ECE-493

FINAL REPORT

NES Handheld Emulation Breakout Board for the BASYS-3

Abstract

There are several methods to emulate the NES, including software or hardware emulation, with the latter being a more authentic emulation. We emulated the NES on a BASYS-3 Artix-7 FPGA and minimized overall cost of components needed to create a portable/handheld device. Our NES FPGA emulation device will be comparable to existing devices in terms of price and capabilities. We used existing functionalities of power systems and of P-MOD accessories for the BASYS-3 to create a singular breakout PCB to interface with the BASYS-3. Additionally, this project is designed to be open source to allow future expansion of capabilities and to provide an opportunity to learn about retro console emulation.

Team Members: Joshua Thomas Riggs, Brooks Aaron Corbett, Anthony Thinh Tang, Samuel Kifle Kebadu, Amilcar Alejandro Paniagua

Date of Submission: 12/4/2022

Faculty Supervisor: Dr. Jens-Peter Kaps

Executive Summary

Current options for Nintendo Entertainment System (NES) emulation, with the requirements of being portable and hardware emulated, are limited and costly. In response, our solution is to design a breakout printed circuit board (PCB) with necessary interfaces and inputs/outputs for the BASYS-3 Artix-7 Field Programmable Gate Array (FPGA), to design an 3D-printed encasing for the purpose of being portable/handheld, and to include cost effective components such as the LCD screen, battery, SD card, and audio speaker. This project results in allowing the BASYS-3 to be a new platform for NES emulation and in providing a new option for DIY NES emulation. Our team is composed of five members, majoring in Computer Engineering and Electrical Engineering:

Team Member	**Relevant Experience**	**Skills**
Samuel Kebadu (CpE) (PM)	ECE 445, 446, 447, 448 FPGA Internship	Verilog, VHDL, C++, C, Computer Architecture
Amilcar Paniagua (CpE)	ECE 286, 447, 448	VHDL, C++, C, Circuit Design, Soldering, PCB Design
Brooks Corbett (EE)	ECE 286, 447, 436 Hardware Engineering Internship	Circuit Design, PCB Design, Power system design, Soldering,
Joshua Riggs (EE)	ECE 448, 445, 436, 447	VHDL, Digital Design, PCB Design
Anthony Tang (EE)	ECE 286, 445, 447, 436	AutoCAD, Circuit Design, Soldering, C

CpE: Computer Engineering
PM: Project Manager
EE: Electrical Engineering

Table of Contents

1. Approach

1.1 Origin

Since original NES consoles are not commercially sold anymore, or are expensive to purchase secondhand, we wanted to develop a cheaper alternative that still maintains portability and simplicity. In addition, some current market alternatives use hardware locked FPGAs; we have planned to have our NES FPGA project to be "unlocked" to allow programmability to the BASYS-3 in case the user desires to use the FPGA platform for other purposes.

1.2 Solution

In order to prototype our project, we started with the BASYS-3 and hardware language. Credit is given to Brian Bennett, who provided his recreation of the NES architecture written in Verilog, available to the public on GitHub [3]. Input and output signals in the constraints file were adjusted to match the P-MOD pins of the BASYS-3. A VGA display, an NES controller, and an audio speaker were required to playtest the emulated NES, along with two BASYS-3 PMOD expansions for audio amplification and SD card reading. The NES architecture can be stored in the BASYS-3 as its default program via USB Universal Asynchronous Receiver-Transmitter (UART) communication, which can also load the NES game ROMs. This functionality was planned to be integrated into the Verilog code to read game ROMs from an SD card through Serial Peripheral Interface (SPI).

Additional components, such as a small LCD screen and a battery power source, were chosen to allow the device to have portable/handheld functionality. To minimize the dimensions of our device, we designed and printed our own breakout PCB that contains the audio amplifier circuit, SD card reading circuit, button interface circuit, video output circuit, and power management circuit.

1.3 Alternative Designs

There were possible several alternatives that were considered:

- External display, controllers, and audio instead of embedded
- Not using the BASYS-3, instead simply using an FPGA soldered to the breakout board
- Using a different storage solution as opposed to an SD card i.e., memory bank
- Using a wired power connection as opposed to battery

Using external components would make the device lose its portability and handheld functionality. Next, switching out the BASYS-3 with a soldered FPGA to the breakout board would minimize the device's dimensions. Another alternative was using a memory bank in the BASYS-3 to store game ROMs; however, this would disallow a user to "plug and play" their own game ROMS via SD card. The last alternative was to use a wired power connection and eliminate the use of a battery. Handheld functionality can be preserved but portability of the device would be lost. These alternatives would have greatly shifted the time and budget specifications of this project.

As mentioned before, Brian Bennett's FPGA project of the NES architecture on GitHub was implemented into the BASYS-3. He reported that his project is missing mappers and the Delta Modulation Channel (DMC). He also provided a C++ windows application named NesDbg that will allow communication to the FPGA via USB UART. NesDbg can be used to run tests and to load game ROMs.

1.4 Team Member Contributions

Our project will consist of five main tasks:

- Circuit design
- PCB design
- Enclosure design
- Game selection menu program
- RTL code for memory

Brooks was responsible for circuit design, with the help of Anthony. Their objective was to design the circuitry and schematics for the breakout board in EasyEDA. Simulations were conducted in computer SPICE software by Amilcar.

Amilcar was responsible for designing the PCB layout, with the help of Brooks. They used the circuits designed by Brooks and Anthony to design a PCB layout in EasyEda. They were responsible for soldering required electronic components to the PCB and testing/debugging the PCB.

When the breakout board dimensions and externals were finalized, Anthony had the responsibility of designing the 3D-printed enclosure with the help of Amilcar. They constructed the device with all the necessary large components such as the LCD screen and the battery in mind.

While the tasks above were sequential, Josh and Sam helped one another on the software/RTL responsibilities. Sam's main responsibility was to create code for RTL functionality for manipulating memory locations in the memory map on the NES architecture. Josh, in tandem, was responsible for programming and designing a game selection menu on boot-up to work with the RTL design.

2. Technical section

2.1 Function Decomposition (Level 0)

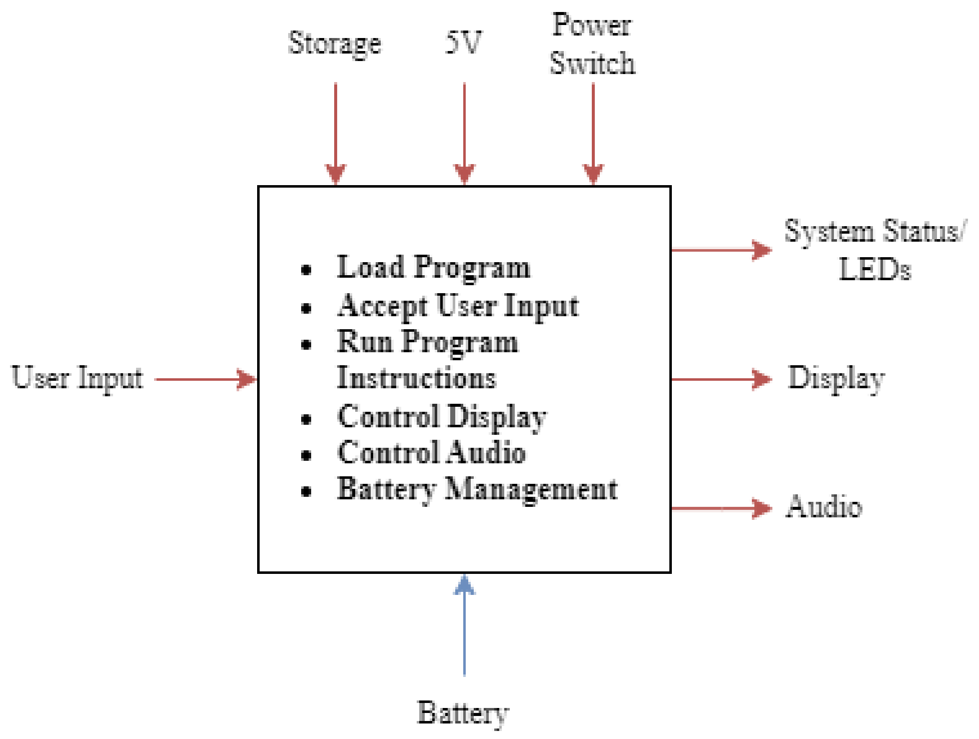


Fig. 1 Functional Decomposition (Level-0)

Fig. 1 shows the level-0 top-down perspective of the NES FPGA; it consists of three main inputs and three main outputs, alongside being electrically powered. The power switch will allow the NES FPGA system to be powered on. Then, the user will input signals via buttons to select a program to load. The NES FPGA will run instructions to output both display and audio to the user, as well as system status LEDs to determine both battery and power status.

2.2 Function Decomposition (Level 1)

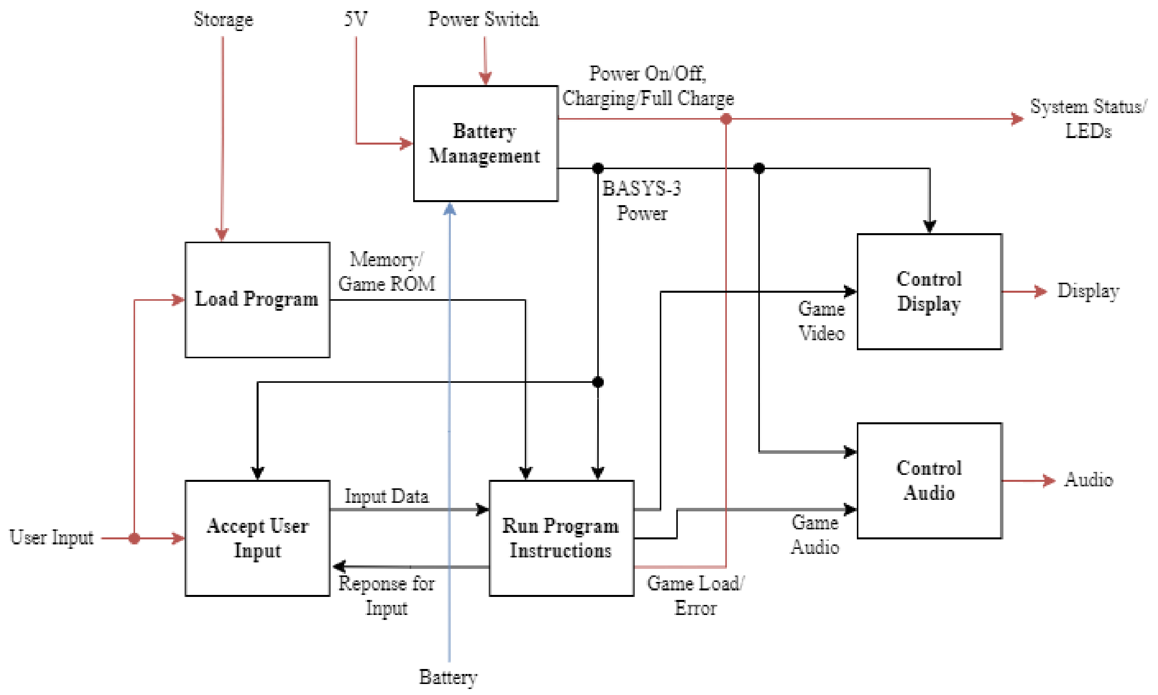


Fig. 2 Functional Decomposition (Level-1)

Fig 2 shows the level-1 Functional Decomposition diagram of the NES FPGA. There are five main modules: Accept User Input, Load Program, Battery Management, Run Program Instructions, Control Display, and Control Audio.

2.3 Function Decomposition (Level 2)

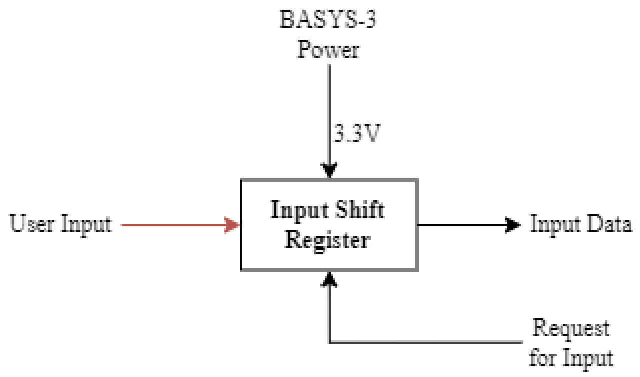


Fig. 3 Functional Decomposition (Level-2) Accept User Input

Fig. 3 shows the level-2 diagram for the Accept User Input module. This function is based on the NES controller, which is composed of a single shift register. The user will input button signals to the shift register, which can be controlled by a request from the program. When the program requests for inputs via a clock signal, the shift register will send input data to the program.

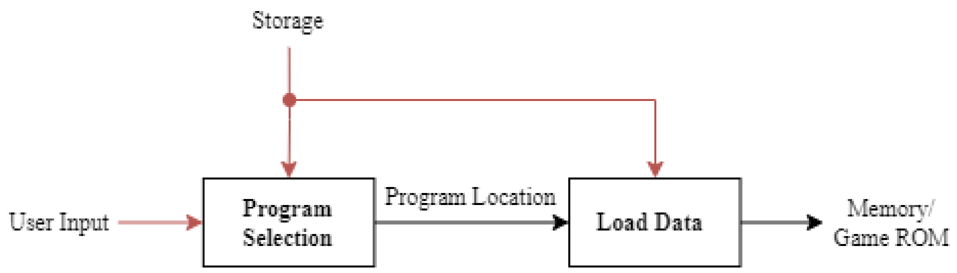


Fig. 4 Functional Decomposition (Level-2) Load Program

Fig. 4 shows the level-2 diagram for the Load Program module. Storage is an insertable SD card that is read by the FPGA through program selection and load data functions. The user will be able to select a program to run. The FPGA will take the location of the program and load it from storage.

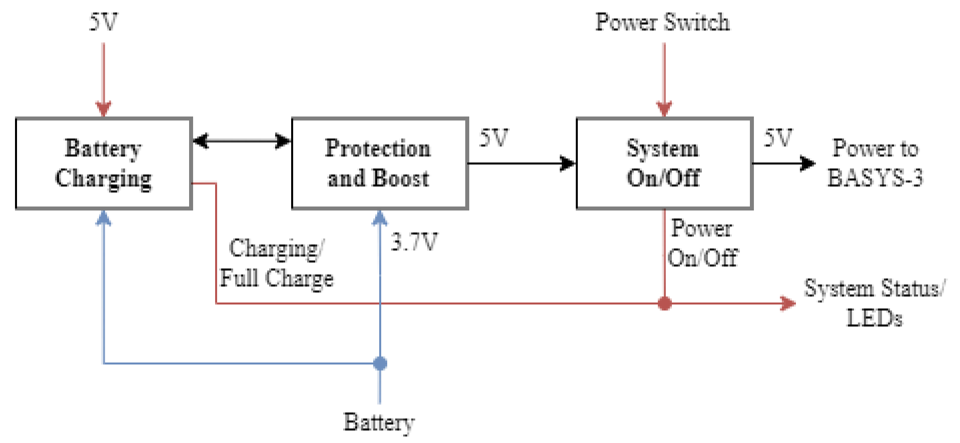


Fig. 5 Functional Decomposition (Level-2) Battery Management

Fig. 5 shows the level-2 diagram for the Battery Management module. 5V of Electric power is sourced from a DC power supply which charges the battery. The battery distributes 3.7V, which is converted back to 5V to power the system.

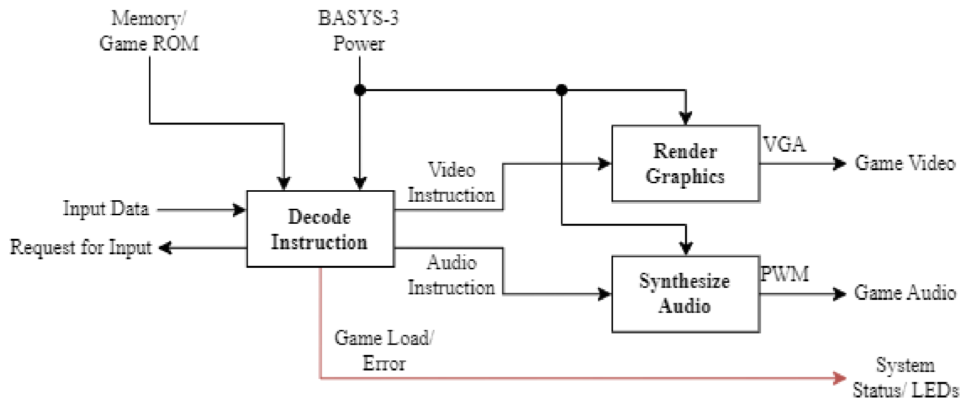


Fig. 6 Functional Decomposition (Level-2) Run Program Instructions

Fig. 6 shows the level-2 diagram for the Run Program Instructions module. This module represents all the inputs and outputs of the BASYS-3 board. The Decode Instructions, Render Graphics, and Synthesize Audio modules exist in the Verilog Code. The BASYS-3 is powered by 5V and receives input data from the user and the memory/game ROM from the SD card. The ROM is decoded and the FPGA renders graphics and synthesizes the audio.

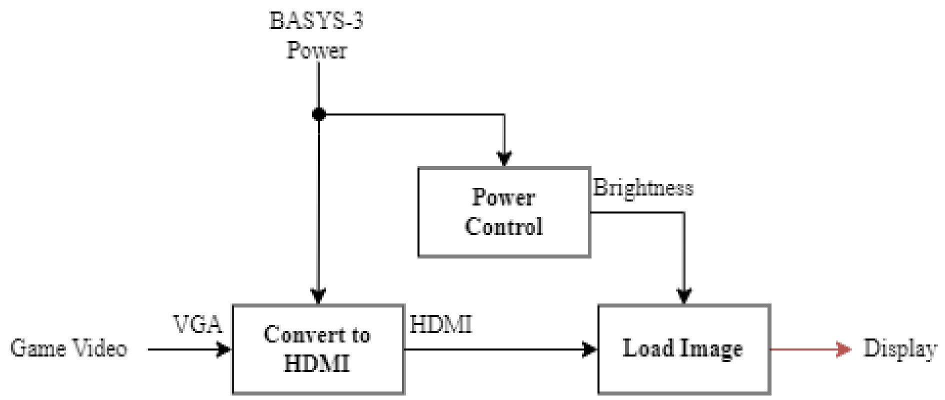


Fig. 7 Functional Decomposition (Level-2) Control Display

Fig. 7 shows the diagram for the Control Display module. The BASYS-3 outputs VGA signals, which is converted to HDMI signals. The LCD screen we chose requires an HDMI signal.

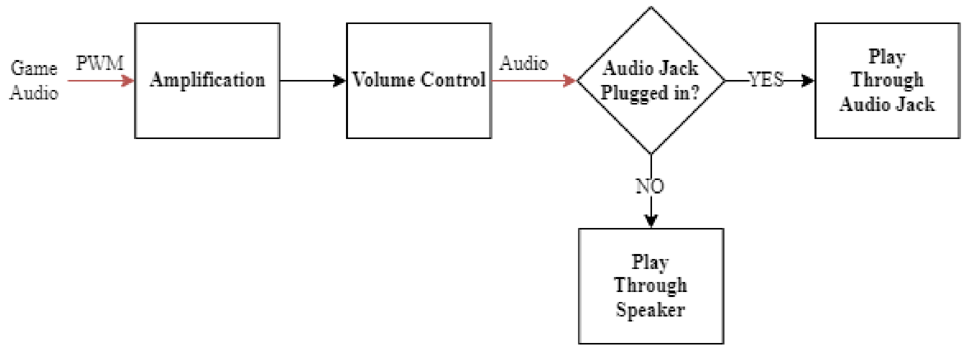


Fig. 8 Functional Decomposition (Level-2) Control Audio

Fig. 8 shows the diagram for the Control Audio module. The PWM audio signal is amplified by 12dB. The volume of this audio can then be adjusted by the user via a potentiometer. The audio jack will have switching functionality: if no device is plugged into the jack, the audio will play through the speaker; if a device is plugged into the jack, audio will play through the device.

2.4 System Architecture

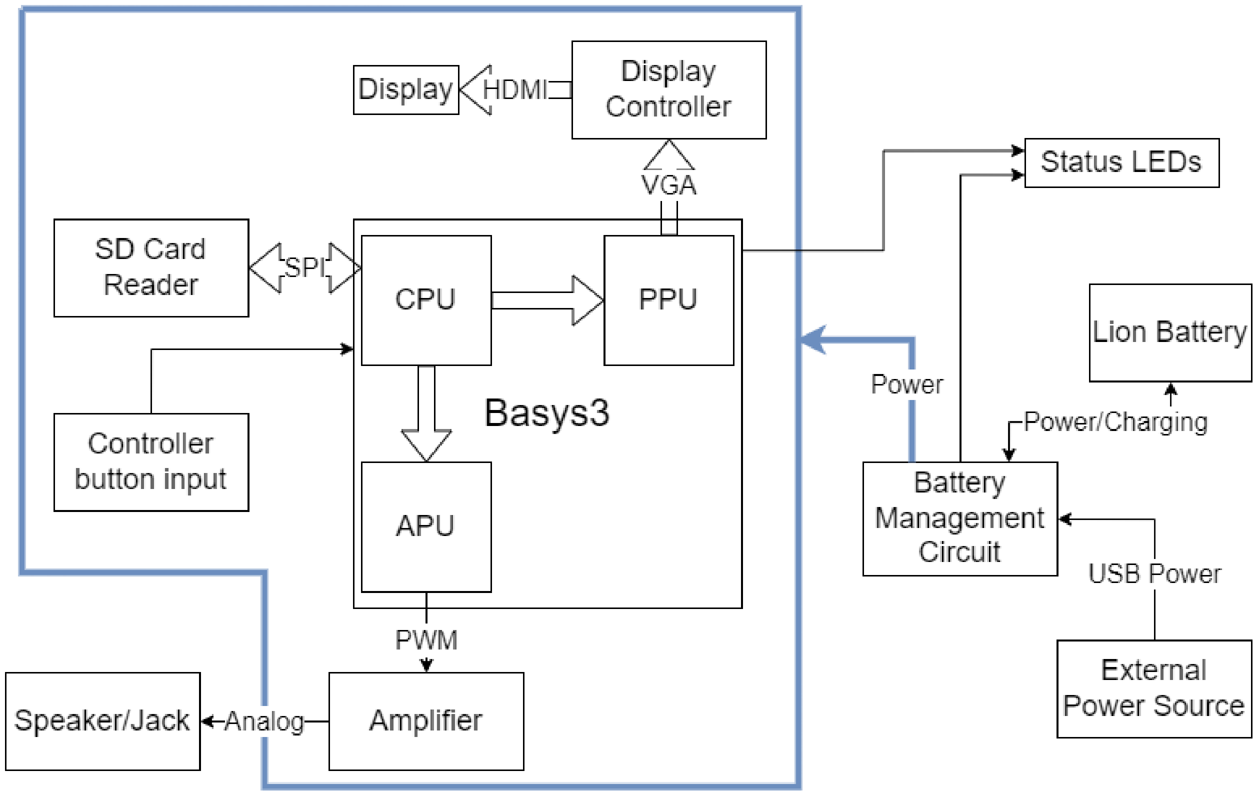


Fig. 9 System Architecture of FPGA NES device

Fig. 9 shows the System Architecture. The blue boxed modules are powered by the Battery Management Circuit module, which can charge a li-on battery via USB from an external power source. Within the BASYS-3 module are three sub-modules which are contained within the NES architecture: the Central Processing Unit (CPU), the Picture Processing Unit (PPU), and the Audio Processing Unit (APU).

2.5 Dataflow

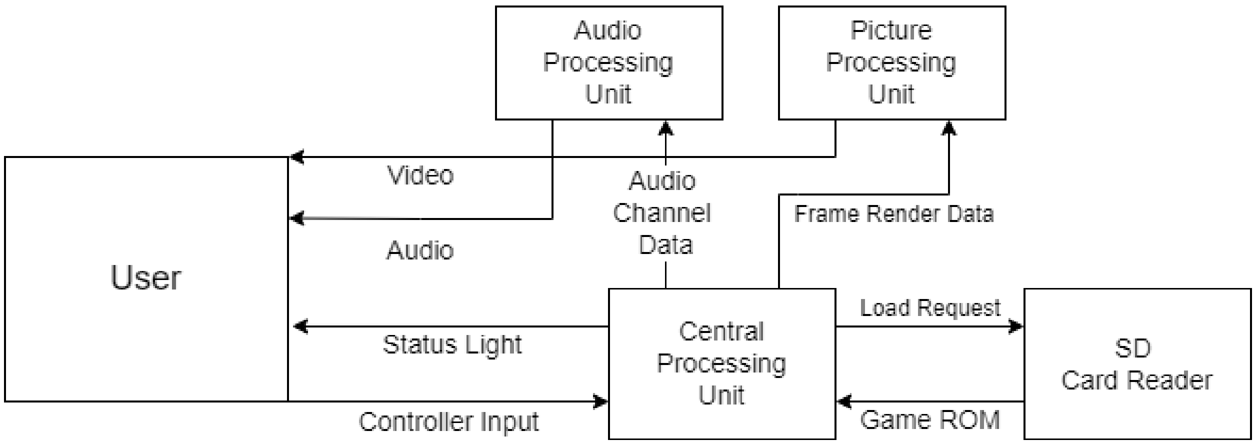


Fig. 10 Dataflow chart of FPGA NES device

Fig. 10 shows the Dataflow chart between the User and the device's modules. Observable data to the user are Video output, Audio output, and Status Lights from the FPGA NES device.

2.6 State Diagram

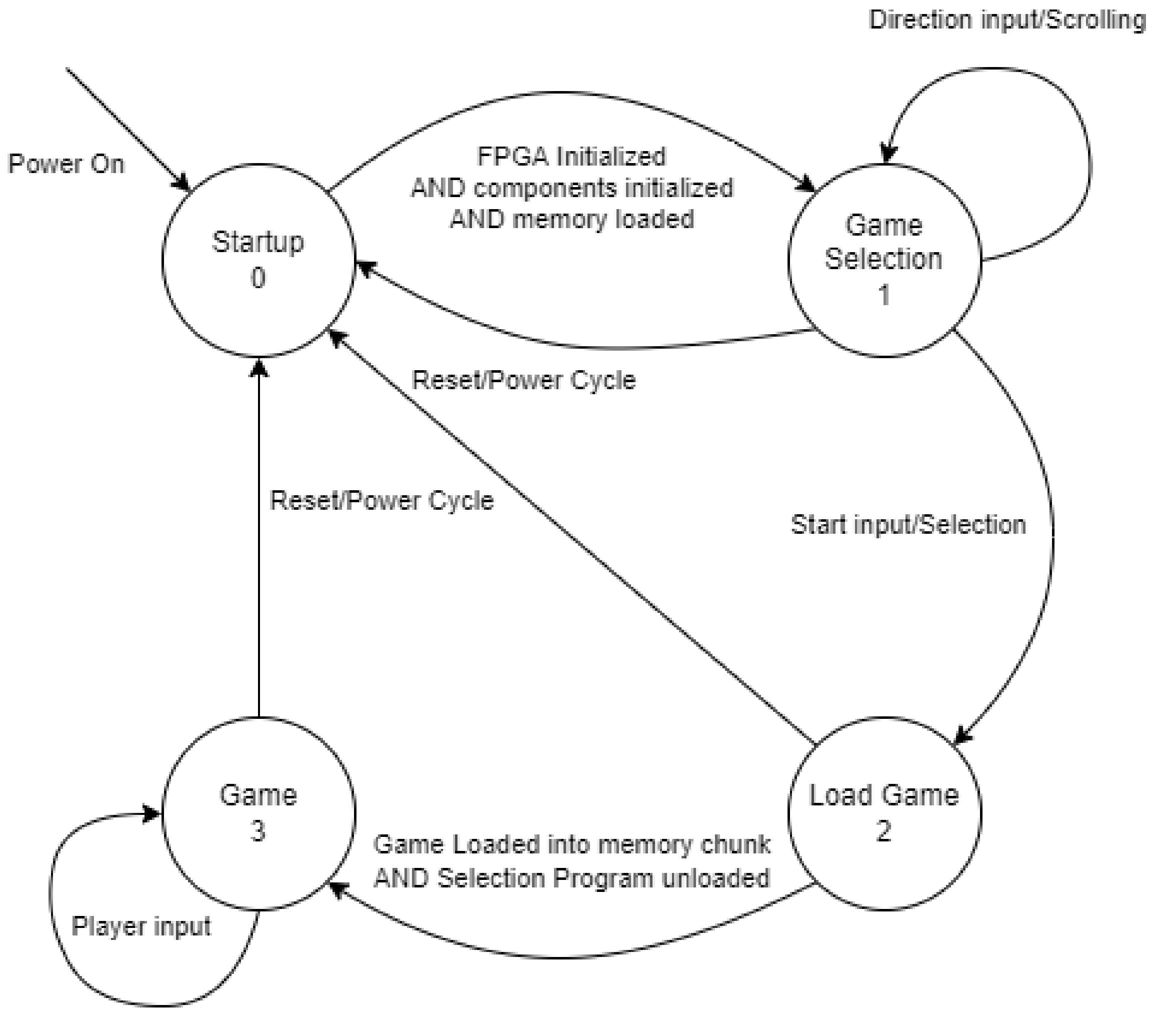


Fig. 11 State diagram of FPGA NES device

Fig. 11 Shows a state diagram of all the possible states the NES device will be in. The NES device will startup via a switch and initialize to a method for game selection. The user will choose the game to run by selecting it with button inputs. After choosing, the game will be loaded and start running, allowing the user to play the game.

