Tony Caldwell - tonjcald@vols.utk.edu

Grant Jennings - gjennin8@vols.utk.edu

Keaton Wyrick - kwyrick4@vols.utk.edu

ECE 351 Final Project Report

11/29/2025

Tetris Doppelgänger

Introduction

The purpose of this project was to create a clone of Tetris utilizing the Basys3 FPGA's capabilities to run game logic, video output, input handling, and audio output all with Verilog HDL. Unlike traditional software video game projects, this Tetris clone will need to operate with parallel modules rather than sequential processes. The hardware requires the game state to constantly be updated and track a score in order for the final win or loss state to be achieved. The project generates a VGA signal to drive a monitor, processes the user inputs via pushbuttons, left, right, and rotation logic, and maintains the correct game board by tracking the various states such as: piece spawning, gravity, locking, line clearing, game over, and score status via the seven segment display. Audio output was created from scratch as well using the AMP2 PMOD.

The primary objectives of this project were to: design a hardware implementation of Tetris with the aforementioned components, achieve a working prototype using the on board memory of the Basys3 FPGA, and to test functionality through simulations and testbenches with the FPGA and Verilog HDL.

Background

Prior to this project, we had wanted to produce something utilizing VGA, and

that could be interacted with. We had contemplated other games such as Pong, as well as something with internet connectivity. We decided on Tetris because it is slightly more complex than Pong, and one of our group members had worked on a Tetris project previously. Traditionally, Tetris is a fast-paced puzzle video game where differently shaped falling blocks (Tetrominoes) drop from the top of the screen. The player must rotate and move these pieces to form complete horizontal lines. When a line is filled, it is cleared and the player earns points. As the game progresses, the pieces fall faster, making it increasingly challenging to prevent the stack from reaching the top, which ends the game. Due to time constraints, memory constraints, and overall complexity, we decided to just develop a single, continuous game state. In doing this, we still have a game functionally mirroring Tetris, but not as complex as a fully complete and polished game.

Regarding related work, we found two online examples of Tetris made entirely on FPGAs. One used a Nexys3 board, and the other used System Verilog. We had hoped to pull some code base from these, but found it to be largely incompatible with our resources. However, they were a good basis to model logic from.

Design Details

The design can be broken down into its various components in the overall project design. Utilizing the on board seven segment display as many of the projects in this class required, seven segment display implementation was the most simple module to begin. The seven segment module increased the score based on the number of cleared lines in a single check. This allowed the user to score a traditional “tetris” event if they so wished. The score was converted into BCD and sent to the seven segment display. Only 2 anodes were required since

the win condition is 10 points. The digits were multiplexed by using a counter which provided an on board option for displaying the score rather than displaying the score on screen through VGA.

An audio module was created for interactivity with the AMP2 PMOD and a Bluetooth speaker (we just used an AUX cable to wire it in), and can be enabled via switch 1. It is effectively a tone generator and sequencer which was used to emulate the recognizable Tetris theme. It includes a PWM DAC, which effectively simulates high frequency output that can be understood by our speaker once the AMP2 amplifies the analog signal. The tone generator creates sawtooth waveforms for the audio tones, and the sequencer uses several hand-calculated note frequencies, pauses, and holds to sequence a hand-composed Tetris theme rendition. It only includes the first two verses of the theme. In addition, an enable state for the AMP2 is passed based on switch 1 and the game's win state. The audio is looped indefinitely until the enable is toggled or the game is won.

The game logic is a finite state machine which checks the board contents, which cells are empty or occupied, as well as the current grid positioning and shape of the tetromino. Additionally, it reviews the next piece, score, win conditions, line clears, etc. The FSM is what sequences through these operations. The board data is then stored in a 2D array and collision detection detects if a rotation or move is legal, when it cannot move, that piece is then locked into the board state and if it fills a row, the FSM triggers the line clear.

The VGA controller was based on lab 4. It generates the horizontal and vertical counter, sync signals, and video flags. Regarding the pixel locations, the color is decided, otherwise it is a 0 value. The

controller forces the resolution to be 640 x 480 at 60Hz.

