Hanoi University of Science and Technology
School of Information and Communication Technology

Project Report: Game Tetris using STM32CubeIDE hardware and software

IT4210E - Embedded Systems

Group Members:

Group 5 - Class: 161346
Luong Ngoc Vu Long - 20235967
Tran Sy Nguyen - 20235985
Nguyen Vu Anh Khoa - 20235957

Lecturers:

Prof. Ngô Lam Trung

January 26, 2026

Contents

1 Introduction

This project aims to replicate the classic arcade game "Tetris" on an embedded system platform. The primary goal is to demonstrate the integration of real-time operating systems (FreeRTOS), graphical user interfaces (TouchGFX), and hardware peripheral control (GPIO, Timers, Interrupts) on the STM32F429I-DISCO development board.

The system features a 240x320 pixel color display, a dedicated audio engine for background music and sound effects using PWM, and physical button controls for game interaction. The software architecture is designed using the Model-View-Presenter (MVP) pattern provided by TouchGFX, ensuring a clean separation between game logic and visual rendering.

2 Hardware Design

2.1 Development Board Specifications

The project utilizes the STM32F429I-DISCO kit, which is built around the STM32F429ZIT6 microcontroller. Key specifications utilized in this project include:

- **Core:** ARM Cortex-M4 with FPU, running at 168 MHz (configured via PLL).
- **Flash Memory:** 2 MB (Storing code and const assets like fonts/images).
- **RAM:** 256 KB Internal SRAM + 64 Mbits External SDRAM (used for Frame Buffers).
- **Display:** 2.4" QVGA TFT LCD via LTDC interface.

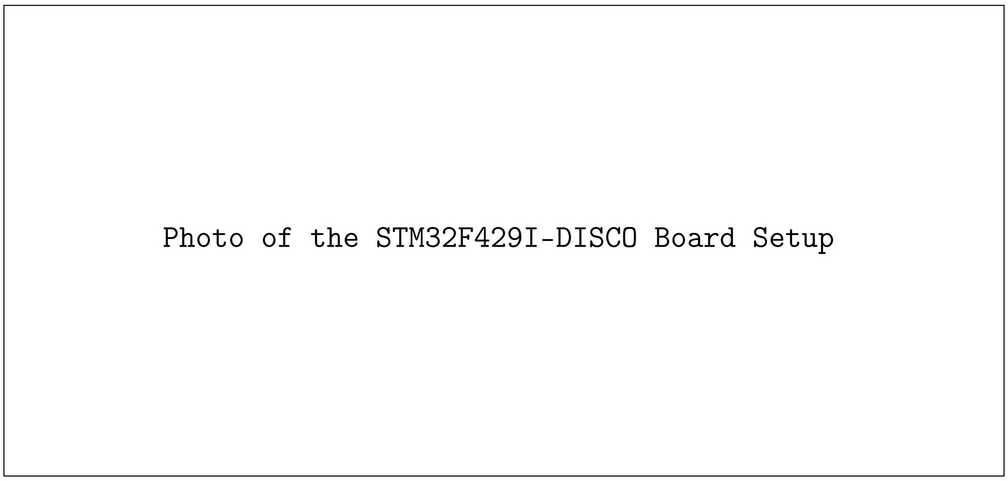

Figure 1: Photo of the STM32F429I-DISCO Board Setup

2.2 Peripheral Configuration

The hardware configuration is generated using STM32CubeMX (inside STM32CubeIDE).

2.2.1 Display Subsystem (LTDC & DMA2D)

The Liquid Crystal Display (LCD) is driven by the LTDC (LCD-TFT Display Controller) peripheral.

- **Interface:** RGB565 (16-bit color depth).
- **Resolution:** 240 x 320 pixels.
- **Memory:** Two frame buffers are allocated in the external SDRAM to support double buffering, preventing screen tearing.
- **DMA2D:** Chrom-Art Accelerator is enabled to handle image copying and blending efficiently, offloading the CPU.

2.2.2 Audio Subsystem (PWM)

Sound generation is achieved using Pulse Width Modulation (PWM) connected to a Piezo Buzzer.

