Nintendo Entertainment System Emulator

Testing Plan

Kajetan Lach

January 21, 2025

1 Introduction

This document outlines the strategy, tools and specific test cases for ensuring the NES Emulator meets its functional requirements and maintains reliability and performance standards. It includes multiple testing types (unit, integration, acceptance) and details on how each type will be conducted.

2 Testing Strategy Overview

2.1 Unit Testing

2.1.1 Goal

Validate individual components (e.g., CPU instruction handling, memory read/write functions) in isolation.

2.1.2 Scope

Core C++ classes and functions responsible for emulation logic.

2.1.3 Tools

Google Test for test framework, potentially **Google Mock** for mocking dependencies.

2.2 Integration Testing

2.2.1 Goal

Ensure that modules (CPU, PPU, APU, input handling, etc.) work togheter as intended when combined.

2.2.2 Scope

Full rendering loop, user input to on-screen behavior and audio output.

2.2.3 Tools

A custom test harness or script (e.g., Python) to run the emulator with various configurations. Automated test ROMs that specifically validate memory timing, sprite rendering and audio synchronization.

2.3 User Acceptance Testing (UAT)

2.3.1 Goal

Validate the emulator against real-world usage scenarios, ensuring it meets user expectations (students, archivists, gamers).

2.3.2 Scope

Loading games, configuring input, adjusting settings and using debug tools from the end-user perspective.

2.3.3 Tools

Manual checklists for testers. Exploratory testing to capture user feedback on usability and performance.

3 Testing Tools and Frameworks

3.1 Google test (gTest)

C++ framework for writing and running unit tests. Provides a wide range of assertions and test fixtures for organizing test cases.

3.2 Google Mock (gMock)

Used for mocking dependencies in unit tests. Allows for testing components in isolation by replacing real objects with mock objects.

3.3 Python

Scripting language for writing test harnesses and automation scripts. Can automate launching the emulator with different ROMs and configurations.

3.4 Manual Testing Checklists

Documented steps to validate core end-to-end scenarios.

3.5 Automated Build & Test (CI/CD)

Github Actions or similar platform to automatically build and run tests on each commit.

4 Test Cases

Below are examples of test cases covering key functionalities. Each includes a description, test steps and expected results.

4.1 Unit Test Example: CPU Instruction Decoding

Verify that the CPU module correctly decodes and executes the LDA (Load Accumulator) instruction with different addressing modes.

Preconditions: CPU instance is initialized. Memory is loaded with test program containing LDA instructions.

Test Steps:

1. Set up CPU registers and memory to known values.
2. Execute the LDA instruction in immediate addresing mode.
3. Execute the LDA instruction in zero-page addressing mode.
4. Check the accumulator and status flags after each operation.

Expected Results: Thge accumulator register contains the expected value after each LDA instruction. The Zero and Negative flags are set/unset correctly based on the loaded value.

4.2 Unit Test Example: Memory Read/Write

Ensure the memory module can read and write bytes at specified addresses.

Preconditions Memory object is created.

Test Steps

1. Write a byte to an address (e.g., 0x2000).
2. Read the byte from the same address.
3. Compare the read value with the written value.

Expected Results The read value matches the written value. No exceptions or errors occur for valid addresses.

Expected Results: Thge accumulator register contains the expected value after each LDA instruction. The Zero and Negative flags are set/unset correctly based on the loaded value.

4.3 Integration Test Example: Full Gameplay Loop

Validate that the emulator can load and run a standard NES ROM from the start screen to in-game, verifying graphics, input and audio.

Preconditions Emulator compiled and installed on a target system. A known working ROM is available.

Test Steps

1. Launch the emulator and load the ROM via the file selection interface.
2. Observe the title screen for correct rendering and audio.
3. Press start or equivalent mapped button to begin gameplay.
4. Check if character movement, background scrolling and sound are synchronized.

Expected Results The title screen, menu and gameplay screens render correctly. Audio plays without stutter or noticeable latency. Controls respond accurately.

Expected Results: Thge accumulator register contains the expected value after each LDA instruction. The Zero and Negative flags are set/unset correctly based on the loaded value.

4.4 User Acceptance Test Example: Debugging Tools

Check if the user can successfully enable and utilize the emulator's debugging interface for learning or development.

Preconditions Emulator is installed and configured. User has a ROM to test.

Test Steps

1. Start the emulator in debug mode.
2. Load the ROM and pause execution at the start screen.
3. Inspect CPU registers, memory usage and breakpoints to observe the game's state.
4. Step through several instructions and note changes in registers and memory.

Expected Results Debug interface displays register and memory states clearly. Stepping through instructions updates information in real time. No crashes or hangs occur during debug operations.

5 Reliability and Performance Validation

5.1 Reliability

Stress Testing: Run the emulator for extended periods to ensure it remains stable without memory leaks.

Error Handling Tests: Confirm that when loading invalid or corrupted ROMs, the emulator provides user-friendly error messages rather than crashing.

5.2 Performance

Frame Rate Tests: Measure the emulator's frame rate when running different games. Ensure it meets or exceeds 50-60 FPS on recommended hardware.

Profiling & Optimization: Use tools like gprof, valgrind or built-in profiling in IDEs to identify bottlenecks, e.g. CPU usage spikes during rendering.