CSE 379 ( **Microprocessors** )

# Lab 7

**Name:** *Thien Phuc Le, Shouvik Das*

**Usernames:** ptle, sdas28

**Lab section:** R1

**Date:** 1/05/2025

## ***Table of contents:***

# Section 1: ( Division of Work )

1) **Thien Phuc Le:**
    a. Responsible for main flow logic, their dedicated flowsheet.

2) **Shouvik Das:**
    a. Responsible for the initialization for startup, paddle movement
    b. Responsible for making the ***Documentation*** for the lab.

# Section 2: ( Program Overview )

## Program Overview:

### Start the Program

- The program implements the classic *1972 Pong* game in ARM Assembly using the Tiva C Series and the Alice Board.
- The game is displayed on a PuTTY terminal using UART0, with paddle and ball updates using ANSI escape sequences.
- Two players compete: The blue user uses the W/S keys, and the red user uses O/L keys on the keyboard to move the paddles.
- Gameplay is driven by GPIO and UART interrupts, and timers manage ball speed and power-up events.

### Game Initialization

- UART0 is configured to send and receive characters to the PuTTY terminal for rendering the game board and handling keyboard input.
- The *RGB LED* is configured to show score-related visual cues.
- A timer is set up to manage ball movement and timed events like power-ups.
- The game board is drawn in PuTTY as an 80x24 grid with walls, two paddles, and a ball centred to play.
- One timer is initialized to control ball movement and power-up timing.

### How to Play

□ **Player 1 (Left Paddle)**:

- Press **W** to move up
- Press **S** to move down

□ **Player 2 (Right Paddle)**:

- Press **O** to move up
- Press **L** to move down

□ The ball moves automatically based on timer ticks, bouncing off paddles and walls.

□ If a player fails to return the ball and the ball goes behind the paddle, the opponent scores a point.

□ The ball resets after every point with a short delay before serve.

### Game Flow

□ Ball moves at fixed intervals via timer interrupt, bouncing off walls and paddles.

□ If ball passes a paddle, the other player scores, ball resets to center.

□ Random power-ups spawn between rounds, granting temporary paddle size boost.

### Game End and Replay

□ The game continues until one of the players reaches the winning score which is decided by the users using the SW2-SW5

□ When this happens:

- A *victory message* is displayed on the terminal.
- All game elements are frozen, and the LEDs start dancing randomly ( by that we mean that they start lighting up randomly )
- Players are then prompted to press a key to either restart or exit the game.

□ If the game is restarted, the board, positions, power-ups, and scores are all reset for a fresh match.

