# FPGA Tetris Game

Michael Lippe, Bhargav Sriram, Garvit Vyas

CSEE 4840: Embedded Systems

Prof. Stephen Edwards

May 13th, 2025

# Overview

- Our project aims to develop a hardware/software system capable of playing Tetris.
- Tetris is a classic puzzle video game revolving around the strategic placement of falling geometric shapes known as Tetrominos.
- The goal is to rotate and arrange these pieces in such a manner that forms complete horizontal lines, which are then cleared from the screen, and points are given based on the number of lines cleared.
- As the game goes on, the falling speed of the blocks increases, and thus so does the difficulty.

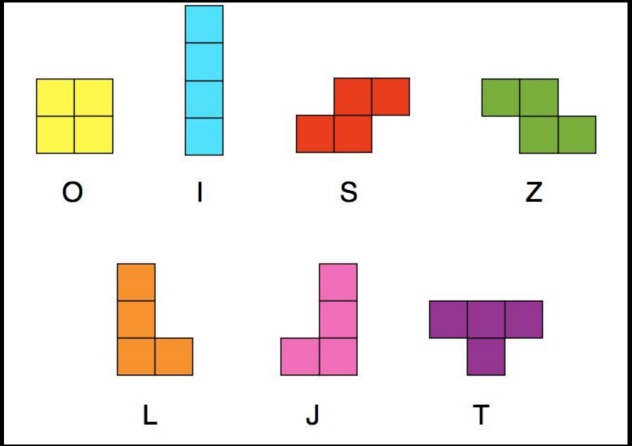


Figure 1: Tetris Pieces

# System Block Diagram

- After verifying the verilog modules for our original design, we realized that we made our system too complex and verifying everything together within a week and a half while also figuring out the compilation and software was not feasible
- As such, we decided to take a tile only approach as tetris does not need sprites
- We decided to pivot and base our design off the provided tile generator so we could focus on the hardware-software interactions and the software
- We modified the existing tile hardware to add a tile map cache to help resolve flickering issues. The cache pulls from the tile map at vblank and stay consistent throughout the frame

