

Project documentation

“ Lotus arcade”



[Project GitHub repository](#)

Project description*:

This project involves the implementation of Tetris on an FPGA (Field Programmable Gate Array) development board using Verilog programming language.

Tetris is a well-known puzzle game in which players must arrange falling geometric shapes—called **Tetrominoes**—within a vertical grid to form complete horizontal lines. The grid size will be **10 columns × 20 rows**, equivalent to a **100 × 200 pixel** area on the display. The Tetrominoes come in **5 distinct shapes**, each made up of four connected blocks.

Players can control the falling shapes using the arrow keys:

Left/Right Arrows – Move shape left or right

Up Arrow – Rotate shape

Down Arrow – Increase falling speed

When a row is completely filled, it is cleared, and the player earns points. The game ends when the Tetrominoes stack up to reach the top of the screen. Additionally, the falling speed increases over time, increasing difficulty and requiring faster reaction times.

The system will be built using finite state machines (FSMs) to manage game logic and a VGA controller to handle graphics rendering and user interface. The gameplay experience is designed to be engaging yet efficient in resource use, showcasing the FPGA’s real-time processing capabilities.

The system will utilize a VGA monitor for visual output and a keyboard for input.

***description cited from project proposal.**

Project stages:

Stage 0: Project proposal (done)

The proposal outlines the general purpose and scope of the project, while highlighting game logic, specifications and software models of the desired result.

Stage 1: Auto-falling game piece (done)

Stage description:

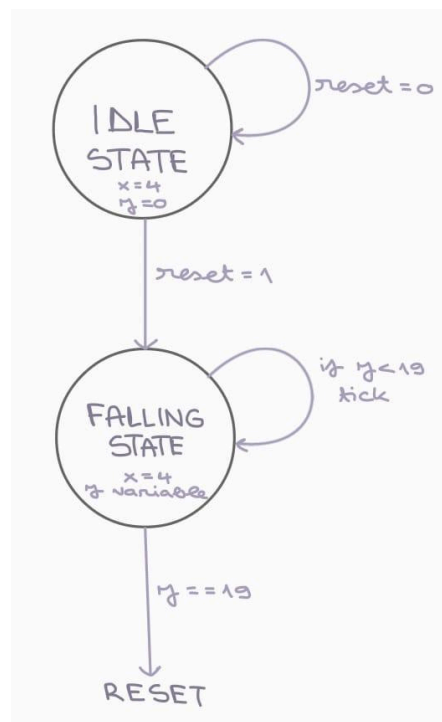
This stage focuses on designing and implementing an automated falling game piece (no user input) on a 10×20 grid. The current system updates the vertical position (y) of the piece periodically using a clock divider and stores the state of the grid.

Module descriptions:

top_module.v

Purpose:

The top module acts as the main controlling unit, coordinating all other modules and managing timing, current piece position and grid interaction.



Functionality:

- initializes the piece at a starting position ($x = 4, y = 0$);
- on each tick, writes the piece's position to the grid module and increments the vertical coordinate by one;
- resets the piece to the initial position (top of the grid) when it reaches the end ($y = 19$).

Inputs: clk (system clock), rst (active high-reset signal)

Outputs: current_x (current horizontal position of the piece- constant for now) current_y (current vertical position of the piece- for auto falling effect)

top_module -> top_module_tb (for simulation)

grid.v

Purpose:

Stores the state (occupied or empty) of each cell in the game grid.

Functionality:

Uses a 2D memory array to store cell values.

Inputs: clk (system clock), rst(reset signal), x (horizontal/column coordinate), y (vertical/row coordinate), state_of_the_cell_input (bit to write to the specified cell: 1 = filled, 0 = empty);

Output: state_of_the_cell_output (returns the value at position (x,y)).

tick_generator.v

Purpose:

This module generates a slower periodic signal (tick) from the system clock. This controls how often the piece falls.

Functionality:

- uses a counter to divide the frequency of the input clock down to a much slower one.
- resets and produces a tick when the counter reaches the set value N.

Inputs: clk (system clock), rst (active-high reset)

Outputs: tick (a single-cycle pulse every N clock cycles)

AFPL.v

Purpose:

The AFPL module expresses the game logic that controls the automatic downward movement of the piece.

Functionality:

- initializes game piece to the starting position (x = 4, y = 0) on reset;
- on every tick pulse, sets write_enable

high and cell_value to 1 to indicate piece's position;

- restarts to the initial position when piece reaches the bottom.

