NES emulation on FPGA

Team A8:
Diego Rodriguez
Nikolai Lenney
Oscar Ramirez Poulat

Use Case

- Faithfully recreate NES console on FPGA
- Play NES games with original controllers
- Support subset of NES games that use mapper-0 (NROM)
- Save game progress onto flash memory
- Load game progress from flash

Use Case - continued

- Why?
  - Bridge gap between software and hardware emulators (save states in hardware)
  - Keep retro consoles alive by porting them to modern technology
- Areas:
  - Hardware Design (CPU, PPU)
  - Signal processing (APU)
  - Software (testing framework)

Requirements

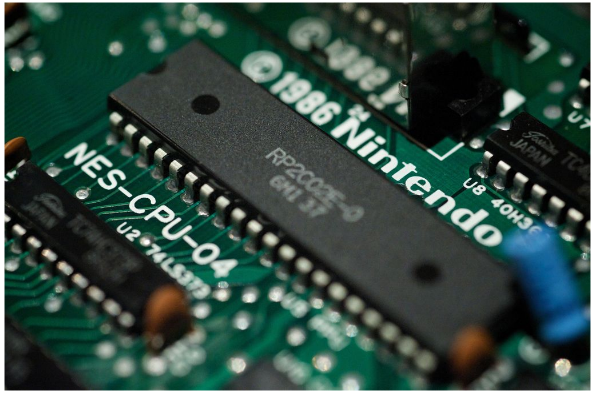
- Player state gets saved to onboard flash memory
- Player state gets loaded from onboard flash memory
- User can use an original NES controller to play a game
- System can load .nes ROMs that use mapper-0
- System will display a 256x240 image to a monitor through VGA
- Frame difference w.r.t. software emulator is less than 1%
- Achieve an average of at least 55 FPS

Requirements - continued

- PPU registers match the values of the emulator at a particular cycle
- CPU registers match the values of the emulator at a particular cycle
- OAM (Sprite RAM) trace (256 bytes) matches emulator at a particular cycle
- CPU RAM trace of load and store matches emulator at a particular cycle
- APU registers for its channels match the emulator at a particular cycle
- Replicate 8-bit audio on a external speaker

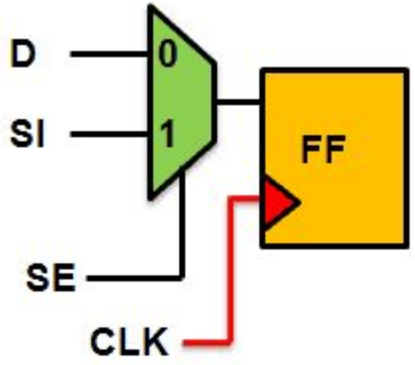
Solution

- CPU: in charge of running the .nes programs, interface with controller
  - Implement instructions set for the 6502 microprocessor, clocked at 1.79 MHz
  - 2 KB of RAM mapped to FPGA SRAM
- PPU: composes game’s frames out of sprites
  - Will generate 256x240 video signal to be sent through VGA to display
  - 10 KB of VRAM mapped to FPGA SRAM
  - Additional memory space to hold color information and screen location