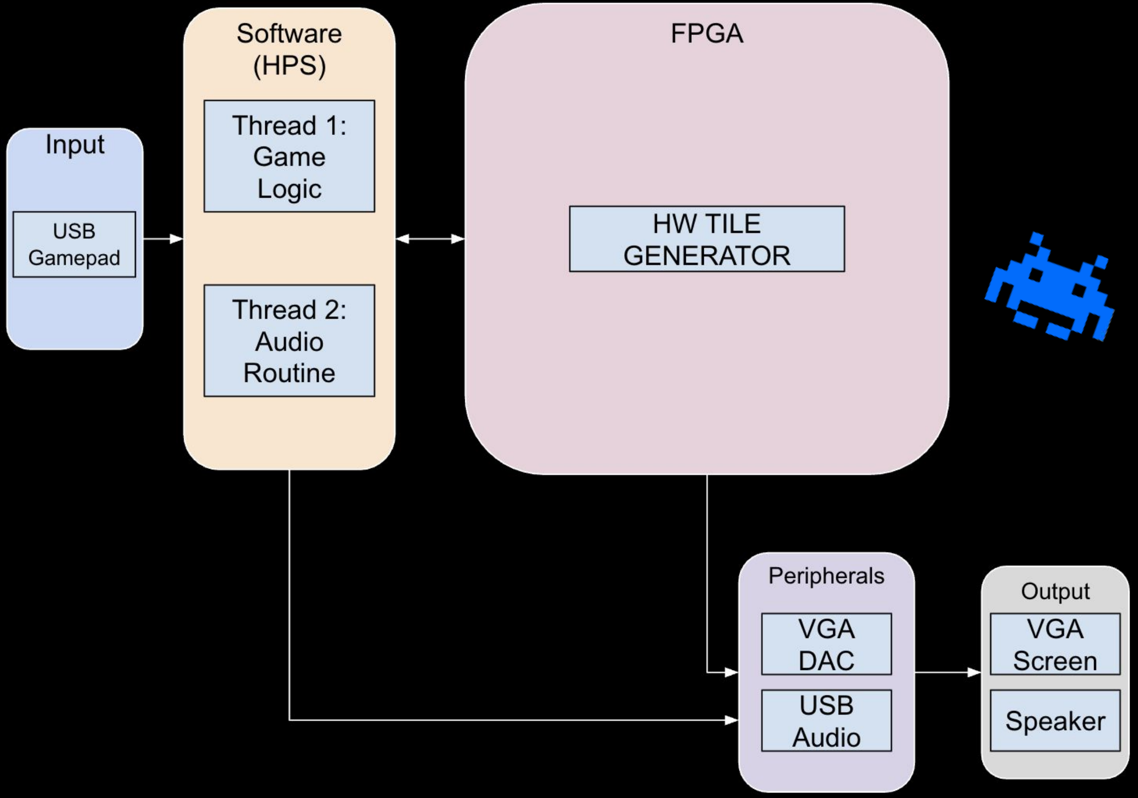


Figure 2: System Block Diagram

# Hardware Block Diagram

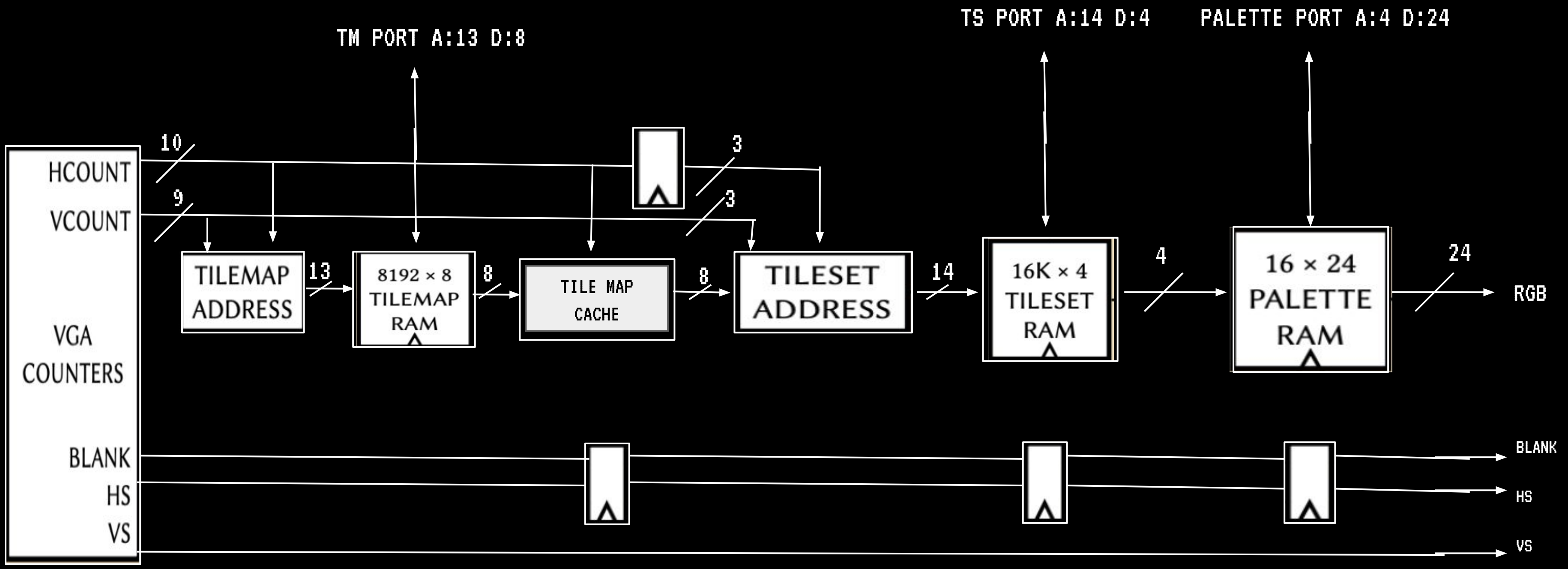


Figure 3: HW Block Diagram (Based on Professor Edwards' diagram [2])