Inputs: clk (system clock), rst (active-high reset), tick

Outputs: x_out (current x coordinate), y_out (current y coordinate), write_enable (signal to the grid to write a new piece), cell_value (value to write into the grid, 1 = filled)

****background.v**

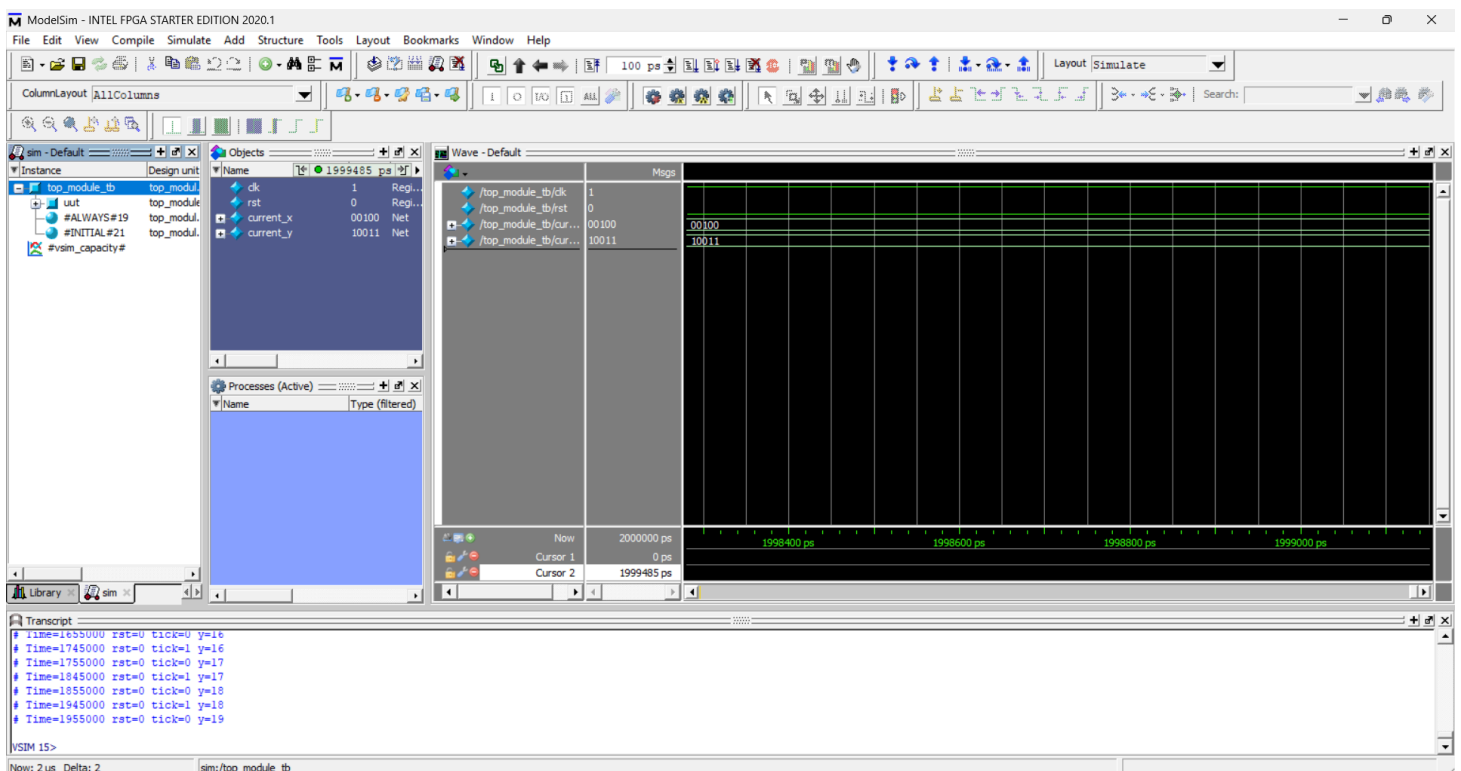
Purpose:

The background module is responsible for generating visual information for the VGA display. It determines what color should be displayed at each pixel location, based on game state (e.g. grid contents or piece positions).

