CSEE 4840 Embedded Systems Design Project — Tetris

Zain Merchant (ztm2105)
Malik Hubbard (jmh2329)
Max Parson-Scherban (map2331)
Eva Gupta (eg3207)

Spring 2023

Contents

1 Introduction

Our Embedded Systems Design project is to develop the hardware and software for the classic game of Tetris on the DE1-SoC FPGA. Tetris is an originally Soviet-developed game where 7 4-block shapes, called Tetrominoes, fall into a playing field. The player controls the orientation of the Tetriminoes as they fall, through some mechanism of user input. Once a horizontal row of Tetromino blocks is complete, the line is deleted, higher blocks fall, and the user obtains a number of points for clearing the row. Upon a specified number of row clearings, the user "levels-up", and the rate of the descending Tetrominoes increases. The game ends when there is no additional space for a falling Tetromino to appear and descend. Our project will utilize the DE1-Soc FPGA, a USB NES controller, a Piezo speaker, and a VGA monitor to recreate this classic game in SystemVerilog and the C programming language.

2 System Block Diagram

Figure 1 contains our System Block Diagram. It is expanded upon in later sections of this report.

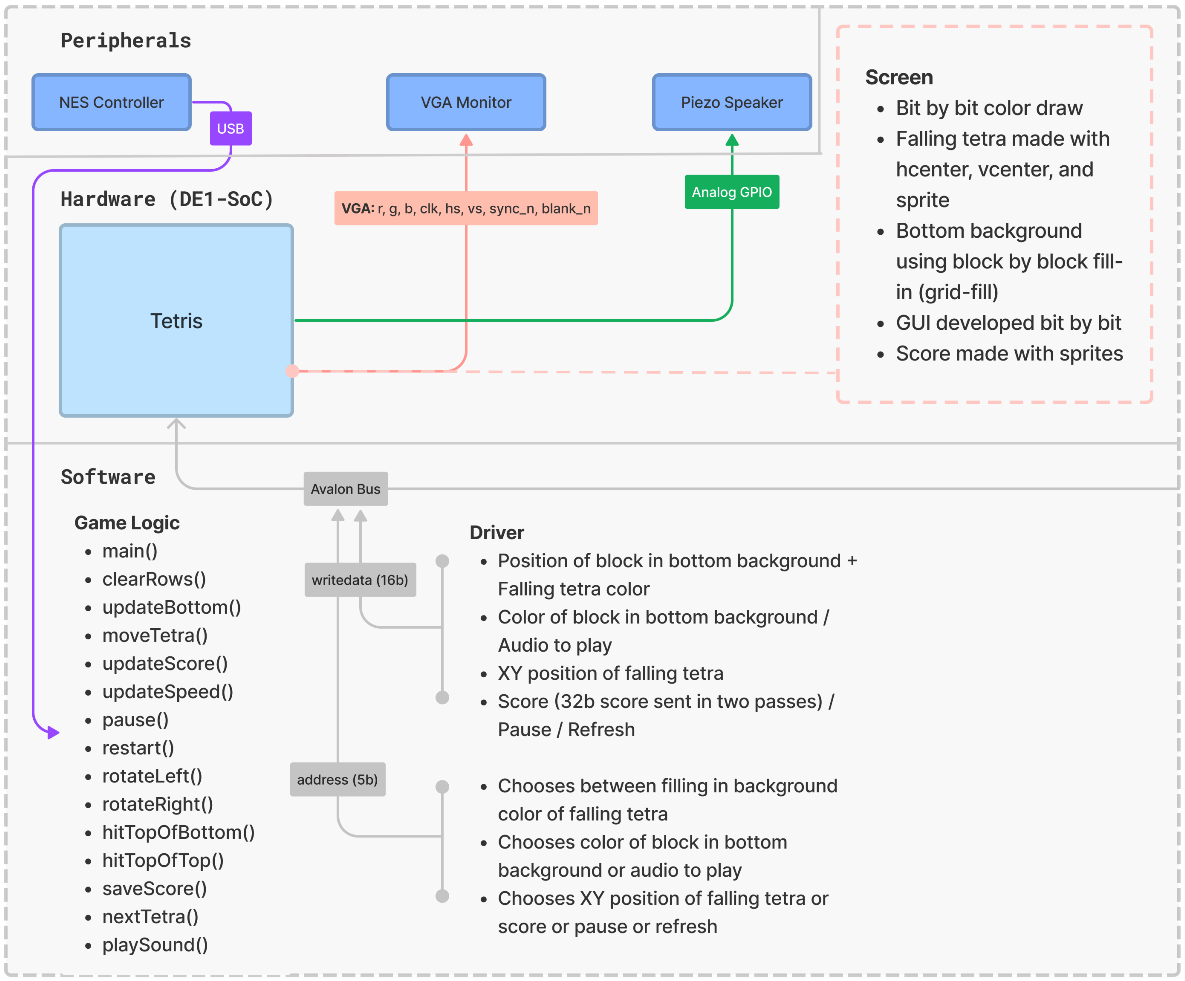
Peripherals
NES Controller
USB
VGA Monitor
Piezo Speaker
Screen
• Bit by bit color draw
• Falling tetra made with hcenter, vcenter, and sprite
• Bottom background using block by block fill-in (grid-fill)
• GUI developed bit by bit
• Score made with sprites
Hardware (DE1-SoC)
VGA: r, g, b, clk, hs, vs, sync_n, blank_n
Analog GPIO
Tetris
Software
Avalon Bus
Game Logic
• main()
• clearRows()
• updateBottom()
• moveTetra()
• updateScore()
• updateSpeed()
• pause()
• restart()
• rotateLeft()
• rotateRight()
• hitTopOfBottom()
• hitTopOfTop()
• saveScore()
• nextTetra()
• playSound()
writedata (16b)
address (5b)
Driver
• Position of block in bottom background + Falling tetra color
• Color of block in bottom background / Audio to play
• XY position of falling tetra
• Score (32b score sent in two passes) / Pause / Refresh
• Chooses between filling in background color of falling tetra
• Chooses color of block in bottom background or audio to play
• Chooses XY position of falling tetra or score or pause or refresh