- Added Tile Map Cache to help reduce tearing/flickering

# Register Mapping

- Table 1:

| REGION | POINTER | BASE ADDRESS | ADDRESS RANGE | SIZE | PURPOSE |
|---|---|---|---|---|---|
| TileMap | TM | 0xFF20_0000 | 0xFF20_0000-0xFF20_1FFF | 8 KiB | 8-bit indices for a 128X64 tile grid |
| Palette | PA | 0xFF20_2000 | 0xFF20_2000-0xFF20_203F | 48 B | 16 entries X 3 bytes (24-bit RGB colors) |
| TileSet | TS | 0xFF20_4000 | 0xFF20_4000-0xFF20_7FFF | 8 KiB | 4 bits-per-pixel graphics (256 tiles X 8X8) |

# Resource Utilization

| Fitter Summary | |
|---|---|
| Fitter Status | Successful - Sun May 11 13:31:59 2025 |
| Quartus Prime Version | 21.1.0 Build 842 10/21/2021 SJ Lite Edition |
| Revision Name | soc_system |
| Top-level Entity Name | soc_system_top |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 416 / 32,070 ( 1 % ) |
| Total registers | 622 |
| Total pins | 362 / 457 ( 79 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 196,992 / 4,065,280 ( 5 % ) |
| Total RAM Blocks | 26 / 397 ( 7 % ) |
| Total DSP Blocks | 0 / 87 ( 0 % ) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |
| Total PLLs | 1 / 6 ( 17 % ) |
| Total DLLs | 1 / 4 ( 25 % ) |

Figure 4: Resource Utilization

# USB Audio

- Tried two open source hardware audio implementations for the DE1-SoC and couldn't get either to work
- Decided to implement audio via USB
- Provided kernel did not have snd-usb-audio module
- Kernel: clone linux-socfpga v4.19 → make socfpga_defconfig
- Config: make menuconfig → enable "USB Audio (snd-usb-audio)" under Sound → USB
- Load modules using modprobe
- Libraries: ALSA + libmpg123 & libao installed
  - Use libmpg to decode mp3 and libao to play decoded audio over usb via ALSA

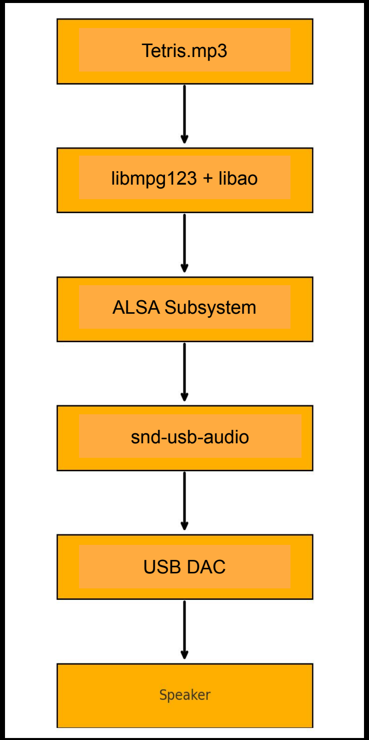


Figure 5: USB Audio Flow

# USB SNES-Style Controller

- Plug a standard SNES-style controller into the DE1-SoC's USB-host port. The FPGA is running a USB-HID stack.
- Button mapping:
  - L, R, or X: rotate piece
  - D-pad left: move the current piece left
  - D-pad right: move the current piece right
  - D-pad down, A, or Y: soft drop
  - B: instantly drop the piece to the bottom (hard-drop)
  - Start: start game from start screen or game over
  - Select: pause game


Figure 6: USB Controller

# Tileset

- 10 tiles used
- Tile 0 is for the background
- Tile 1 is for the playfield and next piece bounding boxes
- Tiles 2 - 7 are for constructing the Tetrominos
- Tile 13 is for the ghost piece and displaying text

