Ray A Torres

Manav Patel

Jesus A Rodriguez Toscano

Final Report: Team 13 PING PONG

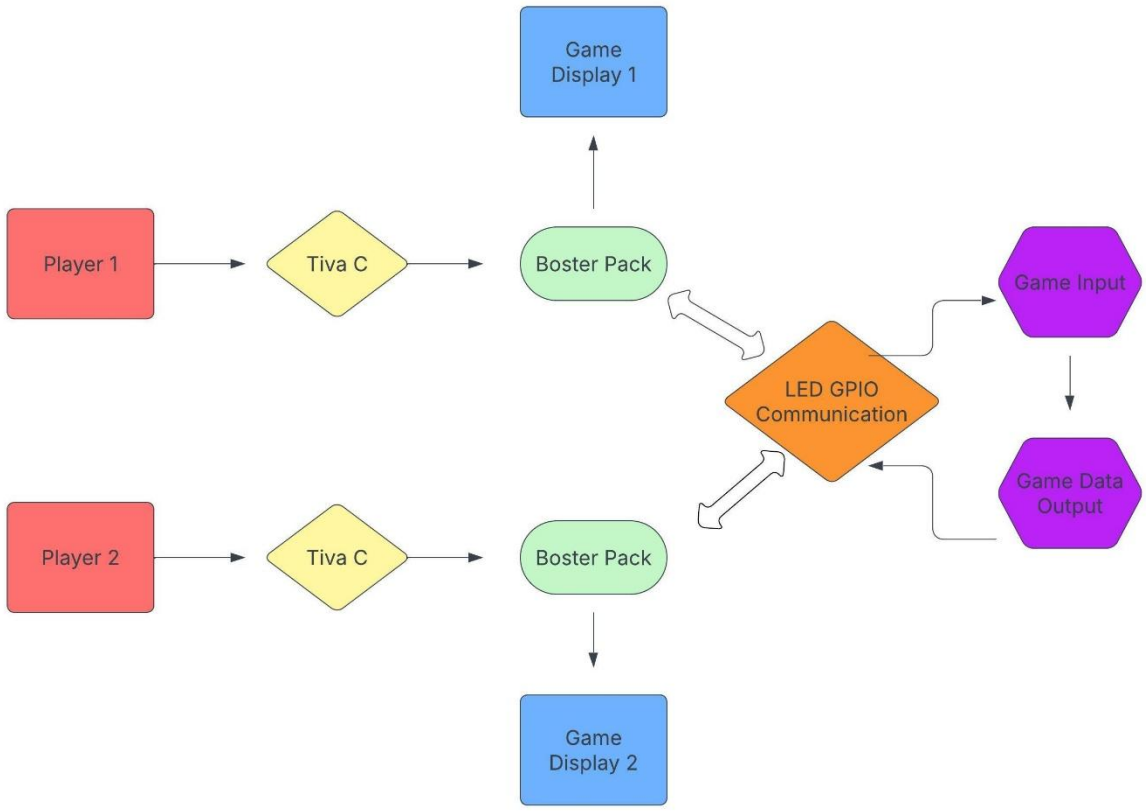
Abstract

This project presents a multiplayer ping pong game implemented on the TM4C123 Tiva-C microcontroller using a custom-built real-time operating system (RTOS). The game allows two players, each with their own Tiva-C board, to interact and play independently while synchronizing gameplay through a shared state change mechanism. This is achieved by the use of GPIO pins, specifically pin PC5; when a signal is sent through this pin the system confirms reception by modifying the LED state, where a transition from OFF to ON signals a trigger to spawn a new ball, visible on both devices. The system supports up to four concurrent balls, enhancing the game's complexity and interactivity. The project emphasizes modular RTOS design, event-driven programming, and lightweight communication without traditional networking protocols like UART or SPI. It demonstrates fundamental real-time concepts such as task management, synchronization, periodic events, and display rendering on a low-power embedded platform.