Figure 1: This is the block diagram of our Tetris system. A USB NES controller provides the input to our software drivers, which are responsible for our overall game logic. The drivers communicate with our VGA and GPIO logic on the FPGA to drive game output to a display. This is discussed further in section 5.

3 Algorithms

3.1 Tetromino Rotation

We will use sprites to display each tetromino. A dual-ported memory unit will be utilized in hardware to save the sprites. Each sprite represents a tetromino and its orientation, thus for the 7 different tetrominoes each with 4 different orientations, there will be 28 different sprites to save. However, because the square tetromino is the same no matter how it is oriented and the 4x1 tetromino is the same for two of its states, we will actually only need to store 24 different sprites. When a falling tetromino is rotated we will simply change the sprite being shown on the VGA screen.

3.2 Bottom Background Addition

Our entire game space, a 10 by 20 block grid, will be represented in software as a two dimensional table. Each entry in the table is a block in the game space. Each entry will hold a numeric value that represents the color in that block. Every time the updateBottom function is called we will scan this array coloring each block in the game space the appropriate color based upon the array's value. Each block in the game space will be numbered from 0 to 255, which will correspond to the index of the 2D array. When a falling tetromino hits the top of the bottom background, based upon which sprite is being displayed the 2D array will be updated to color in the appropriate squares in the games space. The 2D array color values will then be pushed to the VGA display.

3.3 Block Fall/Move

The falling tetromino will exist independently of the bottom background. Similar to Lab 3, the center of each falling tetromino will exist in software and be communicated to the peripheral. Based upon the falling speed, the vertical center of the termino will be incrementing (moving down the screen). Similarly, when the appropriate keycode is received from the NES controller, the horizontal center of the block will be incremented or decremented appropriately while ensuring that it stays aligned with the columns of the game space.

3.4 Row Clear

Clearing rows will take advantage of the software 2D array describing the bottom background. When a row is completed, it will first flash by updating the colors in the corresponding blocks in the 2D array. These changes will be pushed to the VGA display quickly to show the flash. To clear a row, we will update the 2D array then push the update to the VGA display. The 2D array will be updated as follows for row clears: the row that is cleared will take all the colors of the row above it and each row above the cleared row will take the colors of the row above it. This will represent the blocks falling after a row has been cleared. Every row below the cleared row will remain unchanged.

3.5 Next Tetromino

The next tetromino will be created using a random number generator. This generator will develop both the shape of the tetromino and the starting orientation of it. Based upon this randomly generated number, the next tetromino will be clearly defined.

3.6 Audio Generator

Our game audio will be generated completely in hardware similar to the process used in Lab 5 and Lab 6 in the Digital Systems lab. Using a chain of clock dividers and a note decoder we will play music when the appropriate values are written to the hardware.

4 Resource Budgets

See Figure 2. for what the tetrominos and numbers will look like.

Category	Size(bits)	# of images	Total Size(bits)
Tetrominos	1600	23	883,200
Squares	400	7	67,200
Numbers	400	10	96,000
Total			1,046,400

We will be generating our audio waveforms on our fpga so our sounds will not consume memory.

5 Hardware and Software Interface

5.1 Register Descriptions

State of Unmoving Tetrominos

Background Row 0-A - 15 bit int that will describe the color (7 possible) of the rightmost 5 blocks (of 10) in row 0 (the bottom row).

Background Row 0-B - 15 bit int that will describe the color (7 possible) of the leftmost 5 blocks (of 10) in row 0 (the bottom row).

...