The top level module is what ties all of the modules together. It instantiates the 100MHz clock, push buttons, switches, VGA output, seven segment display, and audio logic. It uses the same clock for all of these differing processes but divides the logic which allows for slower or faster signals to work with the game logic, for example the game pieces fall one tick per 50MHz, while maintaining the clock needed for line clearing, VGA implementation, and timing.

Results

Simulation Results

It was very difficult to simulate a project such as this one, due to the game functionality depending on VGA pixel output. We were able to get a testbench to simulate gravity ticks and track a manually spawned piece's y position, as well as a couple other states. The resulting console output is seen in Figure 1.

```
========================================
Mini Tetris Testbench
========================================

TEST 1: Reset
  PASS: Reset complete
  Initial: piece_active=0, score=0, win=0

TEST 2: Piece Spawn
  PASS: Piece spawned at y=1, x=5
  Type=1, Rotation=0

TEST 3: Automatic Falling
  Triggering drops and monitoring y position:
    Drop  1: y= 3, active=1
    Drop  2: y= 5, active=1
    Drop  3: y= 7, active=1
    Drop  4: y= 9, active=1
    Drop  5: y=11, active=1
    Drop  6: y=13, active=1
    Drop  7: y=13, active=0
  PASS: Piece fell and locked at y=13

TEST 4: Next Piece Spawn
  PASS: Next piece spawned
  Position: y=1, x=5, type=2
```

Figure 1 - Testbench console output for 4 tests.

Testbench waveforms were generated as well, which are largely not meaningful for simulating gameplay due to the restrictions in being able to visualize waveforms for a simulated VGA product. However, they do show a few key states such as hsync, vsync, audio out, and amp enable being high, as seen in the corresponding Figure 2.

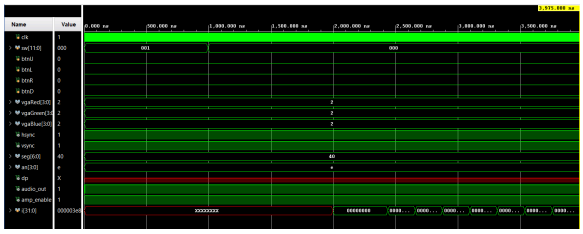

Figure 2 - Resulting testbench waveforms

Experimental Results

The game is output to VGA, and a piece will spawn at the top and begin falling down the screen. The left and right buttons control movement, the up button controls rotation, and the down button controls soft drop. The pieces are blue, and turn green upon collision. The amount of lines cleared is displayed on the 7 segment display, and a win condition is met upon reaching 10 lines cleared. A corresponding win screen is then output to VGA. If your pieces stack up to the top of the screen, the game resets. While the game is running, music is output to a speaker through the AMP2 PMOD. In our provided demo video, a full gameplay demonstration is presented. Below are Figures 3-6 displaying VGA output, as well as the lines cleared counter.