- **Timer:** TIM10.
- **Channel:** Channel 1.
- **Prescaler:** 167 (resulting in a 1 MHz timer clock).
- **Logic:** The frequency of the PWM signal controls the musical pitch (Note), and the duty cycle controls the volume.

2.2.3 Input Controls (GPIO & EXTI)

The game is controlled via four external push buttons connected to GPIO pins configured as external interrupts (EXTI).

Function	**Pin**	**Port**	**Mode**
Rotate Piece	Pin 12	GPIOB	EXTI Rising/Falling
Move Right	Pin 13	GPIOB	EXTI Rising/Falling
Soft Drop	Pin 2	GPIOG	EXTI Rising/Falling
Move Left	Pin 3	GPIOG	EXTI Rising/Falling

Table 1: GPIO Pin Mapping for Game Controls

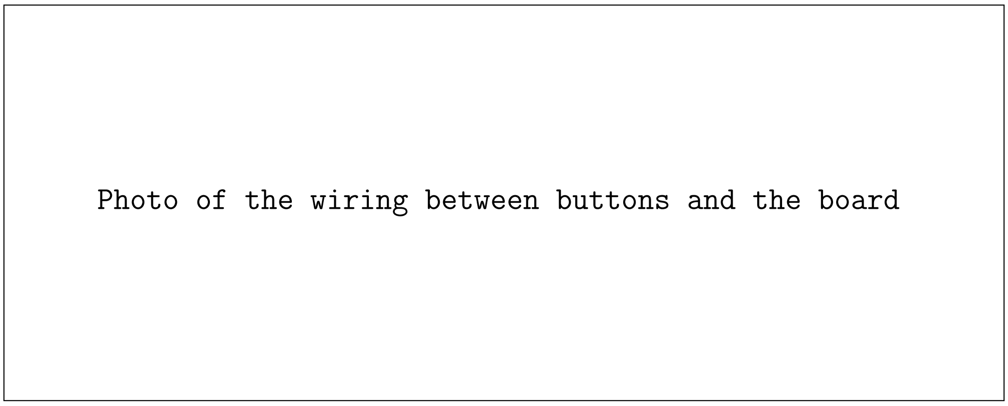

Figure 2: Photo of the wiring between buttons and the board

3 Software Design

3.1 Model-View-Presenter (MVP) Architecture

The software architecture is centered around the Model-View-Presenter (MVP) pattern provided by the TouchGFX framework. This pattern ensures a strict separation of concerns between the visual interface, the underlying game logic, and the synchronization layer that connects them.

- **View:** Responsible for the visual representation and layout.
- **Model:** Contains the core game state, rules, and mathematical logic.
- **Presenter:** Acts as the mediator, handling events from the View and updating the Model, or reflecting Model changes back to the View.

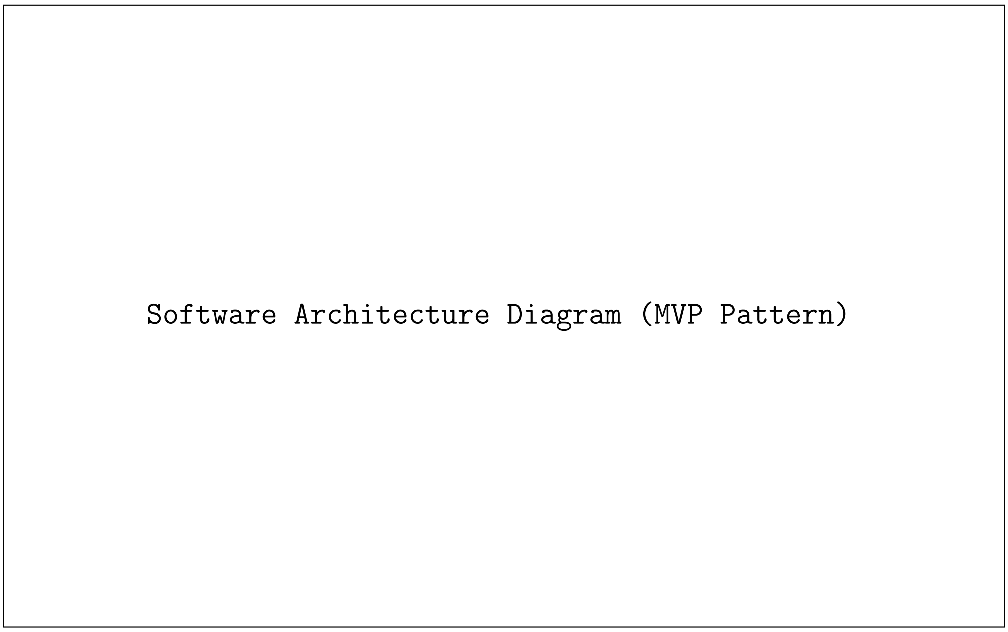

Figure 3: Software Architecture Diagram (MVP Pattern)

