Design Document:Tetris Game Based on DE1-SoC

Xinzi Yu(xy2590), Chuyi Jiang(cj2792)

March 29, 2024

1 Introduction

Our project in Embedded Systems Design aims to recreate the iconic Tetris game by developing both the hardware and software components necessary to run it on the DE1-SoC FPGA platform. Tetris, a game with origins in the Soviet Union, features seven distinct shapes known as Tetrominoes, each made up of four blocks. These Tetrominoes drop into a play area, and players have the ability to change their orientation through some form of user interaction. Completing a full horizontal line of Tetromino blocks removes that line, causes the blocks above to drop down, and awards points to the player. As players successfully clear more lines, they advance to higher levels, which increases the speed of the falling Tetrominoes. The game concludes when there's no longer any room for a new Tetromino to enter and descend.
To bring this classic game to life, our project will incorporate the use of a DE1-SoC FPGA board, along with a joystick for control, a buzzer for audio feedback, and a VGA monitor for visual output.

2 System Block Diagram

The overview of our design is depicted in Figure 1, which consists of three modules - Peripherals, Hardware, and Software.

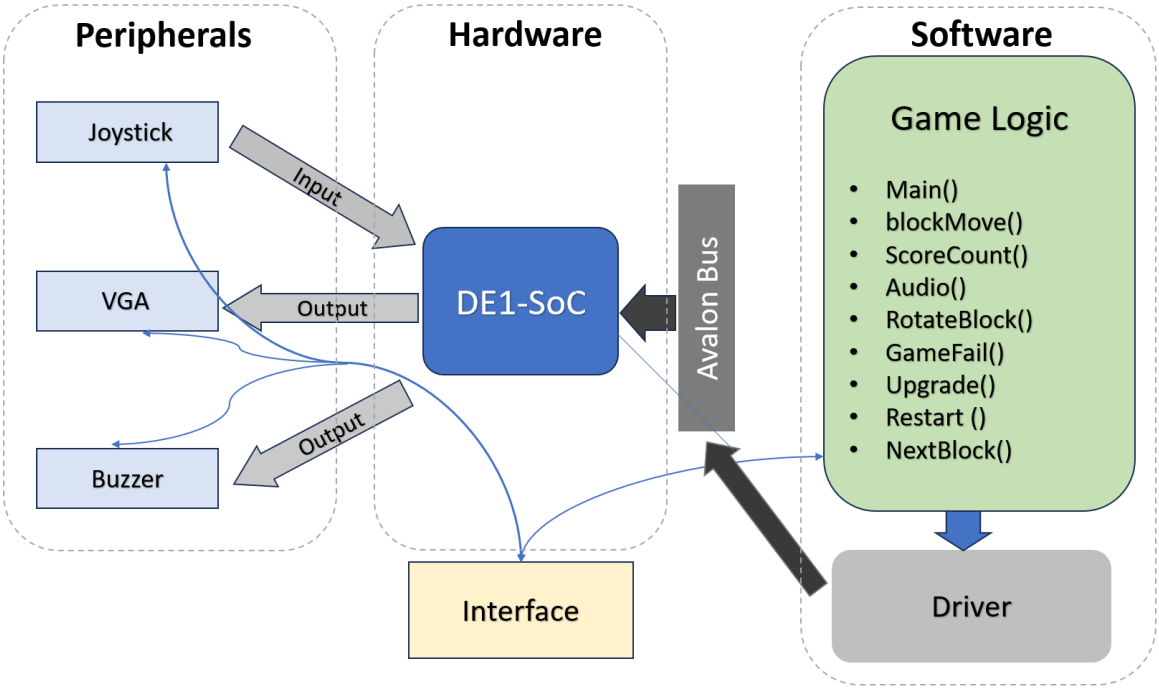

Figure 1: System Block Diagram

3 Algorithms

3.1 Finite State Machine Control Module

The control module operates using a finite state machine, and the State Transition Diagram is displayed in Figure 2.