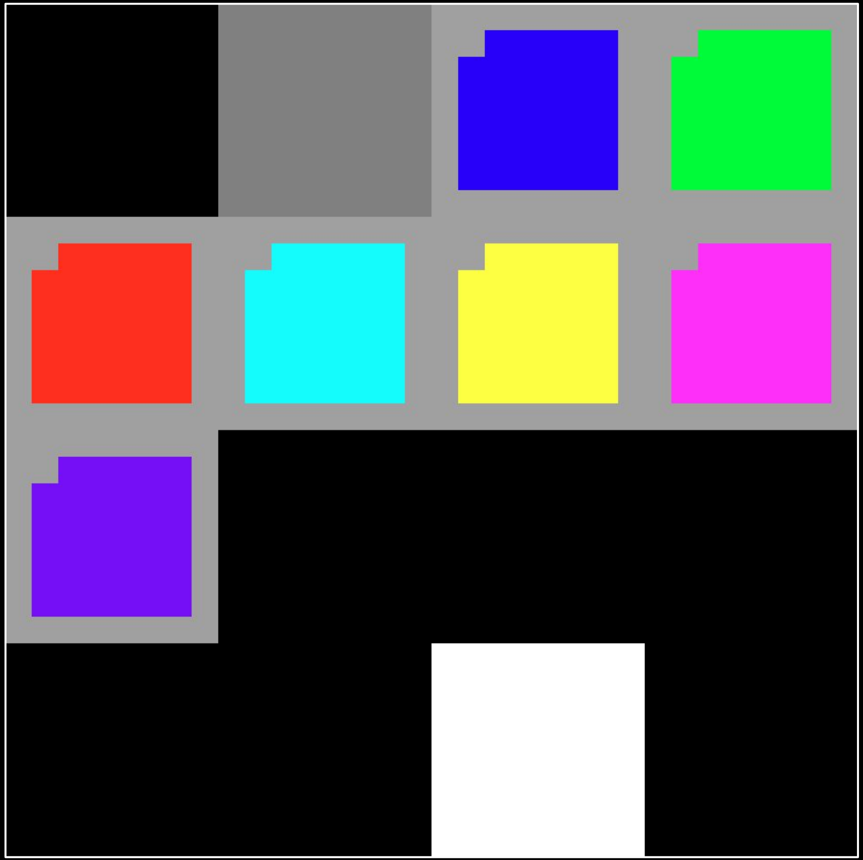

Figure 7: Tetris Tileset

# Start Screen

**FPGA & VRAM Init**

- map_fpga() → mmap TM/PA/TS regions (8 KB tilemap, 64 B palette, 16 KB tileset)
- load_assets() → upload 16-color palette + tileset, clear tilemap

**Render Title & Prompt**

- show_start() → memset(TM,0) → draw_string() at (10,20), (10,40), (10,50)

**Controller Detection**

- open_controller() scans /dev/input/event* for USB SNES pad → returns FD (open device file)

**Wait for START**

- Main loop state=START → poll_input() until EV_KEY code 297 (start button) → state = PLAY


Figure 8: Tetris Start Screen

# SW Game Logic

spawn()

- Position new piece at center - top (px=5, py=0), reset rotation
- If it collides immediately → set game-over

poll_input()

- Read controller events every loop
- Map D-pad to left/right/soft-drop, buttons to rotate/hard-drop/pause

Gravity Tick (step)

- Advance a frame counter
- On each level-dependent interval, attempt to move piece down

Collision Check

- No collision: piece falls one row → back to input
- Collision: piece locks in place

lock_piece()

- Merge tetromino into playfield array
- Trigger clear_lines()

clear_lines() → spawn()

- Remove any full rows, shift above rows down
- Update score, lines cleared, and level
- Call spawn() for next piece (or set game-over on failure)

Game-Over & Reset

- On game-over state: display "GAME OVER"
- Wait for Start button → reset playfield, score, level

# SRS Rotation

**Four Orientations:** Pieces cycle through 0°, 90°, 180°, and 270° states on each turn.

**CW-Only Input:** Every rotation increments the state by one:

$state = (state + 1) \bmod 4$ .