3.2 The View: User Interface

The View layer is implemented using C++ classes generated by TouchGFX. It focuses on rendering the graphical assets and handling the screen layout for the 240x320 portrait display.

3.2.1 Main Menu

Displays the "Tetris Art" logo and options to start the game. The background plays a slower, ambient menu theme.

`Screenshot of the Main Menu Screen`

Figure 4: Screenshot of the Main Menu Screen

3.2.2 Game Screen

The layout is optimized for the portrait screen, dividing the interface into functional zones:

- Center: 10x20 Game Matrix showing active and resting Tetrominos.
- Left Sidebar: HOLD piece slot and Level indicator.
- Right Sidebar: NEXT piece preview and Score display.

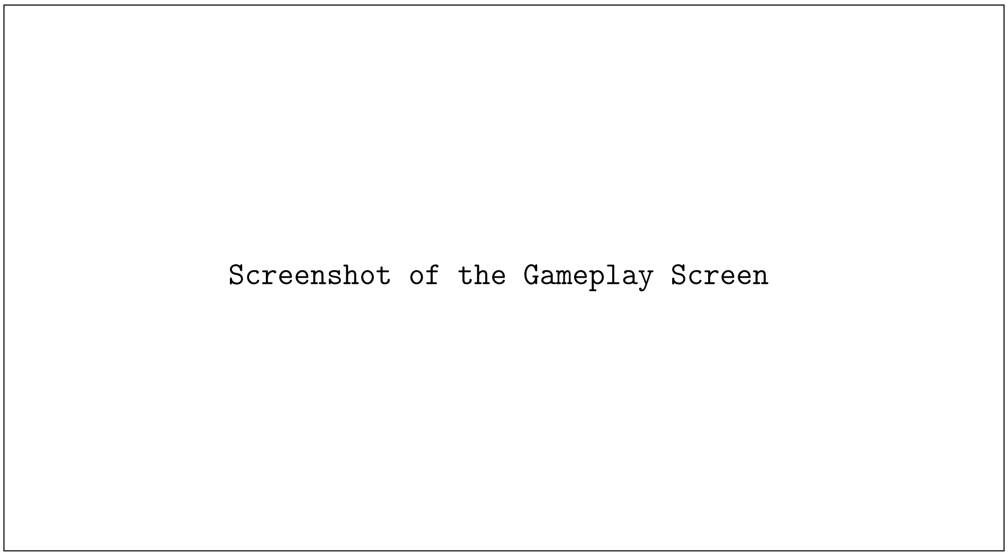

Figure 5: Screenshot of the Gameplay Screen

3.3 The Model: Game Logic

The Model layer maintains the internal state of the Tetris grid and implements the rules of the game. It is independent of the GUI framework, allowing for easier testing and logic updates.

3.3.1 Grid Representation

The 10x20 game board is represented as a 2D array: ‘signed char grid[20][10]‘, where each cell stores a value corresponding to a Tetromino type or remains -1 if empty.

3.3.2 Game Loop (‘tick‘)

The ‘tick()‘ function is the entry point for game updates, called every frame (60Hz). It handles gravity, collision detection via ‘isCollision()‘, and line clearing logic.

```
void Model::tick()
{
    if (isGameOver || isPaused) return;

    tickCounter++;
    if (tickCounter >= dropSpeed)
    {
        tickCounter = 0;
        step(); // Gravity drop
    }
    // ... Input queue processing ...
}
```

3.4 The Presenter: Logic-UI Synchronization

The Presenter layer serves as the bridge. Classes like ‘GameViewPresenter‘ listen for notifications from the Model (via ‘ModelListener‘) and trigger updates in the View.

