Shape

Description automatically generated with medium confidence

EEET2490 – Embedded System: OS and Interfacing, Semester 2025-1

Group Assignment Report

**CLI, SCREEN DISPLAY, AND APPLICATION DEVELOPMENT FOR A BARE METAL OPERATING SYSTEM**

**Lecturer**: Dr Linh Tran – [linh.tranduc@rmit.edu.vn](mailto:linh.tranduc@rmit.edu.vn),

**Team Number: 6**

**Team Members:**

Huynh Ngoc Tai (s3978680)

Tran Quang Minh (s3988776)

Kim Nhat Anh (s3978831)

Vu Thien Minh Hao (s3988776)

**Date** : 26/05/2025

**TABLE OF CONTENT**

[I. INTRODUCTION 1](#_Toc199145491)

[II. WELCOME MESSAGE AND COMMAND LINE INTERPRETER (CLI) 1](#_Toc199145492)

[III. IMAGE, VIDEO, AND TEXT DISPLAY 2](#_Toc199145493)

[IV. APPLICATION DEVELOPMENT 2](#_Toc199145494)

[V. CONCLUSION 3](#_Toc199145495)

[VI. REFERENCES (USE IEEE STYLES) 3](#_Toc199145496)

# I. INTRODUCTION

This report documents the development of a bare-metal operating system running on Raspberry Pi. The assignment focuses on three major tasks: implementation of a welcome message and command line interface (CLI), display of images/videos/text on screen, and the development of an embedded game application. Our objective was to demonstrate understanding of OS-level features, low-level interfacing, framebuffer manipulation, and user interaction design.

# II. WELCOME MESSAGE AND COMMAND LINE INTERPRETER (CLI)

**Requirement:** Implement a CLI system with features such as welcome message, command parsing, history, and execution for basic system commands like help, clear, showinfo, etc. Features like auto-completion and command history were to be added using special key handling.

**Implementation:**

* **Welcome Message:**

An ASCII art was generated using <https://onlineasciitools.com/convert-text-to-ascii-art> and printed to UART during boot.

* **CLI Prompt:**

The prompt always shows “FixingGoodOS>” before each command.

* **Command Handling:**

Input is collected using a character buffer commandBuffer, and processed when Enter is pressed.

Autocompletion is triggered when the Tab key (\t) is pressed. The implementation checks for prefix matches among available commands:

History is stored in a ring buffer. Pressing “–” navigates back, while “=” navigates forward. The buffer is managed to overwrite the oldest command if full. (We don’t use “-” and “+” explicitly because it would require an extra Shift on keyboard for it)

Backspace (127 or \b) removes the last character and redraws the line.

* **Commands Implemented:**

-help: Lists all commands or detailed info if followed by a command name.

-clear: Clears terminal screen using ANSI escape codes.