2.7 Circuit Schematics

2.7.1 Controller

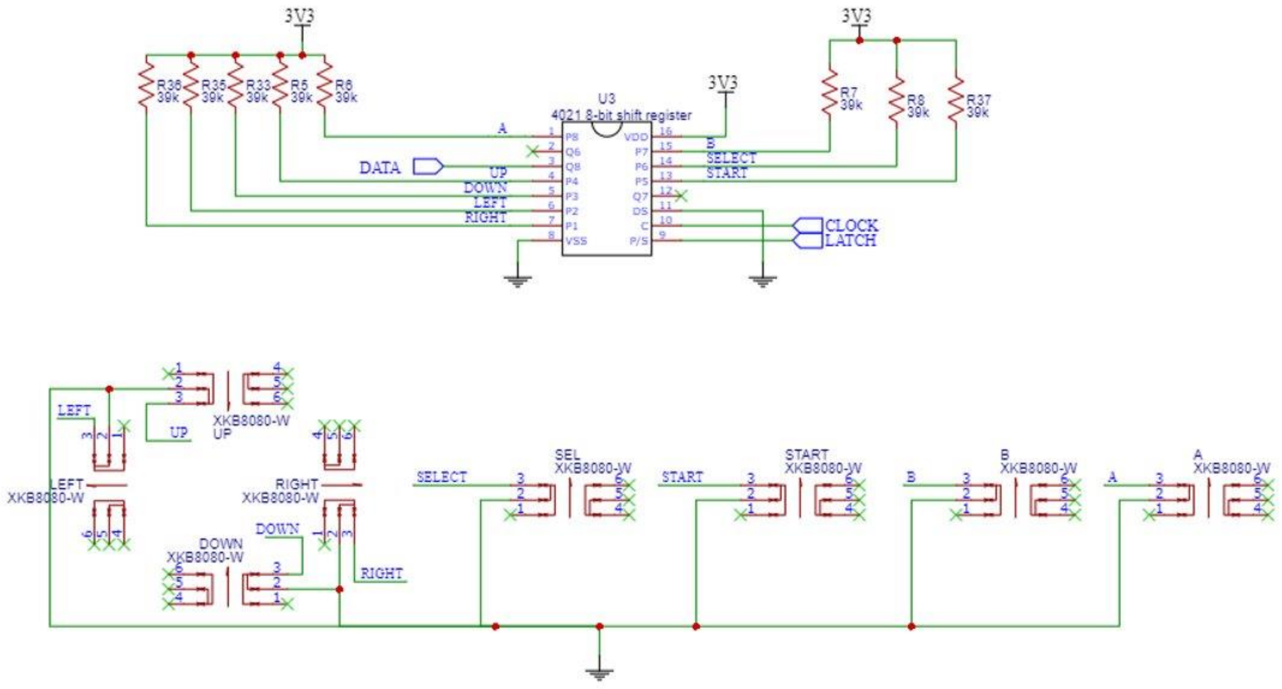


Fig. 12 Circuit Schematic of Controller

The original NES has eight buttons for user input and was referenced for our design [1]. The main IC that makes the controller work is a 4021 shift register that takes eight inputs from the eight buttons, a clock signal, and a latch signal. When the shift register receives the latch signal, it saves the state of all eight inputs. The clock signal then repeats 8 times to read all eight buttons. The recorded input is then sent out of the data pin to the FPGA to be interpreted while the corresponding actions are executed. The controller takes up half of one of the four PMOD ports.

2.7.2 Power/Charging

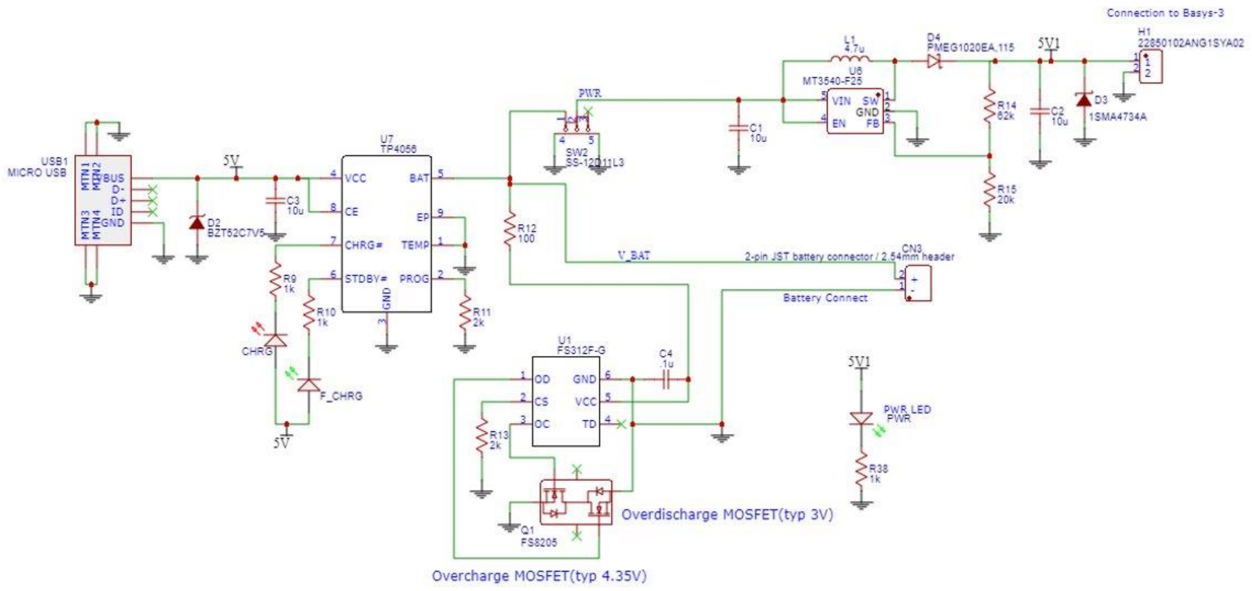


Fig. 13 Circuit Schematic of Power/Charging

In order to achieve portability of the system, a battery needed to be included. The battery is connected to a 5V boost converter in order to supply power to the BASYS-3. A charger IC is also connected so that once an external 5V source is connected, the battery will be able to recharge when needed. There are status LEDs that show when the battery is charging and when the battery is fully charged. Additionally, a protection IC is included so that the battery will not reach a state of overcharge or over discharge. Schottky diodes are included on the main power lines so that if there are any spikes in voltage the connected components will not be damaged.

2.7.3 Audio

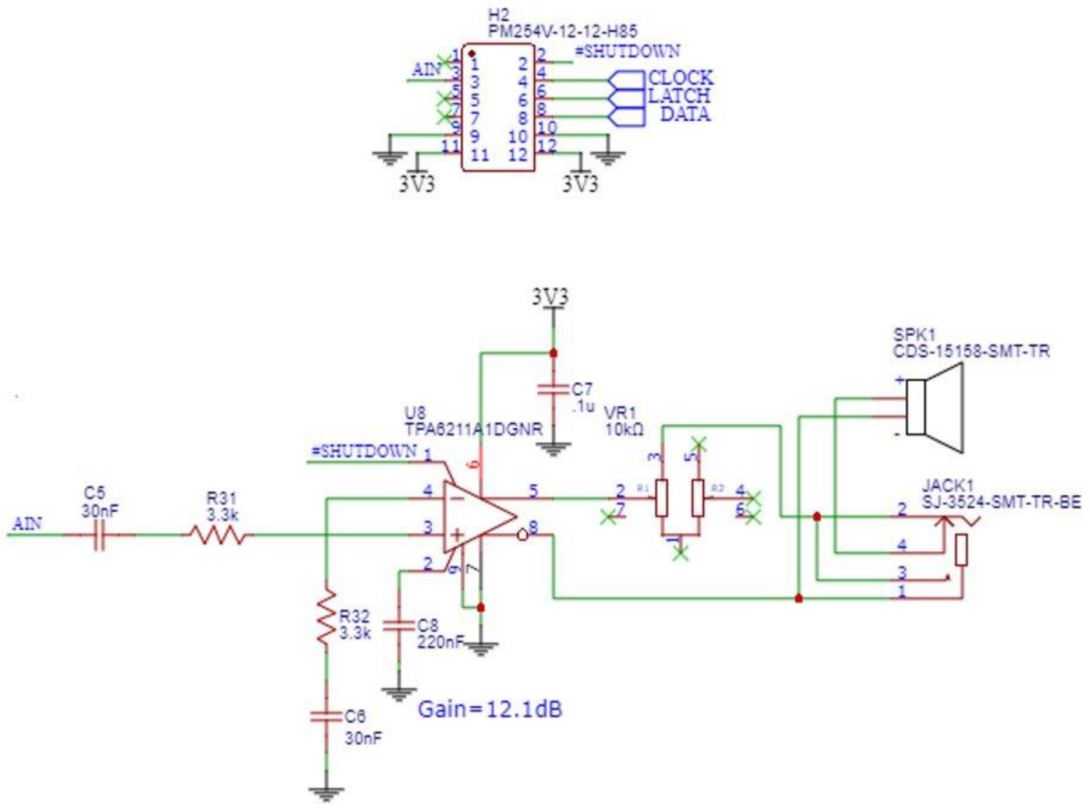


Fig. 14 Circuit Schematic of Audio

The audio that comes out of the BASYS-3 is a PWM signal that is sent through an audio amplifier which applies a gain of 12.1 dB. Once the signal is amplified, the user has the option to control the volume via a potentiometer. There are two different destinations for the audio once the volume is set: the first being the default speaker that is on-board; the second option is when the user plugs a device into the 3.5mm audio jack and the audio will automatically switch over to that device. The audio module takes up the other half of the PMOD port that the controller uses.

2.7.4 Memory

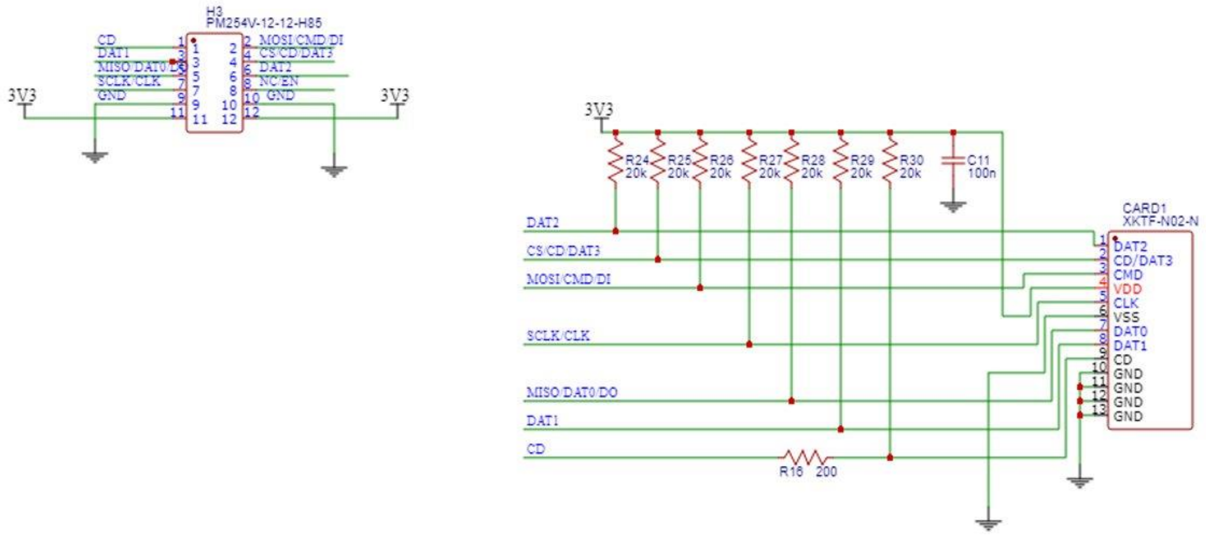


Fig. 15 Circuit Schematic of Memory

This circuit is simply a breakout for the SD card slot. Each signal is pulled high via a 20k resistor. The logic for this module is handled on the FPGA. The memory module takes up an entire PMOD port.

2.7.5 Display

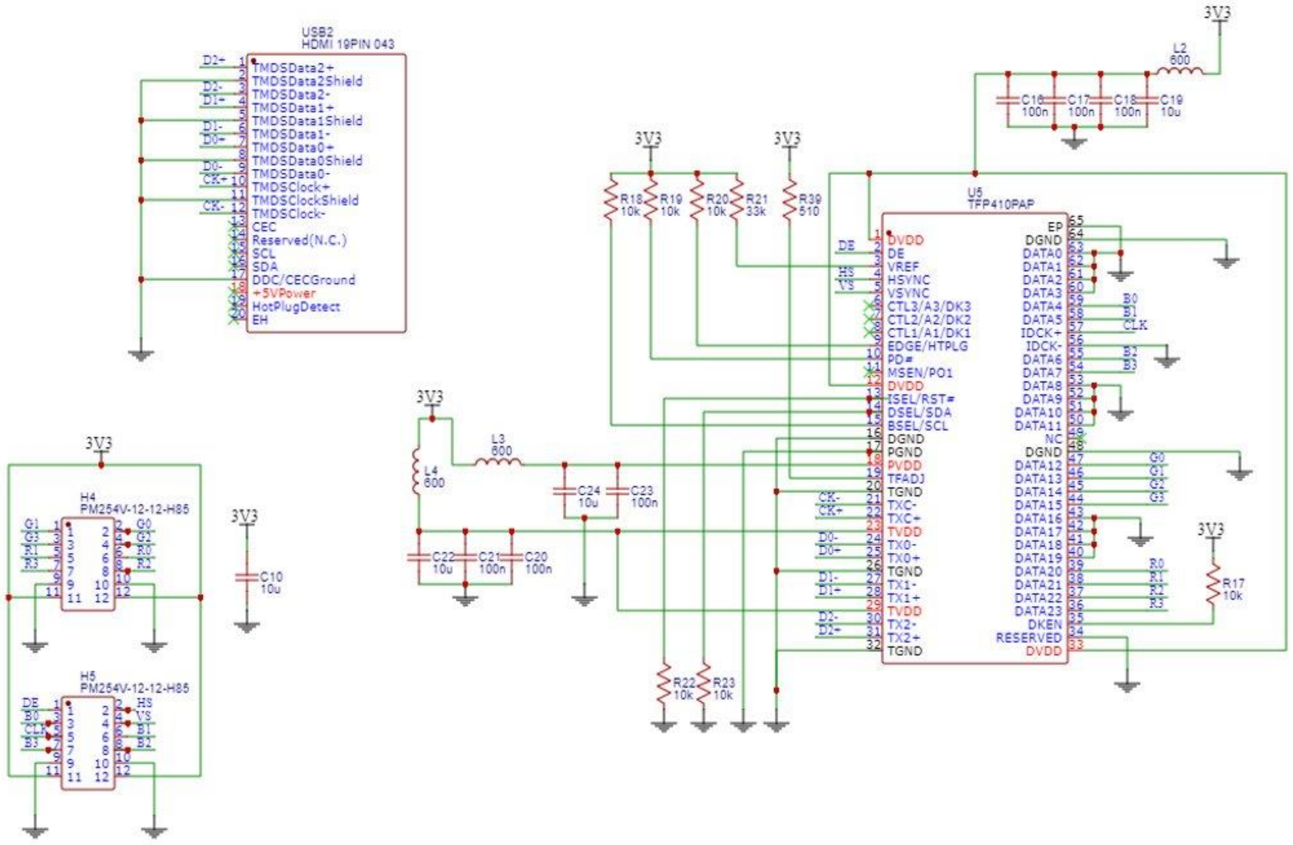


Fig. 16 Circuit Schematic of Display

In order to display onto the LCD, we needed to convert the native VGA signal on the BASYS-3 to a digital signal. This was done using the TFP410PAP IC from Texas Instruments. There is also another IC from Lattice Semi called the SII164 that can be used if the TI chip is unavailable. These ICs take in the red, blue, and green signals along with horizontal and vertical sync and a clock signal to convert to the differential pair signals that works with the LCD's HDMI interface. This digital signal is then connected to a female HDMI port and a male-to-male HDMI ribbon cable is usedto connect the PCB to the LCD. The display module takes up the two remaining PMOD ports on the BASYS-3.

2.8 Software

2.8.1 Rebuild

In order to use the open-source implementation of the NES from Brian Bennett we had to fix and modify the Verilog to get it running on the BASYS-3. One of the issues was that of constraints; the board used by Bennet was the Nexys-6 with a Spartan 6 FPGA [3]. Modifying constraints was trivial because the BASYS-3 had most of the I/O that the Nexys-6 had. One difference was that Bennett was using a breakout board for the NES Joypad controller [3]. This meant that we had to use the PMOD slots on the BASYS-3 directly. Bennett's project didn't make use for all the VGA color channels and so we modified the Verilog to include those on the BASYS-3 [3].

2.8.2 Start Menu

The start menu is implemented using VGA and consists of a module for the generation of all the sync signals for the Vertical and Horizontal axis. This module is also responsible for generating the Pixel Clock and Enable signal. In order to display characters, we made a module for text generation that included a BRAM of ASCII characters. The X and Y axis sync signals were used to access the BRAM, this resulted in a text tile size of 80x30 characters. The start menu also includes a module to use the Joypad controller. This module is separate from the module used in the NES implementation because of clock domain issues. The result of these modules is a start screen of 9 Save game states that can be chosen with the joypad using the UP and DOWN buttons and selected with the A button. The start screen was then multiplexed with the NES Implementation so that the start screen is initialized on boot-up.

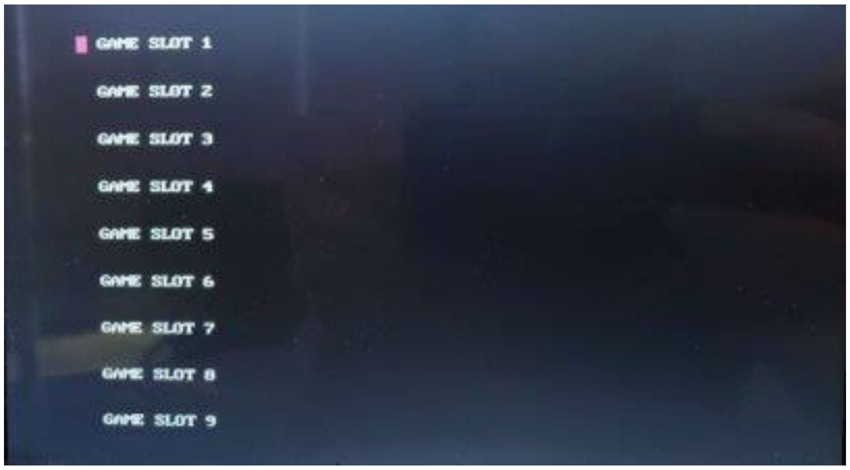


Fig. 17 Game selection menu

2.8.3 Game Storage

Nesdev Wiki provided a plethora of information on programming topics for the NES [4]. It was used when we development started for a game selection menu and for changing memory map values accordingly to each game stored on the SD card.

2.9 Hardware

2.9.1 VGA to HDMI

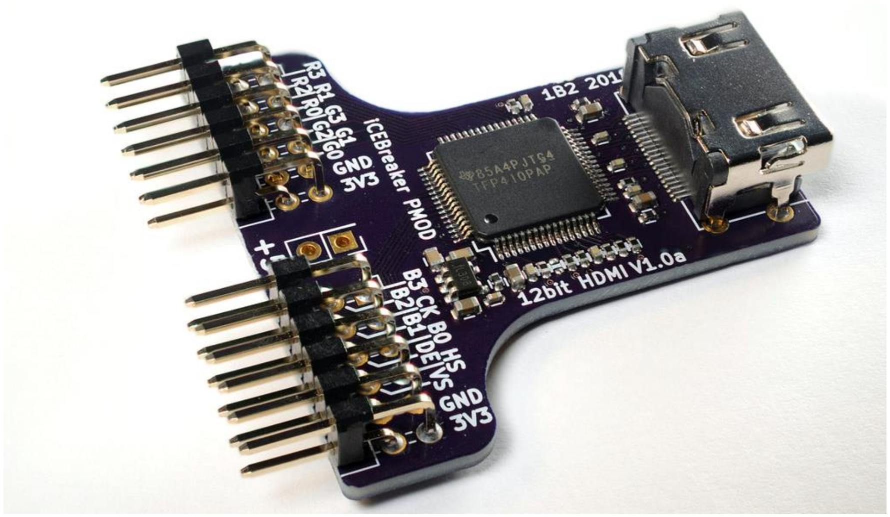


Fig. 18 VGA to HDMI converter PMOD

We based our display circuitry on the PMOD Digital Video Interface by 1BitSquared [2]. It uses two PMOD ports and converts the VGA signal from the BASYS-3 to an HDMI signal.

2.9.2 SD Card

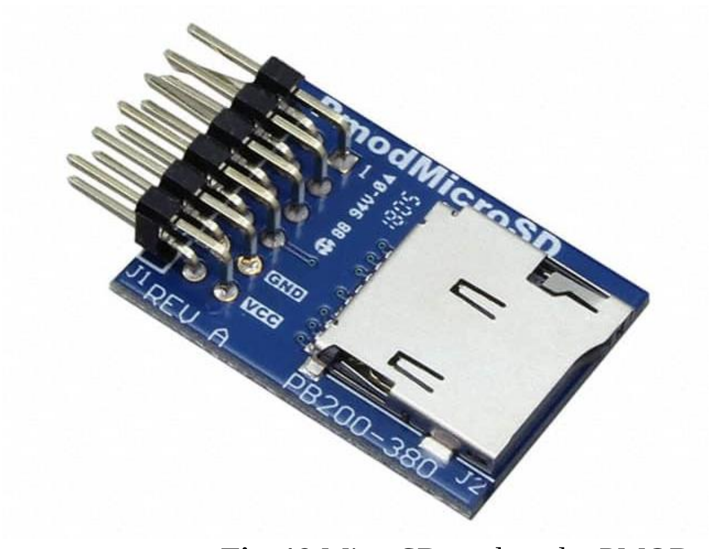


Fig. 19 MicroSD card reader PMOD

We based our memory circuitry on the MicroSD Card Slot by Diligent, a simple breakout board for the MicroSD card.

2.9.3 BASYS-3

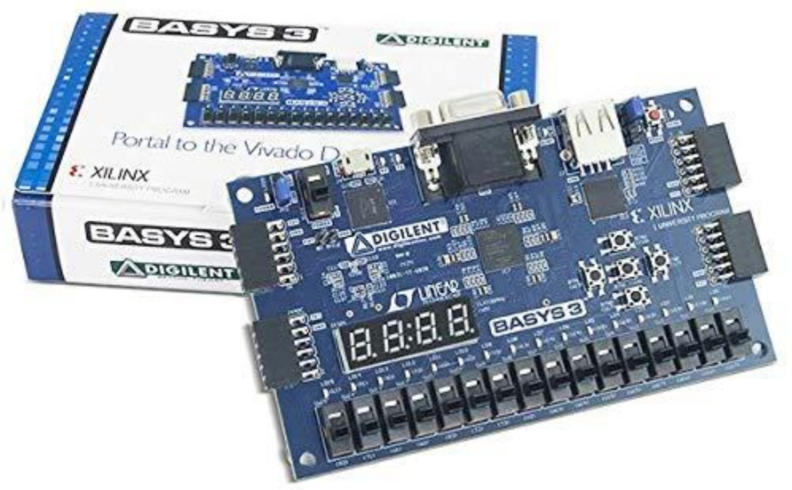


Fig. 20 BASYS-3 FPGA board

Our project revolves around the use of the BASYS-3, an FPGA prototyping board that all CEC Students have at GMU. The BASYS-3 holds the NES CPU architecture and will handle all input and output signals.

2.9.4 Display

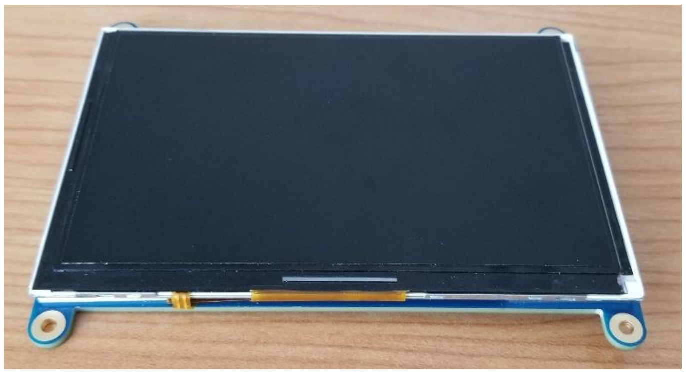


Fig. 21 HDMI 5'' Display Backpack

We chose to have an HDMI 5” display backpack. The NES architecture outputs VGA signal, therefore in the display circuit on the PCB, the VGA signal is converted to HDMI by a TFP410PAP IC.

2.9.5 Battery

Fig. 22 3.7V 4000mAh Li-ion Rechargeable Battery

We used a 3.7V 4000mAh Li-Po rechargeable battery to power our project. Within the PCB circuit, the battery power is boosted to 5V in order to power the BASYS-3.

2.9.6 PCB

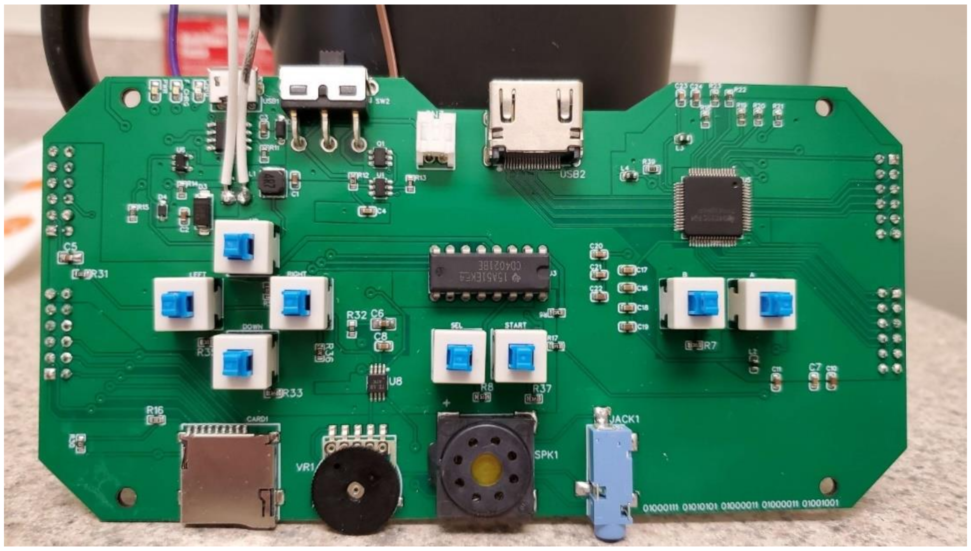


Fig. 23 PCB containing all necessary circuits

The PCB was designed in EasyEDA software and manufactured by JLCPCB. We designed the PCB to contain all necessary circuits needed for controller, power management/ charging, audio, SD card memory, and display output. To interface on top of the BASYS-3, we integrated PMOD headers on bottom of the PCB.

