Faculdade de Engenharia da Universidade do Porto

not NINTENDO

NES Emulator on MINIX

LCOM 2023/2024 - L.EIC

Class 5, Group 5:

Gonçalo Remelhe, up202205318@up.pt

Leonor Filipe, up202204354@up.pt

Ricardo Yang, up202208465@up.pt

Tiago Aleixo, up202004996@up.pt

Index

1. User’s Instructions

1.1 Main menu

At the start of the program, you'll encounter the main menu, showcasing various game options. Each page of the menu displays three available games. To navigate through additional game options, use the arrow keys on your keyboard to switch between menu pages. By moving the mouse and clicking the left button, you can select and enter a game.

To exit the program, the user can press the “ESC” key.

Also, every page has a total of three games on display. If some page does not have all three games, the menu will show a placeholder banner saying “Game not found”. It works like the other banners; however, this placeholder does not redirect you to other pages.

1.2 Play

Although our program can have various games, there are some common instructions if you are not using the controller such as the character's movement with the arrows, key "A" for select, "S" for start and pause and "ESC" to return to the main menu. Additionally, the keys "X" and "Z" have different purposes regarding the game the user is playing. For example, Mario requires the key "X" to jump and "Z" to throw fireballs (only with the super-power).

In short, with the keyboard, the user can play the various games with the keys "ESC", "Z", "X", "A", "S" and arrows.


MARIO
000000
×00
WORLD
1-1
TIME
SUPER
MARIO BROS.
©1985 NINTENDO
1 PLAYER GAME
2 PLAYER GAME
TOP- 000000

2. Project Status

Device	Description	Interrupt?
Timer	Control frame rate	Y
KBD	Menu navigation	Y
Mouse	Tool selection and drawing	Y
Video card	Display menus and screens	N
RTC	Define the date and hour	N
Serial Port	Transfer information with game controller	Y

Timer

- **Functionality:** The Timer module is responsible for controlling the timing and frequency of events within the game. It manages the timer interrupts and provides functions to set the timer frequency.
- **Function name:** *`timer_set_frequency()`*

Keyboard

- **Functionality:** The Keyboard module facilitates game control by handling keyboard input. It captures key presses and releases, allowing the user to interact with the software.
- **Function name:** *`kbd_ih()`*

Mouse

- **Functionality:** The Mouse module enables the use of mouse input for selecting game options. It tracks mouse movement and button presses, allowing users to navigate through the menu.
- **Function name:** *`mouse_int_handler()`*

Video card

- **Functionality:** The graphics driver was used to display the image and the entire visual of the emulator. The refresh rate was set at 30Hz, allowing for smooth viewing of the content and the mouse cursor, especially in “intense” scenarios.

  Every XPM image, excluding the numbers which were hand-drawn, were generated from images in PNG format using convertio (an online converter). These images were pre-treated in Photoshop to narrow down the color palette for each XPM.

  The video mode used in the **menu** was 0x14C, with a resolution of 1152x864 and 32-bit color (8:8:8:8). Each color component RGB can have 256 possible values (0 to 255), allowing for a total of 256^3 possible colors (16,777,216).

  To tackle "screen tearing”, we implemented the double buffering technique. All graphics were rendered onto the back buffer, and upon completion of the previous frame, the *`swap_buffers()`* function was invoked, which copies the information from the back buffer to the display buffer.

- **Function name:** *`set_frame_buffer()`*

RTC

- **Functionality**: The real time clock driver was used to get the current date. It allows the user to see the current time (year, month, day, hours, minutes and seconds). We also implemented RTC interrupts, but they were not used in our project.
- **Function name**: *`rtc_read_date()`*

Serial Port

- **Functionality**: The UART driver was used to get the status from a microcontroller connected to the computer by use that passed in the virtualbox settings, when receiving data it an interrupt is triggered and handled by *uart_ih()* that reads said data and saves it to a Queue in a FIFO way
- **Function name**: `uart_ih()`

3. Code Organization/Structure

main.c - 10%

As the central component of the NES emulator project on Minix, main.c oversees the initialization and execution of the emulator. It coordinates the integration of various modules, including the CPU, PPU, input devices (keyboard and potentially mouse), and ROM loading. main.c orchestrates the emulation loop, where it fetches and executes instructions, updates the display, processes user input, and manages the overall state of the emulator. In the context of the project, main.c holds significant importance as it governs the emulation process and provides the user interface for interacting with NES games. Key data structures within main.c include variables representing the emulator's state, such as the CPU registers, memory map, and display buffer.

bus.c - 2%

Connect all emulator devices below. It is also where ram is stored.