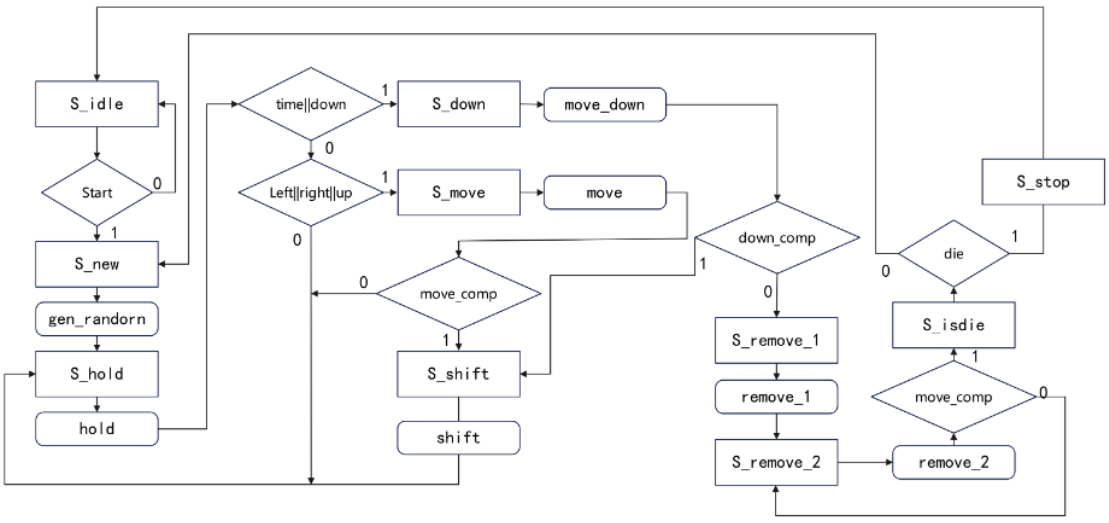

Figure 2: Finite State Machine

The definition of ten states are described as follows.

S_idle: Idle state, meaning the screen display is blank.

S_new: Used to generate a new Tetris block.

S_hold: Hold state. In this state, it times, and when the time reaches a certain interval, it transitions to the falling state (S_down); or it waits for input signals (up, down, left, right), then it transitions to the falling state (S_down) or to the (up, left, right) states.

S_down: Determines if the current Tetris block can move down one grid. If possible, it transitions to the update state; if not, it moves to the update state.

S_move: Determines if the current Tetris block can move according to the instructions specified by the key signal. If possible, it transitions to the update state (S_shift); if not, it moves to the first removal state (S_remove_1).

S_shift: Updates the coordinate information of the Tetris block. Returns to the hold state (S_hold).

S_remove_1: Updates the entire screen's matrix information. Moves to the second removal state (S_remove_2).

S_remove_2: Determines if rows can be cleared, clears the rows that can be cleared, and moves the rows above down one row. This process is repeated until there are no more rows that can be cleared. It then jumps to the game over state (S_isdie).

S_isdie: Determines if the game is over. If it is, it transitions to the stop state (S_stop). If not, it transitions to the Tetris block generation state, generating a new Tetris block.

S_stop: Clears the entire screen and jumps to the game over state (S_isdie).

3.2 Actions of Blocks

Blocks are categorized into inactive and active blocks. Inactive blocks are: (1) blocks that have previously fallen; (2) the result after a block has fallen and been cleared. These are represented by the background matrix. Active blocks are the blocks currently falling, represented by the coordinates and type of the active block.

3.2.1 Block Model

Tetris has seven different shapes of blocks, each with at least one and at most four different ways of rotating. For convenience, both the blocks and their rotation types are numbered, resulting in a total of nineteen blocks. The types of blocks are shown in the block display list in Figure 3.