2.9.7 CAD

CAD design was conducted in FreeCAD software and printed in The Mason Exchange (The MIX) located in Horizon Hall at George Mason University. CAD design was alongside PCB design in order to create a case that would enclose the manufactured PCB and other component as well as leave openings for the user to handle such as the potentiometer for volume control, audio jack, and ports from the BASYS-3 and PCB.

2.9.7.1 3D Models

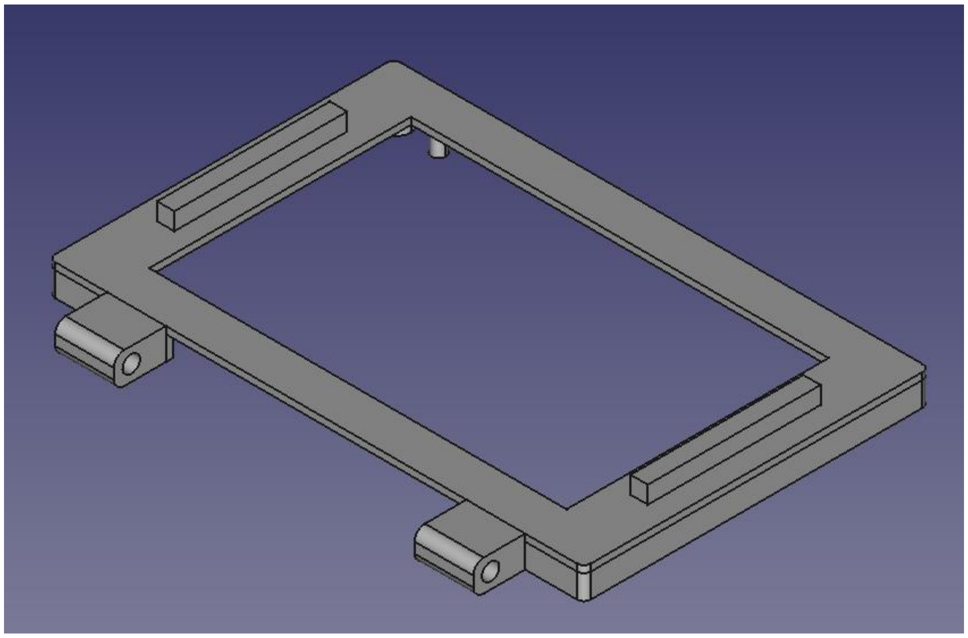


Fig. 24 CAD Design Top Screen Front Panel

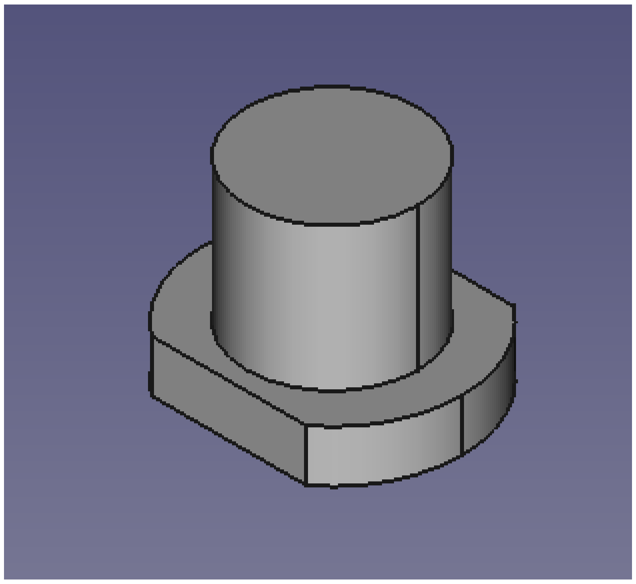


Fig. 25 CAD Design Button

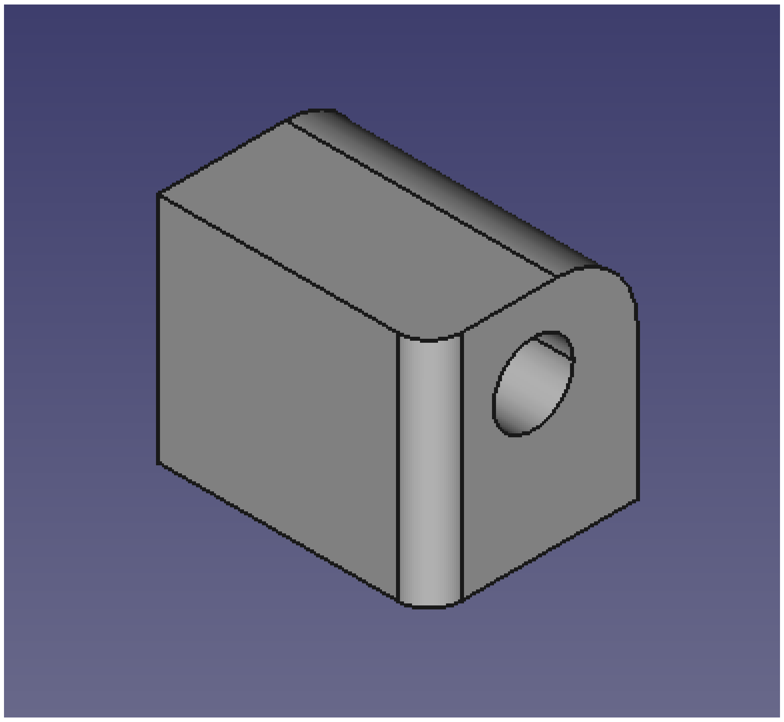


Fig. 26 CAD Design Hinge

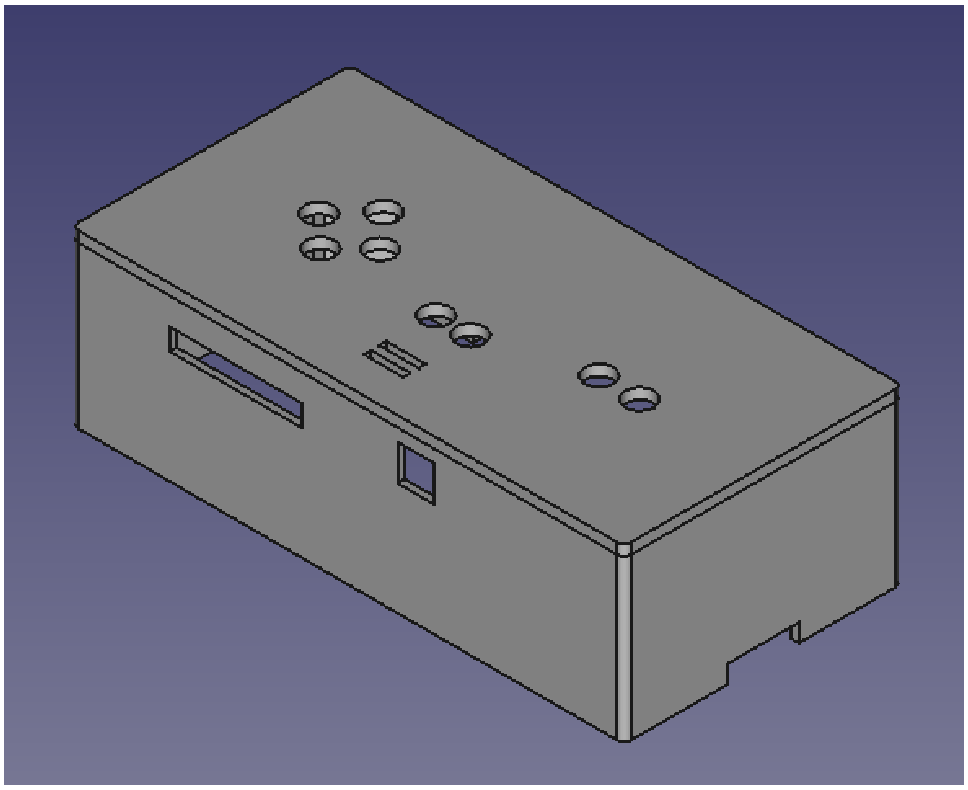


Fig. 27 CAD Design Bottom Case

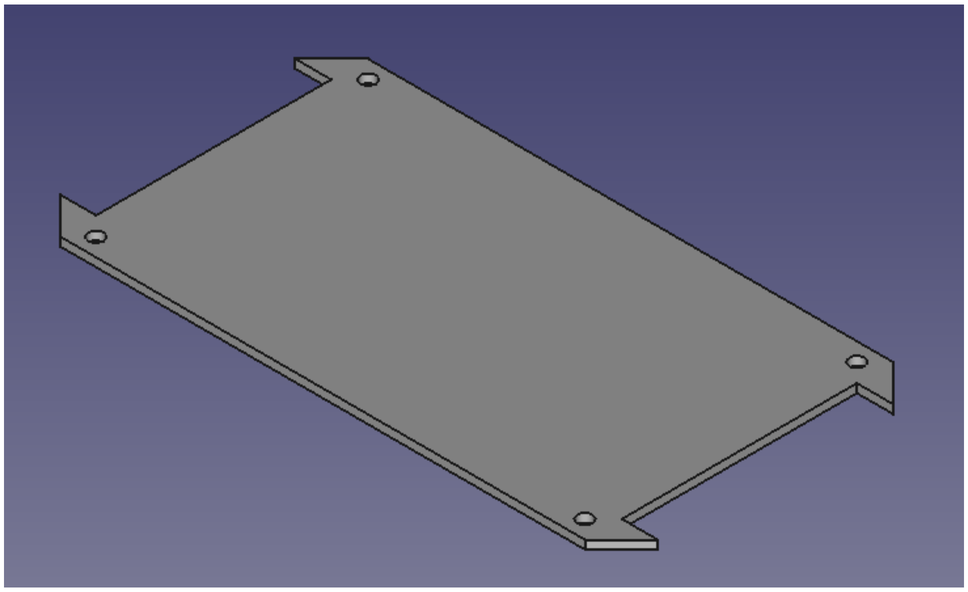


Fig. 28 CAD Design Internal Plate 2

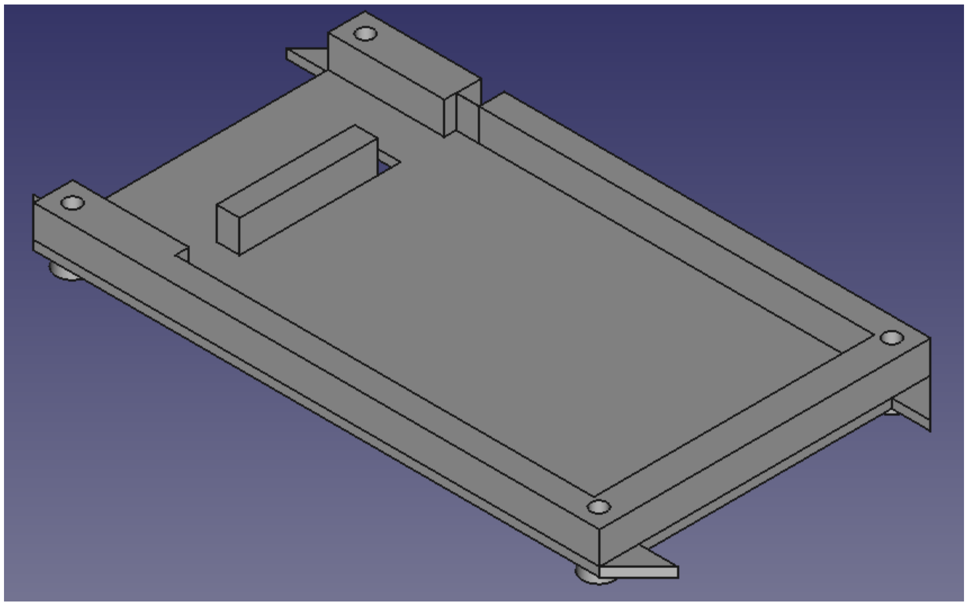


Fig. 29 CAD Design Internal Plate 1 (Battery Hold)

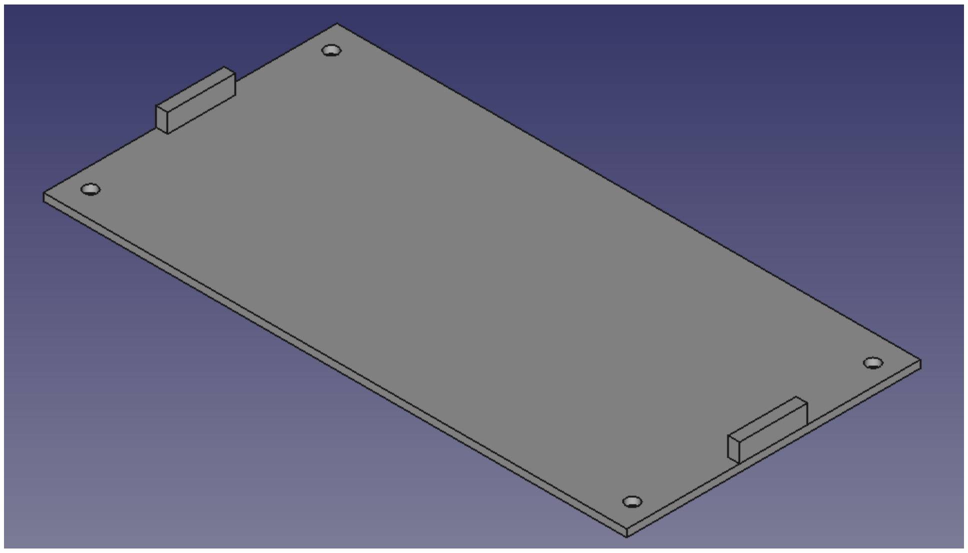


Fig. 30 CAD Design Case Bottom Panel

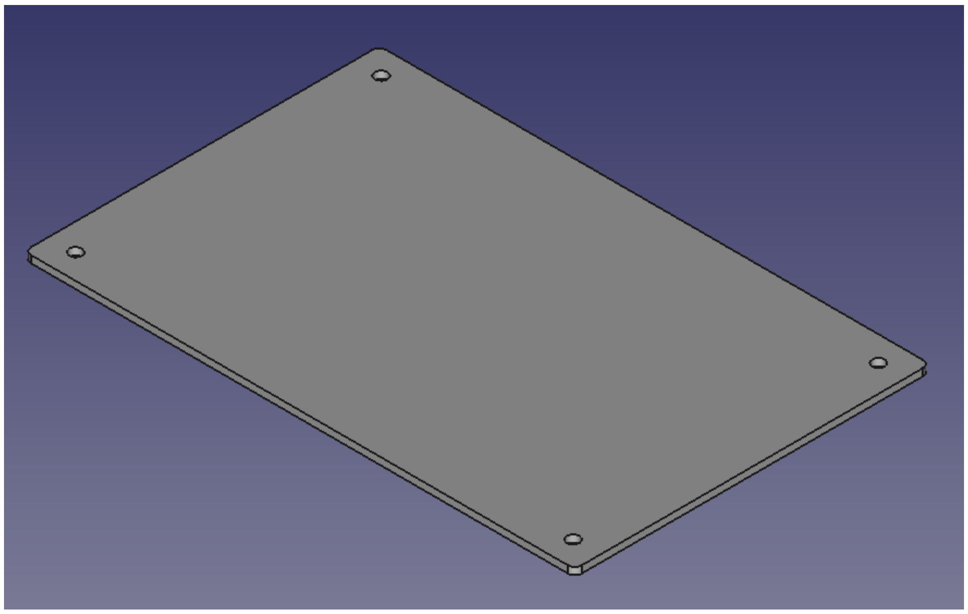


Fig. 31 CAD Design Top Screen Back Panel

2.9.7.2 Orthographic Projections (mm)

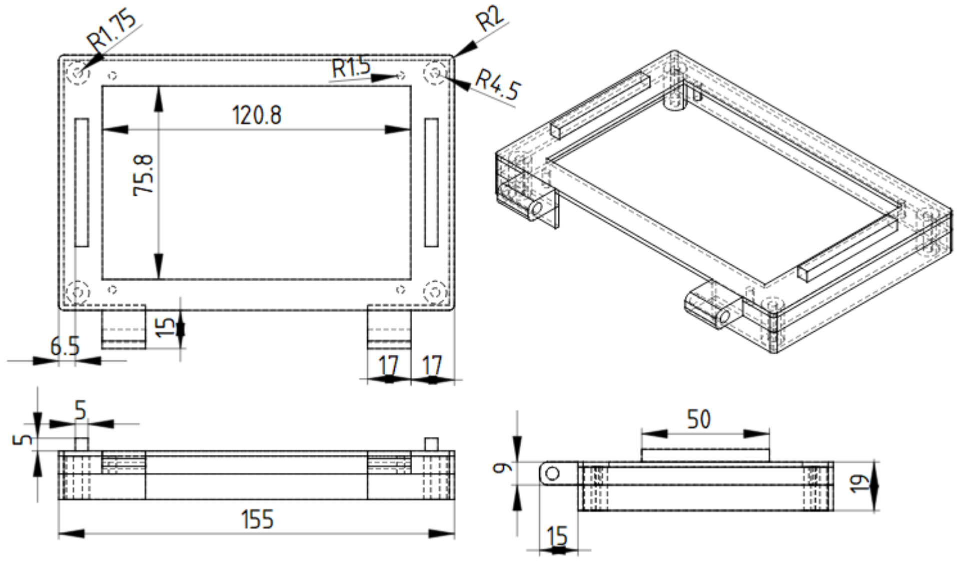


Fig. 32 CAD Design Top Screen Front Panel Orthographic Projection (mm)

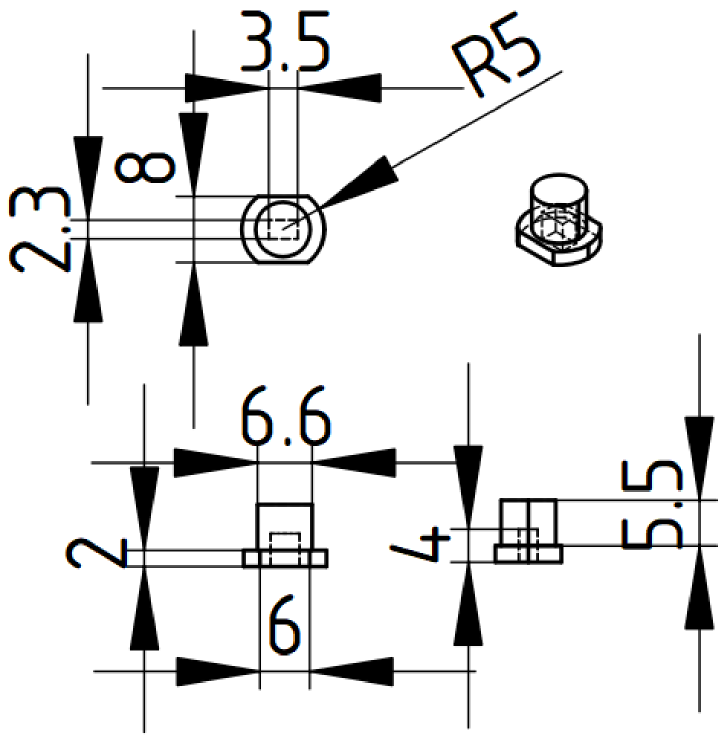


Fig. 33 CAD Design Button Orthographic Projection (mm)

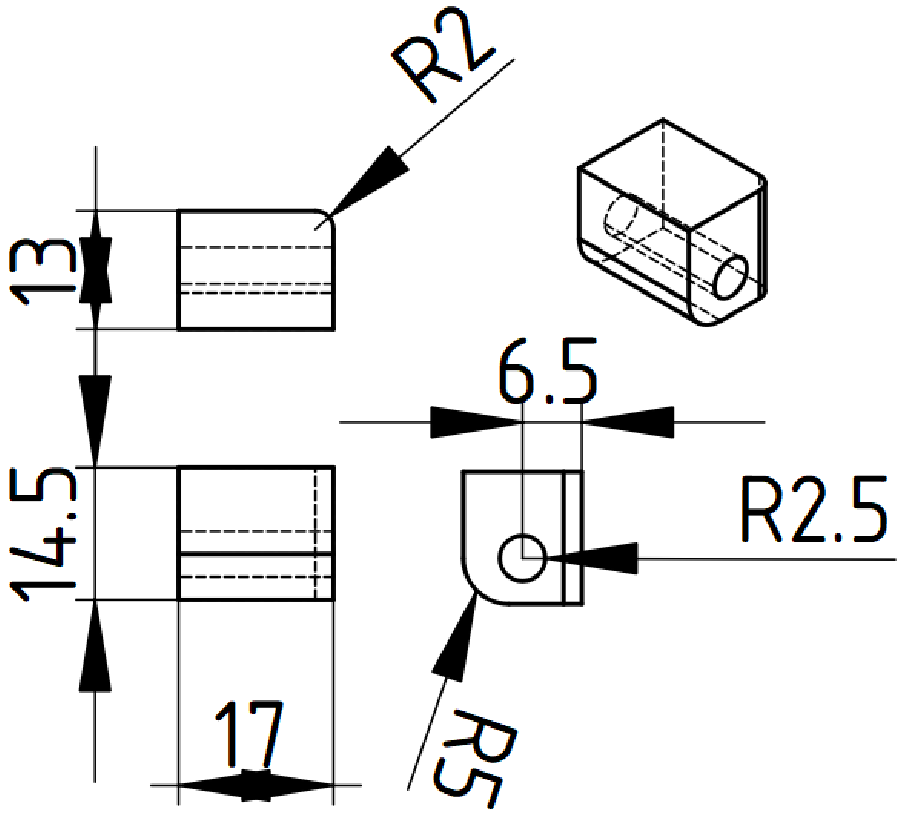


Fig. 34 CAD Design CAD Design Hinge Orthographic Projection (mm)

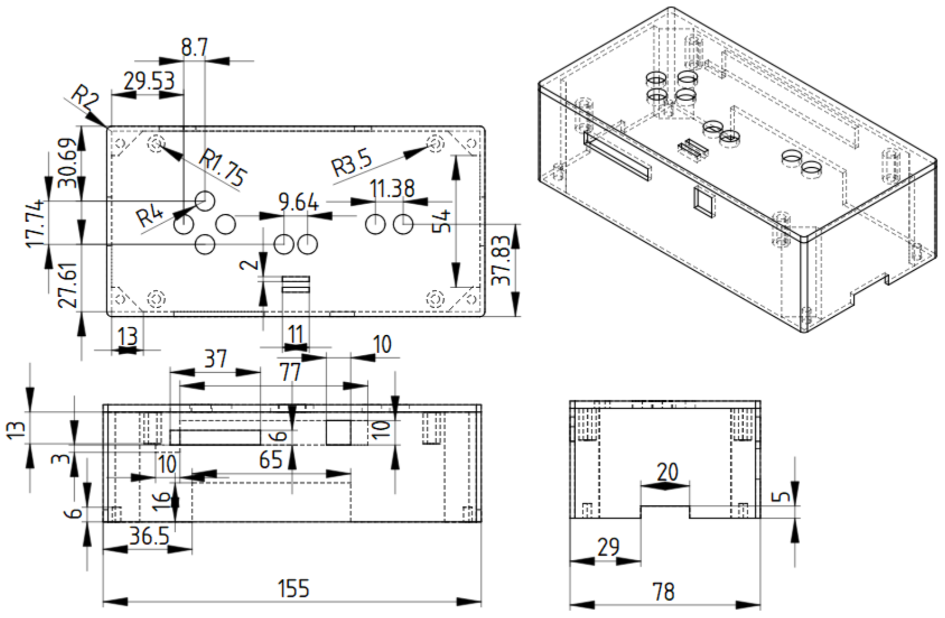


Fig. 35 CAD Design Bottom Case Orthographic Projection (mm)

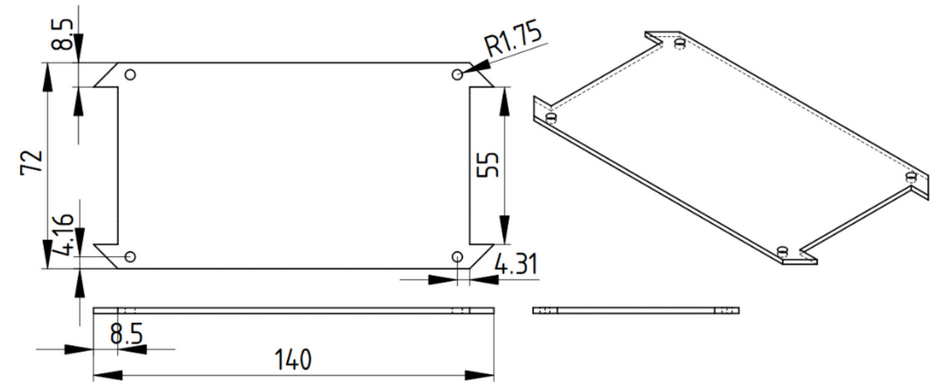


Fig. 36 CAD Design Internal Plate 2 Orthographic Projection (mm)

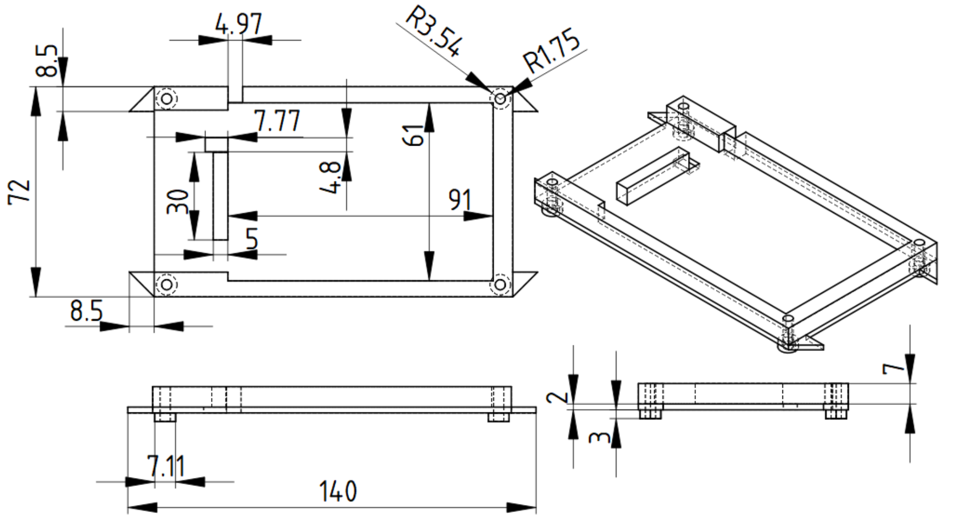


Fig. 37 CAD Design Internal Plate 1 (Battery Hold) Orthographic Projection (mm)

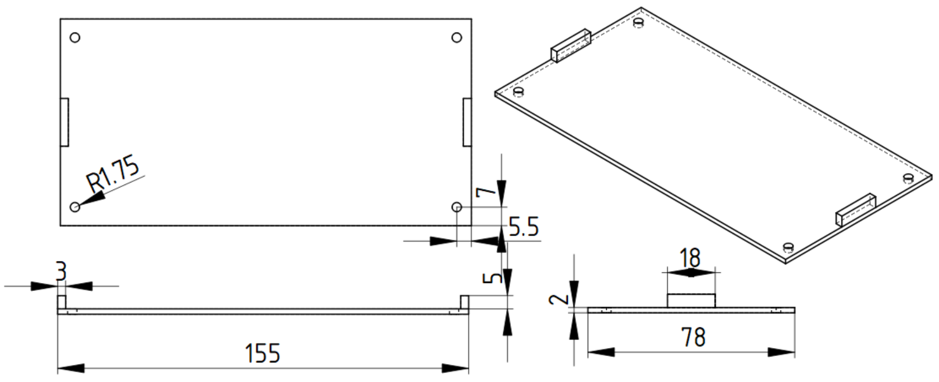


Fig. 38 CAD Design Case Bottom Panel Orthographic Projection (mm)