# Program Summary:

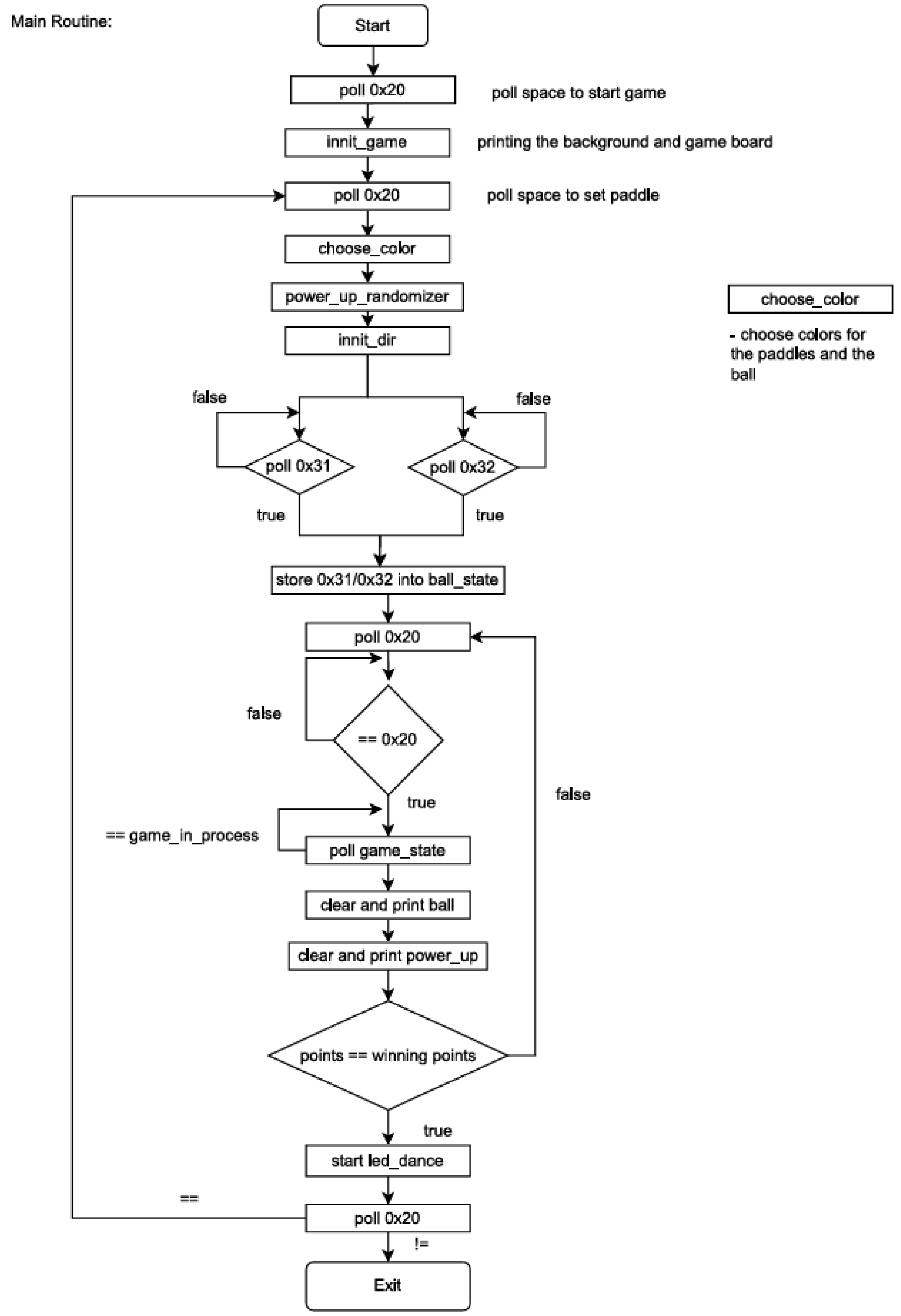
This lab implements the classic 1972 Atari video game Pong using ARM assembly on the Tiva C series. It integrates UART0 for serial communication, GPIO for hardware interfacing, and timer interrupts to create a smooth gameplay. The game is displayed on a PuTTY terminal using ANSI escape codes and runs at 30–60 frames per second based on in-game events.

Two players control vertical paddles: Player 1 uses the **W/S** keys, while Player 2 uses **O/L**. A ball moves across an 80x24 character board, bouncing off walls and paddles. The ball’s speed increases with each successful paddle hit, up to a maximum of 60 FPS. When a player scores by getting the ball past their opponent’s paddle, the RGB LED lights up in their paddle's color, and a new round begins upon a keypress.

Random power-ups  spawn between rounds and grant the last-hitting player a temporary paddle size boost. Four LEDs on the Alice EduBase board act as a countdown timer for the 12-second power-up duration. If a player presses a key while the ball is moving *away* from them, a penalty is triggered, transferring a point to the opponent and ending the round.

Game state is managed through timers, with options to pause and resume using *SW1* on the Alice. The game ends when either player reaches a score selected using *SW2–SW5*, and a winning prompt is displayed with dancing LEDs to celebrate the victory. The user is then prompted to replay or quit the game.

# High level flowchart :



# Section 3: ( Subroutine Descriptions )

## Lab7

- **Purpose:** This is the main routine that initializes and manages the overall game flow for Pong. It waits for user input to begin the game, sets up the game board and colors, initializes ball direction, and continuously loops through game states—checking scores, updating ball and power-up visuals, and determining if the game has ended.
- **Arguments:** None.
- **Return Value:** None.

