Columbia University

Embedded System Design

(CSEE 4840)

Tetris Game Emulator

Instructor: Prof. Stephen A. Edwards

Name	UNI	Contact Information
Zihao Wang	zw2668	zw2668@columbia.edu
Shibo Sheng	ss6372	ss6372@columbia.edu
Shengyue Guo	sg3882	sg3882@columbia.edu

Table of Contents

1 Introduction **3**

2 System Block Diagram **3**

3 Algorithms **4**

3.1 Graphics 4

3.2 Generating Blocks Randomly 4

3.3 Rotation 4

3.4 Audio 4

4 Resource Budgets **5**

4.1 Graphics 5

4.2 Audio 6

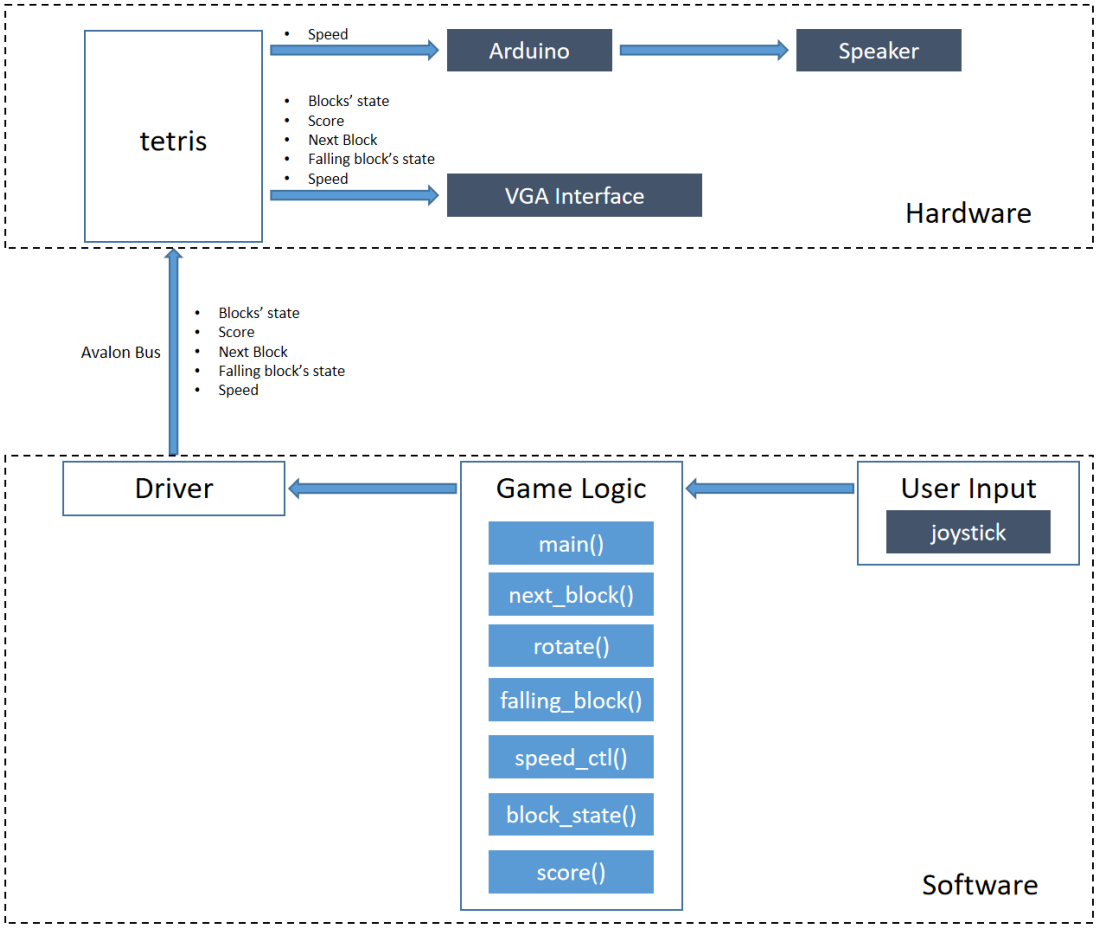
5 Hardware And Software Interface **6**

1 Introduction

Our project goal is to implement a Tetris game emulator. Tetris is a famous game all around the globe. In Tetris, players complete lines by moving differently shaped pieces (tetrominoes), which descend onto the playing field. The completed lines disappear and grant the player points, and the player can proceed to fill the vacated spaces. The game ends when the playing field is filled. The longer the player can delay this outcome, the higher their score will be. In multiplayer games, players must last longer than their opponents; in certain versions, players can inflict penalties on opponents by completing a significant number of lines. [1]

The project will attempt to emulate the Tetris game by using FPGA, SystemVerilog, and other necessary tech components as described below. A highlight of our version of the Tetris game is that the team would implement an audio response system that would give voice feedback based on the game events.