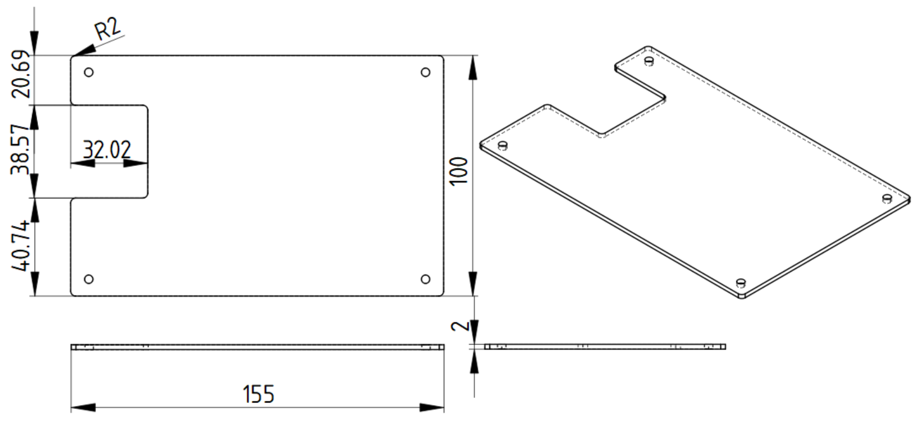


Fig. 39 CAD Design Top Screen Back Panel Orthographic Projection (mm)

2.9.7.3 3D Print Results

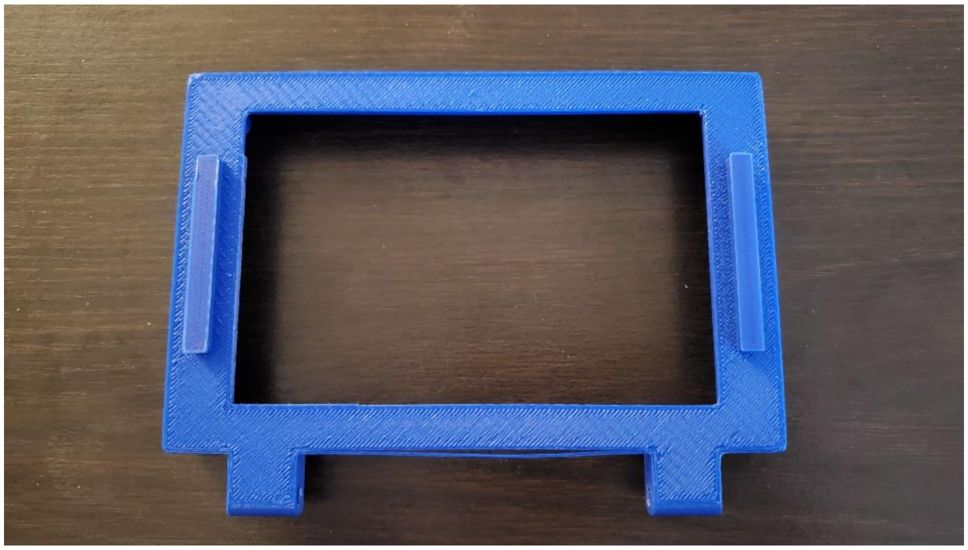


Fig. 40 CAD Design Top Screen Front Panel 3D Print

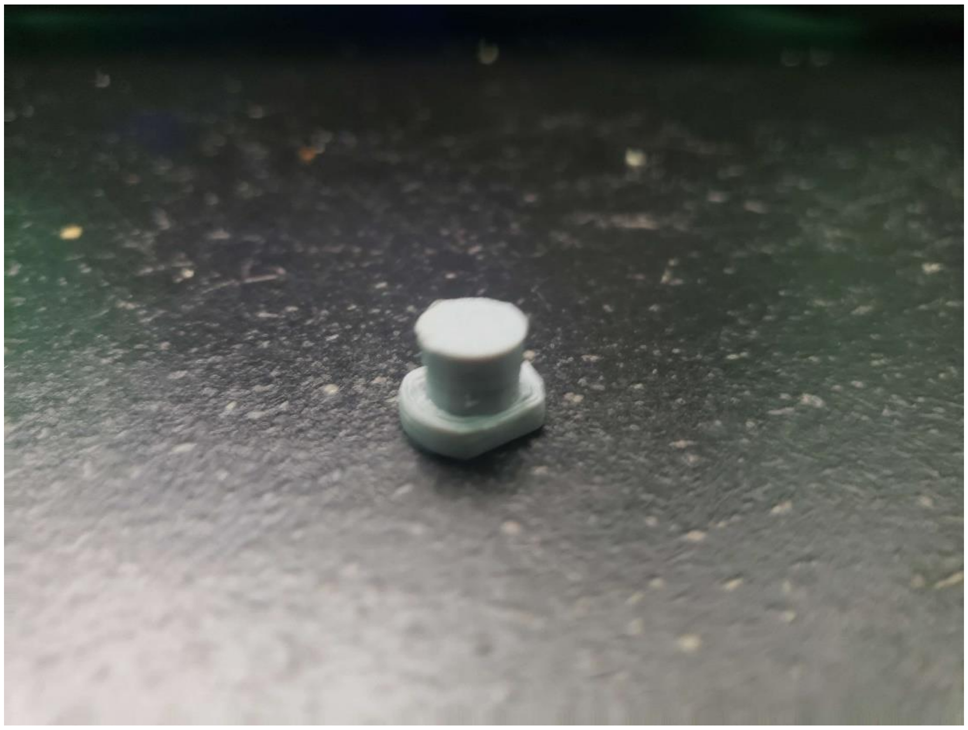


Fig. 41 CAD Design Button 3D Print

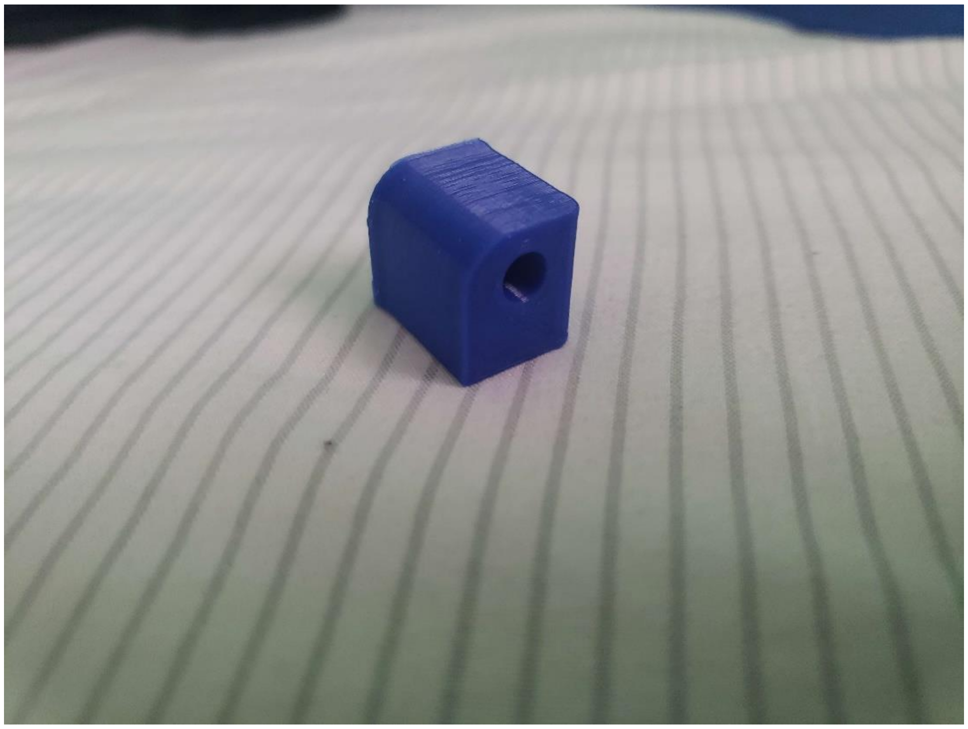


Fig. 42 CAD Design Hinge 3D Print

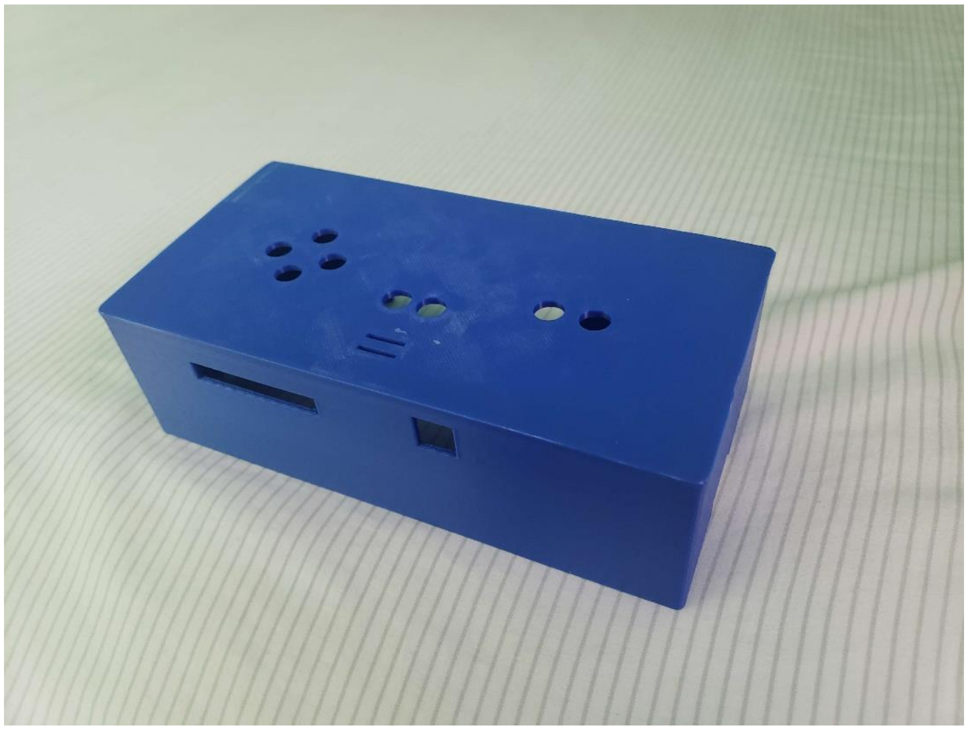


Fig. 43 CAD Design Bottom Case 3D Print

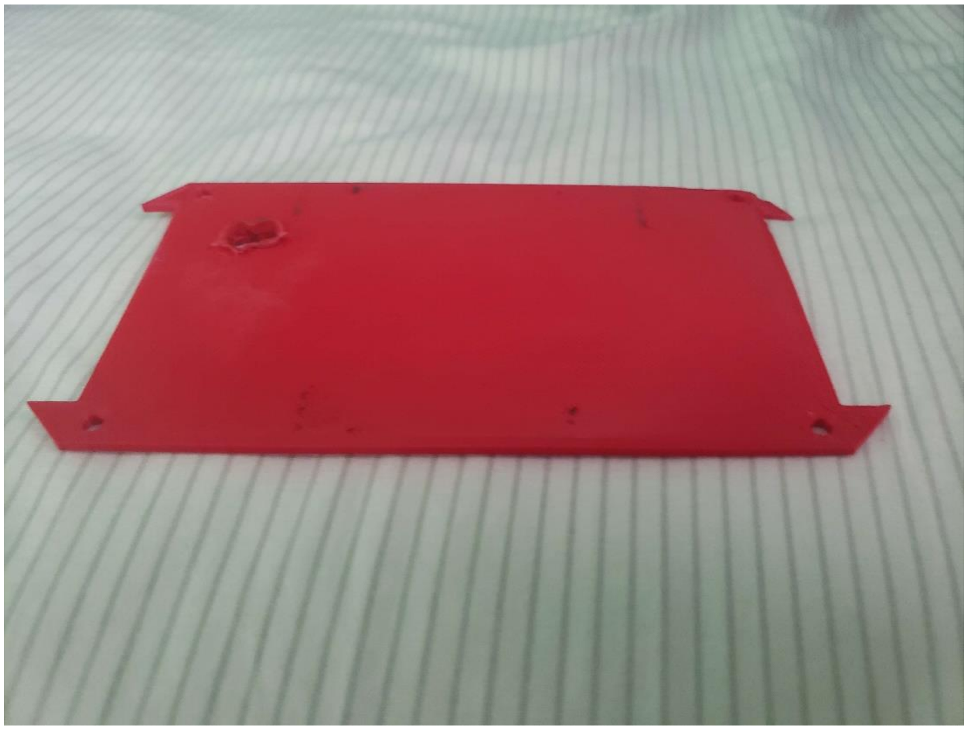


Fig. 44 CAD Design Internal Plate 2 3D Print

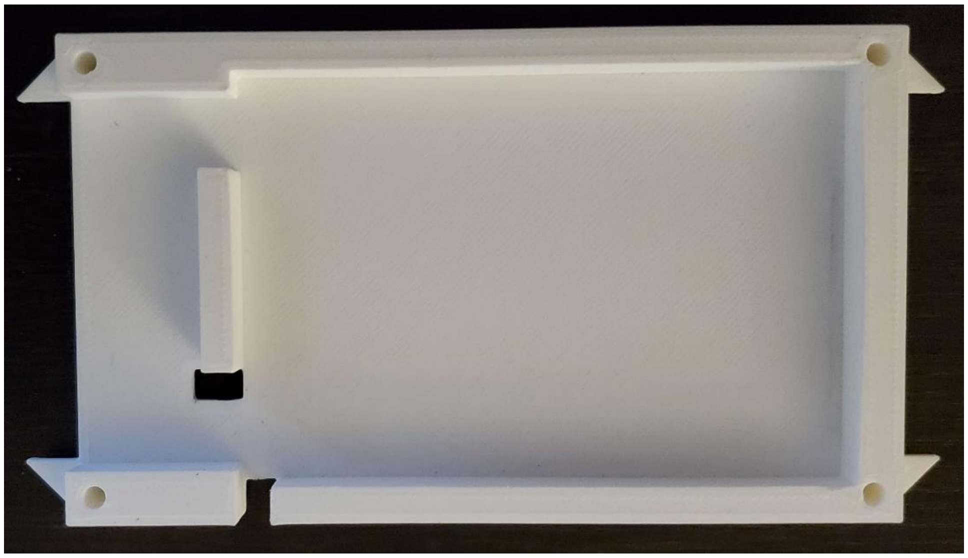


Fig. 45 CAD Design Internal Plate 1 (Battery Hold) 3D Print

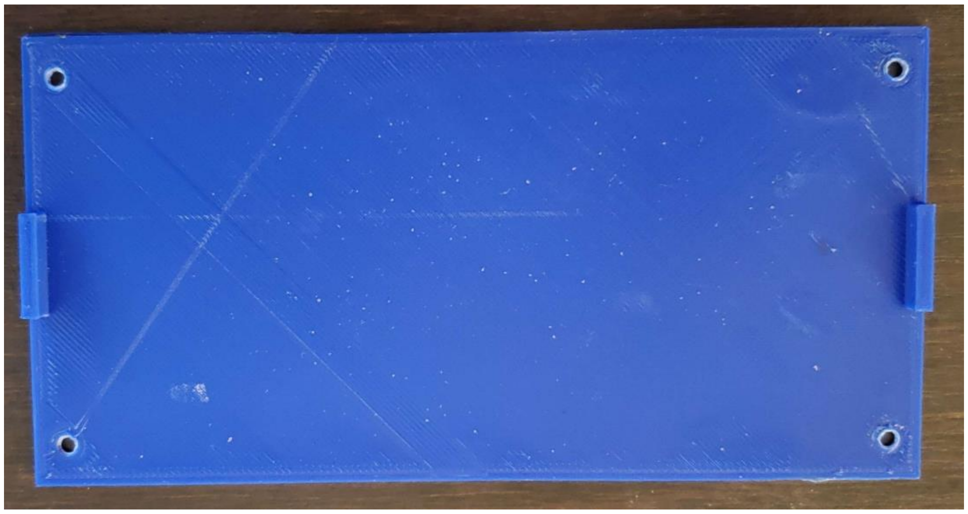


Fig. 46 CAD Design Case Bottom Panel 3D Print 3D Print

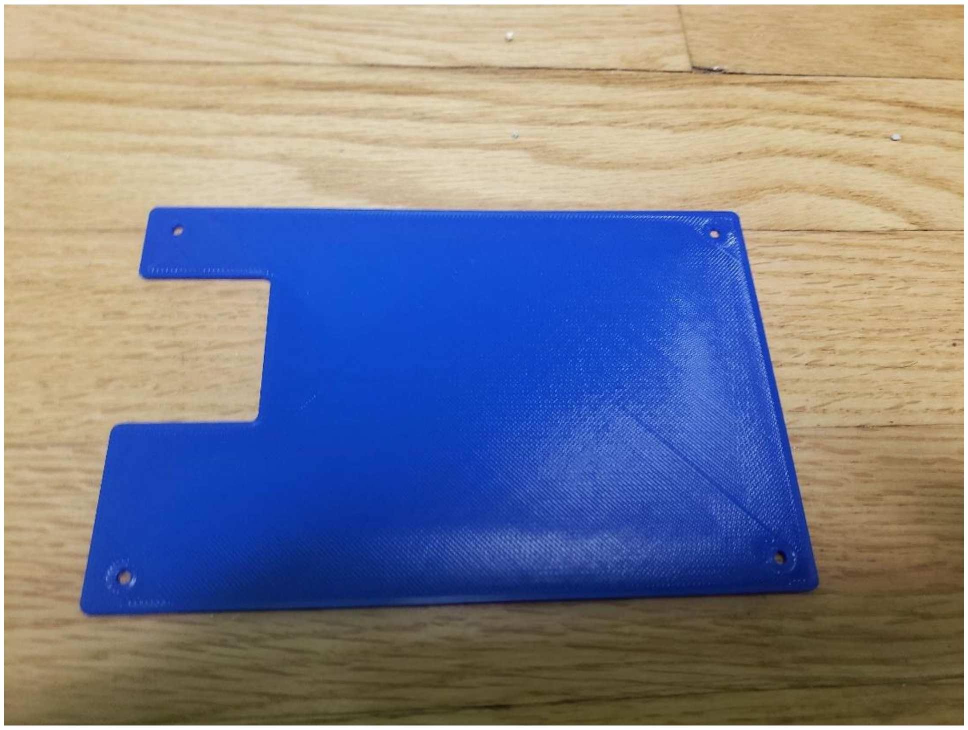
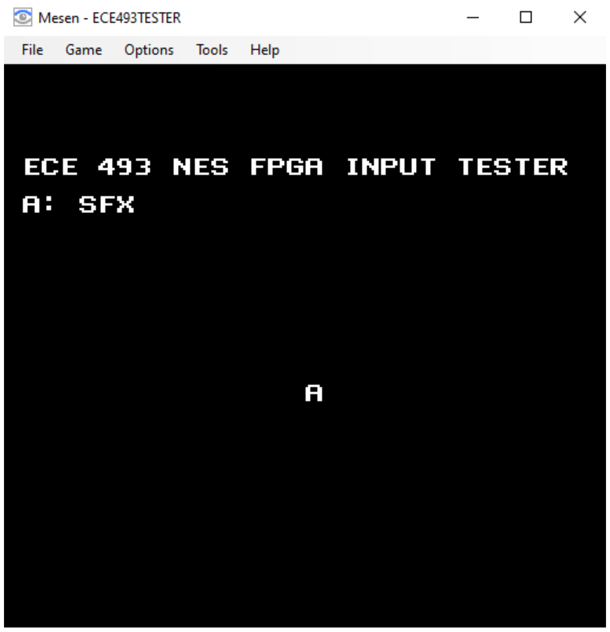


Fig. 47 CAD Design Top Screen Back Panel 3D Print

3. Experimentation

3.1 Introduction

Three experiments were conducted to confirm the playability and the portability of our NES FPGA device: Audio and Video Synchronization, Button Response Time, and Battery Lifetime on Full Charge. The success of these experiments was crucial to the user experience needed to use our NES FPGA device. Before we could measure latency experiments, we first decided to create our own NES “game”, named ECE493TESTER (Fig. 48), for testing purposes only; it is essentially for display button inputs on screen and for playing a sound when a button is pressed. Nesdoug’s tutorial on coding an NES game was used for our testing program [5].



ECE 493 NES FPGA INPUT TESTER
A: SFX

A

Fig. 48 The ECE493TESTER running on the Mesen Emulator while pressing the button A

3.2 Audio and Video Synchronization

3.2.1 Experiment Introduction

The goal of this experiment was to measure the latency between audio and video. In other words, we wanted to observe if audio cues from a game match up with the actions in-game that cause the audio.

3.2.2 Results

We decided to collect data of the audio and video by human observation, and we concluded that at the human level, the audio and the video are synchronized with an undetectable latency. We also observed that the quality of the audio speaker was subpar. Incorrect filtering or speaker component choice are factors that could affect the sound quality.

3.3 Button Response Time

3.3.1 Experiment Introduction

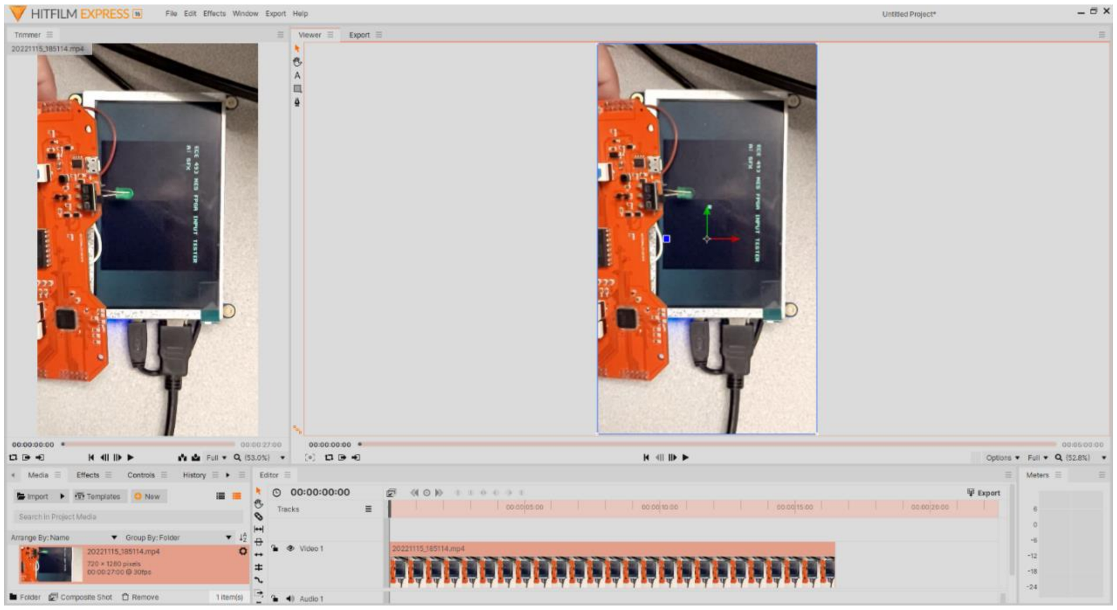


Fig. 49 The video recording was analyzed within HitFilm

The goal of this experiment was to measure the response time between a button press and an on-screen response. To take the measurements, we used a Samsung S10 smartphone with 960 fps and viewed the video in HitFilm, a video editing software (Fig. 49). The number of frames were counted in between, then the response time was calculated:

$$\frac{1}{960\ frame/second} = 1.04\ ms/frame$$

$$Response\ Time = CountedFrames \cdot 1.04\ ms/frame$$

3.3.2 Results

	P1	R1	P2	R2
Counted Frames	85	87	85	75
Response Time (ms)	88.4	90.4	88.4	78

P: Press Average = 86.32 ms
R: Release

Table 1 Measurements of frame data

3.4 Battery Lifetime on Full Charge

3.4.1 Experiment Introduction

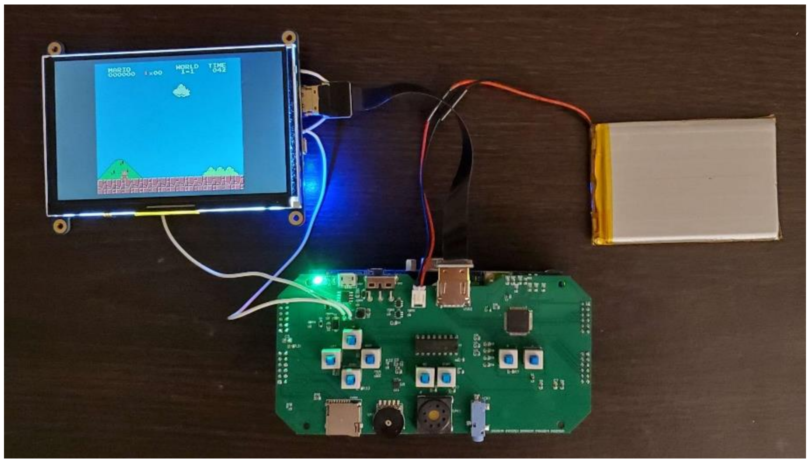


Fig. 50 Super Mario Bros. being played on the NES FPGA

The goal of this experiment was to test and compare our theoretical calculations of the battery life with experimental results of our measured battery life. We set up the experiment by loading the ROM for Super Mario Bros. and starting the game. In the game, Mario has about 2 minutes and 40 seconds to complete a level; when the time runs out, Mario loses a life. Since Mario only has 3 lives, the level needed to be manually reset every 8-10 minutes. This process was repeated until the NES FPGA lost power.

3.4.2 Results

After 3 hours and 3 minutes, the NES FPGA started to lose power to the display. Our initial battery life calculations gave us an estimated battery life of 3.36 hours:

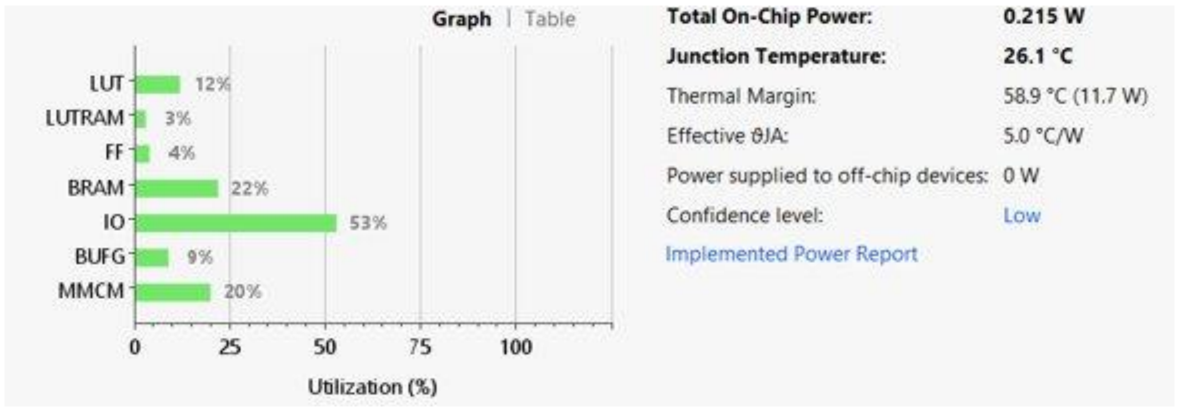
$$LCD\ Watts + Speaker\ Watts + FPGA\ Watts$$

Our experimental results yielded 10% less battery life than we had expected.

4. Experiment Validation

We had collected data on how our system performed but it was also important to validate our system qualitatively. We wanted our project to be as close to the real NES as possible in order to make sure that the user would get the same experience. The process for validating these results was relatively simple, we played Super Mario on the console and made sure to check for any input latency issues, display issues, or audio latency issues. This validation resulted in us concluding that the device offers an acceptably similar experience to the original NES.