Responsible for Starting and Stopping the Emulator.

cartridge.c - 2%

Handles loading to memory, reading and writing to the game ROM.

controller.c - 2%

API to change controller status inside the emulator.

mapper.c - 1%

NES mappers are a way to solve the limited address space in the NES bus, many were developed over the years but for simplicity only mapper 0 is implemented at this time.

For more information read: https://www.nesdev.org/wiki/Mapper

p6502.c - 10%

The 6502 is the processor used in the NES, apple 2, Commodore 64 and others.

This file emulates the behavior of the processor, all address modes, all instructions and interrupt handling.

For more information read: https://www.nesdev.org/wiki/CPU

ppu.c - 15%

The PPU is the Picture Processing Unit, it is the chip responsible for creating graphics in the NES, basically the predecessor to the graphics card, it generates a 256 by 240 grid of pixels, pixel by pixel.

This file emulates the behavior of the PPU,

For more information read: https://www.nesdev.org/wiki/PPU

timer.c - 5%

The Timer module contributes to the project by ensuring precise timing for game events and animations. It manages the timer hardware and provides functions for setting timer frequency. The main data structure in this file is the *count* variable, which tracks the elapsed time.

keyboard.c - 8%

The Keyboard module plays a crucial role in user interaction, capturing keyboard input to control the game or application. It contains functions to handle keyboard interrupts and process key presses. The main data structure is *scan_code*, representing the current scan code received from the keyboard.

mouse.c - 8%

The Mouse module enhances user experience by enabling mouse input for menu navigation or game interaction. It processes mouse interrupts, tracks mouse movement, and detects button presses. The main data structure is *pp*, representing the mouse packet containing information about mouse movement and button status.

graphics.c - 15%

This module was reused from lab5, with several useful features added for our project. Notable improvements include the double buffering technique and functions for drawing XPMs from different positions (top-left corner and bottom-left corner).

The primary data structure is the double buffer for frame management, where each one is represented by a *uint8_t* pointer that stores the color information of each pixel.

sprite.c - 5%

The main objective of this module is to pre-load and store the XPMs used, thus avoiding the constant calling of the `xpm_load` function.

We use an enumeration *ImageName* to identify each image and a structure *XpmData* to store the image data and color map. This preloaded approach allows for faster and more efficient drawing of sprites. The sprites are drawn through calls to graphics functions.

rtc.c - 7%

The RTC module is crucial for managing date and time in the system. It includes functions to handle RTC interrupts (although not used), read the current date and time and functions to enable and disable the various interrupts. The main data structure used is *rtc_date_t*, which represents the current date and time.

uart.c - 10%

The UART driver module enables communication between a computer and a connected microcontroller by managing data transmission and reception through the UART interface. It initializes the UART with configurable parameters such as baud rate, word length, stop bits, and parity. The module subscribes to UART interrupts to handle incoming data, invoking the `uart_ih()` function to read and store received data in a FIFO queue for orderly processing. It also provides functions for sending data, enabling the FIFO, setting interrupt enable registers, and retrieving line and interrupt statuses. By leveraging interrupts and FIFO queues, the UART driver ensures efficient and reliable real-time data communication.

