Space Invaders on FPGA

CECS 361 - Ayleen Perez, Nicholai Agdeppa, Billy Domingo

Overview

Space Invaders is a 2D shooting game where the player controls a spaceship and its goal is to eliminate waves of alien invaders by shooting at them. The game ends when the alien passes the player making it through. Winning the game requires to defeat all the aliens on the screen.

Goal:

- Recreate the game experience using FPGA
- Use Video Graphics Array (VGA) port to render the game's graphic
- Implement hardware level logic
- Build controls

Hardware and Tools

FPGA Development

We used the Nexys A7-100T board to build and run the game. It handles everything, from player input to displaying graphics on a monitor, directly on the hardware, no extra software needed.

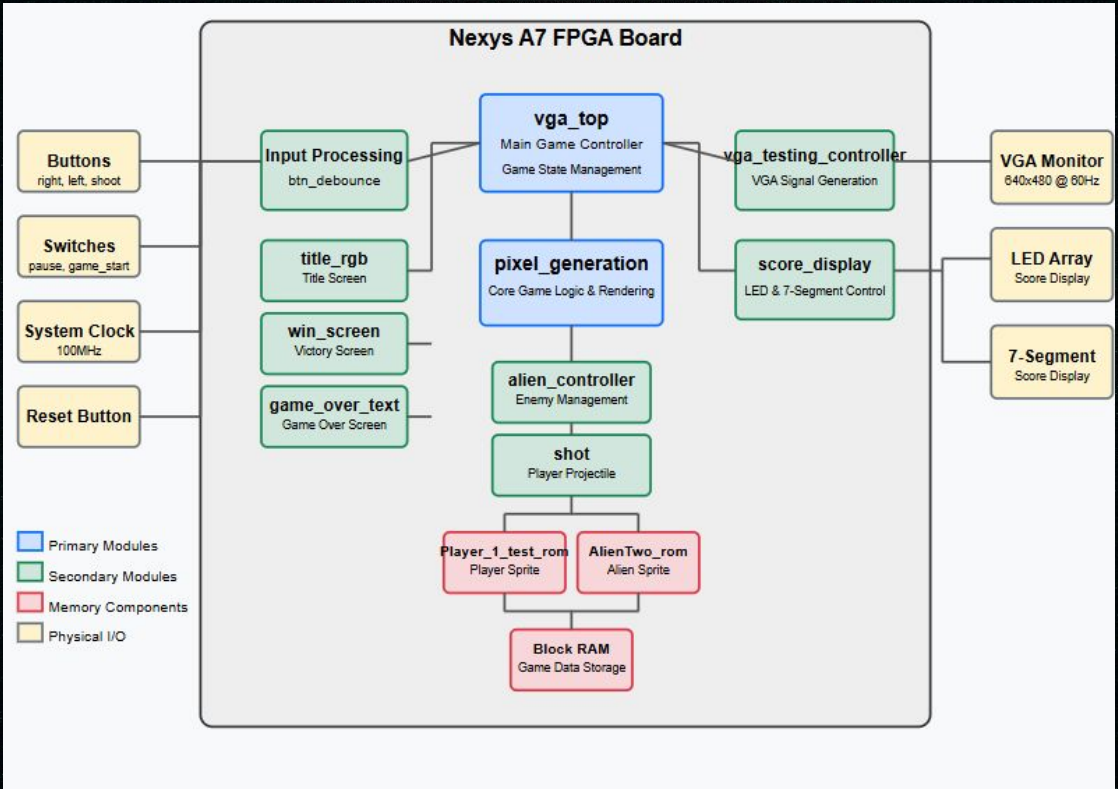
Input Devices

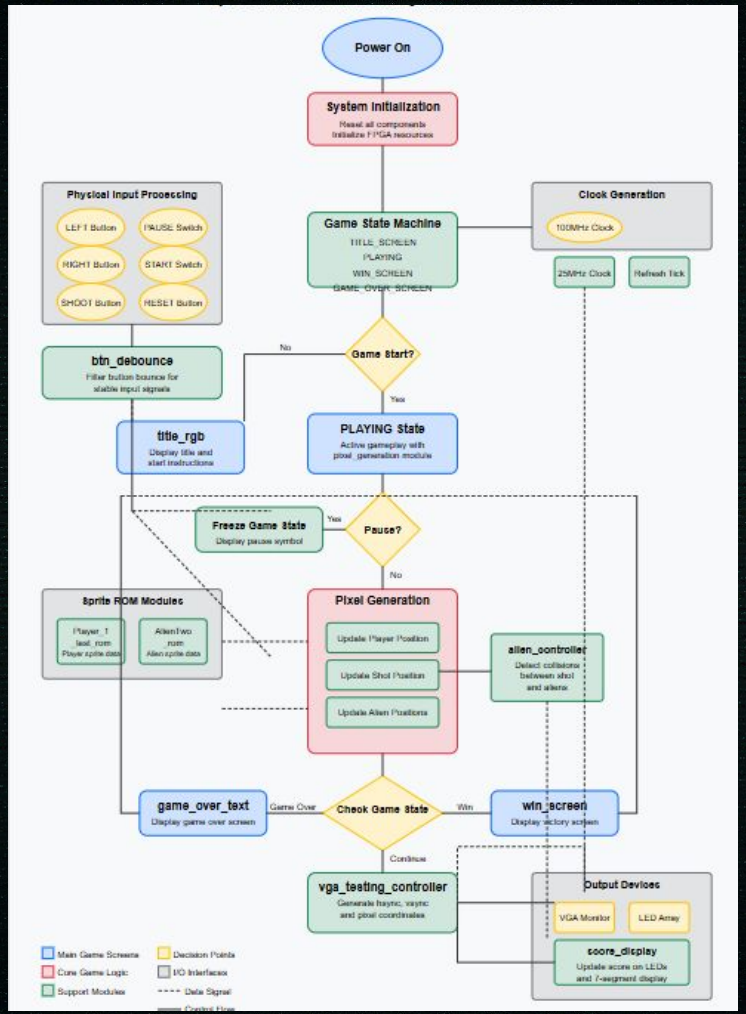
We used the onboard buttons to control the player's ship. Inputs are handled in Verilog, with adding debouncing for more responsive gameplay. Switches are also implemented for start and pausing of the game.

Output devices

We used a VGA monitor to display the game, with our design generating the sync signals needed for a 640x480 resolution. All game elements such as the player, aliens, and bullets, are rendered using pixel coordinates directly in Verilog.