## Timer_Handler

- **Purpose:** This interrupt service routine manages the movement of the ball, checks for paddle and wall collisions, handles scoring, and implements power-up collection and activation. It uses the *ball_state* variable to determine the movement pattern (horizontal, 45° up, or 45° down), and adjusts the direction when borders or paddles are hit. If the ball hits a power-up, the paddle size of the last player to hit the ball is temporarily increased. When a player scores, it stops the ball and resets its position and state for the next round.
- **Arguments:** None.
- **Return Value:** None.

## UART0_Handler

- **Purpose:** Handles keyboard input received via UART. It checks if the game is paused or if paddle movement is allowed. Based on player key presses (**W/S** for Player 1 and **O/L** for Player 2), it moves the corresponding paddle unless a penalty condition is triggered. It also checks if the user intends to quit the game.
- **Arguments: r0**: key pressed gets stored here.
- **Return Value:** None.

## Switch_Handler

- **Purpose:** This interrupt service routine handles the pausing and resuming of the game when SW1 on the Tiva C board is pressed. It checks the current *pause_state*, and either disables or enables the game by updating the clock, paddle movement, and ball movement accordingly. It ensures the game freezes or resumes smoothly, including power-up and timer behaviours.
- **Arguments:** None.
- **Return Value:** None.

## Mov_cursor

- **Purpose:** Moves the cursor in the PuTTY terminal to a specified row and column using ANSI escape codes. This subroutine formats and prints the ANSI string (the format being .string 27, "[Line;ColumnH"). The Line value is passed in r0, and Column value passed in r1. The routine prints the ANSI escape sequence and moves the cursor to the location specified.
- **Arguments: r0**: Holds Line value, **r1**: Holds Column value
- **Return Value:** None.

## Reset_cursor

- **Purpose:** Resets the terminal cursor position to the top-left corner of the PuTTY screen.
- **Arguments:** None.
- **Return Value:** None.

## Second Timer_handler

- **Purpose:** This interrupt routine handles the time-based game mechanics like power-up durations, in-game clock updates, and post-game LED animations. It basically makes sure that the power-up effects expire correctly and controls the LED countdown for power-up timing. It also triggers the celebratory LED dance when the game ends.
- **Arguments:** None.
- **Return Value:** None.

## set_color

- **Purpose:** This subroutine sets the ANSI text color for objects on the UART display (paddles, ball, borders) by loading and printing a colour string from a lookup table(LUT). *There are more details in the next section about the LUT.*
- **Arguments: r0**: Holds the offset value which maps to a certain colour on the LUT.
- **Return Value:** None.

## set_color_ele

- **Purpose:** This subroutine sets the ANSI colour for the Alice EduBase LEDs based on a predefined element lookup table (ELE). *There are more details in the next section about the ELE.*
- **Arguments: r0**: Holds the offset value which maps to a certain colour on the ELE.
- **Return Value:** None.

## penalty_handler

- **Purpose:** Handles penalties when a player violates input restrictions (moving the paddle while the ball is moving away). This routine resets the board for the next round by clearing paddle and ball positions, disabling any active power-ups, and updating the score based on who committed the penalty.
- **Arguments: r0**: None.
- **Return Value:** None.

