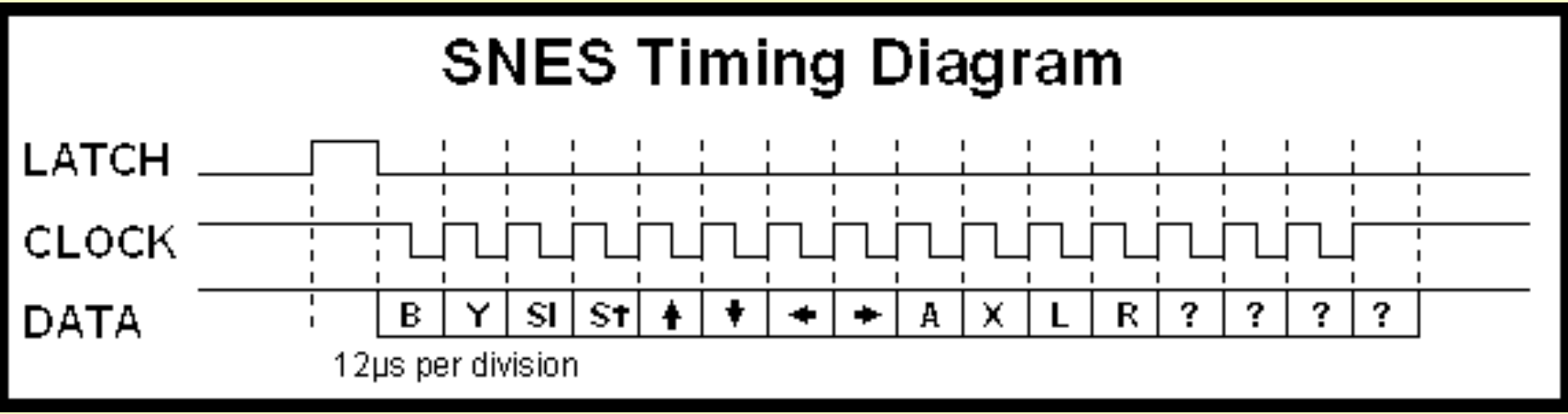
#	STATES (Transitions)	Data (Actions)
1	Latch On 1	
2	Latch On 2	
3	Latch Off	buttons[0] = A state
4	(PULSE STARTS) B On	
5	B Off	buttons[1] = B state
6	Select On	
7	Select Off	buttons[2] = Select state
8	Start On	
9	Start Off	buttons[3] = Start state
10	Up On	
11	Up Off	buttons[4] = Up state
12	Down On	
13	Down Off	buttons[5] = Down state
14	Left On	
15	Left Off	buttons[6] = Left state
16	Right On	
17	Right Off	buttons[7] = Right state

Controllers

- SNES
  - 12 buttons for inputs

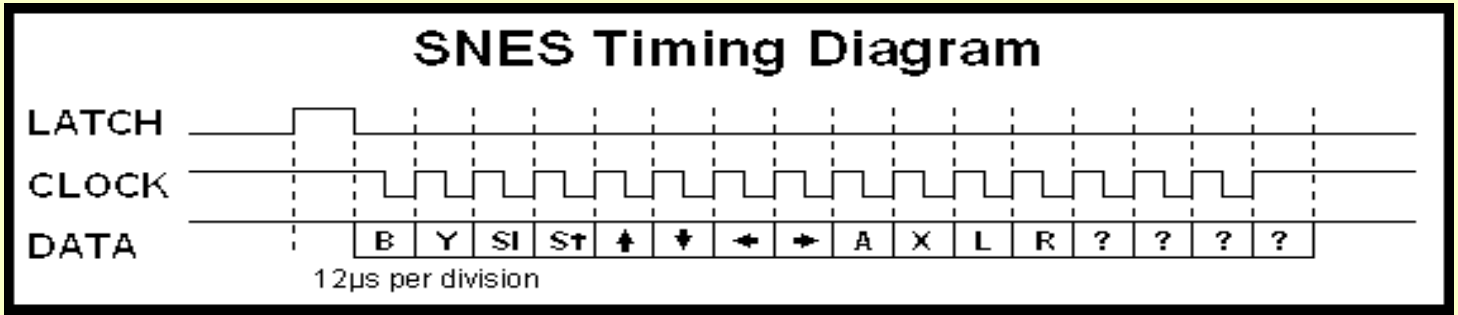
Controllers

- SNES



Controllers

- SNES
  - -34 states total
  - -However, 8 states are inactive
  - -Therefore, 26 active states



Controllers

- Stored bits into array buttons
- NES button array: (Active low)

buttons[0] = A
buttons[1] = B
buttons[2] = Select
buttons[3] = Start
buttons[4] = Up
buttons[5] = Down
buttons[6] = Left
buttons[7] = Right