System Design

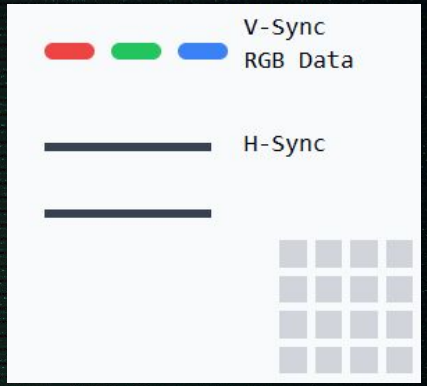



Power On
System Initialization
Reset all components
Initialize FPGA resources
Physical Input Processing
LEFT Button
PAUSE Switch
RIGHT Button
START Switch
SHOOT Button
RESET Button
Game State Machine
TITLE_SCREEN
PLAYING
WIN_SCREEN
GAME_OVER_SCREEN
Clock Generation
100Mhz Clock
25Mhz Clock
Refresh Tick
btn_debounce
Filter button bounce for
stable input signals
No
Game Start?
Yes
title_rgb
Display title and
start instructions
PLAYING State
Active gameplay with
pixel_generation module
Freeze Game State
Display pause symbol
Yes
Pause?
No
Sprite ROM Modules
Player_1
lead_rom
Player sprite data
AlienTwo
_rom
Alien sprite data
Pixel Generation
Update Player Position
Update Shot Position
Update Alien Positions
alien_controller
Detect collisions
between shot
and aliens
game_over_text
Display game over screen
Game Over
Check Game State
Win
win_screen
Display victory screen
Continue
vga_testing_controller
Generate hsync, vsync
and pixel coordinates
Output Devices
VGA Monitor
LED Array
score_display
Update score on LEDs
and 7-segment display
Main Game Screens
Core Game Logic
Support Modules
Decision Points
I/O Interfaces
Data Signal

