Technical Documentation

Authors: Romanov Roman, Skulakova Alisa
Project: Pong Game with Single and Multiplayer modes
Semester: B232 - Summer 2023/2024

Table of Contents

1. Introduction

This project was created as part of the semester project for the APO subject. Its goal is to load and manipulate Ping Pong Game on the screen of the MicroZed board. The classic Ping Pong game, also known as Pong, is a video game where two players control paddles to hit a ball back and forth across the screen. The objective is to score points by preventing the ball from passing your paddle. Program is built using `Makefile`

Software required to run the program:

- Arm Linux C and C compiler with minimal supported version C - C99.

2. Project Structure

The project is primarily divided into two parts: files starting with mzapo (functions which are provided in the project template to access and manipulate hardware), which are located in the mzapo folder, and files containig game logic.

- Files starting with mzapo are provided libraries for accessing the lowest level of hardware on the Micro Zed and were provided.
- All the other files in the main root are created specifically for loading and operating Pong game.

The entry point of the program is the file **`main.c`** , initializing and allocating memory for the hardware controllers and loading start page.

```
Project Root Directory
│
├── mzapo/                      # Hardware Access Libraries
│   ├── mzapo_regs.h            # Initializes hardware controllers
│   └── ...                     # Other mzapo related files
│
├── main.c                      # Entry Point of the Program
│
├── start.c                     # Start-Up Animation and Initialization
│
├── menu.c                      # Main Menu Selection
│   └── botmenu.c               # Bot Difficulty Selection
│
├── pong.c                      # Game Logic for Playing Against a Bot or
Another Player
│   ├── playGameBot()           # Gameplay vs. Bot
│   └── playMultiplayer()       # Gameplay vs. Another Player
│
├── painter.c                   # Graphical Operations and Rendering
│
├── knobs.c                     # Global Knob Input Handling
│
├── led.c                       # Global LED Control and Animations
│
└── end.c                       # End Game Screen and Restart Prompt
    └── menu.c                  # Transition to Main Menu
```

3. main.c

This file has only one method and is responsible for starting the application.

4. start.c

The `start.c` file initiates the application's start-up animation. It prints "Start," draws the background (`drawBackground(0x0000)`), and attempts to initialize knobs (`getKnobsValue()`) for user interaction. If knob initialization fails, the program exits. Otherwise, it displays welcome messages ("START" and "PONG") on an LCD-like display.

The core of the start-up sequence is a loop that waits for any button press, cycling through LED patterns (`ledStartPage(10)`) and dynamically updating a prompt message. Upon detecting a button press, the program transitions to the main menu, ending the start-up animation.

5. menu.c | botmenu.c

The `startMenu()` function in `menu.c` presents a player with a selection menu post-start-up animation. It displays "Player menu," draws a new background, and sets up menu options for single-player and multiplayer modes. The function highlights the current choice based on the selected game mode, toggling between options until a selection is made via button presses.

Initially, the game mode defaults to single-player. As the player navigates through the options using knobs and buttons, the highlighted choice updates, showing the selected game mode. Once a selection is confirmed by pressing a button, the function concludes the menu presentation by calling either `startBotMenu()` for single-player or `playMultiplayer()` for multiplayer, depending on the final choice. A `botmenu.c`is represented similar to `menu.c`, the only difference is between choosing options. Here the user should choose a difficulty level: easy, medium or hard.

6. pong.c

The `playGameBot()` function manages gameplay against a bot, adjusting the bot's speed based on the `botmode`. It initializes the game environment, including the `ball` and `rackets`, and runs until one side scores 5 points. Player input controls the

racket's movement, while the bot's movements are automated. The game state is refreshed in each iteration, including drawing the ball and rackets, and processing player inputs. Scores are updated, and the game ends when a player reaches 5 points.

The `playMultiplayer()` function handles multiplayer gameplay, allowing both players to control their rackets simultaneously using knobs. It follows a similar structure to `playGameBot()`, with game state refreshes and score updates in each iteration. The game concludes when a player reaches 5 points, offering a competitive experience between two human players.