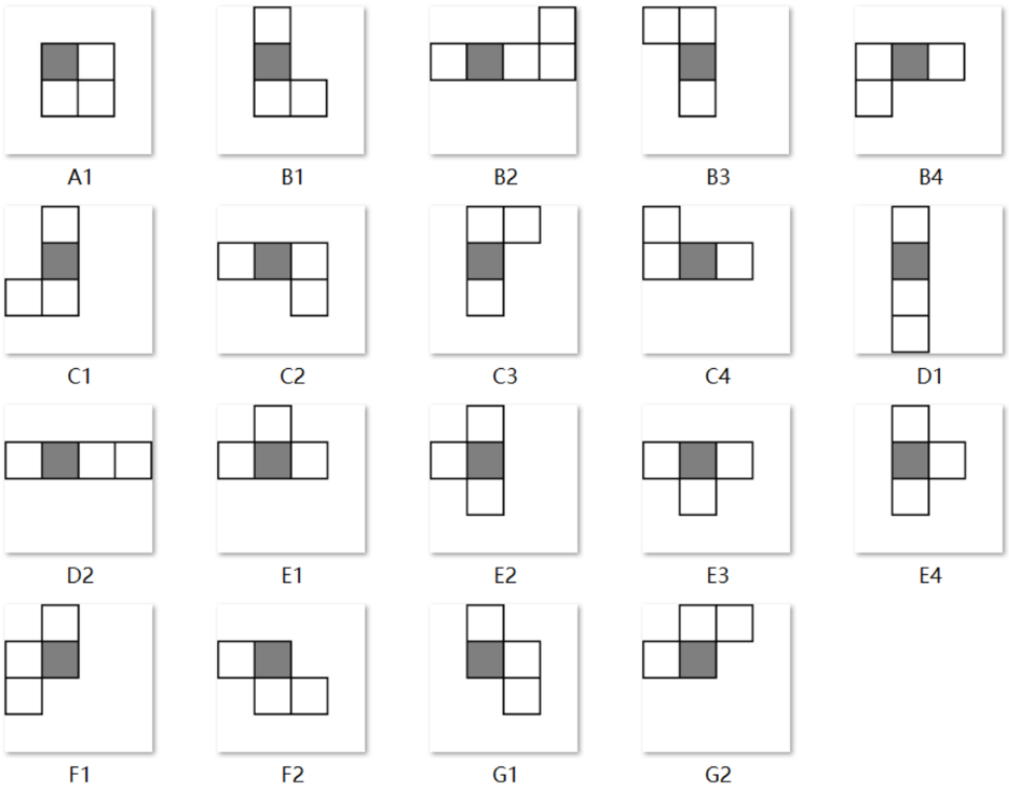

Figure 3: Block Model List

3.2.2 Block Coordinates

`output reg [4:0] n, output reg [3:0] m,`

where `n` and `m` are the row and column pointers for the current active block, pointing to the block's anchor point location. Since we want to rotate the block using the W key, the block's anchor point is the grid point that remains unchanged during rotation, determined by the type of block. Different types have different rotation points.

3.2.3 Block Move

The movement of the block is divided into four types: rotation, falling, moving left, and moving right, controlled by the keyboard. The movement of the block is mainly divided into two parts: (1) judgment; (2) transformation.

The judgment consists of two steps: First, determine whether the transformed block coordinates are legal, that is, whether the transformation will cause the block to go out of bounds. Then, determine

whether there are any background matrix blocks existing in the new position that the transformed block may occupy. If both steps of judgment pass, a success signal is returned; otherwise, it fails.
The transformation process involves moving or reshaping the block. According to the KEYBOARD, when moving, the block coordinates are changed; when reshaping, the block transforms according to its category.

3.2.4 Block Delete

Block elimination is achieved through two states: `remove1` and `remove2`.In the `remove1` state, the position of the active block is copied to `R`, turning it into an inactive block. In the `remove2` state, rows are eliminated and shifted based on their filled status.

3.3 Graphics

3.3.1 Background Matrix

The background matrix, named "Background," is a 20-row by 10-column array of type `reg`, responsible for storing the coordinates of inactive blocks, creating a 20*10 background matrix. When a position in the background matrix is 1, the screen display is white; otherwise, it is black.

3.3.2 VGA Module

VGA supports displaying 16 colors or 256 shades of gray at a higher resolution of 640x480, and can display 256 colors simultaneously at a resolution of 320x240. The human eye is far more sensitive to color than to resolution, so images remain vivid and clear even at lower resolutions. The display image information generated digitally inside the computer is converted by the digital/analog converter in the graphics card into RGB (Red, Green, Blue) primary color signals and horizontal and vertical sync signals. These signals are transmitted to the display device through a cable.