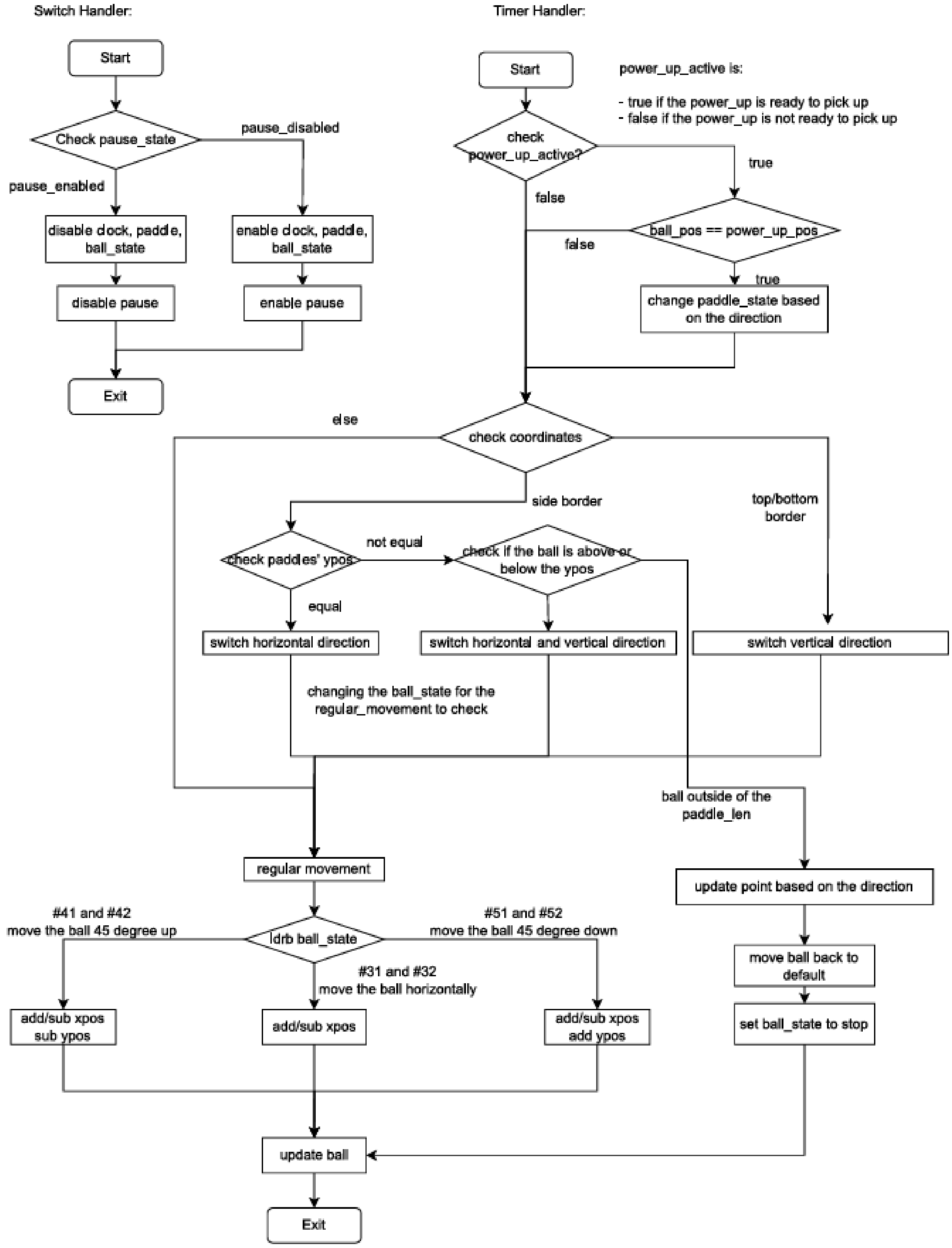
## print_paddle

- **Purpose:** Prints the paddle to the UART display at its current position. It first clears the old paddle region with a black border, then prints the paddle based on its current state: normal length or boosted length.
- **Arguments: r0**: Paddle position
- **Return Value:** None.

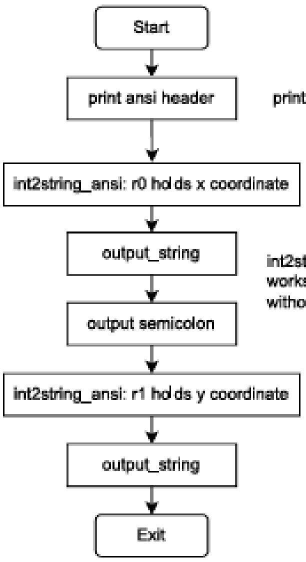
## Power_up ( High Level )

- **Purpose:** This subroutine manages the appearance, activation, and expiration of power-ups during gameplay. After a ball is served, it waits a randomized period before spawning a power-up on the game board. When the ball collects the power-up, the paddle of the player who last touched the ball is temporarily boosted and the Alice LEDs indicate the remaining duration of the effect.
- **Arguments:** None.
- **Return Value:** None.

