Snake Game Implementation on DE10-Lite FPGA Board

EECS 3216 Project Report

Team Members:

- Kanwarjot Singh Bharaj (214284269)
- Fares Trad (217281379)
- Joshua Keppo (210971752)
- Kyle Williamson (218953901)

Abstract

This report details the implementation of a classic Snake game on the DE10-Lite FPGA board with VGA output to an external display. The project successfully combines hardware design using SystemVerilog with real-time data processing to create an interactive gaming experience. Key features include snake movement control, collision detection, score tracking, and progressive difficulty levels. The implementation demonstrates the practical application of digital logic design principles and embedded systems development on FPGA hardware.

1. Introduction

1.1 Project Overview

The Snake game is a classic arcade game where players control a snake that grows in length as it consumes food items. The game ends if the snake collides with itself or the boundaries of the playing field. Our implementation on the DE10-Lite FPGA board showcases hardware-software integration in a real-time application, requiring careful consideration of timing, memory usage, and user interaction.

1.2 Objectives

The primary objectives of this project were to:

1. Design and implement a fully functional Snake game on the DE10-Lite FPGA board
2. Create a VGA controller for real-time display output
3. Implement player control using onboard buttons
4. Design collision detection and score tracking systems

5. Implement progressive difficulty that increases as the player advances
6. Demonstrate mastery of FPGA programming concepts using SystemVerilog

2. System Architecture

The system architecture can be divided into several key components, each handling specific aspects of the game's functionality.

2.1 Top-Level Architecture

The system is built around the DE10-Lite_Snake top module, which integrates all sub-modules and connects to the physical I/O of the FPGA board. The high-level architecture includes:

1. **Input Processing**: Handles debounced key inputs for controlling snake direction
2. **Game Logic**: Manages game state, snake movement, collision detection, and score tracking
3. **VGA Controller**: Generates timing signals for VGA output and manages pixel data
4. **Display Logic**: Determines what to display at each pixel position on the screen
5. **Score Display**: Converts score data to seven-segment display format

2.2 Hardware Components

The DE10-Lite board provides the following hardware components used in our implementation:

- **MAX10 FPGA**: The core computational element
- **50MHz System Clock**: Provides timing for the entire system
- **Push Buttons**: Used for controlling snake direction (clockwise and counter-clockwise rotation)
- **Slide Switches**: Used for game configuration (e.g., border visibility)
- **LEDs**: Provide feedback on button presses and game state
- **Seven-Segment Displays**: Display the current score
- **VGA Port**: Provides output to an external display

2.3 Module Hierarchy

The project is structured with the following module hierarchy:

1. **DE10_Lite_Snake**: Top-level module connecting all components and handling I/O
2. **snake_game**: Core game logic and VGA display generation
3. **vga_controller**: Generates VGA timing signals and coordinates
4. **button_debounce**: Debounces button inputs to prevent unintended multiple presses

3. Design Methodology

3.1 Development Process

The development process followed these sequential steps:

1. **Requirements Analysis**: Defined game specifications and feature set
2. **Architecture Design**: Established the module hierarchy and interfaces
3. **Module Implementation**: Coded individual modules in SystemVerilog
4. **Integration**: Connected modules to form the complete system
5. **Testing**: Validated functionality using simulation and hardware testing
6. **Refinement**: Optimized code and fixed identified issues

3.2 Tools Used

The following tools were utilized in the development process:

- **Intel Quartus Prime**: For SystemVerilog coding, synthesis, and programming the FPGA
- **ModelSim**: For simulation and verification of module functionality
- **DE10-Lite Development Board**: For hardware implementation and testing
- **External VGA Monitor**: For visual output display

3.3 Design Decisions

Several key design decisions shaped the implementation:

3.3.1 VGA Configuration

We implemented a 640x480 @60Hz VGA configuration, a standard resolution that is well-supported by most monitors and straightforward to implement on the DE10-Lite board. The vga_controller module generates the necessary timing signals (h_sync, v_sync) and coordinates (column, row) for display.

3.3.2 Game Grid Design

The game uses a grid-based approach where:

- Each grid cell is 20x20 pixels
- The total grid size is 32x24 cells (640x480 pixels)
- Border cells mark the boundary of the playing field

This approach simplifies collision detection and game logic while providing sufficient resolution for gameplay.