Here, we first divide the system's inherent 100MHz clock, since the display parameters are 640*480 at 60Hz, so we approximate it to 25MHz. To ensure that each point of the image on the transmitter side corresponds correctly to each point on the receiver side, the scanning of both the transmitter and receiver must be synchronized. Sync pulses are stable in period with steep edges. According to the Chinese television standard, the frequency of the horizontal sync pulse equals the line frequency of 15.625KHz, with a line period of 64us. In television technology, 64us is often used as a time unit, represented by H, that is, 1H = 64us. The field sync pulse frequency equals the field frequency of 50Hz, with a field period of 20ms, or 312.5H. The width of the horizontal sync pulse is about 4.7us, and the width of the field sync pulse is between 2.5 to 3Hz.

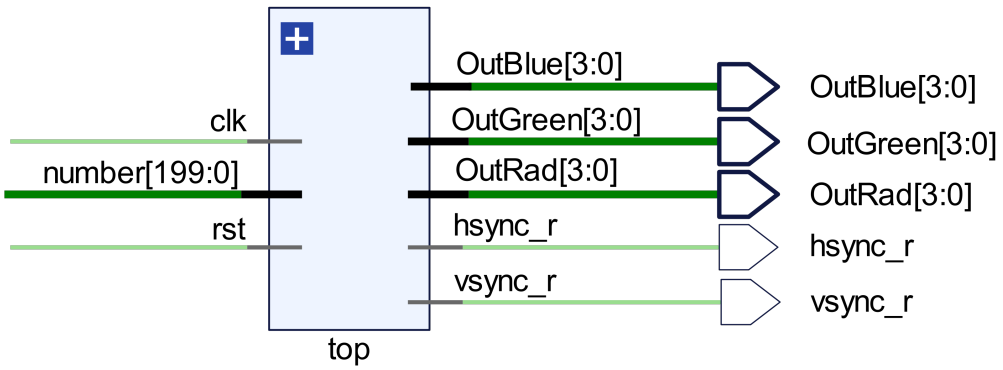

Figure 4: VGA Module

3.4 Audio

Music consists of pitches and durations. By precisely controlling the pitch and duration and accurately driving a buzzer, melodious music can be played. This module generates different music through the

buzzer, controlling the buzzer's frequency according to the pitch of the musical notes in the sheet music.

In digital circuits, the frequency of the overflow signal of a modulo-N counter is 1/N of the count pulse signal frequency. Therefore, a modulo-controllable counter can be designed to achieve different division ratios, allowing the overflow signal frequency to meet different pitch requirements. The division ratios corresponding to the pitches are shown in Table. Since the signal frequencies of the same note name in three octaves differ by a factor of two, the pitch can be raised (or lowered) by an octave without changing the division coefficient, by doubling (or halving) the count pulse frequency, which in turn doubles (or halves) the frequency of the divider's overflow signal.

Table 1: Frequency Division Ratio(FDR) corresponding to pitch

Note	FDR*	Preset	Note	FDR	Preset	Note	FDR	Preset
Bass1	11468	4916	Middle1	5736	10647	High1	2867	13516
Bass2	10222	6168	Middle2	5111	11272	High2	2555	13829
Bass3	9102	7281	Middle3	4552	11272	High3	2276	14108
Bass4	8592	7791	Middle4	4296	12087	High4	2148	11535
Bass5	7653	8730	Middle5	3827	12556	High5	1913	14470
Bass6	6818	9565	Middle6	3409	12974	High6	1704	14678
Bass7	6073	10310	Middle7	3036	13346	High7	1518	14864

The duration of a note is determined by how long the signal of that pitch frequency lasts. A counter can be used for timing the duration, where the count pulse period is the rhythm value from the sheet music, determining the duration of that note unit. Thus, timing for the duration can be implemented with a counter, where the count pulse period is the value of the shortest note in the selected sheet music. The timing for other notes' durations can continue for different count pulse periods based on the multiple relationships between their values and the shortest note duration (unit of measure). For example, if the shortest note in the sheet music has an eighth-note duration, then a quarter note lasts for two clock periods, while a whole note lasts for eight clock periods.

4 Resource Budget

The on-chip memory will be utilized for storing the game state, including the representation of the Tetris playfield, the current and next tetrominoes, scores, and game level.