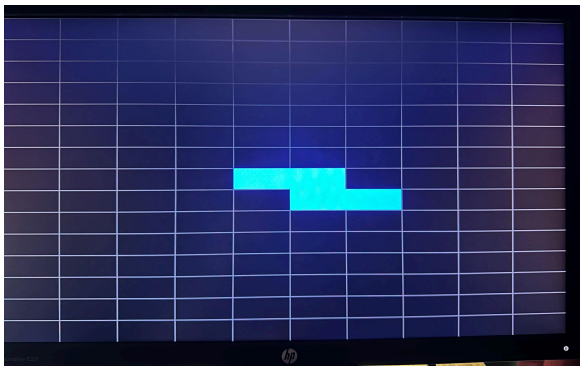

Figure 3 - Start of game, spawned piece falls

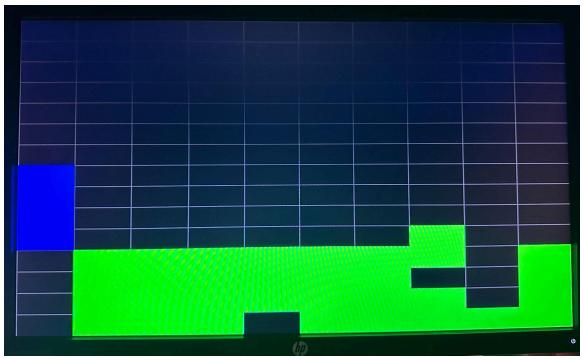

Figure 4 - Mid-game snapshot

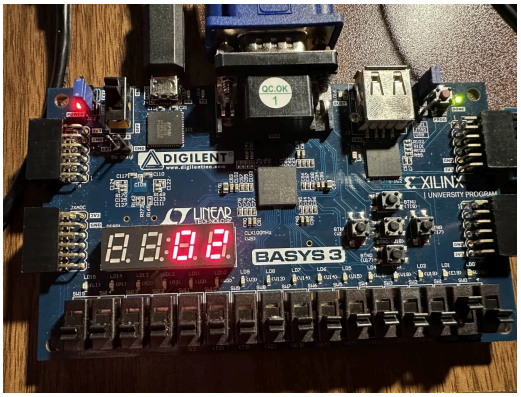

Figure 5 - Lines cleared output (2 in this case)

Figure 6 - Win screen

We also included reset functionality, which is tied to switch 0. Additionally, switch 1 toggles music output on and off.

Synthesis/Implementation

In Figure 7, the resulting synthesized schematic is displayed.

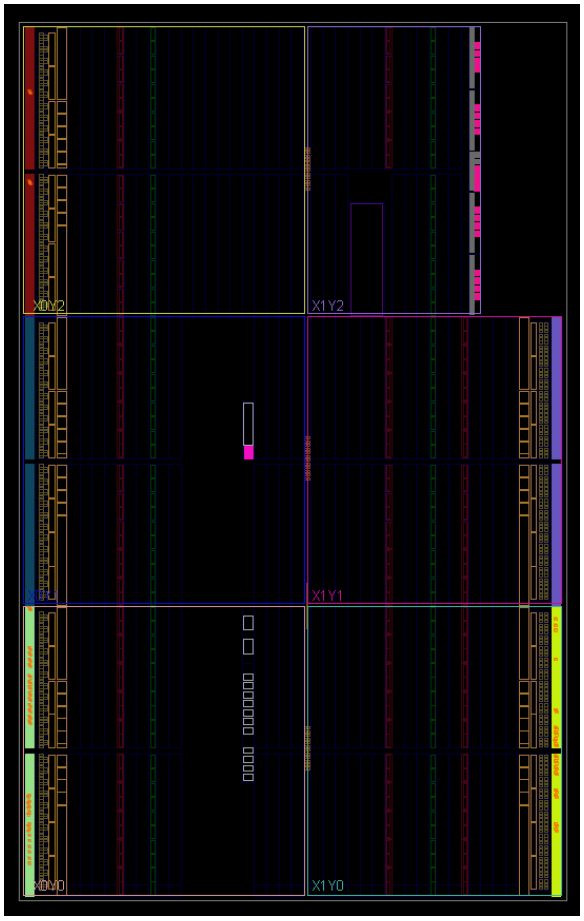

Figure 7 - Synthesized Schematic from top module

Vivado additionally provides utilization, power, and timing reports from the synthesis tab. Figures 8-10 below display the results of running utilization, power, and timing reports on our project file.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
mini_tetris_top	3201	372	107	3	0.5	35	2
music (mustop)	120	103	0	0	0.5	0	0
seg_unit (seven_seg_mux)	10	16	0	0	0	0	0
vga_unit (vga_beh)	115	23	6	0	0	0	0

