SECOND SEMESTER PLAN

Michael Kelly
mkelly01@my.fit.edu

Keith Johnson
kjohns07@my.fit.edu

Eric Wells
wellse@my.fit.edu

Faculty Sponsor
Dr. William H. Allen
wallen@cs.fit.edu

Project Goals

SNO is a Super Nintendo Emulator. It is also a Java applet that can be embedded into a web page, allowing visitors to a website to play a Super Nintendo game inside of their browser. Our main features are:

- Ability to load ROM images of SNES games
- Accurate Video emulation
- Controller input via keyboard
- Ability to embed SNO onto a web page

Notably absent from this list is accurate audio emulation. As we stated in our first plan document, we made audio emulation an optional goal if possible. After a semester of work on this project, we are no longer focusing on audio emulation as a deliverable goal. However, if time permits, we will still attempt to achieve some level of audio emulation within SNO.

Technical Challenges

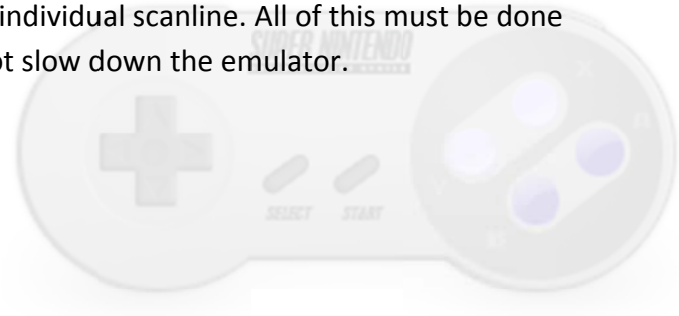
There are three major areas that we face technical challenges in: Timing, Video, and Compatibility.

Timing

Many functions within the SNES depend on precise timing. The processor clocks for the CPU and S-CPU (Sound CPU) run at constant rates that, in some cases, are not precisely known. In addition, much of the console logic dealing with video drawing are timed relative to the scanline and screen dot being drawn by the electron gun in a cathode ray tube television. The challenge lies in mimicking these timings accurately without causing slowdowns in other parts of the emulator.

Video

Video output poses several challenges. Already it has been mentioned that timing of scanline drawing is crucial to accurate emulation. In addition to this, the circuits used to handle drawing in the SNES are complex and involve many factors. Color math, for example, involves performing mathematical transformations on certain pixels depending on their position and priority on the screen relative to other pixels. Another issue is determining which sprites and background tiles are drawn when drawing each individual scanline. All of this must be done while maintaining good performance so as to not slow down the emulator.