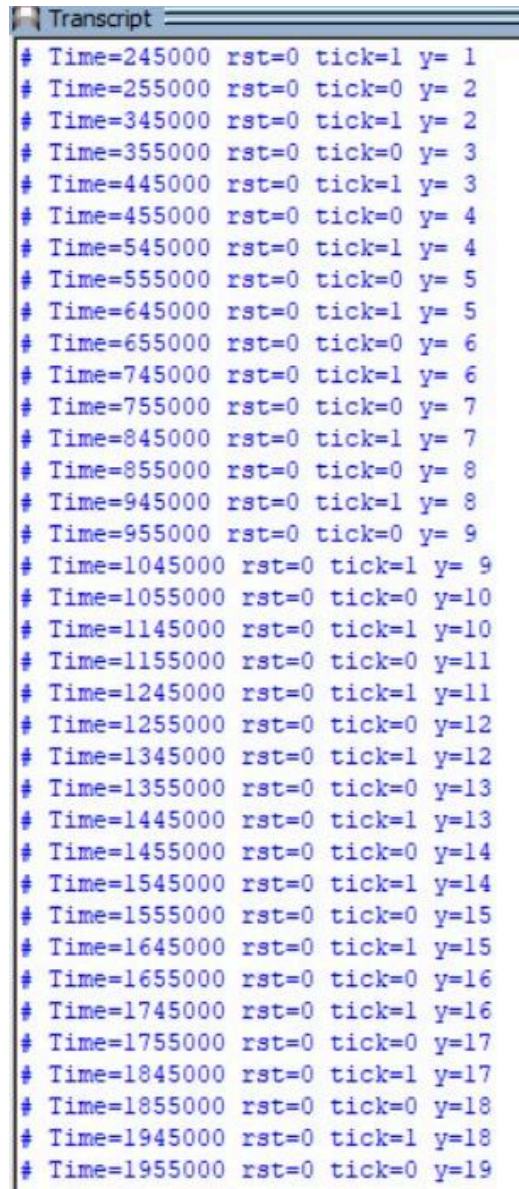
****not currently in use, done for paving the way for future design and VGA implementation.**

Simulation results:

Wave:



Display:



The screenshot shows a window titled "Transcript" with a list of 20 lines of text. Each line represents a game state update at a specific time, with fields for 'rst', 'tick', and 'y'.

```
# Time=245000 rst=0 tick=1 y= 1
# Time=255000 rst=0 tick=0 y= 2
# Time=345000 rst=0 tick=1 y= 2
# Time=355000 rst=0 tick=0 y= 3
# Time=445000 rst=0 tick=1 y= 3
# Time=455000 rst=0 tick=0 y= 4
# Time=545000 rst=0 tick=1 y= 4
# Time=555000 rst=0 tick=0 y= 5
# Time=645000 rst=0 tick=1 y= 5
# Time=655000 rst=0 tick=0 y= 6
# Time=745000 rst=0 tick=1 y= 6
# Time=755000 rst=0 tick=0 y= 7
# Time=845000 rst=0 tick=1 y= 7
# Time=855000 rst=0 tick=0 y= 8
# Time=945000 rst=0 tick=1 y= 8
# Time=955000 rst=0 tick=0 y= 9
# Time=1045000 rst=0 tick=1 y= 9
# Time=1055000 rst=0 tick=0 y=10
# Time=1145000 rst=0 tick=1 y=10
# Time=1155000 rst=0 tick=0 y=11
# Time=1245000 rst=0 tick=1 y=11
# Time=1255000 rst=0 tick=0 y=12
# Time=1345000 rst=0 tick=1 y=12
# Time=1355000 rst=0 tick=0 y=13
# Time=1445000 rst=0 tick=1 y=13
# Time=1455000 rst=0 tick=0 y=14
# Time=1545000 rst=0 tick=1 y=14
# Time=1555000 rst=0 tick=0 y=15
# Time=1645000 rst=0 tick=1 y=15
# Time=1655000 rst=0 tick=0 y=16
# Time=1745000 rst=0 tick=1 y=16
# Time=1755000 rst=0 tick=0 y=17
# Time=1845000 rst=0 tick=1 y=17
# Time=1855000 rst=0 tick=0 y=18
# Time=1945000 rst=0 tick=1 y=18
# Time=1955000 rst=0 tick=0 y=19
```

Stage 2: User input (work in progress)

Stage description:

This stage emphasizes on the advancement of the system, enabling user control for the falling piece. The user input is received through a PS/2 interface, using the PS/2 hardware communication protocol to interpret keyboard scan codes. The user can now interact with the game in real-time to move or rotate the piece.

Modules:

keyboard_decoder.v

ps2_interface.v

tetris_controller.v

References:

Other projects:

[MIT Tetris final project](#)

[GitHub repository on Verilog Tetris](#)

Websites:

<https://www.fpga4student.com/>

<https://nandland.com/>

<https://www.geeksforgeeks.org/>

<https://stackoverflow.co/>