VGA Display & Basics

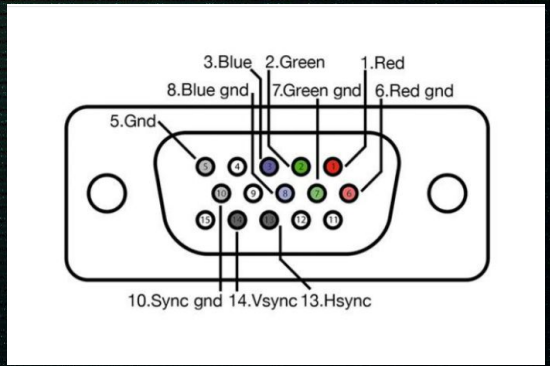
Key Characteristics

The display area is 640×480 pixels. The blanking period includes the front porch (up to 656,490), sync pulse (up to 752,492), and back porch (up to 800,521), adding horizontal and vertical timing for VGA synchronization.



How it works

VGA sends separate analog signals for Red, Green, and Blue color channels, plus horizontal and vertical sync signals. Each pixel's color is determined by varying voltage levels on the RGB channels.



Getting images to Vivado

To display our custom images on the screen, we used a Python script that read BMP files and converted them into a ROM format compatible with Vivado.

- Convert image into a preferred Size
- Convert into a BMP File
- Run Python Script

```
module Player_1_test_rom
    (
        input wire clk,
        input wire [4:0] row,
        input wire [4:0] col,
        output reg [11:0] color_data
    );

    (* rom_style = "block" *)

    //signal declaration
    reg [4:0] row_reg;
    reg [4:0] col_reg;

    always @(posedge clk)
        begin
        row_reg <= row;
        col_reg <= col;
        end

    always @*
    case ({row_reg, col_reg})
        10'b0000000000: color_data = 12'b111111111111;
        10'b0000000001: color_data = 12'b111111111111;
        10'b0000000010: color_data = 12'b111111111111;
        10'b0000000011: color_data = 12'b111111111111;
        10'b0000000100: color_data = 12'b111111111111;
        10'b0000000101: color_data = 12'b111111111111;
```

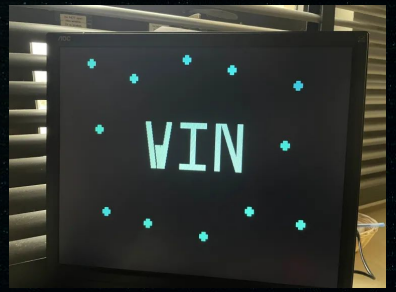
Screen Displays

For our other screen displays:

- game_over_text.v
- flick_switch_text.v
- lebron_credit.v
- title_letters.v
- win_screen.v
- pause_symbol.v

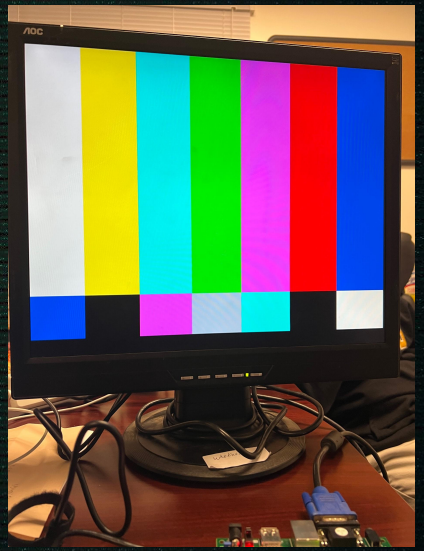
We drew up the images in pixel art website and used a github we found online that takes an input image and converts it to a specified resolution. Processes each pixel and converts the RGB values to an 8-bit format (3:3:2 bit allocation) commonly used in VGA applications. It hardcodes the pixels overall.