Controllers

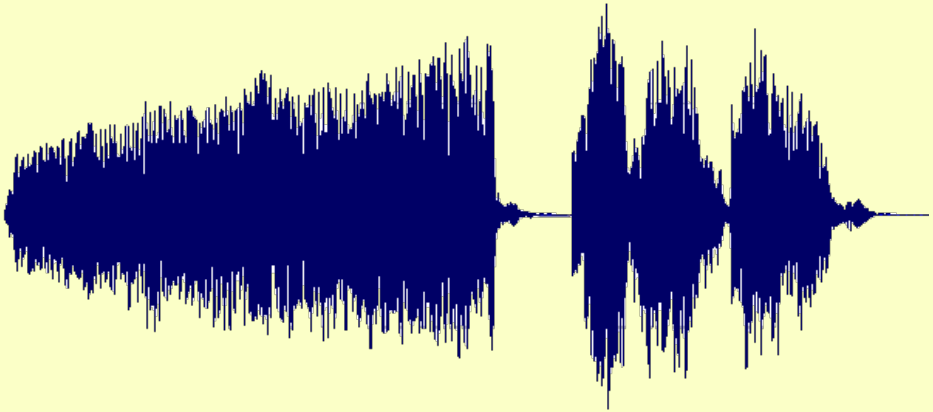
- Example:

  Bits: 01010011 for NES controller have buttons:

  A, B, Up, and Left pressed.

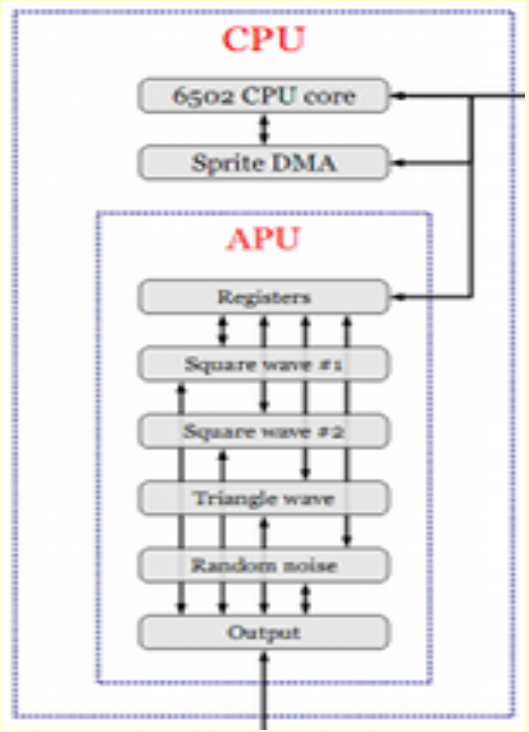
Audio Processing Unit

- How Sounds are generated for NES



Audio Processing Unit

- APU and CPU on same chip



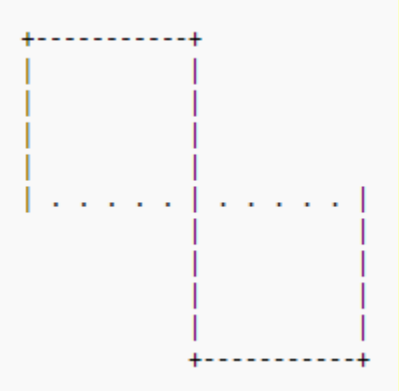
Audio Processing Unit

- 20 Registers
- 5 Channels
  - Rectangle 1
  - Rectangle 2
  - Triangle
  - Noise
  - DMC

Register Name	Size	Address Offset	Initial State	Description
Register4000	8	0	0x10	Rectangle channel 1 register 1
Register4001	8	4	0x00	Rectangle channel 1 register 2
Register4002	8	8	0x00	Rectangle channel 1 register 3
Register4003	8	12	0x00	Rectangle channel 1 register 4
Register4004	8	16	0x10	Rectangle channel 2 register 1
Register4005	8	20	0x00	Rectangle channel 2 register 2
Register4006	8	24	0x00	Rectangle channel 2 register 3
Register4007	8	28	0x00	Rectangle channel 2 register 4
Register4008	8	32	0x10	Triangle channel register 1
Register400A	8	40	0x00	Triangle channel register 2
Register400B	8	44	0x00	Tirangle channel register 3
Register400C	8	48	0x10	Noise channel register 1
Register400E	8	56	0x00	Noise channel register 2
Register400F	8	60	0x00	Noise channel register 3
Register4010	8	64	0x10	DMC channel register 1
Register4011	8	68	0x00	DMC channel register 2
Register4012	8	72	0x00	DMC channel register 3
Register4013	8	76	0x00	DMC channel register 4
Register4015	8	84	0x0f	Status register
Register4017	8	92	0x00	Frame register

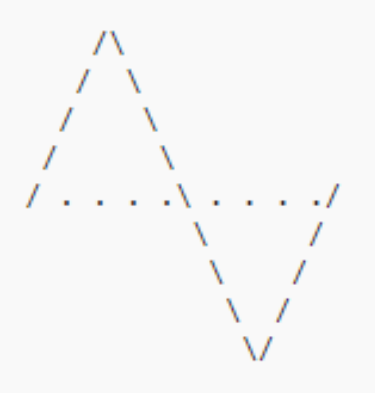
Audio Processing Unit

- Square/Rectangle Channels



Audio Processing Unit

- Triangle Channels



Audio Processing Unit

- Noise Channel

Audio Processing Unit

- DMC Channel

Thank You

Questions?