1.Playfield Storage:
The playfield for a standard Tetris game is 10 columns by 20 rows. Each cell in the playfield needs to store information about whether it is occupied and by which type of tetromino block.Assuming a minimalistic approach where each cell is represented by a single bit (occupied/unoccupied), the entire playfield would require 200 bits. Considering the need to identify tetromino types, utilizing 4 bits per cell to encode up to 16 different states (including empty) would be more practical, resulting in a total of 800 bits (100 bytes).

2.Current and Next Tetromino Storage:
Information about the current and next tetrominoes includes the type of tetromino, its orientation, and position.Given that there are 7 types of tetrominoes and each can have up to 4 orientations, 3 bits for type and 2 bits for orientation are sufficient. Position can be encoded in 5 bits for X (column) and 5 bits for Y (row), totaling 15 bits per tetromino. Thus, storing current and next tetrominoes requires 30 bits.

3.Score and Level Storage:
The score and the game level are integral parts of the game's UI.Assuming a maximum score that fits within a 16-bit register and a level that can be represented by an 8-bit register, this sums up to 24 bits.

4.Miscellaneous:

Additional storage might be needed for game state (e.g., game over flag, line completion flags).32 bits to cover various flags and small state variables.

Total On-Chip Memory Budget

Table 2: Memory Requirement Summary

Component	**Memory Requirement**
Playfield Storage	100 bytes
Current and Next Tetromino Storage	∼4 bytes
Score and Level Storage	∼3 bytes
Miscellaneous	∼4 bytes
Total Estimated Memory Requirement	∼111 bytes

5 Hardware and Software interface

This section delineates the communication protocol between the FPGA's hardware components and the software layer responsible for game logic, input handling, and audio-visual feedback. Our Tetris game leverages the DE1-SoC board's capabilities for a responsive and visually engaging experience.

5.1 Dynamic Block Rendering Interface

1. Dynamic Block Buffer: A designated memory space dynamically renders falling and stationary tetrominoes, allowing for real-time game display updates.

2. Memory Allocation: Dual-buffered scheme with alternating rendering to minimize display flicker.

3. Address Space: Two blocks of 256 bytes each, dynamically allocated; addresses dynamically assigned based on rendering needs.

4. Data Format: 8-bit values representing color and state (active/inactive) for each block.

5.2 Game Mechanics Control Interface

1. Input Command Queue: Hardware registers queue input commands from the joystick, enabling responsive control over game mechanics such as block movement and rotation.

2. Address: 30 for input queue head, 31 for queue tail.

3. Data Format: 8-bit entries encoding direction and action commands.

4. Game State Management: Dedicated registers hold the game's current state, level, and speed, facilitating adjustments based on gameplay progression.

5. Addresses: 32 for game state, 33 for level, 34 for speed.

6. Data Format: Game state and level as 8-bit integers, speed as a 16-bit integer to accommodate fine-grained control.

5.3 Score Management Interface

Score and High Score Registers: Separate registers manage the current score and high score, allowing for easy updates and retrieval.
1. Addresses: 35 for current score, 36 for high score. 2. Data Format: Both as 32-bit integers to support a wide range of scores.

5.4 Audio Interface

5.4.1 Pitch and Duration Control

A division ratio (Frequency Division Ratio, FDR) is applied to the system's clock to achieve the desired note's frequency.
Address Allocation
1. Pitch Control Address: 39 - Modulo-N value for pitch frequency control.
2. Data Format: 16-bit int, representing the FDR values to achieve various pitches.

5.4.2 Duration Timing Control

A separate counter, timed to the shortest note duration in the chosen piece of music, scales to longer durations based on predefined multiples.
Address Allocation
1. Duration Control Address: 40 - Counter preset value for note duration.
2. Data Format: 16-bit int, indicating the count period correlating to note durations.

5.4.3 Audio Triggering

This module controls the triggering of specific sound effects based on game events, enhancing the game's interactive experience.
1. Trigger Address: 41 - Initiates playback of a selected sound effect or melody.
2. Selector Address: 42 - Specifies the sound effect or melody to be played, selected from a predefined set.
Data Format: 1. Trigger: 1-bit flag to start audio playback. **2. Selector:** 4-bit code to choose among various sound effects or melodies.