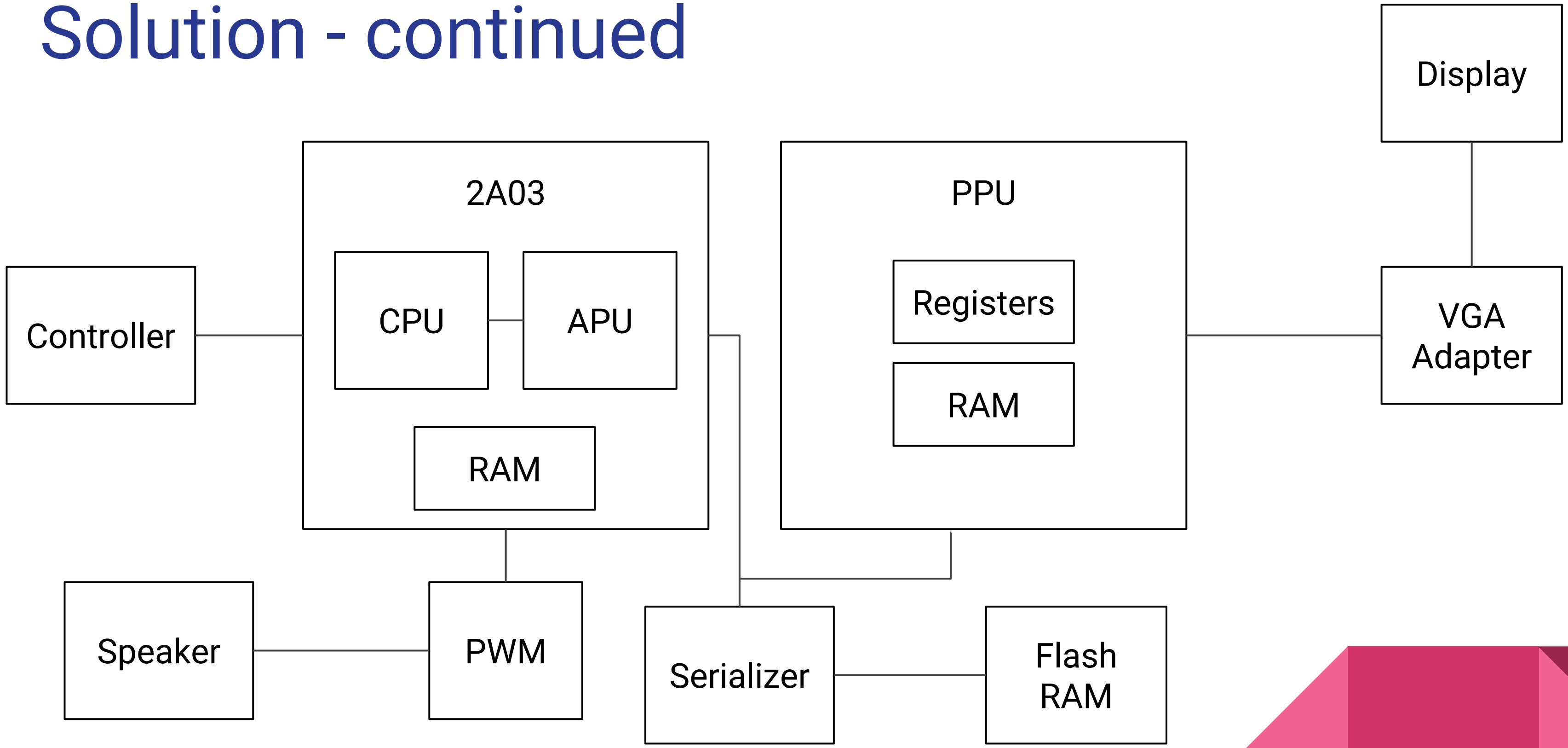


Solution

- APU: generates the game’s music by modulating different channels
  - Has five channels (2 pulse generators, a triangle wave, noise, delta channel)
  - The five channels are modified with designated registers, and combined using a non-linear scheme
- Serialization: serialize game state and write to flash
  - Need to dump game’s stateful information including (RAM, VRAM, registers, OAM)
  - Use a scan chain to serialize our design’s flip flops and store them in memory



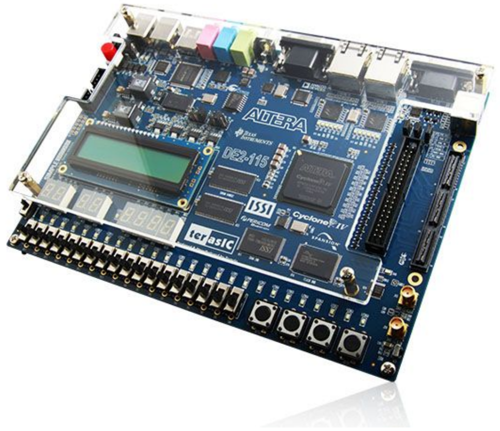
Solution - continued



Solution - continued

Technology:

- Altera DE2-115 FPGA
- Original NES controllers
- Speaker
- Mesen NES emulator
- VGA display