# Section 4: ( Subroutine Flowcharts )


Switch Handler:
Start
Check pause_state
pause_disabled
pause_enabled
disable clock, paddle, ball_state
enable clock, paddle, ball_state
disable pause
enable pause
Exit
Timer Handler:
Start
power_up_active is:
- true if the power_up is ready to pick up
- false if the power_up is not ready to pick up
check power_up_active?
true
false
ball_pos == power_up_pos
false
true
change paddle_state based on the direction
else
check coordinates
side border
top/bottom border
check paddles' ypos
not equal
check if the ball is above or below the ypos
equal
switch horizontal direction
switch horizontal and vertical direction
switch vertical direction
changing the ball_state for the regular_movement to check
ball outside of the paddle_len
regular movement
update point based on the direction
#41 and #42
move the ball 45 degree up
ldrb ball_state
#51 and #52
move the ball 45 degree down
#31 and #32
move the ball horizontally
move ball back to default
add/sub xpos
sub ypos
add/sub xpos
add/sub xpos
add ypos
set ball_state to stop
update ball
Exit

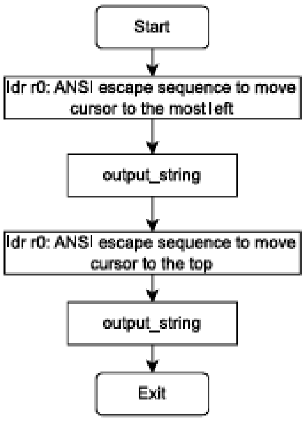
mov_cursor: printing 27,"[L;CH" with L, C as variable


Start
print ansi header
int2string_ansi: r0 holds x coordinate
output_string
output semicolon
int2string_ansi: r1 holds y coordinate
output_string
Exit

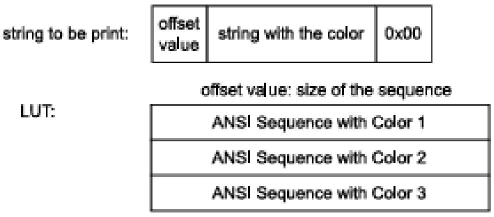
printing .string 27,"["

int2string_ansi:
works like standard int2string but
without the coma

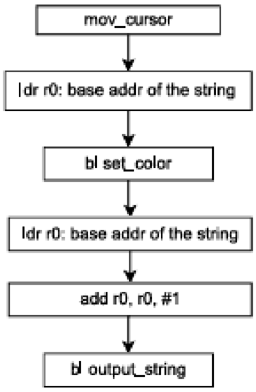
reset_cursor:


Start
ldr r0: ANSI escape sequence to move cursor to the most left
output_string
ldr r0: ANSI escape sequence to move cursor to the top
output_string
Exit

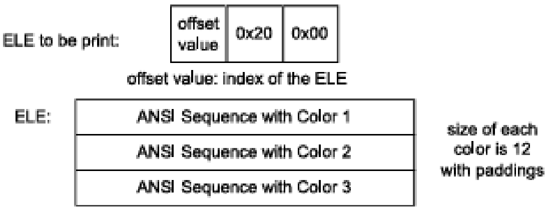
LUT Basics:


string to be print:
offset value
string with the color
0x00
offset value: size of the sequence
LUT:
ANSI Sequence with Color 1
ANSI Sequence with Color 2
ANSI Sequence with Color 3

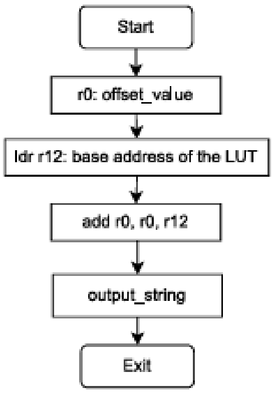
Basic Print:


mov_cursor
ldr r0: base addr of the string
bl set_color
ldr r0: base addr of the string
add r0, r0, #1
bl output_string

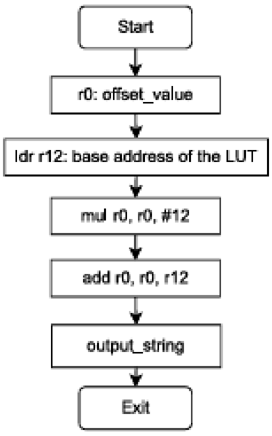
ELE Basics:


ELE to be print:
offset value
0x20
0x00
offset value: index of the ELE
ELE:
ANSI Sequence with Color 1
ANSI Sequence with Color 2
ANSI Sequence with Color 3
size of each color is 12 with paddings