2 System Block Diagram



3 Algorithms

3.1 Graphics

We decided to use tile-and-sprite method to display our graph. We will take the background frame and words that keep unchanged during game play as tile while take blocks and numbers (speed, score) as sprites.

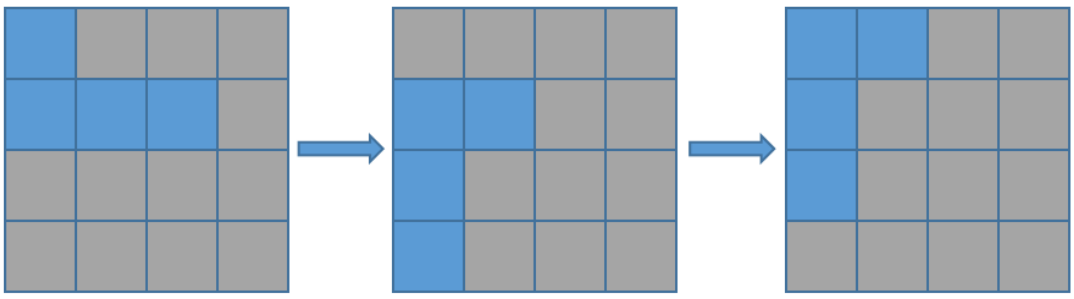
The location of blocks are provided by our software. The falling building block will fall at the speed set by the user. Once it touches the already placed blocks at bottom, it will stop falling. Once a row is filled, this row will disappear and the above blocks will fall down one row.

3.2 Generating Blocks Randomly

We have 7 kinds of building blocks. Our software will generate a random number after a building block has been placed and each number will be mapped to a kind of building blocks.

3.3 Rotation

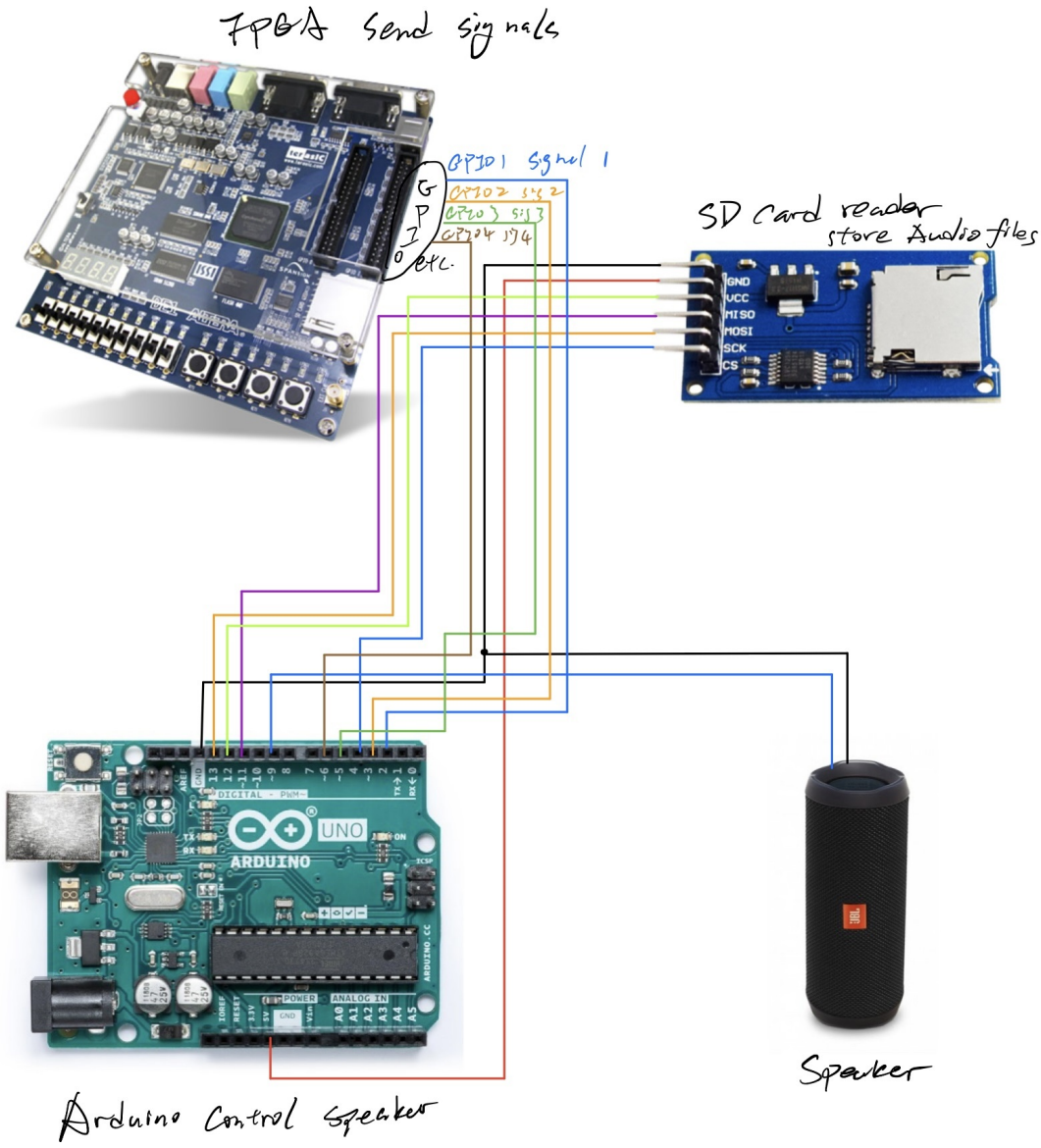
The rotation of the falling building block happens in a 4×4 matrix. This size of matrix can fit all kinds of building blocks. We always put the matrix to the top left corner of the matrix. That is to say, after rotation, we shift all the blocks to the top left corner. If there isn't enough space to do the rotation, our software will ignore the rotation request.