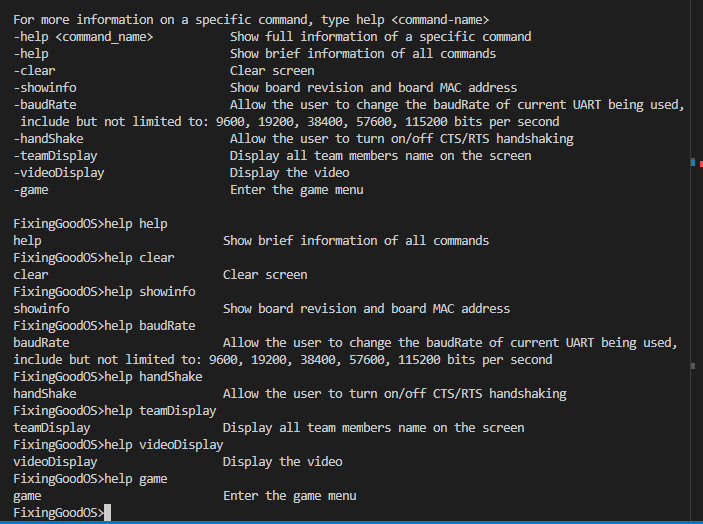
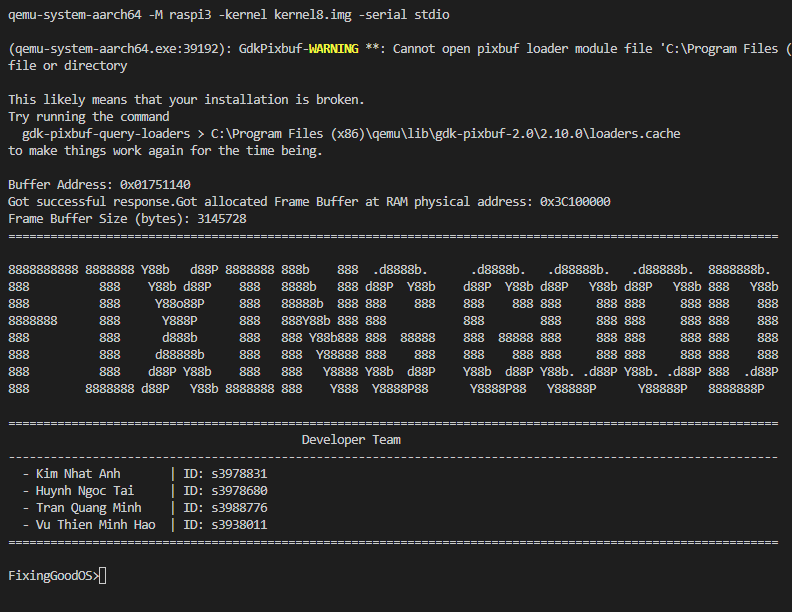
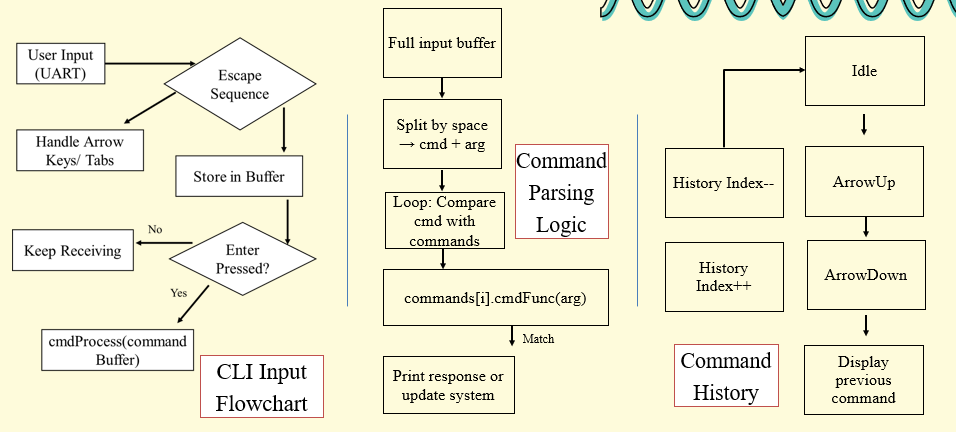
-showinfo: Reads MAC and board revision from specific memory locations.

-baudrate: Accepts integer input and sets UART baudrate via divider calculation.

-handshake: Enables/disables CTS/RTS using GPIO setup.

-teamDisplay and videoDisplay are integration commands used to demonstrate Task 2.

**Result Discussion:** All CLI features function correctly on UART0. The auto-completion significantly improves usability. Command history is intuitive. Limitations include no fuzzy-matching or command arguments parsing.

**Summarized program Flowchart/Diagrams:**

# 

# III. IMAGE, VIDEO, AND TEXT DISPLAY

**Requirement:**

a) Show names of team members with colored text over a background image. Custom font must be used.

b) Display a short video as sequence of images.