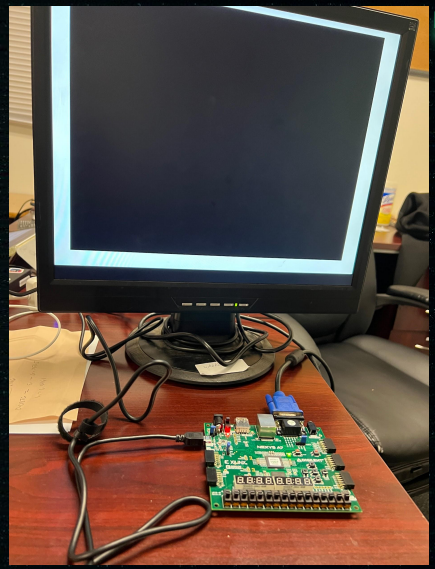
Then in our pixel_generation module we instantiated these screens to work as intend.



Pixel Generation

- Houses many parts of our Project which are instantiated inside of this file.
  - Player movement
  - Aliens
  - Shots
  - Game Boundaries
- This module pieces all of the visual pieces of the project together.
- Allows for the project to be rendered in priority.
- (Pictures shown are early stages of project)



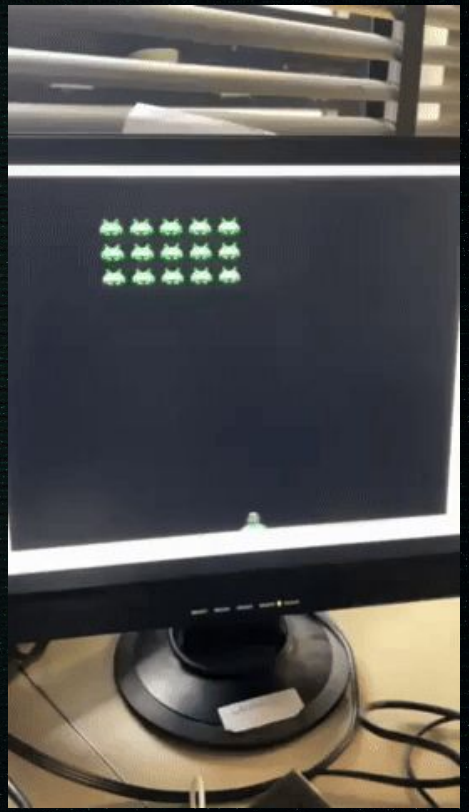


Player Movement

- Module: pixel_generation.v
- Inputs: left, right signals (debounced), clk, reset, pause, and refresh_tick
- Position Tracking:
  - Player's horizontal movement is stored in a register
  - On each refresh_tick, the module checks for left or right input.
  - If left is pressed and player is not at X = 0, move left.
  - If right is pressed and player is not at screen width limit, move right.
- Constraints:
  - Movement only occurs when video is on and game is not paused.
  - Position is clamped to screen edges to prevent going off-screen.

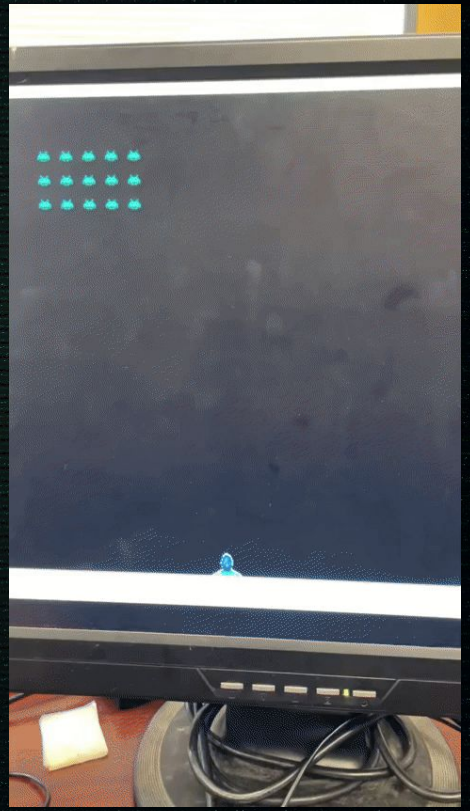
Shooting Module

- Module: shot.v
- Purpose: Handles player projectile creation, movement, and collision with aliens
- Trigger: A shot is created when button is pressed
- Position Tracking:
  - Stores shot's X and Y position based on the player's position at the time of firing.
  - Updates position every refresh_tick by moving the shot upward 3 pixels per frame.
- State Management:
  - Active when shot is in motion.
  - Deactivates if:
    - It reaches the top of the screen.
    - It hits an alien (alien_hit = 1).
    - The game is paused.
  - Performs bound checking to prevent off-screen rendering
  - Uses alien hit detection to stop shot on impact.