3.4 Audio

The team is expected to implement an audio response system onto our Tetris game. When the game starts, an audio would generate indicating that the game has started. When a block touches the base, a sound would be played. When players successfully erase a line of blocks, a different sound effect would be played. An audio response would also be played when the game is over. The team plans to implement this

function by using an extra Arduino Nano, a SD card reader, and a speaker. When the software detects the above event, the FPGA board would send signals to Arduino Nano using GPIOs. After Arduino Nano receives those signals, it would generate different sound effect based on each signal accordingly using the sound files stored in the SD card reader. The speaker would display these sound effect indicating different events.



4 Resource Budgets

4.1 Graphics

To save space, we decide to only use 256 colors.

Category	Size	# of images	Total size (bits)
Block	20×20	1	3200
Number	20×20	9	28800
Score	90×20	1	14400
Speed	90×20	1	14400
Total			60800

4.2 Audio

All of the audio files are stored in an extra SD card outside of the FPGA. The team is planning to purchase an extra 32 or 64 GB SD card.

5 Hardware And Software Interface

For the game module, we need memory cells for every block in a 20×12 grid, each one has one bit of 0 and 1 representing whether this block should be rendered as full or empty. It will be slow to communicate every cell in real time, thus, it can be more efficient to simply communicate about where the current falling object is and is it possible to place the object in the current place. Each time a communication happens, the game module should return whether it is possible for the current falling object to be located at the input position, which should be 1 bit message. A score recorder is also needed to record how many lines the user has successfully achieved, which could be 2 bits with maximal lines of 4 each time. In the last, we need to have registers for the current falling object about its position, rotation and type. The type needs 3 bits for 7 different types, the default bits 000 can be used to represent starting a new game. The rotation needs 2 bits for 4 different orientations. The position needed will be 4 bits for X axis ($12\times$) and 2 bits for Y axis to indicate the falling, where 0 represents horizontal shifts, 1 for normal falling and 2 for directly to the bottom, and always have the origin at the top left corner. That is, the input to this module will be a $3 + 4 + 1 + 2$ bits for the type, position and rotation of the current object, and the module will return with $1 + 2$ bits for whether the falling object can be in the input place, and the change of scores as additional erased lines.

The game module will also communicate to the VGA to render the current game status and the scores. The number of memory cells needed could be as big as 16 bits, for a maximal of 65535 score, which should be enough in most cases. The input from the manipulator will be processed within the top module, as well as the sound effect to transfer the returned information into different sounds.