The goal of this project was to provide a platform on which other systems can be emulated. We had to take careful account of how we used our FPGA's resources in order to allow growth. From Fig. 51, we have a very large margin for growth. Our only real limiting resource would be IO, but the amount of IO needed would remain relatively constant across platforms unless we decided to emulate something far more modern than the NES or SNES.



Total On-Chip Power:	**0.215 W**
Junction Temperature:	**26.1 °C**
Thermal Margin:	58.9 °C (11.7 W)
Effective ϑJA:	5.0 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Fig. 51 Utilization graph for FPGA

5. Project Lifetime

5.1 Maintainability/Maintenance of Final Design

The end user should have little maintenance when it comes to just using the device. The user may have to replace buttons as they wear down or replace the battery as it degrades. The overall casing that houses the system should be able to last a long time, though if necessary, it too can be replaced if the user has access to the 3D model stl file. We expect the electrical components such as the display, Basys-3, and PCB to last throughout the product’s lifetime.

5.2 End of Project’s Lifetime

All parts and components used in this project are easily sourced via electronic suppliers and it is quite unlikely that any given parts will reach the end of their lifetime and stop being manufactured. Moreover, all the parts used have standard alternatives that the user can easily substitute in place of our chosen components. When it comes to disposing of the device it is considered E-waste because of the battery so this means the user must dispose of it correctly. While the constructed project is able disposed of at the end of its “lifetime,” this project is capable of being replicated and improved upon should another choose to do so, so this project does not truly have an end.

6. Administration

6.1 Progress

With the start of ECE 492, we went through our planning and design process for the project. We touched on topics of how the device should visually look and what functionalities it should have. We then came up with solutions to solve our problem statement. In ECE 492, we conducted part selection after we finalized our design. This included searching for parts for the display, audio, battery, and storage. We did some preliminary work on schematic capture for the circuits, which also included prototypes of our controller and audio circuits via breadboards. We also explored some solutions for storing the game ROMs in memory.

During ECE 493, we started on the schematic capture and layout for the PCB that would mount onto the BASYS-3. Alongside the development of the PCB, we started work on the CAD enclosure that would house the BASYS-3, battery, display, and PCB systems, as well as the wires that connect them. The manufactured PCB we designed took two iterations with a few changes done in-between before it was properly functional. The CAD enclosure went through several different iterations, featuring various designs and implementations for each part. We also started writing the RTL for the storage and attempting to save game ROMs onto the SD Card. The RTL was written for the VGA-HDMI converter and for the start menu screen. We ran into issues with saving game ROMs onto the SD Card and were not able to complete that functionality by the end of ECE 493.

6.2 Changes to Design

We had to make several changes to the design especially for the design concerning memory storage of the video game ROMs. Starting out the goal was to use the 6502 core inside the NES implementation to run software that would load the ROMs that the user wanted to play. We decided to not pursue this design and instead opted for a Verilog implementation of a start menu. The start menu works by being the first screen that the user sees when the device is powered on. In the start menu, the user will be able to choose the game to play. This is opposed to the software approach where this start menu would've been entirely in software ran by the 6502 core.

When deciding on how to get our video signal to the LCD, we initially were going to use a ZIF connector on the PCB so that we could use a ribbon cable coming out of the case to take up less space. In order to do that we would have had to use a ZIF to HDMI converter PCB to connect to the LCD. We thought this was not the best solution, so we decided on replacing the ZIF connector on the PCB with a female HDMI and finding a ribbon cable HDMI cord to get the signal to the LCD.

During the circuit design process there were multiple other times where a component was unavailable or not what we needed/wanted. In our audio module we ran into a problem where the chip we wanted to use was out of stock and the only one we had was a very small BGA configuration so we were not confident that we could solder it onto our PCB without damaging the board or creating a short. This issue led us to find another mono-channel audio amplifier that would have an adjustable gain and not consume too much power. Another circuit design issue was the display module. When looking for a solution to make our display portable, we had to look to VGA to HDMI conversion because we could not find a small enough VGA screen. To do this we started by breaking open a cheap VGA to HDMI converter cable and seeing what was inside. After doing that we ran into the problem of not being able to obtain the required chip and its datasheet to know if we were hooking it up correctly or not. Finally, after some more searching and research we were able to find a company that had previously done VGA to HDMI conversion through PMOD and decided to recreate it so it would work with the BASYS-3.

One of our design choices was about determining the physical design or construction of the device. Originally, we went with a single block design, where the controls and screen were on the same face of the case. This would have meant our PCB design would have had to been changed to support both the buttons and display next to each other. We opted to go for a display that is hinged and allows us to close the display. This would have also affected how much filament we would use and how we designed the daughter board, however the effects and impacts of using this design are unknown.

A design choice involving the display was either using an adapter cable for VGA to HDMI or using a chip specially made for converting VGA to HDMI. We opted to go for the chip specially made for this situation. This allowed us to place it on the PCB and save space inside the case of the device. The adapter on the other hand was bulky, unwieldy and too complicated to power inside the case.

6.3 Unplanned Activities

We faced a variety of obstacles through the duration of this project. One of the more impactful obstacles was sourcing the PCB and the components itself. Ordering parts that originate from a different country proved to take longer than we thought, and the shipping is unreliable.

6.4 Funds Spent

Buyer	Quantity	Part	Cost	Cost w/ Quantity	Shipping	Tax	Total	Cumulative Cost	
Amilcar	1	P-MOD Audio Amp	9.99	9.99	4.99	0.6	25.57	Total	Total w/o shipping and tax
Anthony	1	P-MOD MicroSD	6.99	6.99	0	0	6.99	547.97	428.61
Josh	1	P-MOD MicroSD	6.99	6.99	9.99	0	16.98		
Sam	1	P-MOD MicroSD	5.94	5.94	0	0	5.94		
Brooks	1	NES Controller (Potted)	7.79	7.79	0.47	0	8.26		
Brooks	1	NES Controller (Genuine)	8.98	8.98	4.89	0	13.87		
Sam	1	Lithium Battery 3.7v 4000mah	13.99	13.99	0	0.88	14.87		
Sam	1	Digital Caliper	8.99	8.99	0	1.56	10.55		
Sam	1	VGA to HDMI	16.99	16.99	0	0	16.99		
Sam	1	Adafruit Lipoly Charger	12.5	12.5	0	0	12.5		
Sam	1	3pcs DC to DC booster	9.99	9.99	0	0.6	10.59		
Brooks	1	VGA to HDMI	17.99	17.99	0	0.95	18.94		
Brooks	1	Controller Components	1.87	1.87	8.82	0	10.69		
Brooks	1	VGA-hdmi components	1.48	1.48	8.54	0	10.02		
Brooks	1	VGA-hdmi chips	17.56	17.56	6	1.05	24.61		
Josh	2	VGA-hdmi break-out board	19.45	38.9	7	0	52.9		
Sam	1	3 inch embedded display	37.99	37.99	0	6	43.99		
Anthony	2	HATCHBOX 1.75mm True Blue PLA 3D Printer Filament, 1 KG Spool, Dimensional Accuracy +/- 0.03 mm, 3D Printing Filament	24.99	49.98	0	1.5	52.98		
Brooks	1	PCB Prototype	93.75	93.75	20.01	0	113.76		
Sam	1	5 inch display	59.95	59.95	13.42	3.6	76.97		

Table 2 Total funds spent during ECE 492 and ECE 493

6.5 Man-hours Spent

Task	Sam	Josh	Anthony	Amilcar	Brooks	Total
Overhead	53	40	40	44	40	217
Storage	90	15	0	0	0	105
Video	20	73	0	0	0	93
Rebuild	80	80	0	0	0	160
Circuit	0	0	20	36	130	186
PCB	0	0	4	56	25	85
Case	2	0	57	6	2	67
Documentation	15	12	20	80	15	142
Testing	19	19	10	15	5	68
Manufacturing	0	0	62	0	0	62
Component Research	10	10	10	10	40	80
Total Hours/Person	289	249	223	247	257	1265

Table 3 Hours each person spent on tasks

Division	Sam	Josh	Anthony	Amilcar	Brooks	Total
Overhead	68	52	60	124	55	359
FPGA (HDL)	190	168	0	0	0	358
PCB Design	31	29	101	123	202	486
Manufacturing	2	0	119	6	2	129

Table 4 Hours each person spent on division

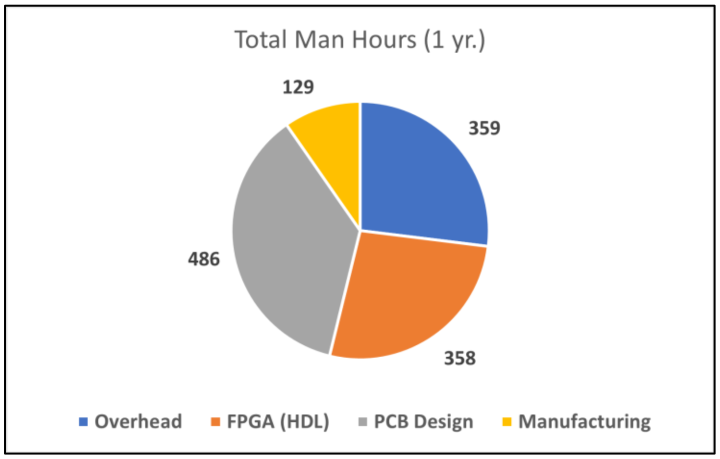


Fig. 52 Chart of total man-hours spent on our project in ECE 492 and ECE 493

7. Lessons learned

7.1 Skills

Time management was essential for the project. During ECE 492 and ECE 493, there were many moments where we would be behind our intended schedule and had to work harder as a result to catch up with our plans. Teamwork was another group skill that we worked through and improve as the year passed. As our members were split into sub-sections, we worked together as each group coordinated with each other as progress was made. Finally, we worked on our documentation skills as we prepared for proposals, progress reports, and presentations throughout the year. There were some challenges with document formatting, providing correct and necessary information, and making our work professional to be given to our faculty supervisor for review.

7.2 Knowledge

7.2.1 Verilog

During this project, we had to learn many different skills. One of the skills we had to learn, in which we had no prior exposure to, was the Verilog HDL. The original NES implementation was entirely in Verilog, and converting such a project to VHDL, a language we had learned in our GMU courses, was unfeasible. We decided on learning Verilog HDL and implementing our solution entirely in Verilog HDL.

7.2.2 PCB Design

During the process of designing the PCB, we learned how to fit all necessary components onto the PCB with a predefined amount of space and only two layers to work with. The use of vias and copper areas helped with these limitations. Also, having to find different components that take up less space also helped.

7.2.3 6502 Architecture

7.2.4 CAD Design

During ECE 493, alongside the PCB design, we had to learn to design, model, and print an enclosure for our system. The main goal was to design an enclosure that would be able to fit all the components we chose for our project and still be a manageable size to be handled. Measurements were carefully taken to allow the PCB design and 3D prints to fit smoothly together and allow the necessary parts such as audio jack and potentiometer be accessible to the user.

8. References

[1] Bob, "How a NES controller works," RetroRGB, 7 March 2020. [Online]. Available: https://www.retrorgb.com/how-a-nes-controller-works.html. [Accessed 29 November 2022].

[2] Esden, "PMOD Digital Video Interface," 1BitSquared, [Online]. Available: https://1bitsquared.com/products/pmod-digital-video-interface. [Accessed 3 December 2022].

[3] B. Bennett, "FPGA-based Nintendo Entertainment System Emulator," 14 July 2012. [Online]. Available: https://github.com/brianbennett/fpga_nes. [Accessed 2022].

[4] "NesDev Wiki," 22 July 2022. [Online]. Available: https://www.nesdev.org/wiki/Nesdev_Wiki. [Accessed 2022].

[5] Dougeff, "How to Program an NES game in C," 2 January 2022. [Online]. Available: https://nesdoug.com/. [Accessed 2022].

9. Appendix A: Proposal (ECE-492)

ECE-492 PROPOSAL

NES handheld emulation breakout board for the BASYS-3

Team Members: Joshua Thomas Riggs, Brooks Aaron Corbett, Anthony Thinh Tang, Samuel Kifle Kebadu, Amilcar Alejandro Paniagua

Date of Submission: 3/11/2022

Faculty Supervisor: Dr. Jens-Peter Kaps

Executive Summary

Options for Nintendo Entertainment System (NES) emulation are limited with the requirements of being portable and hardware emulated. Our solution is to design a breakout board with necessary interfaces and inputs/outputs, design an encasing for the purpose of being portable/handheld, and include cost effective components such as the LCD screen, battery, SD card, and audio speaker. This project will result in allowing the BASYS-3 to be a new platform for NES emulation and in providing a new option for DIY NES emulation. Our team is composed of five members, majoring in Computer Engineering and Electrical Engineering:

Team Member	**Relevant Experience**	**Skills**
Samuel Kebadu (CpE) (PM)	ECE 447, 448 FPGA Internship	Verilog, VHDL, C++, C, Computer Architecture
Amilcar Paniagua (CpE)	ECE 286, 447, 448	VHDL, C++, C, Circuit Design, Soldering
Brooks Corbett (EE)	ECE 286, 447 Hardware Engineering Internship	Circuit Design, PCBA Design, Power system design, Soldering
Joshua Riggs (EE)	ECE 448, 445, 436	VHDL, Digital Design, PCB Design
Anthony Tang (EE)	ECE 286, 445, 447	AutoCAD, Circuit Design, Soldering, C

Problem Statement

NES emulation can vary in range of prices, with each method containing their advantages and disadvantages. We are motivated to seek a more cost-effective method for hardware emulating the NES in a portable package, meant to be hand-operated with a self-contained power source. We are also motivated to bring a new option for NES emulation with the BASYS-3.

As of Q1 2022, original NES home consoles can only be obtained secondhand. The most convenient method to play an NES is emulation by software. However, software emulation runs on personal computers with general architectures, which results in large overhead from trying to interpret hardware instructions meant for one architecture as software instructions. The more authentic solution is to recreate the NES via hardware. Some advantages of genuine replication of the original architecture include zero performance overhead, as well as accurate execution of programs, including any bugs or exceptional cases in either program or hardware. The best way of recreating hardware designs quickly and flexibly is through a Field-Programmable Gate Array (FPGA); a chip of programmable hardware logic. In fact, this is not the first redesign of the NES using an FPGA.

There are a few products in the current market that allow the NES to be played. Nintendo officially sells the NES Classic: a smaller-sized version of the home console. This runs NES games through software emulation on an ARM processor, meaning there are inaccuracies with the emulation, and only works with games purchased in its separate online store. More market options include the Pocket and NT Mini from the company Analogue. Both are High-end hardware “emulators” built on FPGA’s for not only the NES but other consoles as well. While both offer full recreation, and the Pocket is also portable, these systems are priced over $300 and lock their FPGAs and hardware to prevent tampering. For more flexible hardware implementations, there are DIY options. Some DIY projects, however, are complex in such a way that it may require many external parts and multiple levels embedded hardware and circuit design knowledge. One example is the MiSTer FPGA. Our NES FPGA project will allow flexibility within the build; our solution aims to be portable, handheld, cost-effective, hardware unlocked, DIY-friendly, and modular.

Approach

Since original NES consoles are not commercially sold anymore or are expensive to purchase secondhand, we want to develop a cheaper alternative that still maintains portability and simplicity. In addition, some current market alternatives use hardware locked FPGAs; we want our NES FPGA project to be "unlocked" to allow programmability to the BASYS-3 within in case the user wants to use the FPGA platform for another project.

In order to prototype our project, we start with the BASYS-3 and hardware language. Credit is given to Brian Bennett, who provided his recreation of the NES architecture written in Verilog, available to the public on GitHub. Input and output signals will be adjusted to match the pins of the BASYS-3. A VGA display, an NES controller, and an audio speaker will be required to playtest the emulated NES, along with two BASYS-3 PMOD expansions for audio amplification and SD card readding. The NES architecture can be stored in the BASYS-3 as its default program via USB Universal Asynchronous Receiver-Transmitter (UART) communication, which can also load the NES game ROMs. This will be integrated into the Verilog code to read game ROMs from an SD card through Serial Peripheral Interface (SPI).

Additional components, such as a small LCD screen and a battery power source, will be chosen to allow the device to have portable/handheld functionality. To minimize the dimensions of our device, we will construct our own breakout board that will contain the audio amplifier circuit, SD card reading circuit, button interface circuit, and power management circuit.

Time and budget constraints may arise during the prototyping phase. There are several alternatives that can be made:

- External display, controllers, and audio instead of embedded

- Not using the BASYS-3, instead simply using an FPGA soldered to the breakout board
- Using a different storage solution as opposed to an SD card i.e., memory bank
- Using a wired power connection as opposed to battery

Using external components will make the device lose its portability and handheld functionality. Next, switching out the BASYS-3 with a soldered FPGA to the breakout board will minimize the device's dimensions. Another alternative is using a memory bank in the BASYS-3 to store game ROMs, although this would disallow a user to "plug and play" their own game ROMS via SD card. The last alternative is to use a wired power connection and eliminate the use of a battery. Handheld functionality can be preserved but portability of the device will be lost. These alternatives will greatly shift the time and budget specifications of this project.

As mentioned before, Brian Bennett's FPGA project of the NES architecture on GitHub will be implemented into the BASYS-3. He reports that his project is missing mappers and the Delta Modulation Channel (DMC). He also provides a C++ windows application named NesDbg that will allow communication to the FPGA via USB UART. NesDbg can be used to runs tests and to load game ROMs.

Nesdev Wiki provides a plethora of information on programming topics for the NES. It will prove to be useful when we start to develop a game selection menu and to change memory map values accordingly to each game stored on the SD card.

Project requirements specification

Mission Requirements

- The device shall emulate the original NES by creating a breakout board for the BASYS-3 board.
- The device will provide a new platform for emulating NES.

Operational Requirements

Input/Output Requirements

- The device shall output audio to either speaker or headphones.
- The device shall accept user input through push buttons.
- The device shall display video output to an LCD.
- The device shall have indicator LEDs

External Interface Requirements

- The device shall read/load games via SD card/Memory chip.
- The SD card/memory chip should be programmable via micro-USB.

Functional Requirements

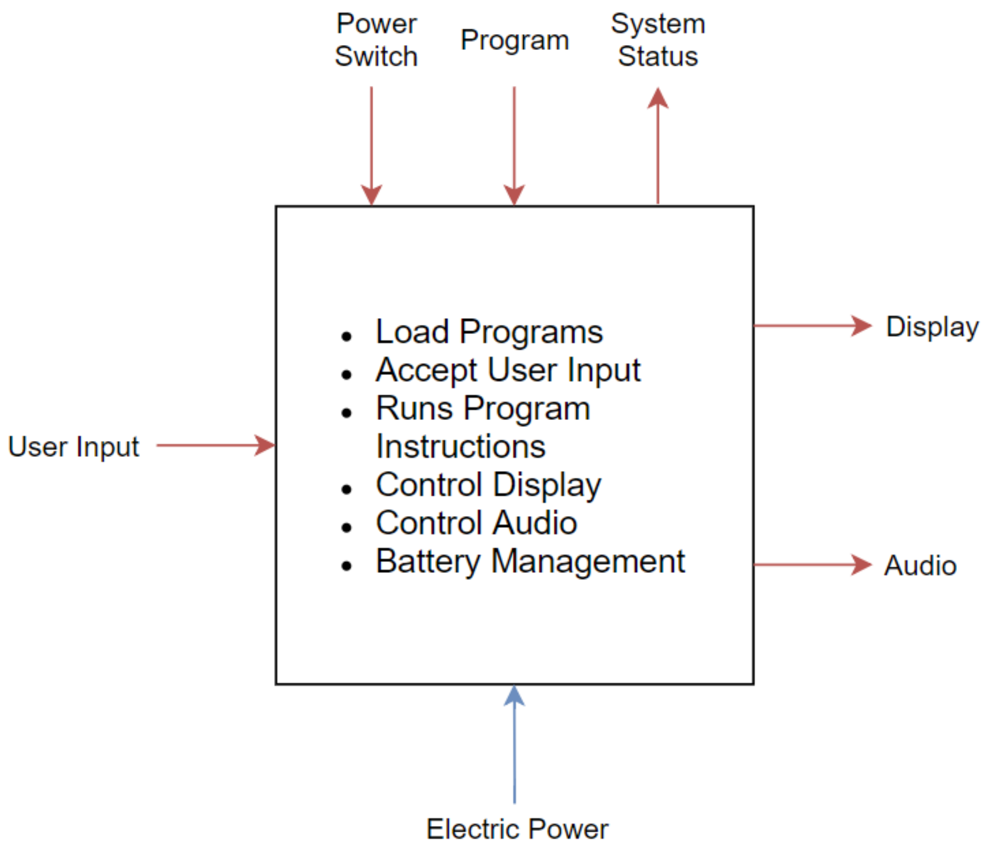
- The device will start on a main menu.
- The device shall allow the user to be able to select a game on the menu.
- The device shall allow the user to adjust audio volume level.
- The device will run games at the same clock speed as the original NES @1.79MHz.

- The device shall be able to continuously play for 3.3 hours on a full charge.

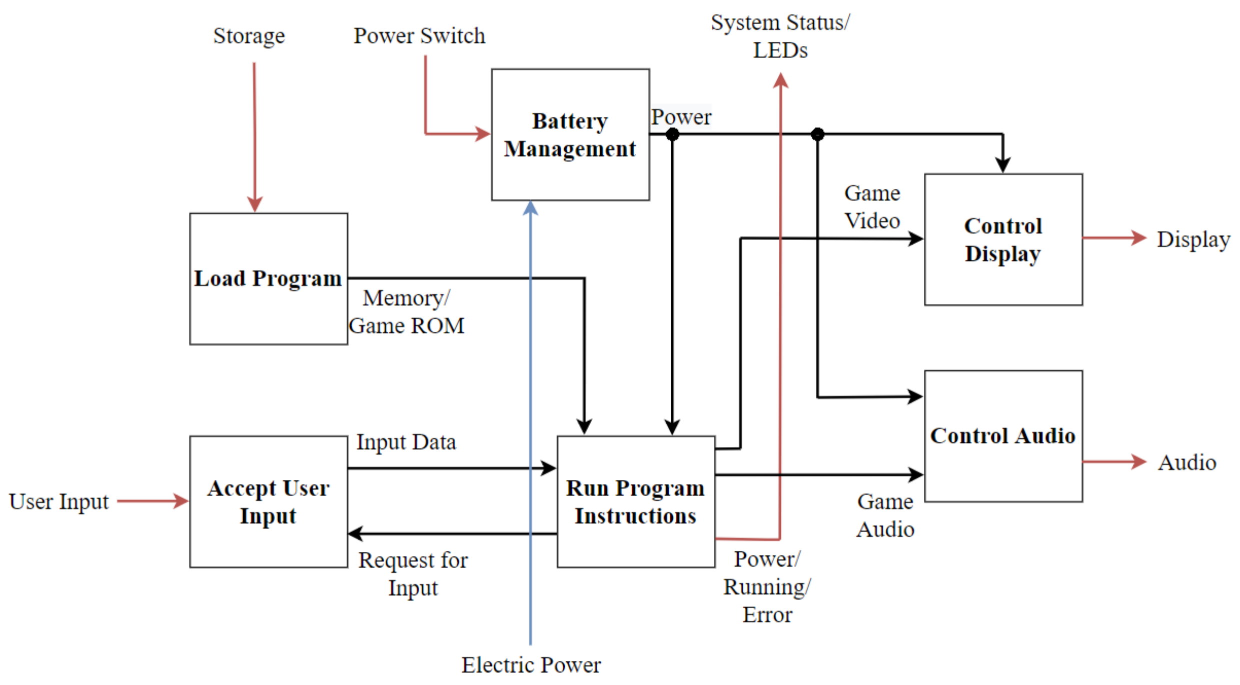
Technology and System-Wide Requirements

- The device should cost less than $150
- The device shall use the BASYS-3 board.
- The device should use VGA as its display interface
- The device should be portable.