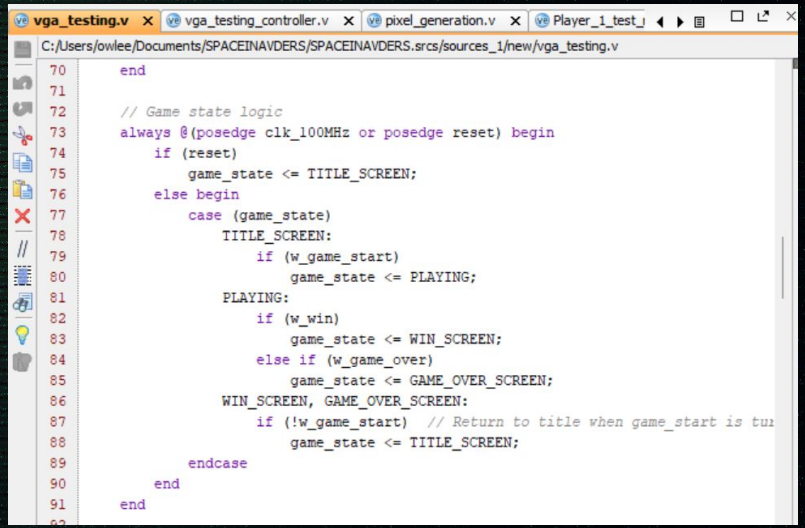
Alien Controller

- Module: alien_controller.v
- Purpose: Manages alien movement, rendering, collision with shots, and win/loss detection.
- Alien Grid: 3×5 array of aliens, each with an "alive" status. Group moves with shared x_offset and y_offset.
- Position Tracking:
  - Moves side-to-side, shifts down at edges.
  - Speed controlled by a counter.
  - Pauses when the game is paused.
- Collision Detection:
  - Detects hits from player shots.
  - Marks aliens as "dead" and triggers shot_hit.
  - Prevents multiple hits in one frame.
- Game Conditions:
  - Win: All aliens are destroyed.
  - Game Over: Any alien reaches the player's Y position.
  - Outputs win and game_over signals to top-level logic.