Project Objectives

- Develop a fully functional ping pong game using the TM4C123 Tiva-C LaunchPad using the RTOS designed in Lab 3.
- Implement multiplayer functionality by using GPIO pins and shared LED state transitions to synchronize gameplay across two independent microcontrollers.
- Design a real-time game loop using periodic event threads to manage paddle control, ball physics, and screen updates.
- Support dynamic gameplay with up to four active balls, each responding to collisions with walls and paddles.
- Enable modular and maintainable code structure with separate source files for paddle, ball, wall, and game logic.
- Ensure stable rendering and state consistency through effective use of RTOS primitives such as semaphores and periodic events.

Functional block Diagrams



Functional Description

The system is designed as a multiplayer ping pong game running on two independent TM4C123 Tiva-C microcontrollers. Each microcontroller executes the same RTOS-based game code and displays the gameplay on a connected LCD using the BoosterPack interface.

Players interact with the game using onboard joysticks to control their paddles. The core game logic, including ball movement, collision detection, and screen updates, is handled through periodic event threads managed by a custom RTOS. The paddle position is updated based on joystick input, and the ball bounces off walls, the top boundary, and the player's paddle.

Multiplayer functionality is implemented by monitoring a shared LED state that changes based on the communication done via pin PC5. When either player presses their input button, the LED changes from OFF to ON, triggering both devices to spawn a new ball simultaneously. This synchronization mechanism allows both microcontrollers to

independently reflect a shared game state without requiring direct communication via UART or SPI. The game allows up to four active balls at once.

The system uses real-time scheduling, clean screen rendering, and logic isolation across multiple source files (ball.c, paddle.c, walls.c) to ensure smooth gameplay. The wall boundaries are redrawn as needed to prevent visual artifacts. Game updates are processed at 30 Hz using periodic event threads for stable and consistent performance.

Software Development Environment

The software was developed using Keil µVision 5 with the ARM Compiler version 5.06 for the TM4C123GH6PM microcontroller. Code was written in C and structured into modular files for game logic (ball.c, paddle.c, walls.c) and system control.

A custom real-time operating system manages thread scheduling and periodic events for smooth gameplay. The game visuals are displayed on an TI BOOSTXL-EDUMKII using the BoosterPack interface, and input is captured via the onboard joystick and buttons.

Development and testing were performed on physical hardware using the Tiva-C LaunchPad with a BoosterPack LCD display attached.

Hardware Used

- **TM4C123GH6PM Tiva-C LaunchPad(2x)**: Core microcontroller used for running the game logic and RTOS.
- **TI BOOSTXL-EDUMKII BoosterPack (2x)**: Used to display the game screen, including the paddle, ball, and walls.
- **Joystick (onboard)**: Provides analog input for player paddle movement.
- **User Buttons (onboard)**: Used to trigger ball spawning via shared LED state.
- **Onboard RGB LED**: Monitored for state changes to enable multiplayer synchronization.
- **USB Cable**: For programming and debugging via the ICDI interface.

High-level / Pseudo Code

// Initialize Custom RTOS Resources

CREATE Periodic Event Thread Game_Updater (30Hz)