Background Row 20-A - 15 bit int that will describe the color (7 possible) of the rightmost 5 blocks (of 10) in row 20 (the top row).

Background Row 20-B - 15 bit int that will describe the color (7 possible) of the leftmost 5 blocks (of 10) in row 20 (the top row).

Background Next - 5 bit int that will select which tetromino sprite will be displayed as next

State of Falling Tetromino

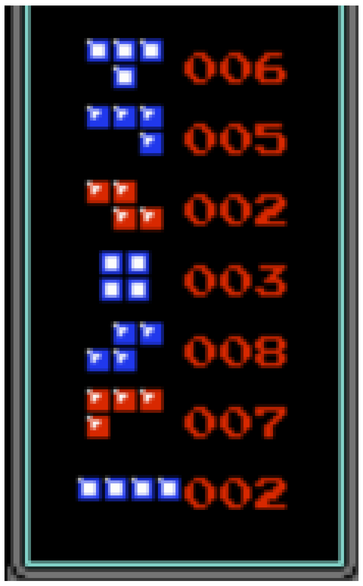

Figure 2:
These are the tetromino sprites we need for our game. The block color sprites are simply sprites of single squares of the various colors and textures in the tetrominos.

Falling Sprite - 5 bit int that will select the sprite for the falling tetromino.

Falling Vertical - 16 bit int that will set the vertical position for the falling tetromino.

Falling Horizontal - 4 bit int that will set the horizontal position for the falling tetromino. This is just selecting one of 10 columns.

State of Score

Current Score low - 16 bit int that will set the 1st through 4th decimal digits of the current score, the 0th digit of the score is always 0.

Current Score high - 16 bit int that will set the 5th through 9th decimal digits of the current score.

High Score low - 16 bit int that will set the 1st through 4th decimal digits of the current score, the 0th digit of the score is always 0.

High Score high - 16 bit int that will set the 5th through 9th decimal digits of the current score.

State of Music

Music Enable - 1 bit that will signal for the sound indicated by Music Sound to be played

Music Sound - 4 bits that will select which sound to play.

5.2 Game Logic Function Descriptions

clearRow - This function will be called whenever a 10 block row is filled in the bottom background. This row will flash blue and white for 500 ms before disappearing. Each row above the deleted row will drop down using the updateBackground() function.

updateBackground - This function will be called whenever a new tetromino hits the bottom background or whenever a row is cleared. When the bottom of a tetromino hits the top of the bottom background there will be a 250 ms pause where the user can rotate the tetromino. After this pause, the tetromino will join the bottom background by calling the updateBackground function. The updateBackground function will fill in the appropriate blocks in the bottom background grid with the same colors as the falling tetromino in its final position.

moveTetro - This function will be responsible for moving the falling tetromino left, right, and down. Each time the appropriate signal comes from the NES controller, the moveTetro() function will use ioctl to update the position of the falling tetromino. This function will also be called to make the tetromino fall down the screen, with the new position being updated according to the tetromino fall speed at that moment.

updateScore - Whenever a row is cleared, updateScore() will be called. The score variable will be increased and communicated to the peripheral which will then update the appropriate sprites showing the score on the VGA screen.

updateSpeed - As the game progresses, the blocks should fall faster. Thus, we will implement a 1 minute timer which whenever it ends will reset and increase the tetromino fall speed.

rotateleft - This function will handle the case when the user wants to rotate the tetromino left. We will be communicating the center of the tetromino to the peripheral and its type. When the user wants to rotate left we will first check in software if this is within the bounds of the screen. If it is, then the type of sprite (sprite type encapsulating the shape and orientation of the tetromino) shown will be updated. This will then be communicated to the hardware so that the falling block is now shown as rotated. rotateRight() will work similarly, just with a different sprite type being shown when the appropriate button is pushed on the NES controller.

hitTopOfBottom - This function is called when the bottom of the falling tetromino hits the top of the bottom background. This function will make a new tetromino appear at the top of the screen, generate a new next tetromino, and call updateBackground().

hitTopOfTop - This function is called when enough bricks have stacked up so that the top of the last placed tetromino is at the top of the game space. This means the player has lost. The score value will be automatically reset to zero, and the game will restart. If the player's score is higher than the last saved high score, the saveScore() function will be called.

nextTetro - This function will be responsible for generating the next tetromino that will fall down the screen. A random number generator from 1 to 28 will be used. Based upon what the number generator creates, the type of tetromino and its starting orientation will be determined. This tetromino will be supplied to the player when hitTopOfBottom() is called.

playSound - This function will play game sound whenever a row is cleared, a high score is beaten, a block is rotated, or a block hits the top of the bottom background. This function will write to the peripheral which will then generate the appropriate sound in hardware and send it to a piezo speaker.

saveScore - This function will be called whenever the game has ended and the current score is higher than the last saved high score. This function will update the high score to be the current score.