3.3.3 Snake Control Mechanism

We implemented a rotational control system using two buttons:

- KEY[0]: Rotate snake clockwise
- KEY[1]: Rotate snake counter-clockwise

This control scheme was chosen over direct cardinal direction controls due to the limited number of buttons available on the DE10-Lite board. It also provides an interesting gameplay mechanic that requires planning and anticipation.

3.3.4 Difficulty Progression

Game difficulty increases as the player scores points:

- Snake speed increases with each food item consumed
- The maximum speed is capped to maintain playability
- The speed adjustment is implemented through a countdown counter that triggers snake movement

3.3.5 Score Display

The score is displayed on the seven-segment displays using binary-coded decimal (BCD) conversion:

- Score is stored in binary format
- Converted to BCD for display on HEX3-HEX5
- A custom function converts BCD digits to seven-segment display patterns

4. Implementation Details

4.1 Core Game Logic

The core game logic is implemented in the snake_game module, which handles:

```
// Game state enumeration
typedef enum logic [1:0] {
    IDLE,
    RUNNING,
    GAME_OVER
} game_state_t;

// Direction enumeration
typedef enum logic [1:0] {
    DIR_RIGHT = 2'b00,
    DIR_DOWN  = 2'b01,
    DIR_LEFT  = 2'b10,
    DIR_UP    = 2'b11
} direction_t;
```

The game state machine transitions between IDLE, RUNNING, and GAME_OVER states based on player actions and game events. The snake movement is controlled by direction states that are updated based on button inputs.

4.2 Snake Implementation

The snake is represented as arrays of X and Y coordinates:

```
logic [$clog2(GRID_WIDTH)-1:0]  snake_x [0:GRID_WIDTH*GRID_HEIGHT-1];
logic [$clog2(GRID_HEIGHT)-1:0] snake_y [0:GRID_WIDTH*GRID_HEIGHT-1];
logic [100:0] snake_length;
```

This approach allows for:

- Dynamic growth of the snake when food is consumed
- Efficient collision detection with itself and boundaries
- Simple rendering of the snake on the display

4.3 VGA Controller

The VGA controller generates timing signals based on standard VGA parameters:

```
vga_controller #(
    .h_pixels(H_PIXELS),
    .h_fp(H_FP),
    .h_pulse(H_PULSE),
    .h_bp(H_BP),
    .v_pixels(V_PIXELS),
    .v_fp(V_FP),
    .v_pulse(V_PULSE),
    .v_bp(V_BP)
) vga_inst (
    .pixel_clk(pixel_clk),
    .reset_n(reset_n),
    .h_sync(VGA_HS),
    .v_sync(VGA_VS),
    .disp_ena(vga_disp_ena),
    .column(vga_column),
    .row(vga_row)
);
```

The controller operates at a 25MHz pixel clock (derived from the 50MHz system clock) and provides:

- Horizontal and vertical sync signals
- Display enable signal
- Current pixel coordinates

4.4 Input Debouncing

To ensure reliable button input, a debouncing module was implemented:

```
button_debounce db_key0 (
    .clk(MAX10_CLK1_50),
    .rst(rst_sync),
    .button(~KEY[0]),
    .pulse(key0_pulse),
    .stable(key0_stable)
);
```

The debounce module:

- Synchronizes the input signal to prevent metastability
- Uses a counter-based approach to filter out bounces
- Generates a single-cycle pulse on button press
- Provides a stable output signal for LED display

4.5 Random Number Generation

For food placement, a Linear Feedback Shift Register (LFSR) was implemented:

```
always_ff @(posedge clk or negedge reset_n) begin
    if (~reset_n) begin
        lfsr <= 16'hACE1;  // Initial seed
    end else begin
        lfsr <= {lfsr[14:0], lfsr[15] ^ lfsr[13] ^ lfsr[12] ^ lfsr[10]};
    end
end
```

This provides pseudo-random coordinates for food placement without requiring complex random number generation hardware.

4.6 Display Generation

The pixel color is determined based on the current pixel's position relative to game elements:

```
always_comb begin
    // Default - black background
    pixel_color = COLOR_BLACK;

    // Calculate grid position of current pixel
    grid_x = vga_column / GRID_SIZE;
    grid_y = vga_row / GRID_SIZE;

    // Check what the current pixel represents
    is_border = border_visible && (/* border conditions */);
    is_snake_head = (grid_x == snake_x[0] && grid_y == snake_y[0]);
    is_snake_body = /* loop through snake body */;
    is_apple = (grid_x == apple_x && grid_y == apple_y);

    // Set pixel color based on what it represents
    if (is_border) pixel_color = COLOR_BLUE;
    else if (is_snake_head) pixel_color = COLOR_GREEN;
    else if (is_snake_body) pixel_color = COLOR_DARK_GREEN;
    else if (is_apple) pixel_color = COLOR_RED;
end
```

This combinational logic determines the color of each pixel in real-time based on the current game state.