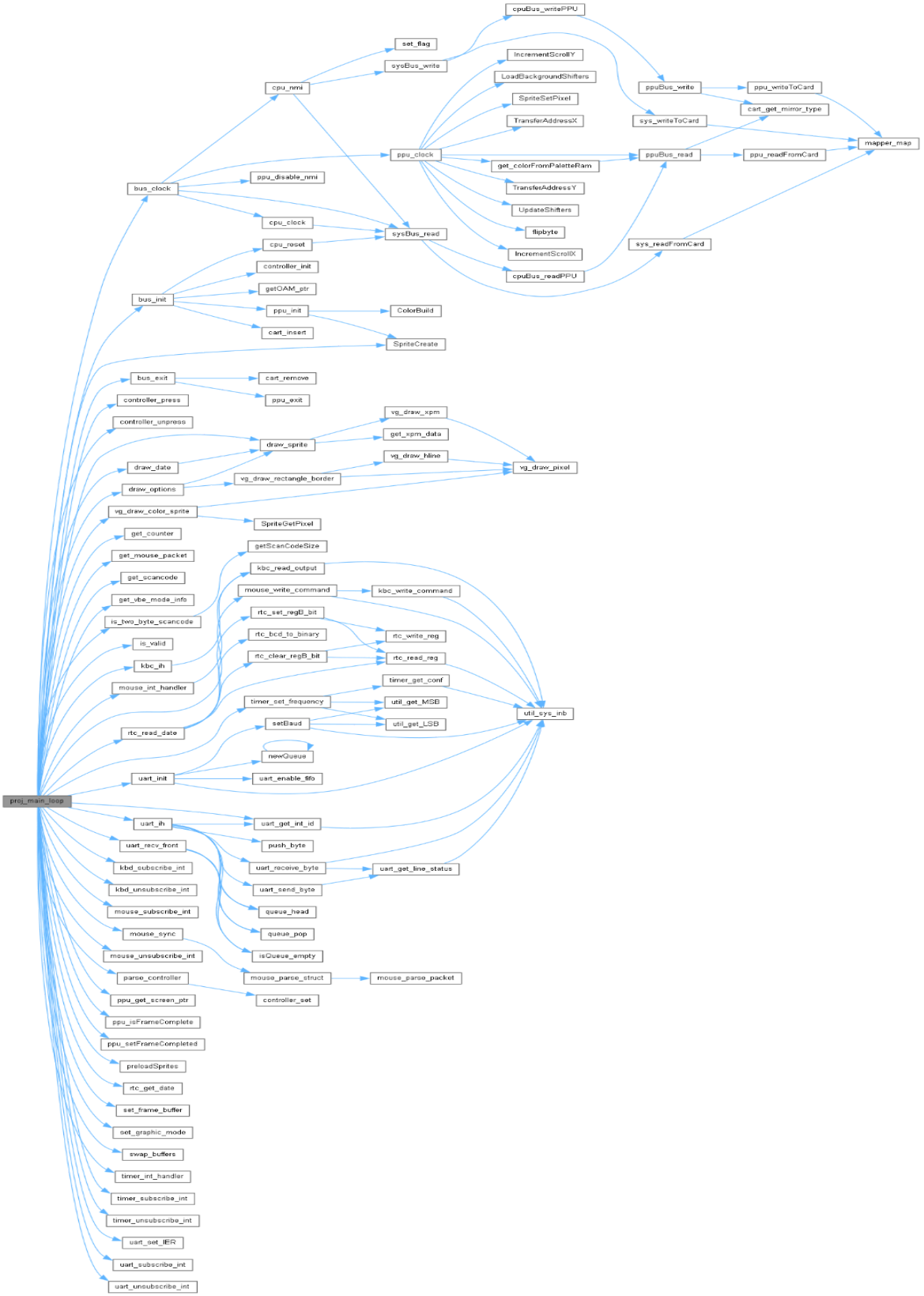
For this there was also developed a script to run in a ESP8622 microcontroller hat

communicates with the fiscal controller reading and sending its state through the serial connection.

xpm_image.c - 1%

This module contains the XPM images used throughout the project. These images are stored in the *xpm_row_t* format, which is a specific representation suitable for handling XPM data. This module centralizes all graphical assets, providing a consistent and organized way to manage and access the images required by the project.

Function call graph



4. Implementation Details

As recommended, we decided to develop the code in layers in order to facilitate the usage as well as development of more complex functions.

The RTC was implemented without the usage of interrupts, even though in our code there are functions to set alarms to every second. We have decided to not use them for simplicity.

5. Conclusions

This project encountered several challenges and delays that led to some functionalities not being implemented. It is also important to note that this project was really ambitious.

A potential future feature is a more dynamic use of the RTC. Currently, the RTC only shows the date and time on the menu screen. In the future, it could be used to create a dynamic background that changes from "sunshine" to "night" and automatically switch the emulator's theme between light and dark modes based on the actual time. This would enhance the visual appeal and user experience.

Additionally, incorporating a new mapper (for example, mapper no. 1) would allow for the inclusion of more games, broadening the project's scope and utility.

Despite the challenges, the project achieved several significant milestones such as successful implementation of a basic emulator framework capable of running select games with mapper 0 and the creation of a foundational codebase that allows for future expansion and feature addition.

Finally, we learned about teamwork and many other soft-skills. Effective communications and collaboration among team members were crucial in order to overcome integration challenges.

6. Appendix

After compiling the program, the user can use different commands in order to run the program as they want. Running the command ‘lcom_run proj’ will make the program run as the default mode: playing with the gaming controller as the player 2 and the keys of the keyboard as the player 1.

However, if the user runs the program with ‘*lcom_run proj “--no-uart”* ‘, the program will run without the use of the serial port.

Additionally, running the program with the command ‘*lcom_run proj “--player-1-serial”*’, it inverts the default flag and makes the player 1 as the player with the gaming controller.

Finally, the flag *“--vmode 0x115”* changes the resolution of the emulator (the menu stays with the same resolution).