**Implementation:**

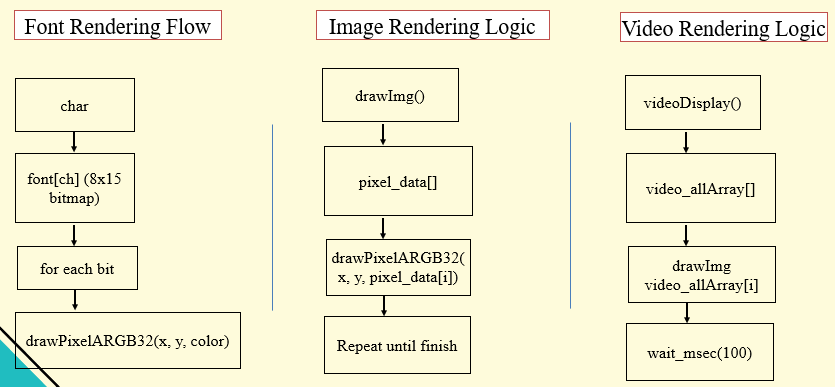
* **Font:**

We implemented a custom font system where each character glyph is represented as an 8x15 bitmap. These bitmaps were extracted from the Tamzen8x15.bdf file in the Tamzen font repository - <https://github.com/sunaku/tamzen-font/tree/master/bdf>.

The BDF file was parsed and converted into a C-style header file containing a two-dimensional array, where each glyph is stored as a sequence of bytes. During rendering, each bit in the glyph data is read and drawn to the screen pixel-by-pixel using drawPixelARGB32().

* **Background Image:** An image was converted to ARGB32 using https://javl.github.io/image2cpp/ and displayed full-screen on qemu.
* **Text Overlay:** Each team member's name is drawn in a different color at fixed positions with the background image that’s draw beforehand using drawPixelARGB32() function.
* **Video Display:** A short video was converted to 31 BMP frames using FFmpeg, then each frame converted to ARGB32 arrays. These arrays are shown sequentially using a videoDisplay() loop.

**Summarized program Flowchart/Diagrams:**

****

**Result Discussion:** Text and images render as expected. Video playback is smooth but limited to 20 FPS due to processing overhead.

**Summary of features implemented in both Tasks 1 & 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature Group** | **Command/ Feature** | **Implementation** | **Testing (any issues/limitations)** |
| **CLI Basic Features** | welcome screen | complete | Displays on UART boot |
| help | complete | Shows usage with and without args |
| clear | complete | Clears screen using ANSI codes |
| showinfo | complete | Displays MAC and board revision |
| baudrate | complete | Changes UART0 baudrate live |
| **CLI Enhancement** | OS name in CLI | complete | Static prompt FixingGoodOS> |
| Auto-completion in CLI | complete | Matches prefix and fills input |
| Command history in CLI | complete | Works with “–“ (going back) and “=” (going forward)  (We don’t use “-” and ”+” because it would require an extra shift on keyboard) |
| **Image, Video, and Text Display** | Background image and text display | teamdisplay | Shows names of 4 members in color with the RMIT background. |
| Video display | videodisplay | Plays 31 frames @ ~20 FPS |