5. Testing and Validation

5.1 Simulation Testing

The project was initially validated through simulation using ModelSim. The testbench (snake_game_tb.sv) provided a controlled environment to verify:

- VGA signal generation
- Button input processing
- Game state transitions
- Snake movement and growth
- Collision detection

The testbench included a sequence of inputs that simulated:

- Game initialization
- Direction changes

- Food consumption
- Border toggling

5.2 Hardware Testing

After simulation testing, the design was implemented on the DE10-Lite board for hardware validation. The following aspects were tested:

1. **VGA Output**: Verified correct display on external monitor
2. **Button Controls**: Tested responsiveness and accuracy of direction changes
3. **Game Logic**: Validated snake movement, growth, and collision detection
4. **Score Display**: Confirmed accurate score tracking and display
5. **Progressive Difficulty**: Tested speed increases as score advanced

5.3 Validation Results

The game successfully passed all testing phases with the following results:

1. **VGA Output**: Clean, stable display at 640x480 @60Hz
2. **Control Response**: Consistent button response after debouncing
3. **Game Mechanics**: Snake movement, growth, and collisions function as expected
4. **Scoring System**: Accurate score tracking and display on seven-segment displays
5. **Progressive Difficulty**: Appropriate increase in game speed as score advances

5.4 Performance Metrics

The final implementation achieved the following performance metrics:

1. **Resource Utilization**:
    - Logic elements: Approximately 30% of available resources
    - Memory bits: Minimal usage for snake position storage
2. **Timing Performance**:
    - VGA refresh rate: 60Hz (standard)
    - Game update rate: Variable based on score (decreases from ~4Hz to ~16Hz)
    - Input latency: <20ms after debouncing
3. **Visual Performance**:
    - Stable display without flickering
    - Clear distinction between game elements
    - Smooth snake movement

6. Challenges and Solutions

6.1 VGA Timing Challenges

Challenge: Implementing precise VGA timing signals to ensure stable display.

Solution: We utilized a well-tested vga_controller module with configurable parameters for different resolutions. The module was carefully verified through simulation before hardware testing.

6.2 Snake Movement and Collision Detection

Challenge: Efficiently representing the snake's growing body and detecting collisions.

Solution: Implemented the snake as arrays of coordinates with a variable length counter. This approach simplified collision detection and allowed for efficient updates during movement.

6.3 Random Food Placement

Challenge: Generating random positions for food placement without placing food on the snake.

Solution: Implemented an LFSR for pseudo-random number generation and added validation to ensure food wasn't placed on the snake. If an invalid position was generated, the algorithm would adjust the position until a valid one was found.

6.4 Button Debouncing

Challenge: Physical buttons produced multiple spurious signals on a single press.

Solution: Implemented a robust debouncing module using synchronization and a counter-based approach to filter out bounces. This provided clean, single pulses for direction changes.

6.5 Speed Progression

Challenge: Implementing a smooth progression of game speed without making it unplayable.

Solution: Designed a variable countdown timer that decreases with each food item consumed, with lower and upper bounds to ensure playability throughout the game.

7. Lessons Learned

Throughout the project, several valuable lessons were learned:

1. **Modular Design**: Breaking the system into well-defined modules with clear interfaces simplified development and debugging.

2. **Simulation Before Hardware**: Thorough simulation testing identified and resolved many issues before hardware implementation, saving time and effort.

3. **State Machine Design**: Carefully designed state machines provided a clean framework for managing game logic and transitions.

4. **Resource Management**: Efficient use of FPGA resources through careful algorithm design allowed implementation of all required features.

5. **Signal Synchronization**: Proper synchronization of signals between clock domains prevented metastability issues and ensured reliable operation.

6. **User Experience Considerations**: Balancing game difficulty progression to maintain playability while increasing challenge required careful tuning.

8. Future Improvements

Several potential improvements have been identified for future iterations:

1. **Additional Input Methods**: Implement PS/2 keyboard support for more intuitive directional control.

2. **Enhanced Visuals**: Add animations, textures, or effects to improve visual appeal.

3. **Game Modes**: Implement different game modes (e.g., time trial, maze obstacles) for variety.