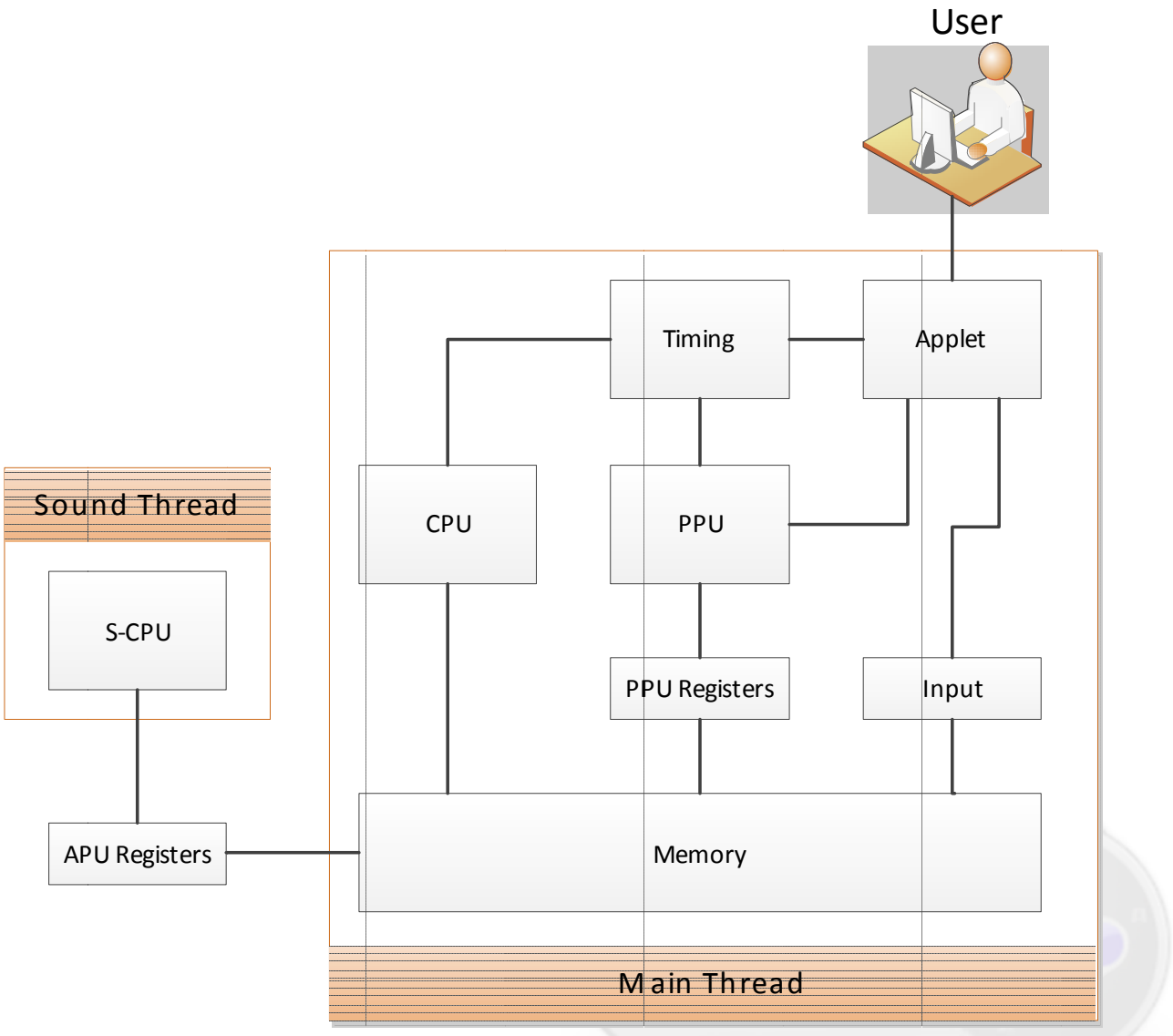


Compatibility

Compatibility refers to issues that are not primary features we intend to implement, but are require to emulate certain games. The best example of this is the S-CPU. While we consider sound an optional feature of SNO, most games use the S-CPU in some way. Most games will block execution until the S-CPU code is loaded and certain values are sent back to the CPU. This causes games to freeze while loading, making the game unplayable. Thus, although we do not intend to emulate sound fully, we are required to implement the S-CPU in order to successfully play many games.

Design

The diagram below summarizes the important components in the SNO architectural design.



The diagram is composed of two threads: The main thread and the sound thread. Because the S-CPU is an entirely separate CPU, it runs in its own thread and communicates with the main program thread via a set of memory-mapped registers.

Each box in the diagram represents a component in our system, and may be composed of multiple Java classes and packages. The connections between components indicate interaction between components as described below.

Components

- **Applet:** The Applet class and UI components that are presented to the user. The Applet receives input from the user via the keyboard and mouse, and displays the graphics generated by the PPU.
- **Timing:** Handles the timing between CPU cycles as well as signaling the PPU to draw scanlines. Also handles the VBlank timing and signals the PPU to draw a frame to the applet.
- **CPU:** Core of the SNES. Processes instructions stored in Memory.
- **PPU(Picture Processing Unit):** Handles drawing scanlines to a buffer. When signaled by the Timing component, the PPU sends this buffer to the Applet to be displayed to the user. Stores graphics data internally in VRAM and exposes VRAM to the CPU via a set of special memory-mapped registers in the PPU Registers component.
- **PPU Registers:** Special locations in memory that the CPU can write to, which in turn affects the PPU. The CPU uses the PPU Registers to load graphics data as well as controlling settings that affect how the PPU draws to the screen.
- **Input:** Stores the state of the virtual controller and translates input sent from the Applet to data stored in Memory for the CPU.
- **Memory:** Stores ROM data and code to be executed by the CPU. Also provides an interface for the CPU to write to memory-mapped registers that control other parts of the system.
- **S-CPU:** Executes code stored in the APU Memory as a second processor. Ideally, this also controls the DSP, which generates sound to be played by the Applet.
- **APU Registers:** Set of memory-mapped registers that facilitate communication between the CPU and the S-CPU.

Progress Summary

Feature	Completion	Todo
CPU Emulation	90%	Bugfixes, Timing
Video / PPU Emulation	40%	Sprites, Background Scrolling, Other Modes
GUI / Applet	50%	User Experience: Keymaps, Options
Controller Input	50%	Support for multiple controllers, cross platform fixes

Milestone Tasks

Milestone 4 – February 21st

- Improve Video
- Compatibility Fixes

Milestone 5 – March 28th

- Improve Video
- Compatibility Fixes
- User Experience Improvements
- Showcase Poster

Milestone 6 – April 25th

- Demo of SNO
- User Manual
- Demo Video
- Compatibility Fixes

Milestone 4 Task Matrix

Task	Michael K.	Keith J.	Eric W.
Improve Video	34%	33%	33%
Compatibility Fixes	34%	33%	33%

Milestone 4 Task Summaries

- **Improve Video:** Several areas of video rendering have yet to be implemented:
  - **OAM(Object Attribute Memory):** Stores data on objects, which are sprites that can be precisely positioned on screen using XY coordinates.
  - **Color Math:** Certain backgrounds and sprites can be marked to participate in color math, where their pixels are modified before being displayed
  - **Background Scrolling:** Backgrounds can be scrolled around the screen.
  - **HBlank:** An interrupt can be specified that runs after each scanline is drawn.
  - **Window Masks:** A window can be defined (and changed during HBlank) that masks out drawing in certain areas.
  - **Modes:** There are several "Modes" of drawing, ranging from Mode 0 to Mode 7. Currently, none are properly implemented.
- **Compatibility Fixes:** Several parts of SNO need to be tested and fixed to improve ROM compatibility:
  - **S-CPU:** The sound CPU has been implemented, but is not functioning correctly in some cases, causing games to freeze during loading.
  - **HiROM:** HiROM loading is still incomplete, making it difficult to load most ROMs.
  - **DMA:** We have only implemented DMA in one direction, which is enough for most games, but for full compatibility it needs to function bi-directionally.
  - **HDMA:** An extension of DMA that runs between every scanline, it is required by a few games for more complicated drawing.
  - **General Bugfixes:** We don't write perfect code, so part of compatibility is to run games, and fix any bugs we encounter in things we previously expected to be working properly.

Faculty Sponsor Approval

I have discussed with the team and approve this project plan. I will evaluate the progress and assign a grade for each of the three milestones.

Signature: _____ Date: _____