Figure 8 - Utilization report for project files

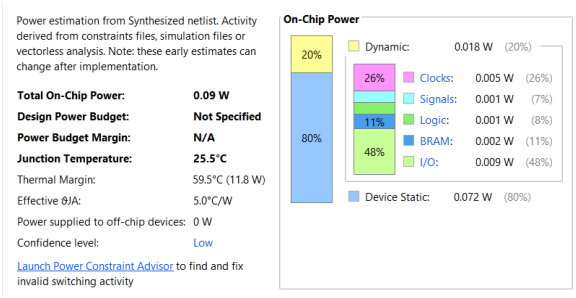

Figure 9 - Estimated power report for design

Figure 10 - Design timing report

The utilization report indicates that we only take up roughly 15% of the Basys3's Look-Up Tables, and only a small percentage of its flip-flops, 2-to-1 muxes and 4-to-1 muxes. It additionally only uses half of a memory block. If we were to revisit this project, we would likely have the resources available on the Basys3 to implement a more polished Tetris. Moving to the power report, only 20% of power usage is dynamic. Nearly half of this power is dedicated to input and output, which is logical due to our use of a PMOD and VGA in parallel. The total on-chip power sums to 0.09W, which is 80% static. Finally, regarding the timing report, it appears that signals during setup are arriving late, which applies to 150 endpoints. This negative slack means that the board is running too fast for some logic delays. However, nothing critical occurs when our bitstream is run on the Basys3, so it could be that these errors are corrected in subsequent clock cycles, or the chip is faster than these worst-case scenarios.

Discussion

Overall, the performance of our Tetris implementation was on-par with our expectations going into the project, especially considering the constraints of writing parallel code for sequential logic. The core elements of our game (VGA rendering, input handling, audio sequencing, and scoring logic) all functioned reliably and accurately.

One of the most difficult aspects of the development was creating a simulation that could simulate entire game states. We ended up having to just simulate gravity ticks, the active game piece, and the coordinate of the game piece. These pieces of information allowed us to determine if the game was having any major issues before generating the bitstream. Even though we were able to simulate these results, the majority of testing was done manually on the hardware.

Despite having multiple large processes running in parallel to compute the game states and audio, our utilization and timing reports report that our game was running with decent efficiency. Because of this, our design could allow for more expansion of the project in the future since the majority of the board's resources were unused. Expansions could include adding more game pieces, a more complex scoring system, and the full tetris game theme. While efficient, the negative slack shows that certain paths were operating close to the limit of the Basys board. Although we did not have any issues regarding the negative slack in live testing, this could cause issues if the game were expanded too much.

Our biggest success of the project was finally getting the audio to work through the PMOD. Since this required hand-calculating frequencies and multiple trials with terrible noises coming from the speaker, completing this made the project feel more like a game we were creating than a project assigned for school.

Conclusion

This paper presented the design and implementation of a general Tetris clone on the Xilinx Basys3 FPGA using Verilog HDL. The project integrated a VGA controller, FSM for the game and board state, rendering logic, input handling and debouncing, and audio implementation in

order to create a fully hardware based, real time video game. Simulations confirmed the correct board state, input response and stable game mechanics.

During this project we gained practical experience in designing and creating a complex digital system, working with real-time VGA output, and audio implementation. Future improvements would include adding all of the various tetraminos and their shapes as well as true 90 degree rotations in all pieces. We could also improve the audio to provide alternative themes or tones. We could explore alternative hardware architectures which allow for more PMOD use or board logic. Overall, this Tetris project serves as a practical exercise in digital logic design in Verilog and the Basys3 FPGA based design.

Appendix

- Link to Nexys3 Tetris github: https://github.com/lavingiasa/verilogTetris
- Link to System Verilog Tetris github: https://github.com/afifmoh6923/fpga-tetris-basys3
- **Link to our project github:** https://github.com/Insullica/FPGA-Tetris-Doppelganger
- **Link to our demonstration video (also included in project submission folder):** https://drive.google.com/drive/folders/1lUQdbeLD5JlcwKiz0GinJHmZnkl9iQeY?usp=sharing
- Link to Tetris piano scheme used for music composition: https://www.youtube.com/watch?v=nBd_h0hB6Xk
- Link to note sequence guide: https://www.piano-keyboard-guide.com/how-to-play-the-tetris-theme-song-easy-piano-tutorial-korobeiniki
- Link to AMP2 PMOD: https://digilent.com/shop/pmod-amp2-audio-amplifier/?srsltid=AfmBOooEFFLISorylIbaFqMk6eC14cHX0IgeS-4IWzg7r6T7mcEFcFUZ
- Link to our in-class presentation: https://docs.google.com/presentation/d/1HkG_pKeNzqaQa32GNvgZSf5JFSxRMykCcGjyrUjZxrs/edit?usp=sharing