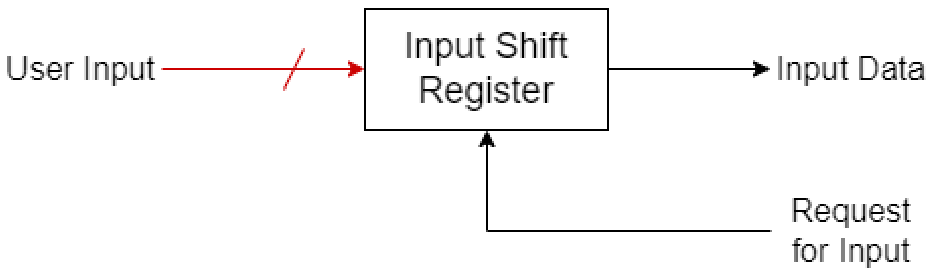
System Design



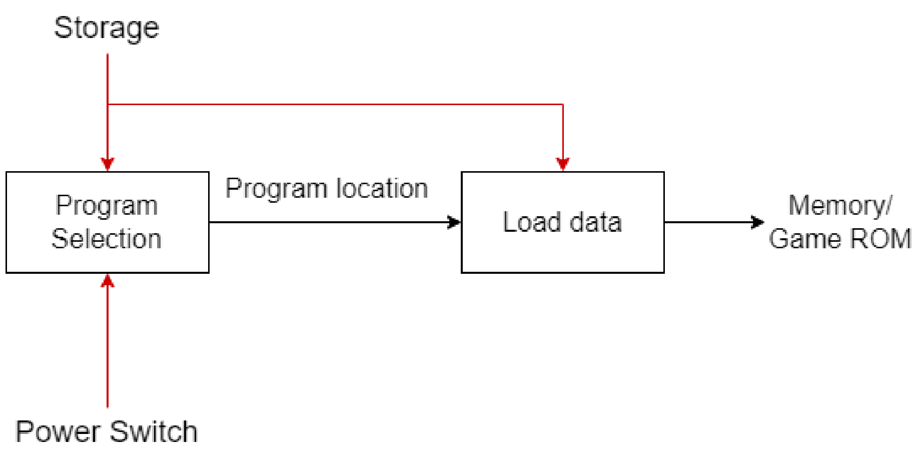
Functional Decomposition (Level-0)



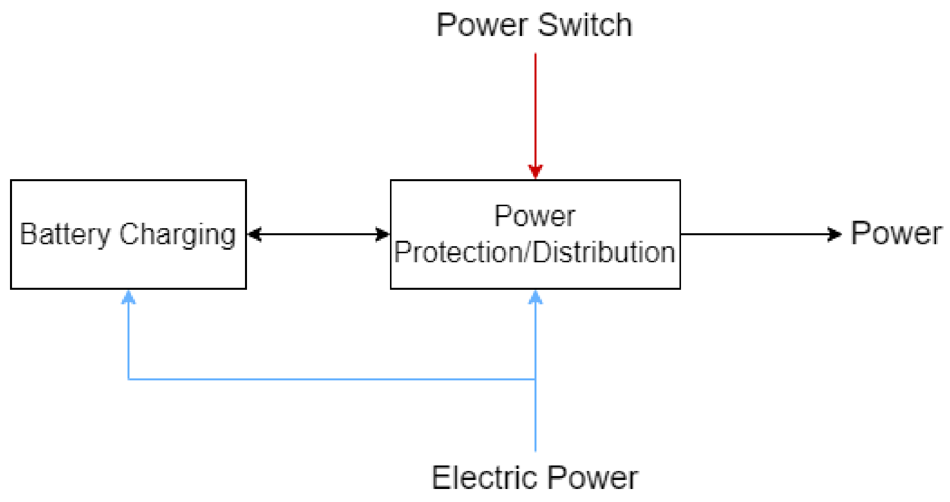
Functional Decomposition (Level-1)



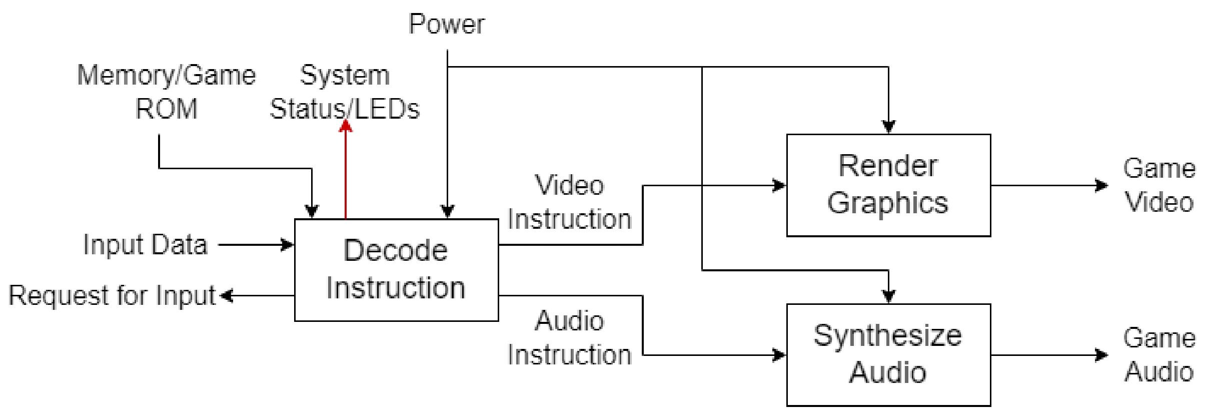
Functional Decomposition (Level-2): Accept User Input



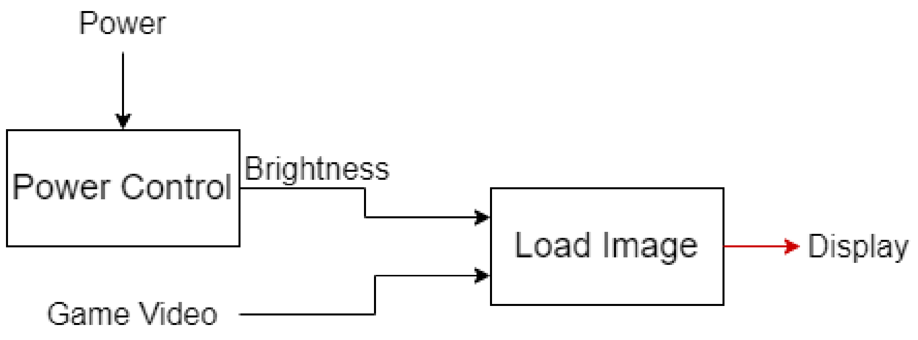
Functional Decomposition (Level-2): Load Program



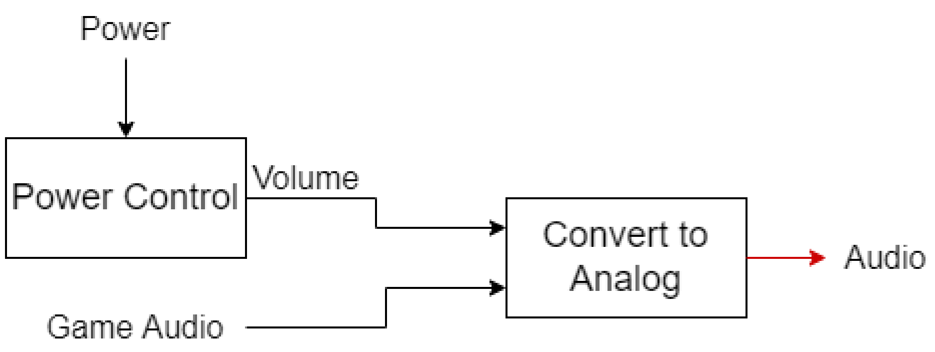
Functional Decomposition (Level-2): Battery Management



Functional Decomposition (Level-2): Run Program Instructions

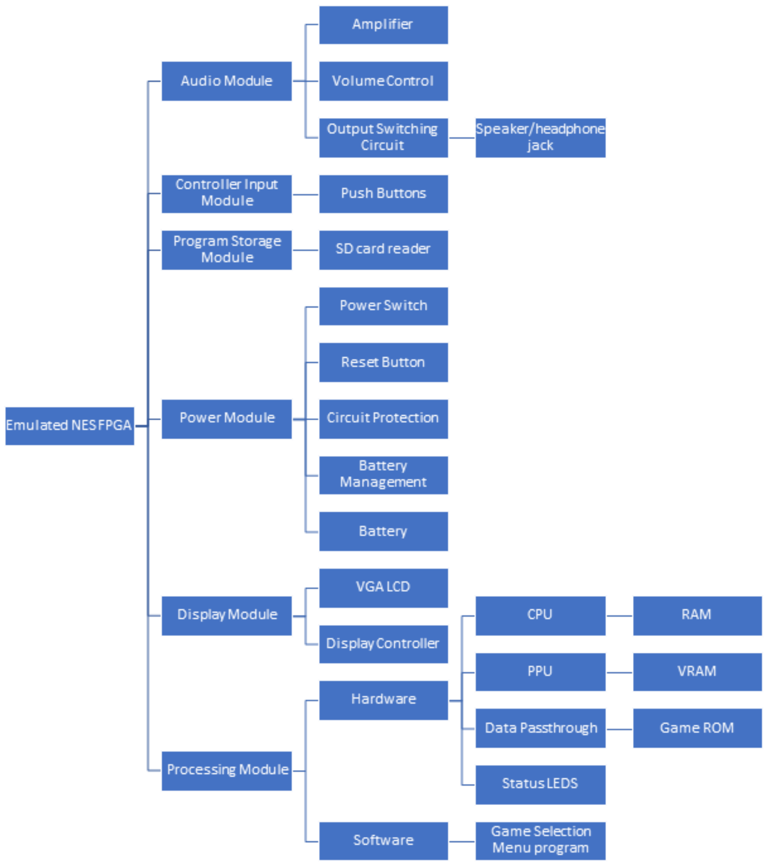


Functional Decomposition (Level-2): Control Display

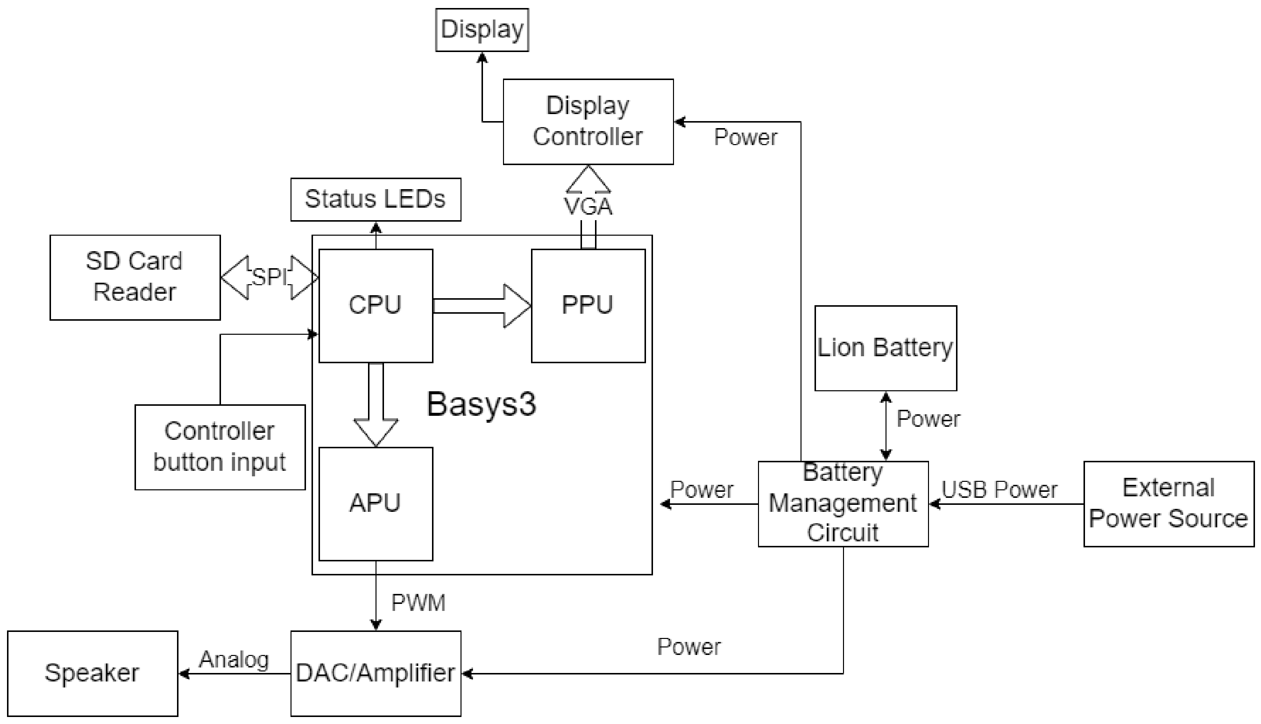


Functional Decomposition (Level-2): Control Audio

Physical Architecture



System Architecture



Preliminary Experimental Plan

Problems may arise when testing the prototype. To validate the requirements of our project, several experiments can be conducted:

- Audio and video synchronization
- Button response time
- Battery lifetime on full charge

For the audio and video synchronization experiment, a general test at the human sense level can be performed. A high frame rate video (at least 60 fps) and audio recording of the VGA display may be used in video editing software to confirm that audio, such as sound effects from a game, are synchronized with the actions within the game.

A button response time experiment can be done in the same manner of the experiment above, with a high framerate video recording (at least 60 fps) of the VGA display and a button being pressed. An estimated response time may be measured based on the amount of video frames it takes for the VGA display to change.

Lastly, a battery life experiment will help ensure we reach our calculated 3.3 hours of battery life for the device. The battery will initially be fully charged, then the experiment can be divided into sub-experiments:

- Idle battery lifetime
- Playtime battery lifetime

All the above experiments will need to be reconducted on the finalized device in ECE 493.

Preliminary Project Plan

In ECE 493, we will begin to develop our finalized device. Our project will consist of five main tasks:

- Circuit design
- PCB design
- Enclosure design
- Game selection menu program
- RTL code for memory

Brooks will be responsible for circuit design, with the help of Anthony. Their objective will be to design the circuitry for the breakout board. With the help of computer SPICE software, simulations can be conducted.

Amilcar will be responsible for designing the PCB layout, with the help of Brooks. They will need to use the circuit designed by Brooks and Anthony to design a PCB for the breakout board. They will also be responsible for soldering required electronic components to the PCB.

Once the breakout board dimensions and externals are finalized, Anthony will have the responsibility of designing the 3D-printed enclosure with the help of Amilcar. They will then construct the device will all necessary large components such as the LCD screen, battery, and speaker.

While the tasks above are sequential, Josh and Sam will be helping one another on the software/RTL responsibilities. Josh's main responsibility will be to create code for RTL functionality for manipulating memory locations in the memory map on the NES architecture. Sam, in tandem, will be responsible for programming and designing a game selection menu on boot-up to work with the RTL design.

Potential Problems

During or before ECE 493, we will need to learn skills for our corresponding responsibility. Most notably, CAD software for both 3D modeling the enclosure and for designing the PCB. Programming-wise, our knowledge of VHDL and assembly code may help facilitate learning Verilog and 6502 assembly with respect. Those programming skills will be required to implement the SD card to read and load game ROMs. Without that implementation, we will have to fall back to NesDbg to load the game ROMs.

There are known missing functionalities in Brian Bennett’s NES architecture. One of the missing functionalities is support for mappers. Some NES games use mappers to expand the normal limitations of their circuitry and memory in the ROM. Games that utilize mappers may not work as intended when play tested on the recreated NES architecture. Another missing functionality is the Delta Modulation Channel (DMC). The DMC is utilized in some NES games to be able to play a “sampled” sound, such as a vocal phrase, or a real-world acoustic instrument. Playable games in the recreated NES architecture will simply not be able to output the audio from the DMC.

From our testing of Brian Bennett’s NES architecture, we noticed that the VGA display contains a yellow tint filter. We also noticed that some character sprites, such as Mario in Super Mario Bros., contain slight visual errors. We hypothesize that the visual errors may be caused by the type of display we used during play testing.

References

- Literature references
  - https://github.com/brianbennett/fpga_nes
  - https://wiki.nesdev.org/w/index.php?title=Nesdev_Wiki
  - https://www.retrorgb.com/how-a-nes-controller-works.html
  - https://taywee.github.io/NerdyNights/nerdynights/nesarchitecture.html
  - https://www.masswerk.at/6502/6502_instruction_set.html
- People contacted
  - Professor Jens-Peter Kaps

10. Appendix B: Design Document (ECE-492)

ECE-492

DESIGN DOCUMENT

NES Handheld Emulation Breakout Board for the BASYS-3

Abstract

There are several methods to emulate the NES, including software or hardware emulation, with the latter being a more authentic emulation. We will emulate the NES on a BASYS-3 FPGA and we will minimize overall cost of components needed to create a portable/handheld device. Our FPGA NES emulation device will be comparable to existing devices in terms of price and capabilities. We will be using existing functionalities of power systems and of Pmod accessories for the BASYS-3 to create a singular breakout PCB to interface with the BASYS-3.

Team Members: Joshua Thomas Riggs, Brooks Aaron Corbett, Anthony Thinh Tang, Samuel Kifle Kebadu, Amilcar Alejandro Paniagua

Date of Submission: 5/9/2022

Faculty Supervisor: Jens-Peter Kaps

TABLE OF CONTENTS

1. Problem Statement

Within the past decade, older/retro games are becoming much more difficult to acquire. As such, many players in the gaming community turn to emulation to preserve their beloved games. Video game console emulation can vary in range of prices, with each method containing their advantages and disadvantages. We are motivated to seek a more cost-effective method for hardware emulating the NES in a portable package, meant to be hand-operated with a self-contained power source. We are also motivated to bring a new option for NES emulation with the BASYS-3.

As of Q1 2022, original NES home consoles can only be obtained secondhand. The most convenient method to play an NES is emulation by software. However, software emulation runs on personal computers with general architectures, which results in large overhead from trying to interpret hardware instructions meant for one architecture as software instructions. The more authentic solution is to recreate the NES via hardware. Some advantages of genuine replication of the original architecture include zero performance overhead, as well as accurate execution of programs, including any bugs or exceptional cases in either program or hardware. The best way of recreating hardware designs quickly and flexibly is through a Field-Programmable Gate Array (FPGA), a chip of programmable hardware logic. In fact, this is not the first redesign of the NES using an FPGA [3].

There are a few products in the current market that allow the NES to be played. Nintendo officially sells the NES Classic: a smaller-sized version of the home console. This runs NES games through software emulation on an ARM processor, meaning there are inaccuracies with the emulation, and only works with games purchased in its separate online store. More market options include the Pocket and NT Mini from the company Analogue. Both are High-end hardware “emulators” built on FPGA’s for not only the NES but other consoles as well. While both offer full recreation, and the Pocket is also portable, these systems are priced over $300 and lock their FPGAs and hardware to prevent tampering. For more flexible hardware implementations, there are DIY options. Some DIY projects, however, are complex in such a way that it may require many external parts and multiple levels embedded hardware and circuit design knowledge. One example is the MiSTer FPGA. Our NES FPGA project will allow flexibility within the build; our solution aims to be portable, handheld, cost-effective, hardware unlocked, DIY-friendly, and modular.

2. Project requirements specification

In our preliminary brainstorming we came up with a few requirements that our design requires. Mission Requirements are abstract and design independent and focus on how our design relates with external uses. Operation requirements specify what the input and outputs are, what the external interfaces are, any specific functional requirements and any technology requirements. These requirements were chosen with input from our stakeholder Dr. Kaps.

2.1 Mission Requirements

- The device shall emulate the original NES by creating a breakout board for the BASYS-3 board.
- The device will provide a new platform for emulating NES.

2.2 Operational Requirements

Input/Output Requirements

- The device shall output audio to either speaker or headphones.
- The device shall accept user input through push buttons.
- The device shall display video output to an LCD.
- The device shall have indicator LEDs

External Interface Requirements

- The device shall read/load games via SD card/Memory chip.
- The SD card/memory chip should be programmable via micro-USB.

Functional Requirements

- The device will start on a main menu.

- The device shall allow the user to be able to select a game on the menu.
- The device shall allow the user to adjust audio volume level.
- The device will run games at the same clock speed as the original NES @1.79MHz.
- The device shall be able to continuously play for 3.3 hours on a full charge.

Technology and System-Wide Requirements

- The device should cost less than $150
- The device shall use the BASYS-3 board.
- The device should use VGA as its display interface
- The device should be portable.

3. System Design

This section will demonstrate our FPGA NES device as three levels from a top-down modular perspective. Level-0 is the highest representation, with each level diving deeper into the modules of the device. All levels show all respective inputs and outputs with named functional "black box" modules in between. This section will also include the physical and system architecture representations of our device. The physical architecture diagram shows components and functions in a hierarchical manner while the system architecture diagram shows specific modules and their data relevance with each other.

As seen in Fig. 1, our FPGA NES system is composed of three inputs and two outputs. A user will be able to input controls via buttons and will be able to power on the system with a power switch. Program data will be sourced from an external SD card storage. The system will give status information through LEDs and will render both video and audio to be displayed and played through a speaker/headphones output. Fig. 2 shows all six of the modules that are contained within our system. The user input module is shown in Fig. 3. The user can provide input via push buttons which are read by an 8-bit shift register and fed to the system as input data. As shown in Fig. 4, the user will be able to choose a game through a program selection method. The game data at the chosen address location within the storage will then be loaded. Since our system will be portable, a battery management module, shown in Fig. 5, will be used. This module consists of a charging circuit, a protection circuit, and a boosting circuit. At the center of the system, in Fig. 6, the game ROM data will be decoded and will render the graphics and audio. The user will then be able to interact with the game by using the input module. As seen in Fig. 7, game video data will be loaded to a display that has its own power control system. Finally, Fig. 8 shows that the game audio data will be converted to analog data before becoming an output to a speaker/headphone. The volume of the audio will be adjustable via a potentiometer. Fig. 9 shows the physical architecture, with the higher hierarchical concept to the left. Fig. 10 shows the system architecture, where modules can be seen externally and internally from the BASYS3.

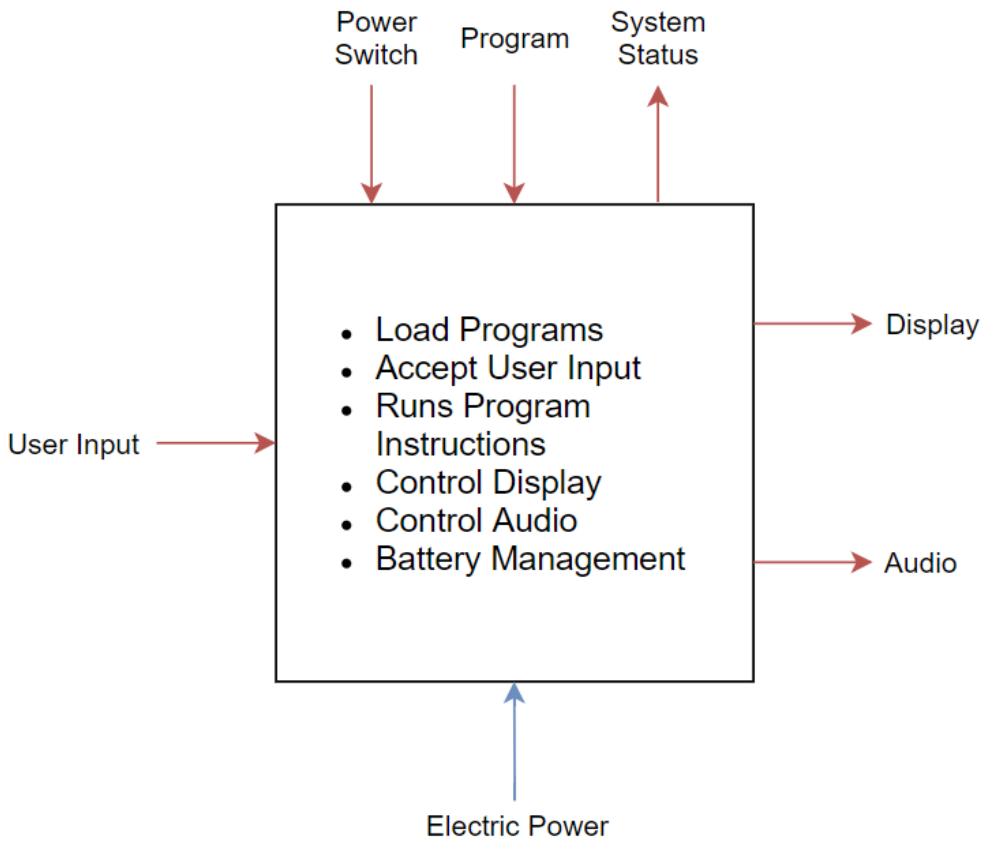


Fig. 1 Functional Decomposition (Level-0)

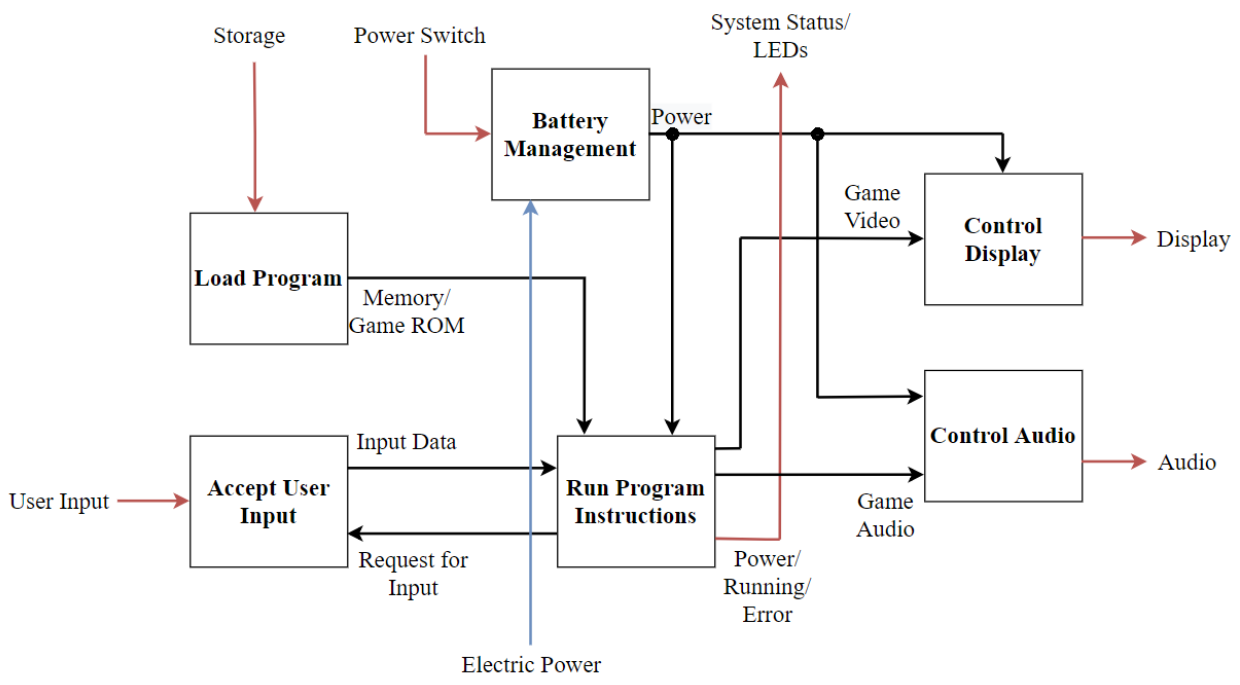


Fig. 2 Functional Decomposition (Level-1)