**Defined Pivot:** Each tetromino spins around its designated origin within a 4X4 block grid.

**Wall-Kick Trials:** If the raw rotation collides, SRS sequentially tests up to five translation offsets.

**JLSTZ Kick Table:** J, L, S, T, Z pieces use offsets (0,0), (-1,0), (-1,+1), (0,-2), (-1,-2)

**I-Piece Kicks:** The I tetromino's own sequence: (0,0), (-2,0), (+1,0), (-2,-1), (+1,+2)

**O-Piece Exception:** The O tetromino rotates in place with no kicks applied

**First-Valid Wins:** The first offset that resolves the collision is accepted; if none

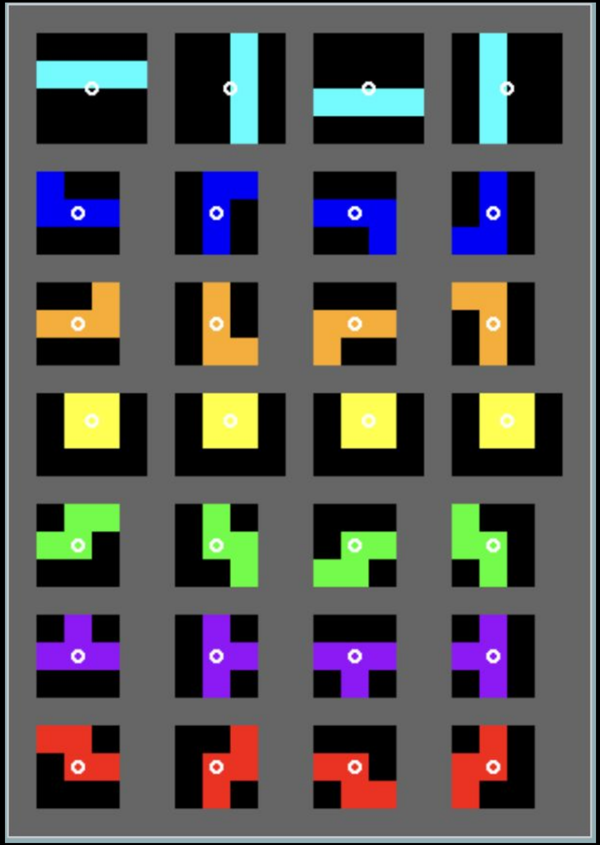

Figure 9: Tetris SRS Rotation [4]

# Gameplay Screen

Game Update: tetris.step() handles gravity, collision, rotation, line clears

Render Pipeline

- draw_borders() → static frame around playfield & next-box
- draw_playfield() → draw playfield and settled blocks
- draw_piece() → overlay active tetromino
- draw_next() → preview next piece in box
- draw_hud() → update SCORE & LINES via draw_string()

State Transition : if tetris.game_over() → state = OVER, call show_game_over()

Frame Timing

- usleep(16666) → ≈60 Hz update rate


Figure 10: Tetris Gameplay Screen

# End Screen

- Clear Playfield: memset(TM,0)
- Draw Text:
    - draw_string(10, 10, "GAME OVER")
    - draw_string(10, 20, score) & draw_string(10, 30, lines)
- Prompt Restart: draw_string(10, 40, "START: RESTART")
- Wait for Start


Figure 11: Tetris End Screen

# References Used


- [1] https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
- [2] https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.tar.gz
- [3] https://github.com/milon/Tetris
- [4] https://harddrop.com/wiki/SRS
- [5] https://github.com/altera-fpga/linux-socfpga/tree/v4.19
- [6] https://github.com/Ameba8195/Arduino/blob/master/hardware_v2/cores/arduino/font5x7.h

# DEMO: GAME START