When the Model state changes (e.g., a piece moves), the Presenter is notified and calls the corresponding View methods to refresh the display, ensuring the visual layer stays perfectly synchronized with the logical state.

```
// Example from GameViewPresenter.cpp
void GameViewPresenter::modelStateChanged()
{
    view.updateBoard(); // Signal the View to redraw based on new Model
     data
}
```

3.5 System Integration (RTOS & Audio)

The application logic sits atop a real-time foundation that manages hardware resources and asynchronous background tasks.

3.5.1 FreeRTOS Task Management

Three primary tasks ensure smooth operation:

- **GUI_Task:** Executes the TouchGFX engine and MVP logic (High Priority).
- **DefaultTask:** Handles button debouncing and system monitoring.
- **SoundTask:** Processes audio requests in the background (Low Priority).

3.5.2 Audio Engine

To maintain performance, audio is handled via a Producer-Consumer pattern. The Game Model (Producer) sends 'TrackID' requests to a queue, which are processed by the 'SoundTask' (Consumer) to drive the PWM-based buzzer.

4 System Integration & Challenges

4.1 Input Debouncing

Directly reading GPIO pins caused erratic behavior due to mechanical switch bounce. We implemented software debouncing using FreeRTOS Timers.

- When a button press interrupt occurs, a timer starts (e.g., 50ms).
- The input is only registered if the signal remains stable until the timer callback executes.

5 Results and Conclusion

5.1 Project Outcomes

The project successfully delivers a playable Tetris clone on the STM32F429I-DISCO.

- Performance: The game runs at a stable 60 FPS.
- Audio: Background music plays smoothly without interrupting game logic.

- Gameplay: All standard Tetris rules (rotation, wall kicks, line clears, leveling) are implemented.

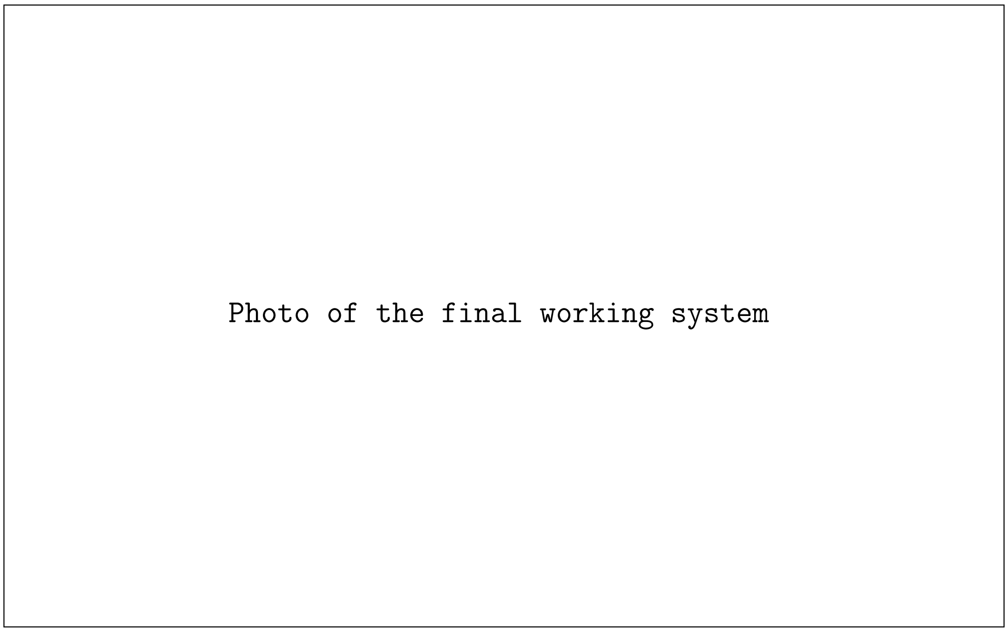

Figure 6: Photo of the final working system

5.2 Future Improvements

- Implement "Ghost Piece" visualization (indicated in design but currently simplified).
- Save high scores to internal Flash memory to persist after power loss.
- Add support for using the on-board Gyroscope (SPI5) for experimental tilt controls.