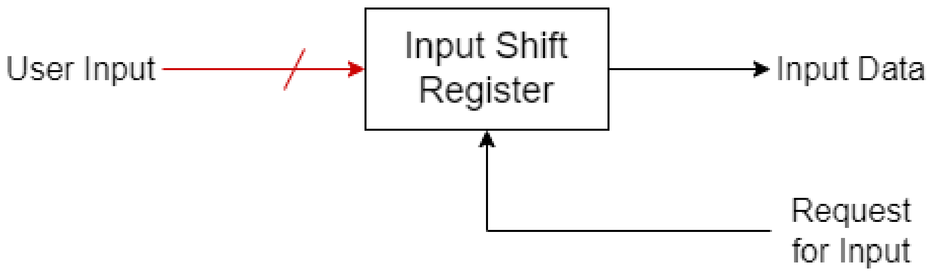


Fig. 3 Functional Decomposition (Level-2) Accept User Input

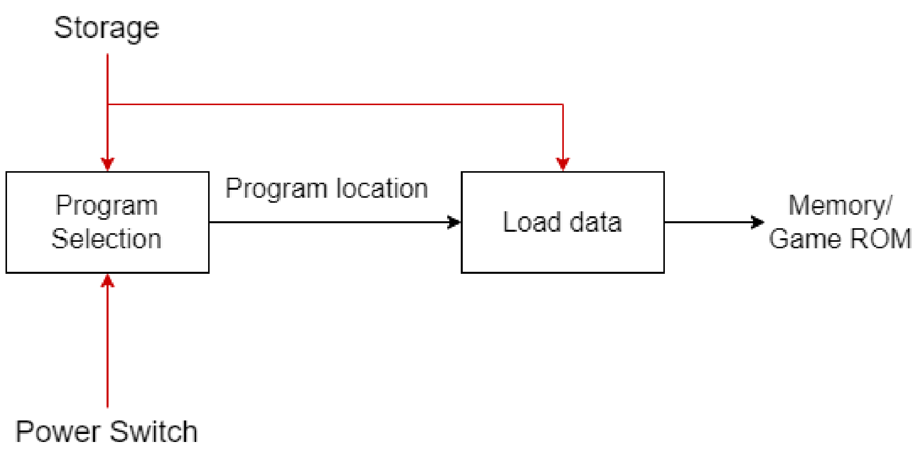


Fig. 4 Functional Decomposition (Level-2) Load Program

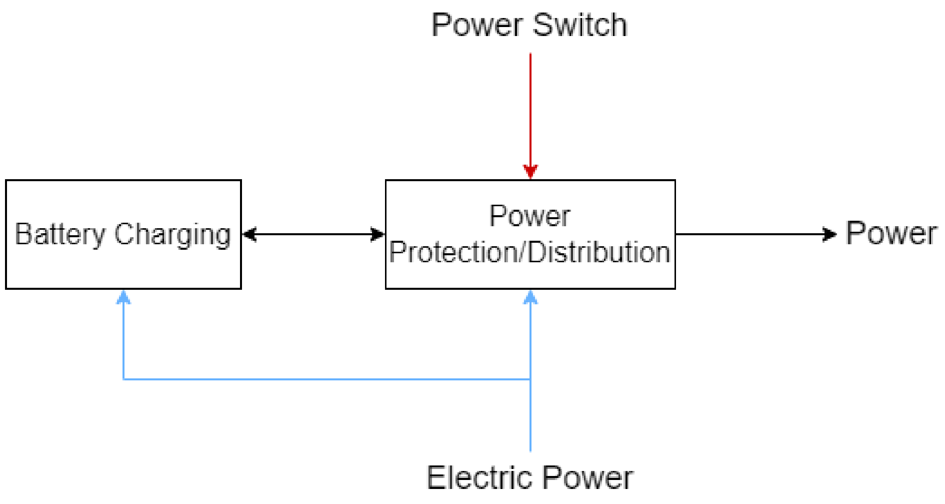


Fig. 5 Functional Decomposition (Level-2) Battery Management

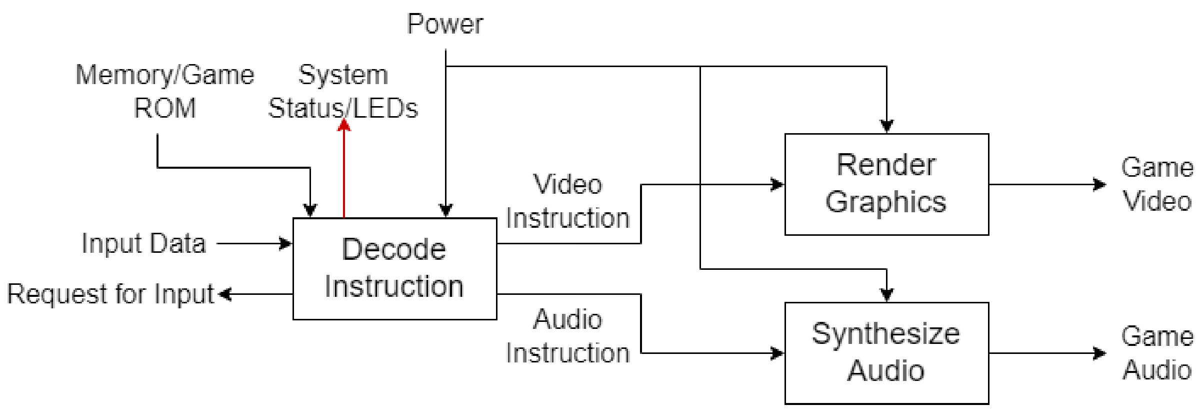


Fig. 6 Functional Decomposition (Level-2) Run Program Instructions

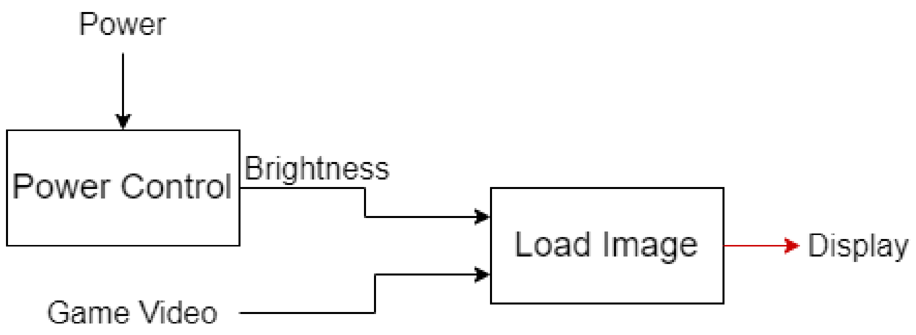


Fig. 7 Functional Decomposition (Level-2) Control Display

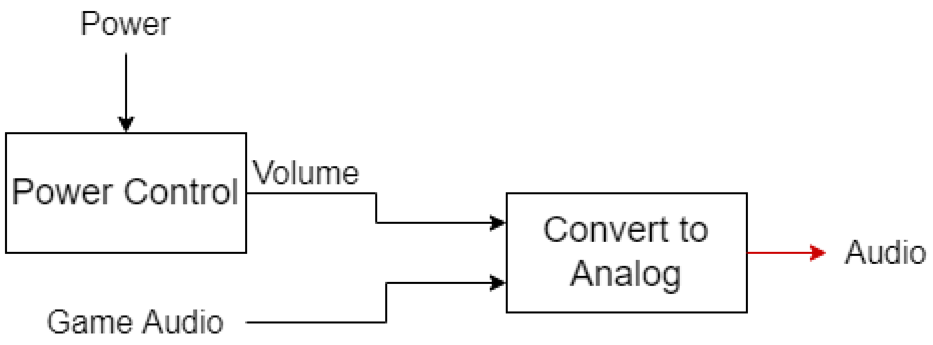


Fig. 8 Functional Decomposition (Level-2) Control Audio

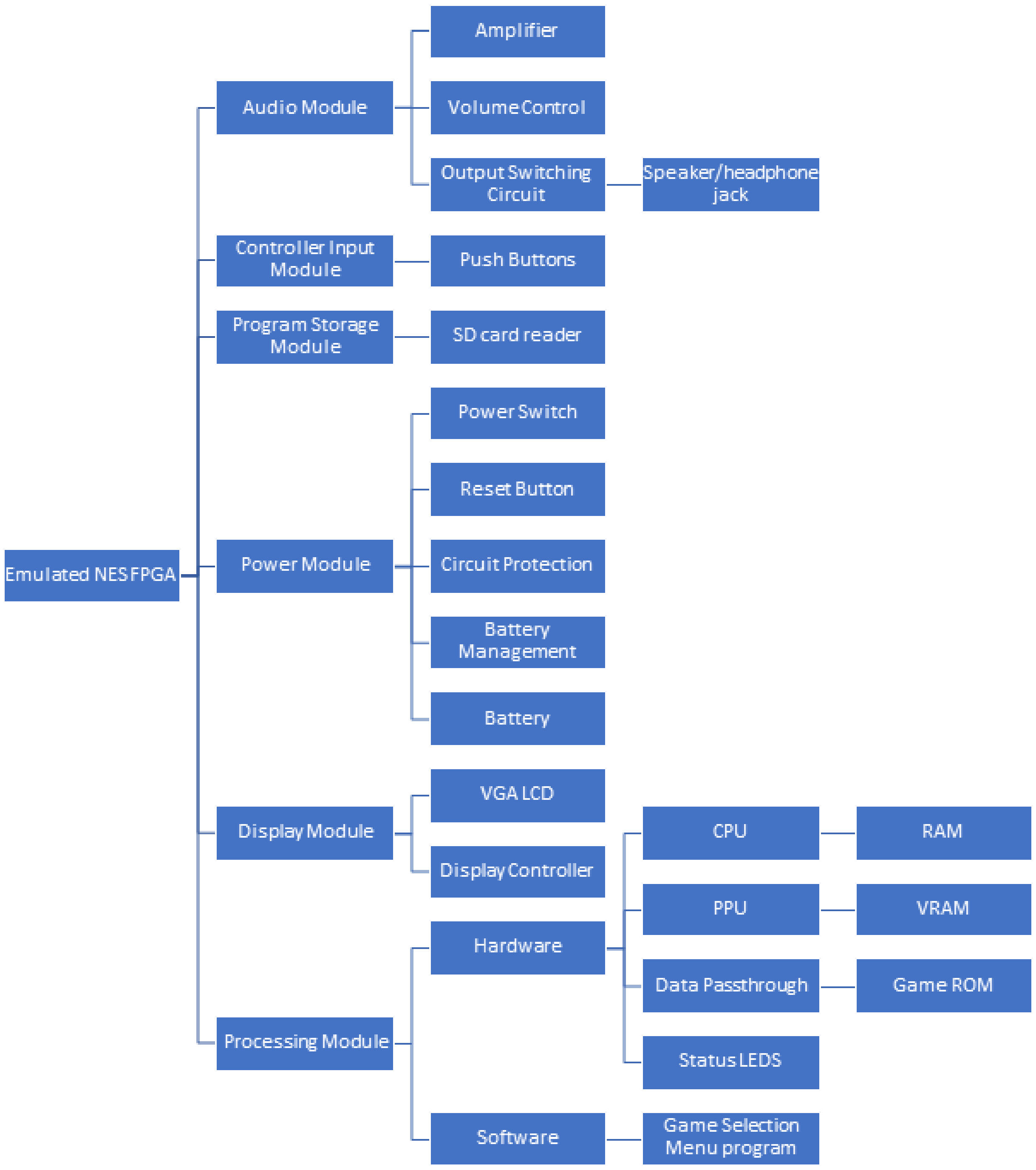


Fig. 9 Physical Architecture

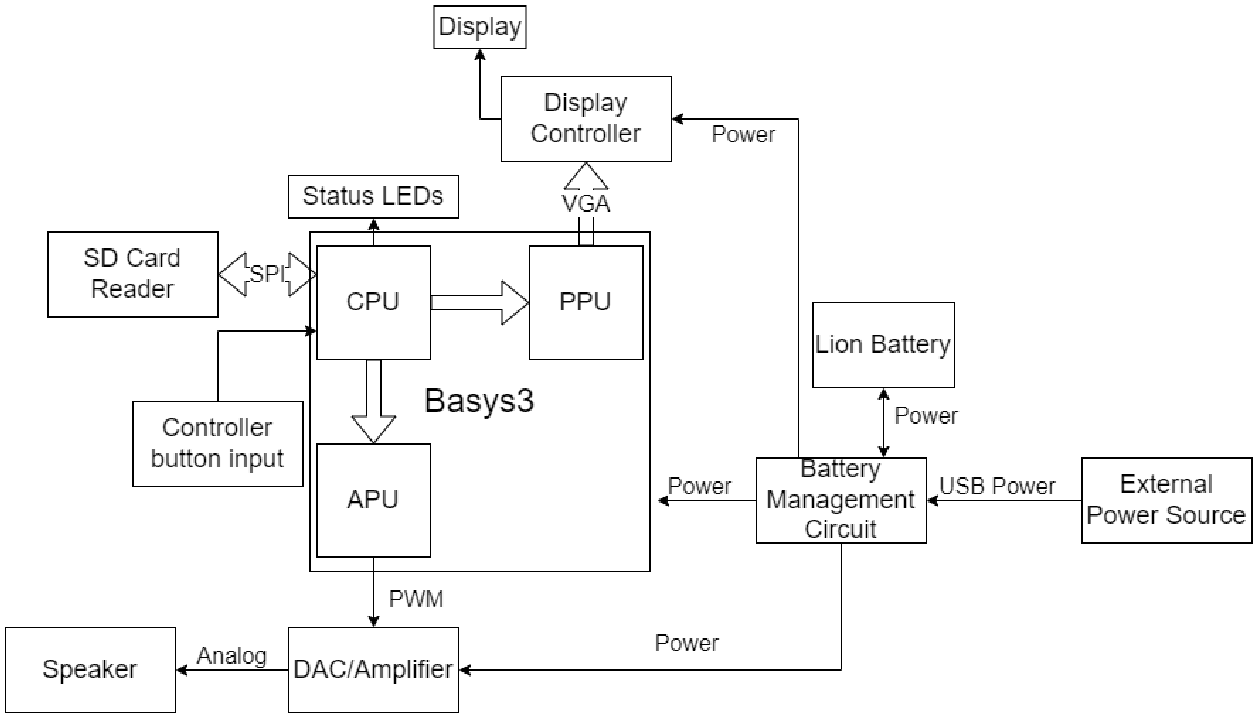


Fig. 10 System Architecture

4. Background Knowledge

4.1 VGA Protocol

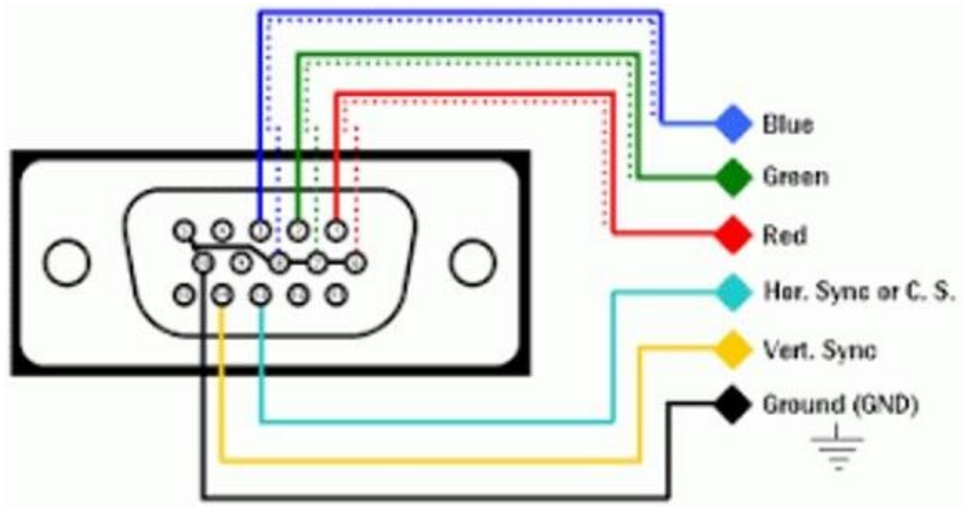


Fig. 11 VGA pin signals

The BASYS-3 has one female VGA connection and can be programmed to output VGA for display. The VGA protocol consists of 5 signals: RED, GREEN, BLUE, HSYNC, and VSYNC. The RED, GREEN, and BLUE color signals send color data to the pixel determined by a pixel clock signal. The HSYNC and VSYNC are used to synchronize the start and stop of the pixel lines of every frame.

4.2 SPI Protocol

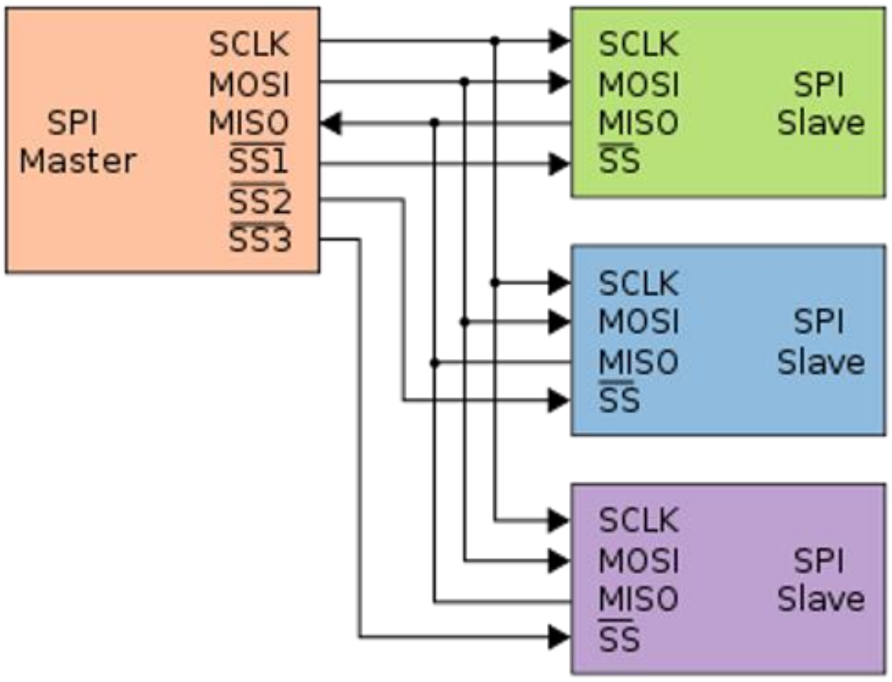


Fig. 12 SPI Master and Slave connections

This project will require the use of the SPI which stands for Serial Peripheral Interface. SPI is a synchronous, Full-Duplex communication protocol that has four pins and they consist of Serial Clock (SCLK), Master Out Slave In (MOSI), Master In Slave Out (MISO) and Slave Select or Chip Select (SS). The function of SCLK is self-explanatory, MOSI is used for sending data, MISO is used for receiving data and Chip Select is used to select the destination. SPI will be used to interface with the SD card to store NES video games.

4.3 6502 CPU arch

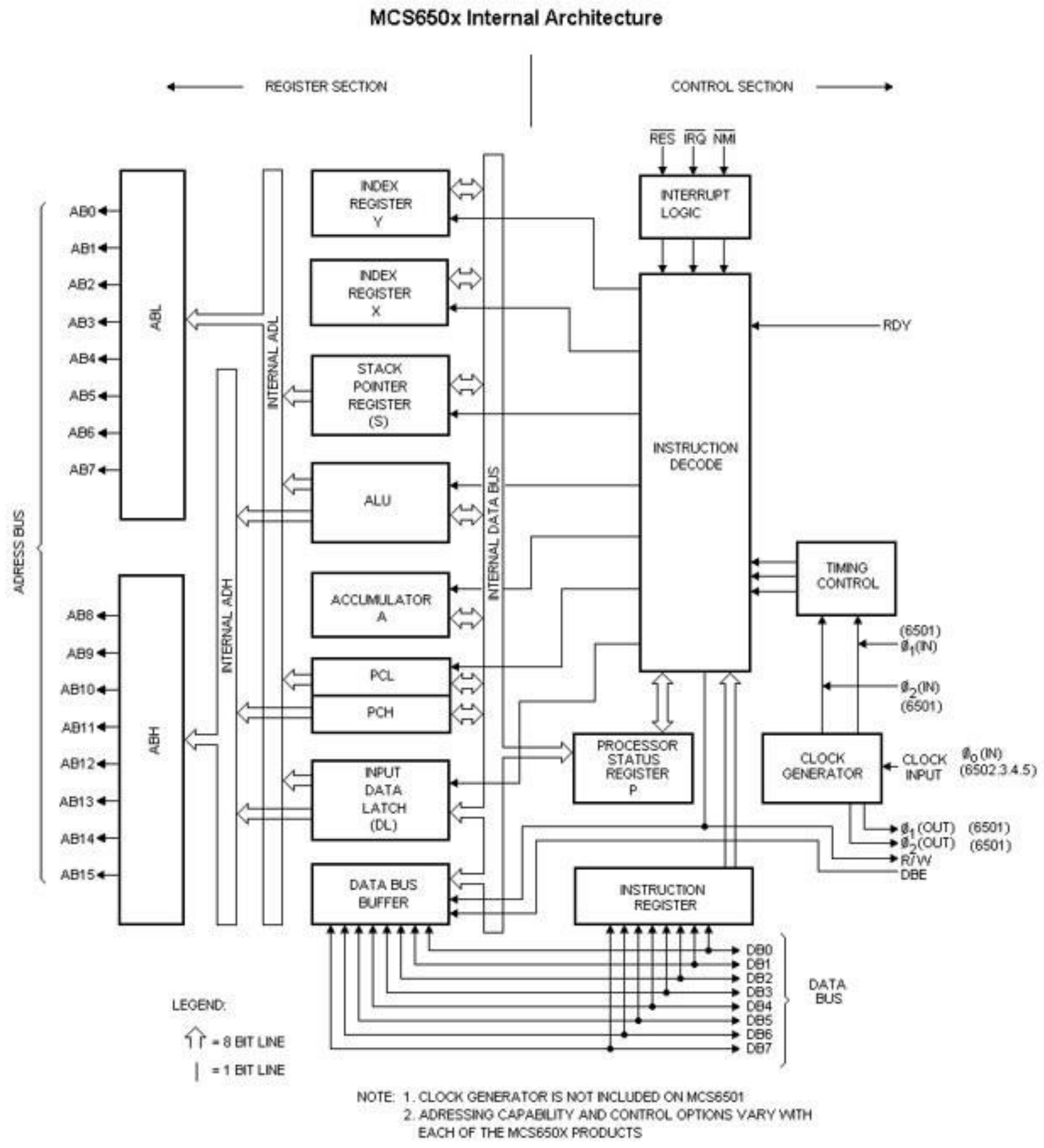


Fig. 13 Processor Architecture of MCS650x

The NES is centered around a modified version of the popular 6502 Central Processing Unit (CPU) designed by MOS Technology in 1975. Our design in turn will be built off a hardware recreation of this architecture. The 6502 is a little-endian 8-bit processor with a 16-bit external address bus [2]. Within the main architecture, there are 3 major bus interfaces: the instruction fetch bus, the data read bus, and the data write bus. These connect the CPU shell that handles external system calls to the CPU core. This allows for both calling up to 65 KB of memory, and for direct external bus interfaces to other processing units, such as the PPU and APU found on the NES [1][5]. Later on, MOS Technology would create a second version of the 6502 in 1981 that is CMOS-based: the 65c02.

4.4 Power/battery Management

There are 4 main components in our battery management system: a charging circuit, a circuit protection circuit, a voltage booster, and the battery itself. The BASYS-3 needs 5-volts to be able to run so a battery is needed to supply 5-volts. Instead of getting a battery that was above 5-volts and dropping the voltage down, a rechargeable 3.7-volt Li-Ion battery was chosen. A voltage booster was used to get the voltage up to 5-volts and produce a current output of up to 4 amps. A circuit protection aspect needs to be considered so that in the case of a spike in current or voltage, the circuit will not be damaged. Finally, the battery will need to be charged in order to keep using it repeatedly. A charging circuit with a charging IC allows this to happen with a 5-volt input.

4.5 Bennet Open-Source NES

We will be building on top of a previous design of the NES by Brian Bennet, written in Verilog. Bennet constructed his version of the NES using an open-source implementation of the 6502 processor. Bennet uses open-source software, called NesDBG, that is used for debugging the NES and for loading the game ROMs into the FPGA Block Random Access Memory (BRAM). Bennet implemented his design on a Nexys 3 Spartan-6 FPGA and used a breakout board, that is no longer in production, to connect the NES controller to the NES itself. Bennet also used a Pmod amplifier and a speaker for the audio[3].

5. Detailed Design

In order to better model our design and describe system behavior we decided to make a data flow diagram and a state diagram. The data flow diagram is used to better describe the data that is moving throughout our design. The diagram shows how the video, audio, status lights and controller input interact with the user. It also shows the individual components of the system such as the audio processing unit, picture processing unit, central processing unit and SD card reader. The state diagram shows what tasks the system must accomplish and what state it will be in to accomplish those tasks. The diagram shows how the system moves from the game selection screen through user input to loading a game from the SD card.

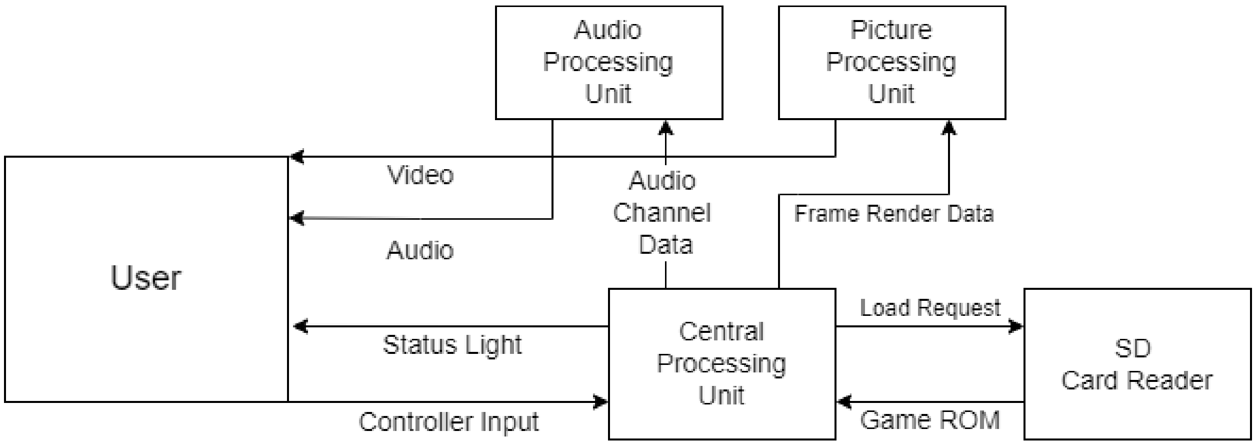


Fig. 14 Dataflow chart of FPGA NES device

In Fig. 14, the user will be able to both select a game and play the games with controller input. A status light will give the user information about the status of the battery and the loading process. The Central Processing Unit is responsible for loading the games and rendering both audio and video using the audio processing unit and the picture processing unit. The Central Processing Unit will also be responsible for making requests to the SD card reader to load game ROMs.

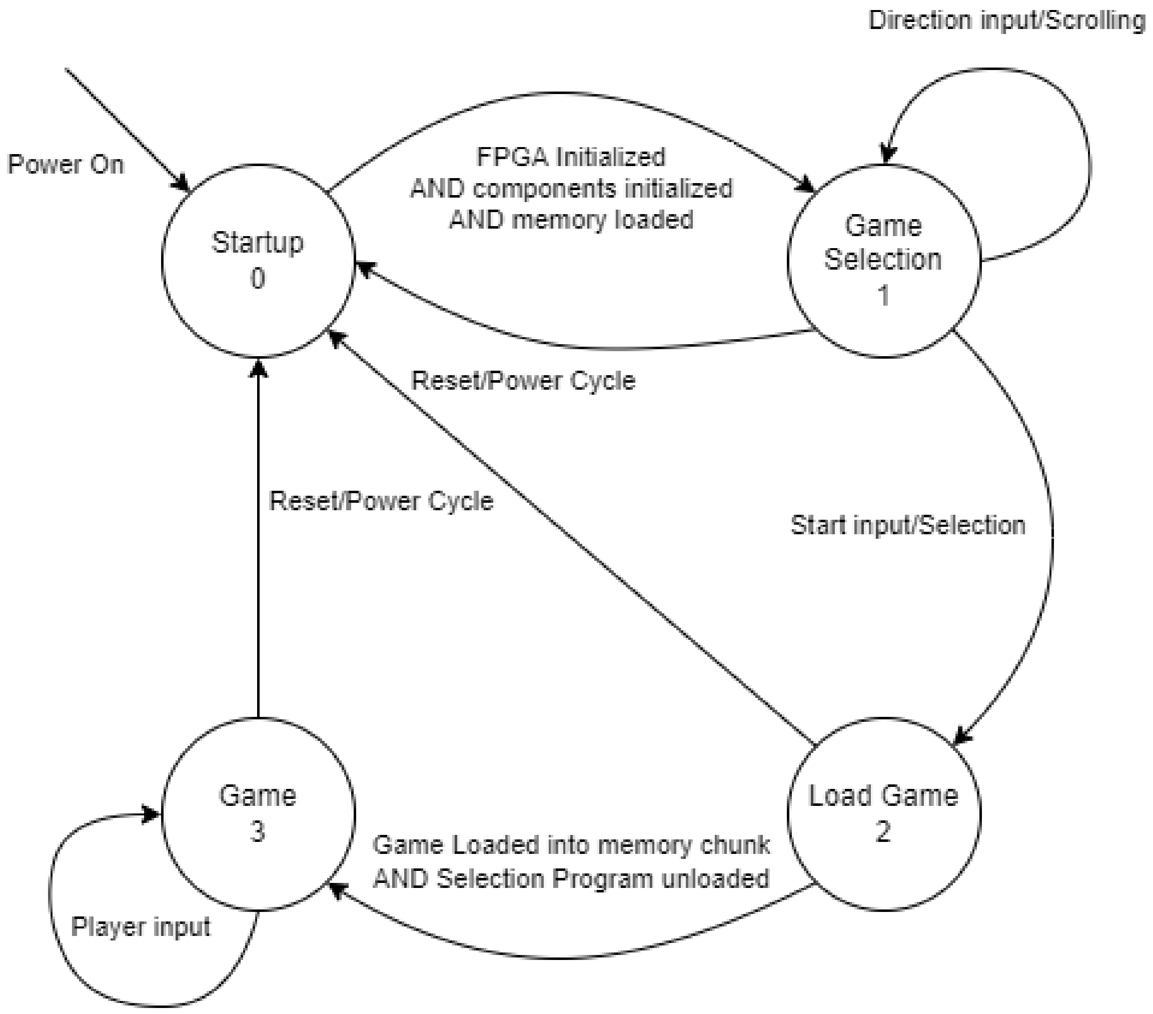


Fig. 15 State diagram of FPGA NES device

Fig 15. Shows the state machine of the FPGA NES device. On the Startup state, the FPGA is initialized directly to the Game Selection state where the user will input scroll direction and selection of a game in the menu. Once a game is selected, the Load Game state is entered where

the ROM data will be loaded. Finally, the selected game will begin, and the user can play in the Game state. All states can be reset back to the Startup state to choose a new game.

5.1 Software

The goal for the software of this project is to run an NES program on startup that reads a list of loaded games in the SD card and presents them in a menu for the player to choose from. Once the player selects the game they want to load, the program will then load the game into RAM, then jump to the game instruction location to start the game. This software will be written in the 6502’s assembly language and rely on accessing game emulation ROM files through a fat32 structure. The 6502 assembly language does not have built in compatibility with accessing file structures, but open-source libraries are available for fat32 file structure navigation [2]. These libraries come with their own set of issues with NES hardware that is discussed later in this paper. The external SD Card component being read from or written to is capable of communication using SPI.

5.2 Hardware Components

This section contains schematics that are provided by the respective retailers of the components purchased for our prototype. These schematics will aid in our future circuit design since each electrical component is detailed.