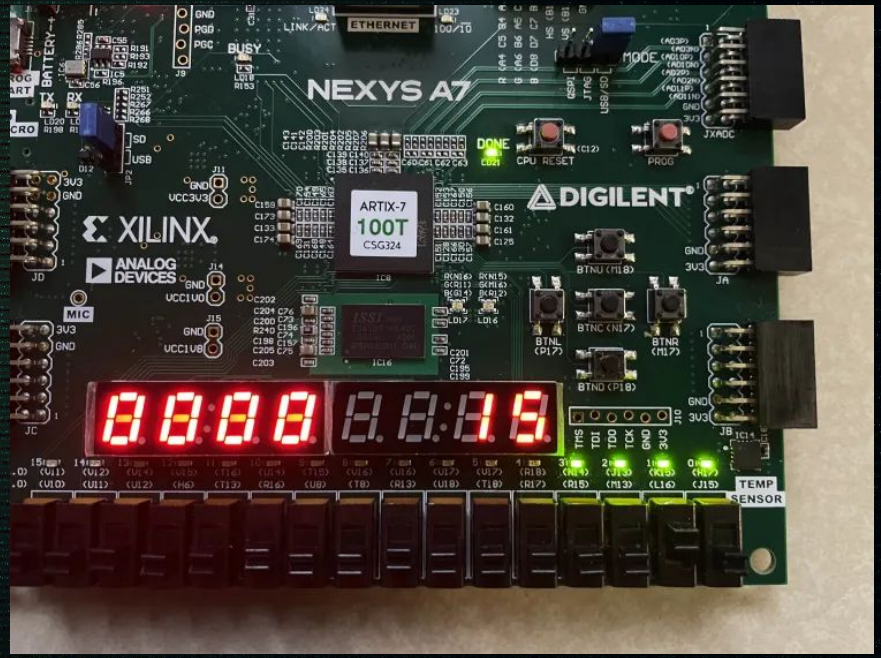
Start, Pause, and Game Over Screens

- **Start Screen:**
  Displayed when the switch is **OFF**. Shows the game title using title_rgb. Transitions to gameplay when switch is turned ON.
- **Pause Screen:**
  Activated with a pause switch. Freezes game logic but still renders a pause symbol (pause_symbol.v) on screen.
- **Game Over Screen:**
  Shown when the alien reaches a certain y position. Overrides game display to show “GAME OVER” using game_over_text.v. Game resets on switch toggle or reset signal.



BCD 7-Segment & LEDS Score Counter

- Module: score_display.v Shows score on both LEDs (binary) and 7-segment (decimal) display.
- Increments by 1 for each alien hit (max score: 15).
- Uses digit multiplexing and BCD decoding to show tens and ones digits.
- Runs on 100 MHz clock with active-low signals.
- Based on our CECS 201, project 5, adapted for this game.



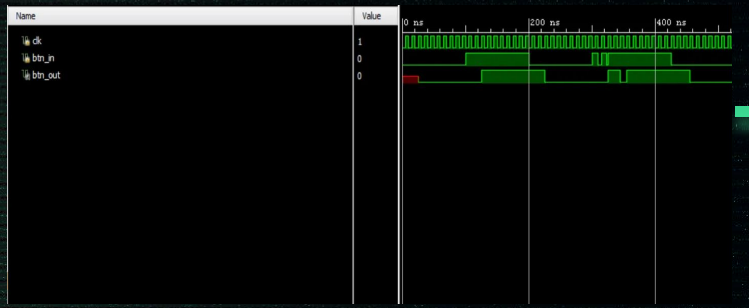
Simulations

Button Debounce Simulation

Goal: Show how mechanical button bounces are filtered out.

Observation: Raw input (btn_in) had noisy transitions. Debounced output (btn_out) only changed after stable input for several cycles.

Result: Proved that our btn_debounce.v module works reliably, preventing accidental multiple inputs during gameplay (e.g., multiple shots or jerky movement).

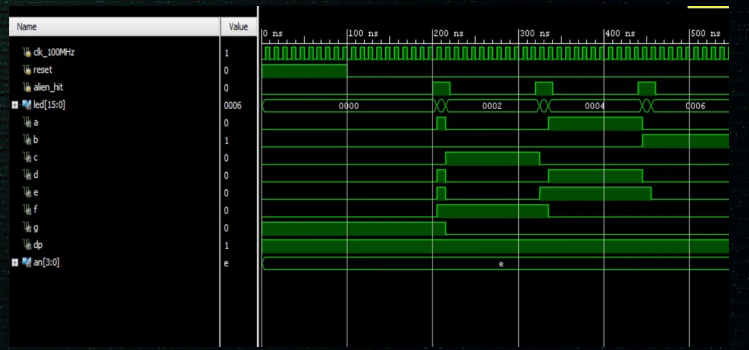


Score Display Simulation

Goal: Test how score updates when aliens are hit.

Observation: alien_hit signal triggered 3 times; score updated from 0 → 1 → 2 → 3 on LEDs and 7-segment display.

Result: Confirmed the score system (score_display.v) increments correctly and displays values as intended, enhancing game feedback.



Implementation

- The game is written in Verilog and built from modular files (e.g., pixel_generation.v, alien_controller.v, shot.v, score_display.v).
- Modules are instantiated hierarchically:
  - pixel_generation.v includes alien_controller, shot, and sprite ROMs.
  - Display modules like title_rgb, win_screen, and game_over_text are selected in vga_top.v based on game state.
  - btn_debounce.v filters button inputs for reliable control.
- The top-level module vga_top.v connects everything and manages game states using a state machine.
- vga_testing_controller.v handles VGA timing for a 640×480 display at 60Hz.
- A VGA cable connects the Nexys A7-100T directly to a monitor. All logic runs on hardware using the board's buttons, LEDS, and switches.

Demo


pause screen

Thanks