7. painter.c

The **painter.c**, file contains functions that handle graphical operations for a game or application, focusing on drawing characters, pixels, rectangles, and scores on a screen. Here's a concise overview of its functionalities:

Drawing Text

- `draw_word()`: Draws a string of characters at a specified position, with an optional offset and scaling factor.
- `draw_char()`: Draws a single character at a given position, utilizing a font descriptor (`fdes`) to determine the character's width and pixel representation.

Pixel Operations

- `draw_pixel_big()`: Draws a large pixel (scaled version) at a specific location, useful for creating larger shapes or filling areas.
- `draw_pixel()`: Draws a small pixel at a specified location, directly manipulating framebuffer (`fb`) memory.
- `highlightCurrentChoice()`: Highlights a rectangular area by changing its background color, leaving the text color unchanged.

Shape Drawing

- `drawRacket()`: Draws a racket object, iterating over its dimensions to fill the area with a specified color.
- `drawBall()`: Similar to`drawRacket()` , but for drawing a ball object.
- `drawRectangle()`: Draws a rectangle with alternating colors along its edges, creating a striped effect.

Background and Rendering

- `drawBackground()`: Fills the entire screen with a specified color, resetting the visual canvas.
- `renderLCD()`: Writes the contents of the framebuffer to the LCD, updating the display.

Utility Functions

- `char_width()`: Calculates the width of a character based on the font descriptor, useful for aligning text.
- `showScores()`: Formats and draws the scores of two players, updating the display with the current score difference.

These functions collectively provide a comprehensive toolkit for rendering graphics, text, and interactive elements on the screen, suitable for games or applications requiring visual feedback.

8. knobs.c | led.c

The `knobs.c` and `led.c` files contain functions that interact with hardware peripherals, specifically knobs and LEDs, to provide user input and visual feedback in a game or application. In `knobs.c`, `initKnobs()` maps physical memory addresses for knob control, and `getKnobsValue()` reads knob positions and button states from mapped memory. These functions facilitate reading user inputs from knobs, enabling game controls or settings adjustments.

In contrast, `led.c` focuses on controlling LED indicators.`ledInit()` initializes LED control by mapping physical memory addresses, and `ledLineClean()` resets all LED lines. `ledRGBClean()` clears RGB LED colors, preparing them for new color assignments. `colorRGB()` sets the RGB color of the LEDs, allowing for customizable lighting effects. Functions like `ledWin()` and `ledStartPage()` use these capabilities to create visual cues, such as victory animations or startup sequences, by cycling through LED colors and patterns. These visual elements enhance the user interface, providing clear indications of game status or outcomes.

9. racket.c | ball.c

The `racket.c` and `ball.c` files define the behavior and interactions of rackets and balls within a game, likely resembling a Pong-like setup. `racket.c` includes functions

for initializing and moving rackets, with `initracket()` setting initial positions and colors based on the racket's number, and `moveRacket()` adjusting the racket's vertical position within bounds. Collision detection with the ball is handled by checking if the ball's horizontal or vertical movement would result in a collision with any racket, adjusting the ball's direction accordingly. `ball.c` manages the ball's properties and movement, including initialization with random or predefined speeds for multiplayer or bot-controlled games. The `moveBall()` function updates the ball's position, checks for collisions with rackets and the game boundaries, and adjusts the ball's direction upon hitting a racket or reaching the edge of the screen. Collision detection is facilitated by `checkCollisionX()` and `checkCollisionY()`, which determine if the ball's path intersects with a racket's boundaries, crucial for gameplay dynamics and scoring.

10. end.c

The `end.c` file defines the `showEnd()` function, which is responsible for displaying the end-of-game screen, determining the winner, and prompting the player to restart the game. Upon concluding a match, it first prints "End" to the console and then draws a black background. It formats messages to display the winning player and the final scores. The screen is populated with these messages, instructing the player to press any button to restart the game. The function waits for any button press, toggling the message color between white and black every few seconds to indicate readiness for a new game. Additionally, it cycles through an LED pattern to visually signal the end of the game. Once a button is pressed, the game resets the scores, clears the old scores, and calls `startMenu()` to begin a new game session.

10. Reference and Knowledge Base

[APO classes materials]