4. **Two-Player Mode**: Add support for two-player competitive gameplay.

5. **Sound Effects**: Integrate audio feedback for game events using additional hardware.

6. **High Score Storage**: Implement non-volatile storage for high scores using the FPGA's flash memory.

7. **Menu System**: Add a game menu for configuration options and mode selection.

8. **Power Optimization**: Implement power-saving features for prolonged gameplay.

9. Conclusion

The Snake game implementation on the DE10-Lite FPGA board successfully demonstrates the application of digital design principles in creating an interactive gaming system. By combining hardware modules for VGA control, input processing, and game logic, we created a complete embedded system that provides an engaging gaming experience.

The project showcases how modern FPGAs can be used for real-time applications requiring precise timing, user interaction, and visual display. The modular approach to design allowed for incremental development and testing, resulting in a robust final product.

The skills and knowledge gained through this project—including hardware description language programming, state machine design, timing management, and system integration—provide a strong foundation for more complex digital design work in the future.

10. References

1. Altera DE10-Lite User Manual
2. Intel MAX 10 FPGA Device Datasheet
3. VGA Signal Timing Standard
4. SystemVerilog Language Reference

Appendix A: Module Interfaces

A.1 DE10_Lite_Snake Module

```
module DE10_Lite_Snake(
   // Clock inputs
   input logic         ADC_CLK_10,
   input logic         MAX10_CLK1_50,
   input logic         MAX10_CLK2_50,

   // Input devices
   input logic  [1:0]    KEY,
   input logic  [9:0]    SW,

   // Output devices
   output logic [9:0]    LEDR,
   output logic [7:0]    HEX0,
   output logic [7:0]    HEX1,
   output logic [7:0]    HEX2,
   output logic [7:0]    HEX3,
   output logic [7:0]    HEX4,
   output logic [7:0]    HEX5,

   // VGA outputs
```

```
    output logic [3:0]    VGA_R,
    output logic [3:0]    VGA_G,
    output logic [3:0]    VGA_B,
    output logic          VGA_HS,
    output logic          VGA_VS
);
```

A.2 snake_game Module

```
module snake_game(
    input logic clk,
    input logic reset_n,
    input logic [1:0] KEY,
    input logic [9:0] SW,
    output logic [3:0] VGA_R,
    output logic [3:0] VGA_G,
    output logic [3:0] VGA_B,
    output logic VGA_HS,
    output logic VGA_VS,
    output logic [$clog2(GRID_WIDTH*GRID_HEIGHT)-1:0] score
);
```

A.3 vga_controller Module

```
module vga_controller #(
   parameter h_pixels   = 640,
   parameter h_fp       = 16,
   parameter h_pulse    = 96,
   parameter h_bp       = 48,
   parameter h_pol      = 1'b0,
   parameter v_pixels   = 480,
   parameter v_fp       = 10,
   parameter v_pulse    = 2,
   parameter v_bp       = 33,
   parameter v_pol      = 1'b0
)(
   input pixel_clk,
   input reset_n,
   output reg h_sync,
   output reg v_sync,
   output reg disp_ena,
   output reg [31:0] column,
   output reg [31:0] row
```

```
);
```

A.4 button_debounce Module

```
module button_debounce (
    input logic clk,
    input logic rst,
    input logic button,
    output logic pulse,
    output logic stable
);
```

Appendix B: Game Parameters

```
// VGA Parameters
parameter H_PIXELS = 640;
parameter H_FP = 16;
parameter H_PULSE = 96;
parameter H_BP = 48;
parameter V_PIXELS = 480;
parameter V_FP = 10;
parameter V_PULSE = 2;
parameter V_BP = 33;

// Game parameters
parameter GRID_SIZE = 20;
parameter GRID_WIDTH = 32;
parameter GRID_HEIGHT = 24;
parameter BORDER_SIZE = 1;
parameter INIT_SNAKE_LEN = 1;
parameter GAME_SPEED_MAX = 12000000;
parameter GAME_SPEED_MIN = 3000000;
parameter GAME_SPEED_DECREMENT = 500000;
parameter MAX_SNAKE_LENGTH = 100;

// Color definitions (4-bit RGB)
parameter [11:0] COLOR_BLACK = 12'h000;
parameter [11:0] COLOR_GREEN = 12'h0F0;
parameter [11:0] COLOR_RED = 12'hF00;
parameter [11:0] COLOR_BLUE = 12'h00F;
parameter [11:0] COLOR_DARK_GREEN = 12'h080;
```