CREATE Dummy Foreground Threads for RTOS setup

```
// Initialize Hardware
SETUP Tiva C (Player 1 or 2)
SETUP BoosterPack LCD Display
SETUP Joystick for Paddle Control
SETUP Buttons for Ball Spawn Trigger
SETUP RGB LED (used for multiplayer ball sync)

// Define RTOS Tasks and Periodic Event
TASK Game_Updater()
LOOP FOREVER
  CALL Paddle_Update()
  CALL Ball_Update()
END LOOP
END TASK

FUNCTION Paddle_Update()
  READ joystick X value
  MAP X to paddle position
  ERASE previous paddle
  DRAW new paddle
END FUNCTION

TASK CommThread()
  INIT prevLevel ← false
  INIT riseCount ← 0
  LOOP FOREVER
```

```
    WAIT on CommSema   // OS_Wait(&CommSema)
    READ level ← Comm_CheckReceived()
    LED_Set(level)
    IF prevLevel == false AND level == true THEN
      INCREMENT riseCount
      IF riseCount > 1 THEN
        CALL Ball_SpawnNew()
      END IF
    END IF
    SET prevLevel ← level
  END LOOP
END TASK

TASK CommSignalThread()
  LOOP FOREVER
    SIGNAL CommSema   // OS_Signal(&CommSema)
    RETURN  // Must return immediately (RTOS rule for periodic threads)
  END LOOP
END TASK

FUNCTION Ball_Update()
  FOR each active ball
    ERASE old ball position
    UPDATE ball position using dx/dy
    IF ball hits wall
      BOUNCE ball
```

```
      IF ball hits paddle
        BOUNCE ball upward
      IF ball missed
        RESET ball to center
      DRAW ball at new position
      IF previous position overlapped right wall
        REDRAW right wall
    END FOR
    IF LED state transitioned from OFF → ON
      IF less than max ball count
        SPAWN new ball on both boards
    END IF
END FUNCTION

// Main Entry Point
MAIN()
    DISABLE interrupts
    INIT clocks, LCD, joystick, buttons, and LED
    INIT paddle, ball, and wall rendering
    CALL OS_Init()
    CALL OS_AddThreads(CommThread, Dummy2, ...)  // Required for OS
    CALL OS_AddPeriodicEventThread(Game_Updater, 33)  // 30 Hz
    CALL OS_AddPeriodicEventThread(CommSignalThread, 33)  // 30 Hz

    CALL OS_Launch(10000)  // Start RTOS
END MAIN
```

RTOS Routines

The game utilizes a custom cooperative real-time operating system (RTOS) to manage task scheduling, synchronization, and periodic execution. The following RTOS features were used:

- **OS_Init()**
  Initializes RTOS structures, prepares thread management, and sets up system tick timing.
- **OS_AddThreads()**
  Adds a set of dummy foreground threads and communication signal semaphore thread (CommThread) to satisfy the RTOS's scheduler requirements. In this project, all game logic is handled in a periodic thread.
- **OS_AddPeriodicEventThread()**
  Registers a periodic event thread (Game_Updater and CommSignalThread) that runs at 30 Hz. This thread performs all game logic, including paddle movement, ball physics, and collision detection.
- **OS_Launch()**
  Starts the RTOS scheduler with a defined time slice (e.g., 1ms). Once launched, the periodic thread begins execution.
- **Semaphores and Mailboxes**
  Multiplayer functionality was achieved via hardware LED state monitoring and use of semaphores.

This RTOS-based approach provides consistent timing for gameplay updates, simplifies synchronization, and ensures responsive input handling using low-overhead scheduling.

How the Project Will Be Demonstrated (Interactions of UI, Sensors, Screen, LED, Sound)

The final project will be demonstrated on two Tiva-C LaunchPads, each running an identical copy of the game. Each board features a BoosterPack LCD for displaying the game screen and uses the onboard joystick for paddle movement.

During gameplay, the following interactions will be shown:

- **Joystick Control:** Players move their paddle left and right using the onboard joystick, with paddle position updates reflected in real-time on the LCD screen.

- **Ball Physics and Collisions:** The ball moves continuously across the screen, bouncing off walls and the paddle. If the ball is missed, it resets to the center.
- **LED-Based Synchronization:** A button press causes the onboard LED to transition from OFF to ON. Both microcontrollers monitor this state change, and if detected, a new ball is spawned on each board simultaneously. This enables multiplayer interaction without direct communication.
- **Visual Output:** The LCD displays all game elements, including the paddle, ball(s), and boundary walls. Up to four balls can be active on-screen.

The demonstration will highlight smooth gameplay, real-time responsiveness, and hardware-based synchronization between two independent systems.

Hardware and Software Development Environment Readiness Date
The hardware and software environment for this project was fully prepared and operational by April 10, 2025. This included:

- Keil µVision 5 configured and functional for compiling and flashing
- RTOS kernel initialized with periodic thread scheduling working as expected
- Paddle movement, ball physics, and rendering routines successfully tested
- Joystick input and LCD output integrated with responsive UI
- Multiplayer synchronization via LED state transitions verified on both Tiva-C boards