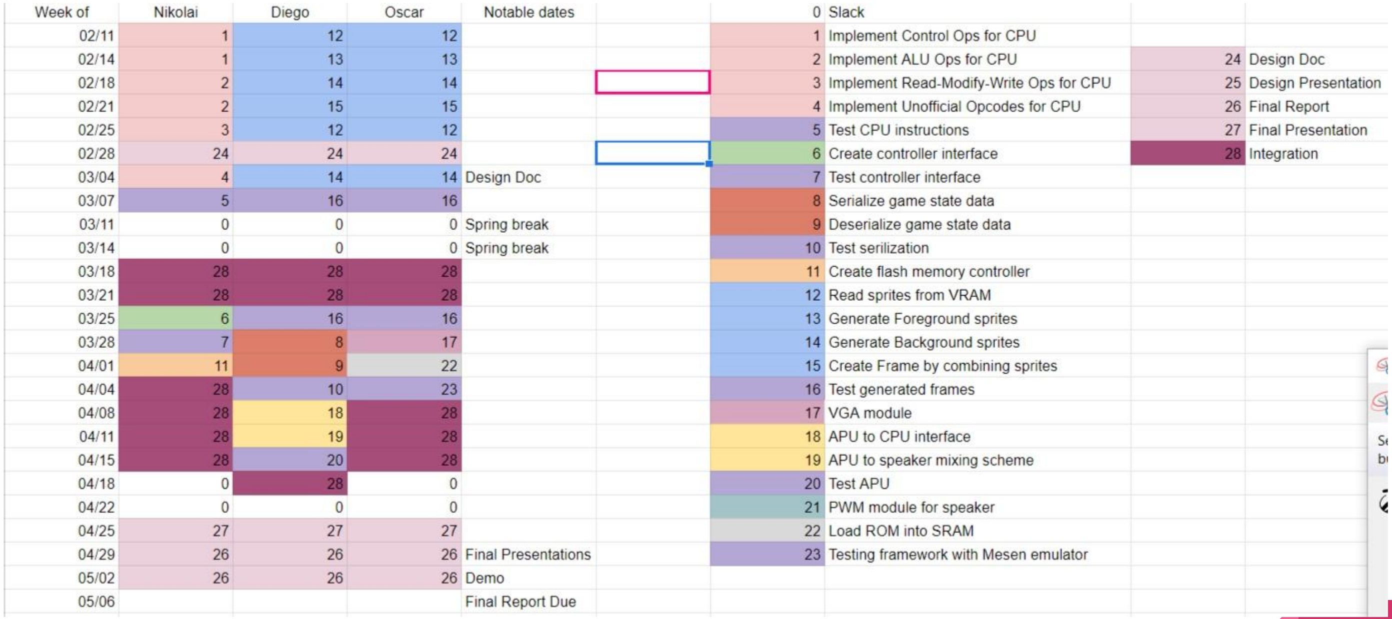
Testing, Verification and Metrics

- Saving/Loading
  - Write and read to Flash memory using Altera software
- Outputs for audio and video
  - Create dummy images and steady frequencies
- NES ROMS
  - Instantiate .nes files in SRAM and observe game bootup
- Controllers
  - Use test harness to detect the different button inputs from controller
- Frame Rate
  - Insert counter to keep track of how many frames are sent to display
- Execution
  - Compare CPU state to that of Mesen emulator

Tasks and Division of Labor

- Nikolai:
  - Implement CPU instruction set
  - Interface with controllers
  - Module for writing/loading save state to flash memory
- Diego:
  - Implement the PPU rendering pipeline
  - Implement APU audio pipeline
  - Implement PWM module for speaker
  - Save State serialization
- Oscar:
  - Implement the PPU rendering pipeline
  - Implement VGA driver
  - ROM loader into SRAM
  - Testing framework for emulator comparison

Schedule



Week of	Nikolai	Diego	Oscar	Notable dates
02/11	1	12	12	
02/14	1	13	13	
02/18	2	14	14	
02/21	2	15	15	
02/25	3	12	12	
02/28	24	24	24	
03/04	4	14	14	Design Doc
03/07	5	16	16	
03/11	0	0	0	Spring break
03/14	0	0	0	Spring break
03/18	28	28	28	
03/21	28	28	28	
03/25	6	16	16	
03/28	7	8	17	
04/01	11	9	22	
04/04	28	10	23	
04/08	28	18	28	
04/11	28	19	28	
04/15	28	20	28	
04/18	0	28	0	
04/22	0	0	0	
04/25	27	27	27	
04/29	26	26	26	Final Presentations
05/02	26	26	26	Demo
05/06				Final Report Due

0	Slack
1	Implement Control Ops for CPU
2	Implement ALU Ops for CPU
3	Implement Read-Modify-Write Ops for CPU
4	Implement Unofficial Opcodes for CPU
5	Test CPU instructions
6	Create controller interface
7	Test controller interface
8	Serialize game state data
9	Deserialize game state data
10	Test serilization
11	Create flash memory controller
12	Read sprites from VRAM
13	Generate Foreground sprites
14	Generate Background sprites
15	Create Frame by combining sprites
16	Test generated frames
17	VGA module
18	APU to CPU interface
19	APU to speaker mixing scheme
20	Test APU
21	PWM module for speaker
22	Load ROM into SRAM
23	Testing framework with Mesen emulator

24	Design Doc
25	Design Presentation
26	Final Report
27	Final Presentation
28	Integration

https://docs.google.com/spreadsheets/d/10u9kZImvUQW7v8SjNf5pVW5hULkV7jVWW4Zvj-4kfc8/edit?usp=sharing