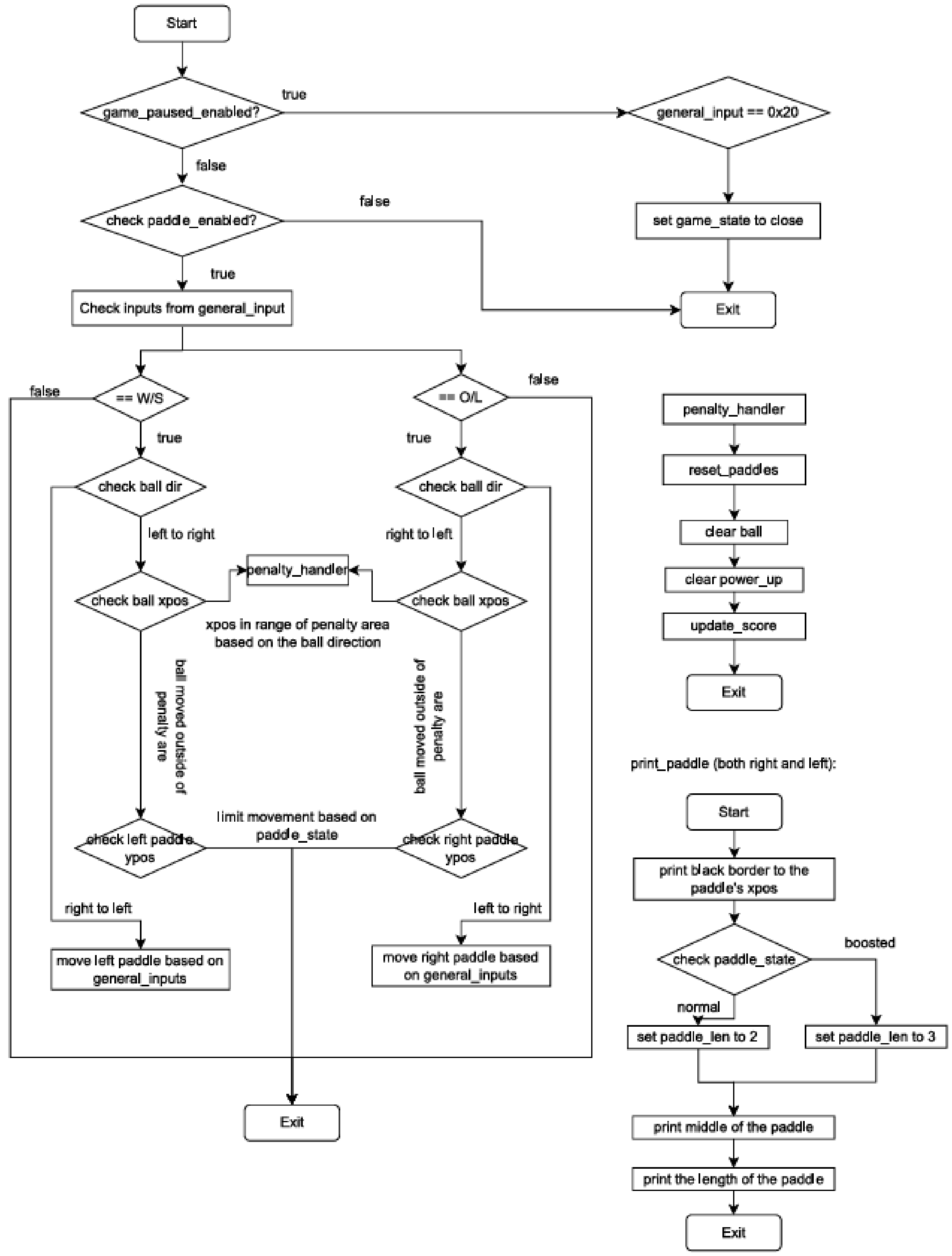
set_color:



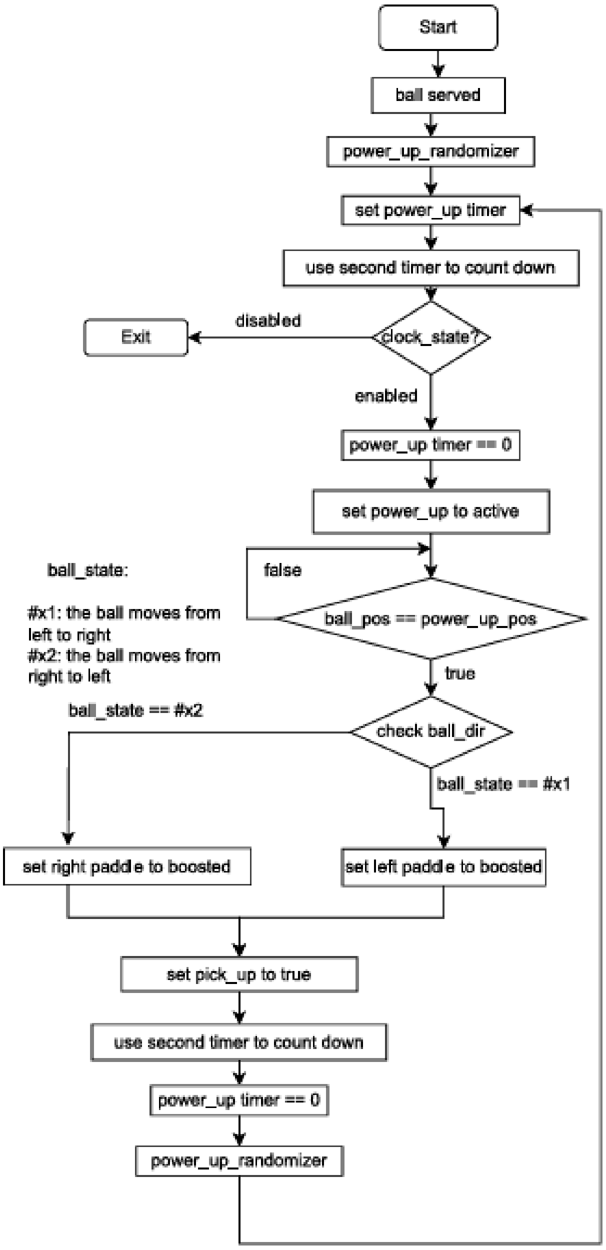
set_color_ele:



UART_Handler:



power_up high level:



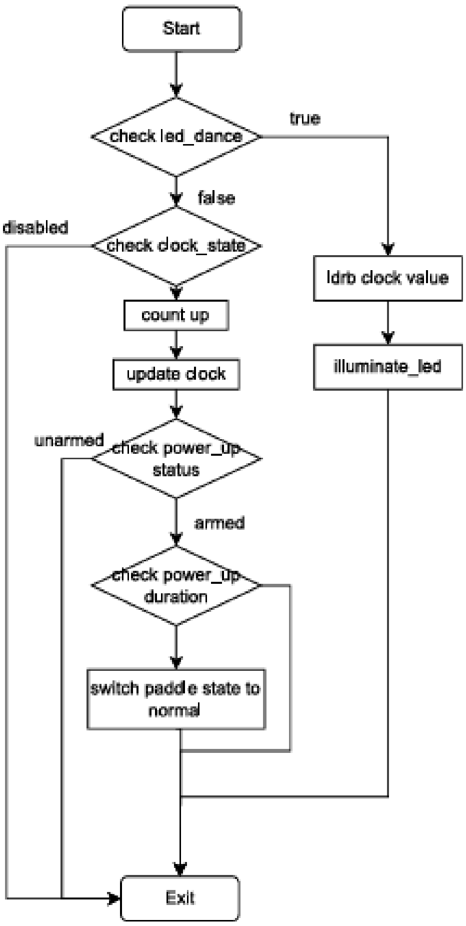
ball_state:

#x1: the ball moves from left to right
#x2: the ball moves from right to left

power_up_randomizer

- set xpos, ypos, and arm_timer based on the clock value

Second Timer Handler:


Start
check led_dance
true
false
check clock_state
disabled
ldrb clock value
count up
illuminate_led
update clock
unarmed
check power_up status
armed
check power_up duration
switch paddle state to normal
Exit