# IV. APPLICATION DEVELOPMENT

**Game: Basketball Star**

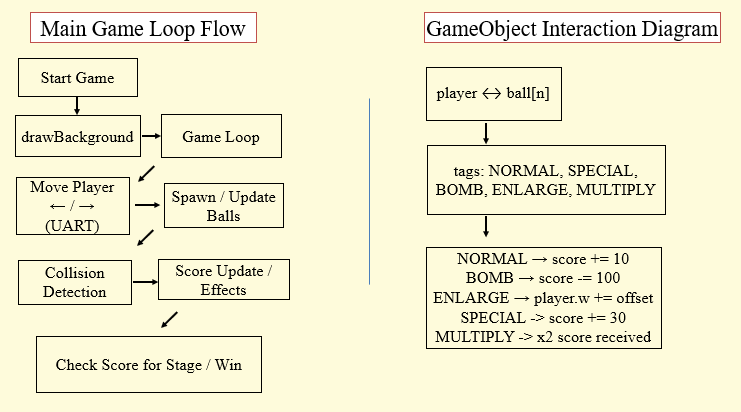
**Gameplay:**

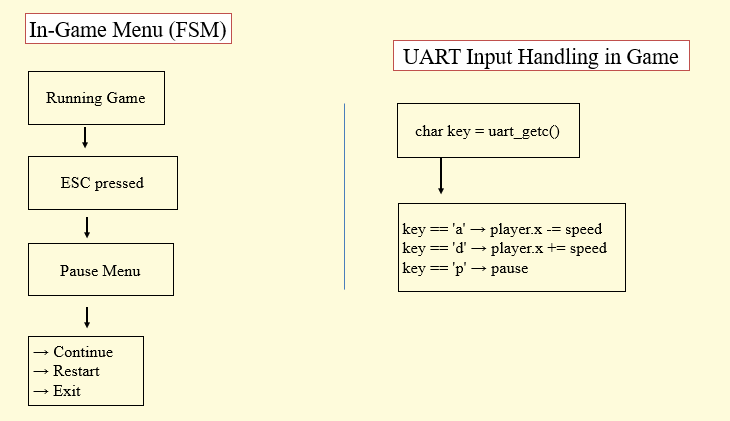
* Player controls a basketball hoop at the bottom of the screen.
* Balls fall from the top. Player must move left/right to catch them.
* Different ball types: normal, special (bonus), bomb (penalty).

**Design:**

* **Background:** Static image displayed once.
* **Sprites:** Hoops and balls are drawn using pixel data.
* **Physics:** Balls fall vertically; collision is checked against the hoop.
* **Controls:** Arrow keys are mapped through UART for player movement.
* **Game Logic:** Score increments or decrements based on ball type.
* **Communication:**
  + UART logs each command received.
  + Acknowledges valid commands with ACK.
  + Logs current score and number of balls caught.

**Summarized program Flowchart/Diagrams:**





**Result Discussion:** Game is functional with responsive controls and smooth visuals. Limitation: no pause/resume functionality, and fixed difficulty.

# V. CONCLUSION

This assignment provided valuable experience in low-level OS development, framebuffer rendering, and interactive application design. We deepened our understanding of UART, CLI parsing, framebuffer structure, and embedded graphics handling. Working with the Raspberry Pi in a bare-metal environment helped bridge theory and practical implementation. Looking forward, we aim to explore real-time scheduling and multi-threading for more advanced systems.

# VI. REFERENCES (USE IEEE STYLES)

[1] Raspberry Pi Board Revision Info, https://www.raspberrypi-spy.co.uk/2012/09/checking-your-raspberry-pi-board-version/  
[2] MAC Address Info, https://www.javatpoint.com/what-is-mac-address  
[3] ASCII Art Generator, https://onlineasciitools.com/convert-text-to-ascii-art  
[4] Image to C Array Converter, https://javl.github.io/image2cpp/  
[5] Scratch Game Tutorials, www.youtube.com/watch?v=jFVJdRLZoQ4, www.youtube.com/watch?v=QXru0rSV2ZQ  
[6] PCS Font Tutorial, https://github.com/bztsrc/raspi3-tutorial/tree/master/0A\_pcscreenfont  
[7] Framebuffer Tutorial, <https://github.com/babbleberry/rpi4-osdev/tree/master/part5-framebuffer>

https://github.com/sunaku/tamzen-font/tree/master/bdf

*For each part of the assignment, please introduce the* ***requirement*** *(what will be implemented), then following by* ***implementation*** *(clearly explain how you implemented it, including important information for readers to understand), and finally* ***result discussion*** *(result and any limitation if it has).*

*For the code explanation, you don't need to go line by line, but should explain the way you do it (could be done in similar way of the lab guide). Flowcharts or diagrams could be used to support/illustrate your explanation if necessary.*