Fig. 16 is a Pmod amplifier that uses a single SSM2377 audio amplifier chip along with 330, 10k, and 0-ohm resistors, 4700pF, 10uF, and .01F capacitors for sufficient filtering. Fig. 17 is a Lithium Ion and Lithium Polymer battery charger based on the MCP73833. It uses a USB mini-B for connection to any computer or any USB power supply. Charging is performed in three stages: first, a preconditioning charge, then a constant-current fast charge, and finally a constant-voltage trickle charge to keep the battery topped up. The fast-charge current is 500 mA by default but is easily adjustable from 100 mA up to 1000 mA by soldering a through-hole resistor onboard. Fig. 18 is a DC-to-DC voltage booster with an input range of 3-35 volts and a variable output voltage of 5-40 volts. This booster's efficiency has a maximum of 94% with a switching frequency of 220kHz. The output voltage is determined by the choice of resistors R1 and R2. Fig. 19 is an SD card reader that interfaces directly with the Pmod port and does not have any other ICs that it interfaces with. In prototyping we have decided to use a Pmod microSD that has the option to add a current limiting IC, but we are not using it currently. Fig. 20 is the original NES controller that consists of an 8-bit register and some carbon printed resistors [4]. For our design we have used the same 8-bit register (the MC14021BCPD) that was taken out of an original NES controller. The value of the carbon resistors was 40k-ohm in the original controller, so we have replaced them with 40k-ohm ceramic resistors.

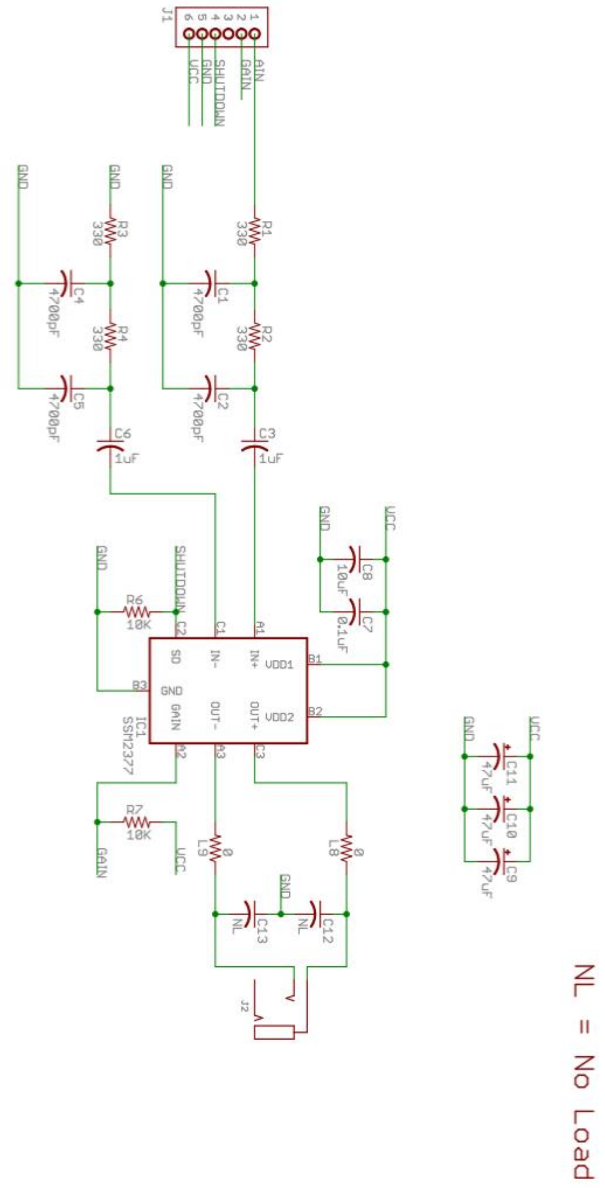


Fig. 16 Audio Amplifier

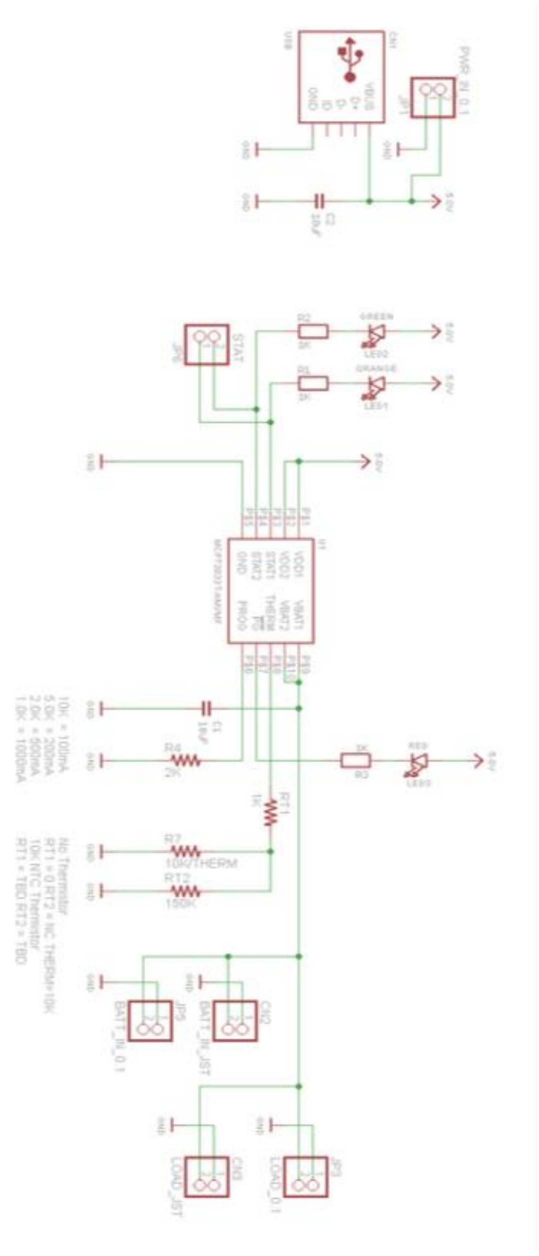


Fig. 17 USB LiIon/LiPoly Charger

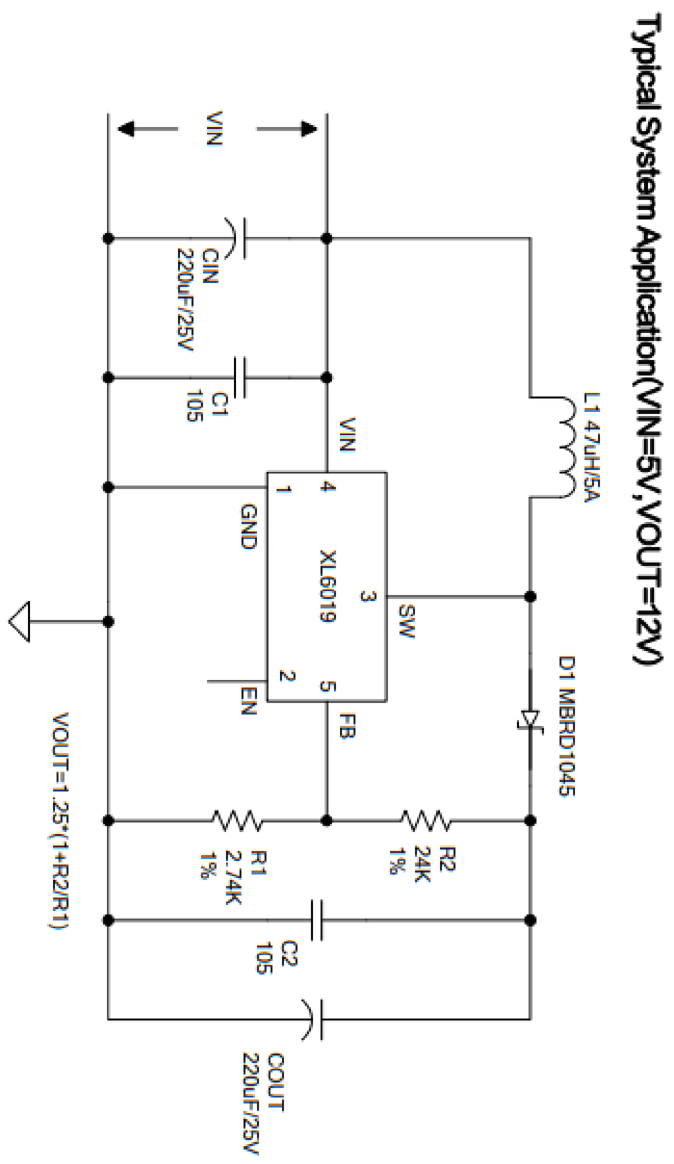


Fig. 18 DC-to-DC Booster

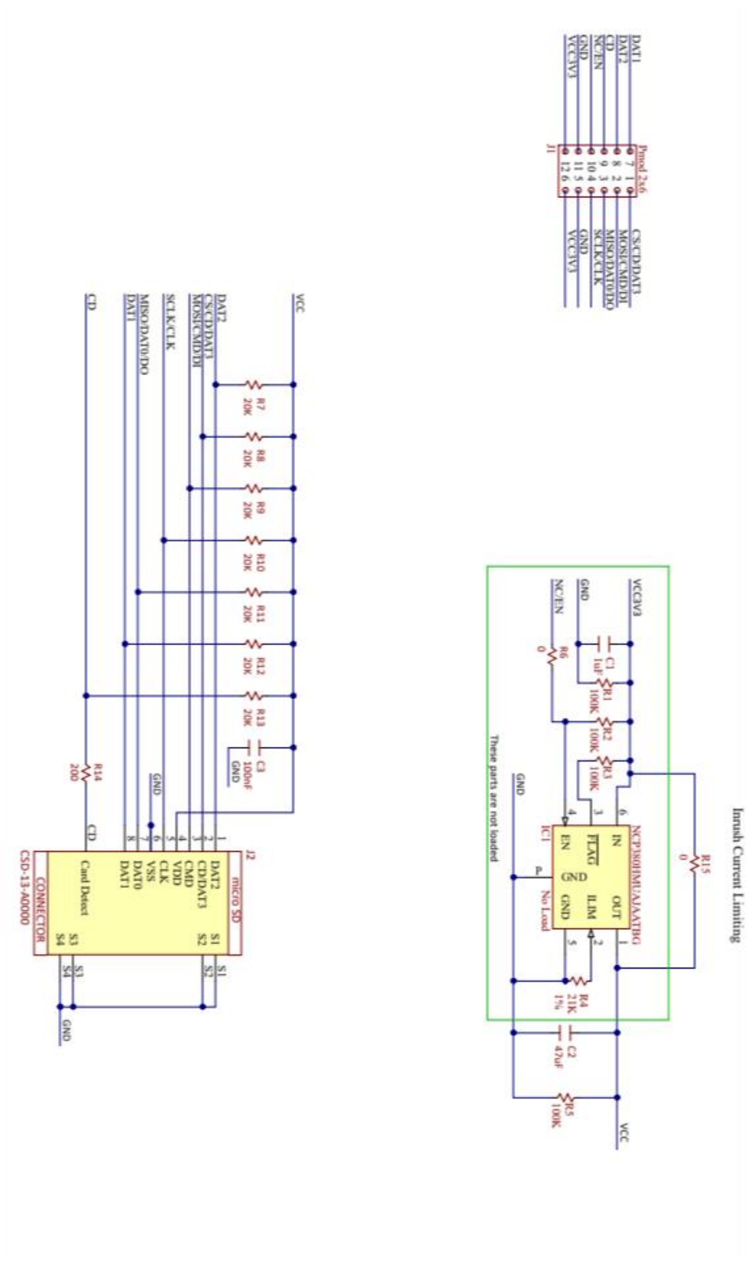


Fig. 19 SD Card

Fig. 20 Controller

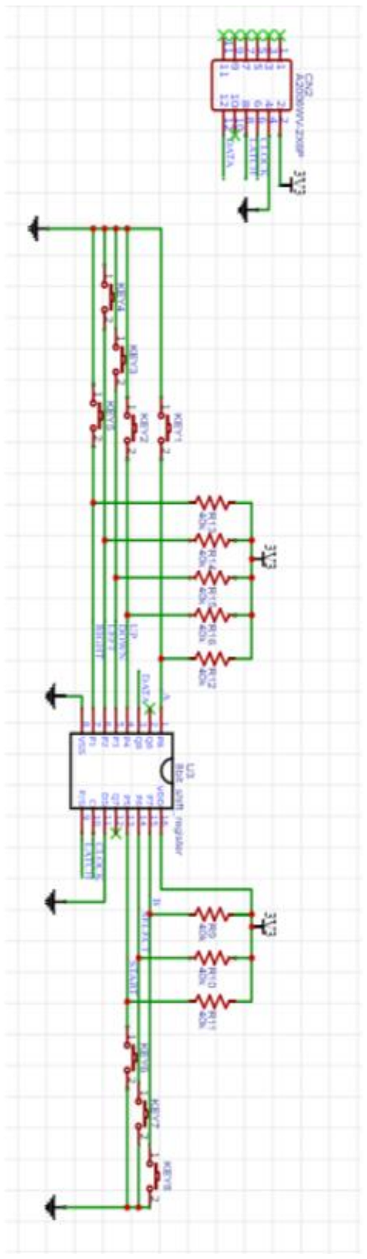


Fig. 20 Controller

6. Prototyping Progress Report

This section will list all the components we used and how they were used in our experimental setup and prototyping. A timeline of our experimental setup throughout the semester is also included in this section, alongside a reflection of what we have learned from our prototyping efforts. Next, this section will explain the roadblocks we've encountered along the way, our testing plan with experiments, and the task allocation for ECE 493.

6.1 List of components

- BASYS-3
- Pmod Audio Amp
- Pmod MicroSD
- 16GB SD Card
- 8Ω Speaker
- NES Controller
- Lithium Battery 3.7v 4000mAh
- VGA to HDMI
- Adafruit LiPoly Charger
- DC to DC Booster
- 480x320, 3.5-inch Touch Screen IPS TFT LCD

6.2 Current experimental setup

NES emulation is currently achieved by the BASYS-3 with the support for an NES controller, a VGA display, and audio, powered by a lithium battery. Games can be loaded by

NesDgb via micro-USB and audio can be listened to successfully with a speaker. Each main component was individually tested.

Controller

- Breadboard controller in progress
- Currently using a premade NES controller

Display

- We are planning on using a 480x320, 3.5-inch Touch Screen IPS TFT LCD

Audio

- Audio Amplifier Pmod is interfaced
- Speaker and volume controller included
- Audio is amplified by 12dB

SD Card

- The SD Card is connected via the Pmod pins on the Basys-3
- It will be used to store one game of our choosing
- The SD Card will interface via SPI and read and write raw bytes with no file structure.

Battery

- Currently using a 4000mAh battery.
- Theoretical calculations show that we will get 7 hours of battery life.

6.3 Prototyping History

Version 4.15.2022

- Power circuit and battery tested
- Mini VGA display tested

Version 4.8.2022

- Controller breadboard tested

Version 4.1.2022

- Soldered Pmod audio jack tested with 8 Ω speaker

Version 2.25.2022

- Projector display tested for the first time
- Conference room audio aux tried for the first time
- NES controller tested for the first time

6.4 What we learned

From our testing of Bennett's NES architecture, we noticed that on certain VGA monitors, the visual display contains a yellow tint filter. We also noticed that some character sprites, such as Mario in Super Mario Bros., contain slight visual errors. We discovered that the visual errors are caused by the type of display we used during play testing and their native resolutions. We hypothesize that monitors with larger resolutions are more prone to visual errors.

6.5 Current Roadblocks

We are currently attempting to implement the SD card to read and load game ROMs by modifying the existing Verilog. Without that implementation, we will have to fall back to NesDbg to load the game ROMs. Our current approach to implementing the ability to load a game from an SD card is to read and write to the SD card over an SPI interface as opposed to the official SD card interface. We decided on SPI because learning the SD card protocol may be time consuming and unnecessary for the scope of this project. To store a game in memory we will try to use the NesDbg software to load a game like it was originally supposed to but instead of storing the game in the cartridge ROM it will copy to the SD card. On startup, the Verilog code will detect a flag that signals loading from the SD card rather than the NesDbg software.

We are also currently attempting to implement the system on a breadboard. These parts include the shift register that the NES controllers use, the Adafruit battery management system, the DC-to-DC booster, the lithium battery and the speaker. Our prototyping goals for ECE 492 will include the implementation of all the components via breadboard and to be able to read at least one game from the SD card.

6.6 Predicted Roadblocks

During or before ECE 493, we will need to learn skills for our corresponding responsibility. Most notably, CAD software for both designing the printed circuit board (PCB) and for 3D modeling the enclosure. It's realized that designing the 3D enclosure will require a completed design of the PCB. Soldering-wise, we will need to practice using a heat gun to solder components only available as surface-mount technology (SMT) ball grid array (BGA). Another roadblock that we will run into is the fact that the fat32 file system library that we will use to implement our start menu requires the 6502 to have a certain amount of RAM. The problem is that the NES has 2 KB of ram, and the library requires 5 KB of ram. This means that we will have to increase the amount of RAM in the NES Verilog implementation or if that doesn't work then we will need to find another way to implement the start menu functionality. Another problem is that the fat32 library was developed for the 65c02 and not the 6502, which means that there could be other unforeseen problems or bugs with it.

6.7 Bonus Roadblocks

There are known missing functionalities in Bennett's NES architecture that will not affect our finalized project. These bonus roadblocks would be nice to finish but aren't our priority. One of the missing functionalities is support for mappers. Some NES games use mappers to expand the normal limitations of their circuitry and memory in the ROM. Games that utilize mappers may not work as intended when play tested on the recreated NES architecture. Another missing functionality is the Delta Modulation Channel (DMC). The DMC is utilized in some NES games to be able to play a "sampled" sound, such as a vocal phrase, or a real-world acoustic instrument. Playable games in the recreated NES architecture will simply not be able to output the audio from the DMC.

6.8 Testing Plan

Problems may arise when testing the prototype other than debugging the designed PCB. To validate the requirements of our project, several experiments can be conducted:

- Audio and video synchronization

- Button response time
- Battery lifetime on full charge

For the audio and video synchronization experiment, a general test at the human sense level can be performed. A high frame rate video (at least 60 fps) and audio recording of the VGA display may be used in video editing software to confirm that audio, such as sound effects from a game, are synchronized with the actions within the game. Success will be achieved when any video/audio sync issues are not noticeable by whoever is playing the game.

A button response time experiment can be done in the same manner as the experiment above, with a high framerate video recording (at least 60 fps) of the VGA display and a button being pressed. An estimated response time may be measured based on the amount of video frames it takes for the VGA display to change. Success would be achieving a latency of 10ms or less, ideally a latency that is difficult to notice during play.

Lastly, a battery life experiment will help ensure we reach our calculated 3.3 hours of battery life for the device. The battery will initially be fully charged, then the experiment can be divided into sub-experiments:

- Idle battery lifetime
- Playtime battery lifetime

The success for this experiment will be achieved when we feel the game console has a sufficient life span, both idle and with constant play. At the moment, we are determining success for the idle time to be at least the calculated 3.3 hours.

Alongside previous tests, testing of the game select screen is necessary. This will be conducted once the RTL code and game selection UI is completed. The test will check if we are able to properly read an SD card for game data and select it on the LCD display, and success will be determined if the game is correctly loaded after selection.

All the above experiments will need to be reconducted on the finalized device in ECE 493.

6.9 List and Description of Tasks

In ECE 493, we will begin to develop our finalized device. Our project will consist of five main tasks:

- Circuit design
- PCB design
- Enclosure design
- Game selection menu program
- Register transfer level (RTL) code for memory

Brooks will be responsible for circuit design, with the help of Anthony. Their objective will be to design the circuitry for the breakout board. With the use of computer SPICE software, simulations can be conducted.

Amilcar will be responsible for designing the PCB layout, with the help of Brooks. They will need to use the circuit designed by Brooks and Anthony to design a PCB for the breakout board. They will also be responsible for soldering required electronic components to the PCB.

Once the breakout board dimensions and externals are finalized, Anthony will have the responsibility of designing the 3D-printed enclosure with the help of Amilcar. They will then construct the enclosure will all necessary large components such as the LCD screen, battery, and speaker implemented.

While the tasks above are sequential, Josh and Sam will be helping one another on the software/RTL code responsibilities. Josh’s main responsibility will be to create code for RTL functionality for manipulating memory locations in the memory map on the NES architecture. Sam, in tandem, will be responsible for programming and designing a game selection menu on boot-up to work with the RTL design.

7. Schedule and Milestones

7.1 Next Semester Plans

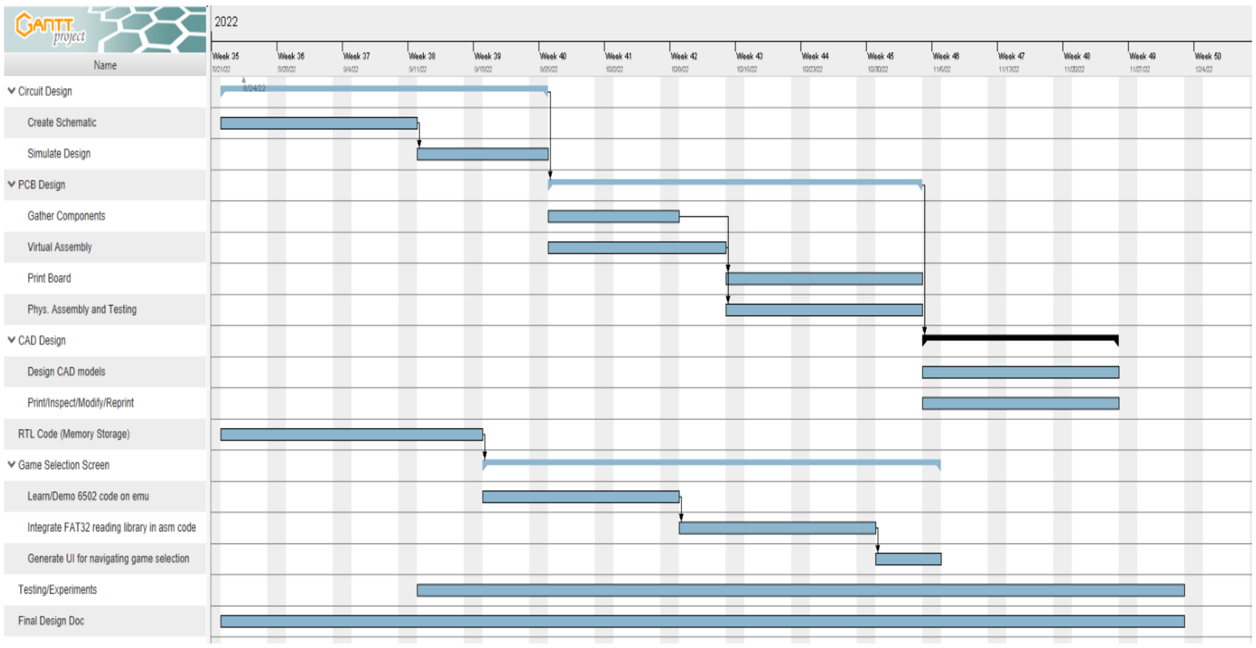
Our plans for next semester are to move on from the prototyping stage and design an enclosure that will house the BASYS-3, battery, display, speaker, and a PCB. We will design a breakout PCB that will integrate the boost converter, battery management system, SD card reader, controller buttons, audio control, and circuit protection. We will also expand the storage functionality of our project by enabling the 6502 CPU to read from the SD card using a file system. To do this we will need to write a program in assembly that will act as our "start menu" that can choose games from the filesystem on the SD card.

7.2 Milestones

- Circuit Design
  - Schematic creation
  - Schematic design simulation
- PCB Design
  - PCB Virtual Assembly
  - Printing PCB
  - PCB assembly and testing
- CAD Design
  - Designing CAD model
  - Printing CAD design and testing
- Completion of RTL Code

- Game Selection Screen
    - Integration of 6502 code into emulator
    - Design and generate UI for game selection

7.3 Gantt Chart



8. References

- Literature references

[1] “2. NES Architecture Overview — Nerdy Nights 0.0.1 documentation.” https://taywee.github.io/NerdyNights/nerdynights/nesarchitecture.html (accessed May 05, 2022).

[2] “6502 Instruction Set.” https://www.masswerk.at/6502/6502_instruction_set.html (accessed May 05, 2022).

[3] B. Bennett, *brianbennett/fpga_nes*. 2022. Accessed: May 05, 2022. [Online]. Available: https://github.com/brianbennett/fpga_nes

[4] “How A NES Controller Works | RetroRGB.” https://www.retrorgb.com/how-a-nes-controller-works.html (accessed May 05, 2022).

[5] “NESdev Wiki.” https://www.nesdev.org/wiki/Nesdev_Wiki (accessed May 05, 2022).

11. Appendix C: Software printout – if required by FS

11.1 ECE493TESTER.c

```
// Written by nesdoug
// Modified by Amilcar Paniagua

#include "LIB/neslib.h"
#include "LIB/nesdoug.h"

#define BLACK 0x0f
#define DK_GY 0x00
#define LT_GY 0x10
#define WHITE 0x30
// there's some oddities in the palette code, black must be 0x0f, white must
be 0x30

#pragma bss-name(push, "ZEROPAGE")

// GLOBAL VARIABLES
// all variables should be global for speed
// zeropage global is even faster

unsigned char i;
unsigned char pad1;

const unsigned char text[]="ECE 493 NES FPGA INPUT TESTER
A: SFX"; // zero terminated c string

const unsigned char palette[]={
BLACK, DK_GY, LT_GY, WHITE,
0,0,0,0,
0,0,0,0,
0,0,0,0
};

// example of non-sequential vram data
const unsigned char space_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
' ',
NT_UPD_EOF}; // data must end in EOF

const unsigned char space1_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
' ',
NT_UPD_EOF}; // data must end in EOF
```

```
const unsigned char u_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'U',
NT_UPD_EOF}; // data must end in EOF

const unsigned char l_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'L',
NT_UPD_EOF}; // data must end in EOF

const unsigned char r_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'R',
NT_UPD_EOF}; // data must end in EOF

const unsigned char d_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'D',
NT_UPD_EOF}; // data must end in EOF

const unsigned char a_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'A',
NT_UPD_EOF}; // data must end in EOF

const unsigned char b_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'B',
NT_UPD_EOF}; // data must end in EOF

const unsigned char st_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'+',
NT_UPD_EOF}; // data must end in EOF

const unsigned char sl_letter[]={
MSB(NTADR_A(16,17)),
LSB(NTADR_A(16,17)),
'-',
NT_UPD_EOF}; // data must end in EOF

enum {SFX_JUMP, SFX_DING, SFX_NOISE};
```

```
void main (void) {

      ppu_off(); // screen off

      pal_bg(palette); //     load the BG palette

      // set a starting point on the screen
      // vram_adr(NTADR_A(x,y));
      vram_adr(NTADR_A(1,5)); // screen is 32 x 30 tiles

      i = 0;
      while(text[i]){
            vram_put(text[i]); // this pushes 1 char to the screen
            ++i;
      }

      // vram_adr and vram_put only work with screen off
      // NOTE, you could replace everything between i = 0; and here with...
      // vram_write(text,sizeof(text));
      // does the same thing

      ppu_on_all(); //  turn on screen

      ppu_wait_nmi(); // waits until the next nmi is completed, also sets a
VRAM update flag
                              // the text will be auto pushed to the PPU
during nmi


      while (1){
            // infinite loop
            ppu_wait_nmi(); // wait till beginning of the frame
            pad1 = pad_poll(0); // read the first controller

            if(pad1 & PAD_UP){
                  set_vram_update(u_letter); // set a pointer
            }
            else if(pad1 & PAD_LEFT){
                  set_vram_update(l_letter); // set a pointer
            }
            else if (pad1 & PAD_RIGHT){
                  set_vram_update(r_letter); // set a pointer
            }
            else if (pad1 & PAD_DOWN){
                  set_vram_update(d_letter); // set a pointer
            }
            else if (pad1 & PAD_A){
                  set_vram_update(a_letter); // set a pointer
                  sfx_play(SFX_DING, 0);
            }
```

```
		else if (pad1 & PAD_B){
			set_vram_update(b_letter); // set a pointer
		}
		else if (pad1 & PAD_START){
			set_vram_update(st_letter); // set a pointer
		}
		else if (pad1 & PAD_SELECT){
			set_vram_update(sl_letter); // set a pointer
		}
		else {
			set_vram_update(space_letter); //set a pointer
		}
		ppu_wait_nmi();
	}
}
```