Snake Game

Final Project - Embedded Systems

Cristopher Marte (cjm2301), Somya Mehta (sm5169), Rohan Rabbi (mr4280), Saher Iqbal (si2443)

Contents:

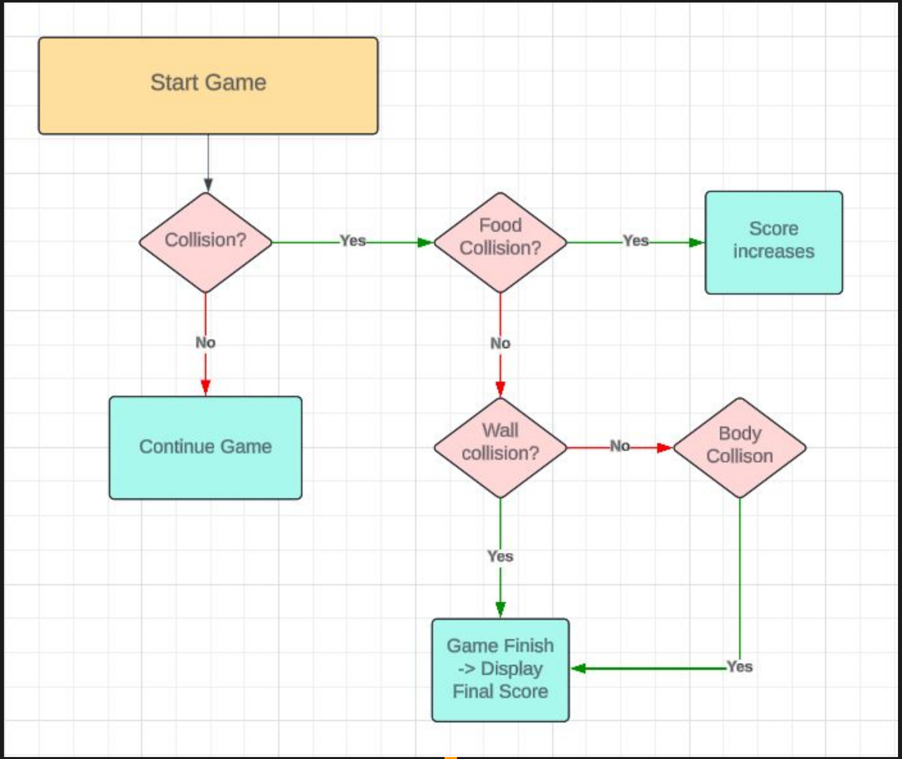
Project Introduction

The main part of the game is the snake which the user is supposed to navigate throughout the screen. While navigating, as the snake eats a fruit, it grows in length. The score is displayed based on the length of the snake. The game ends if the snake hits its own body or hits the wall.

Game Components:

The Snake Sprite: The user controls this using a controller.
Wall Sprites: The boundary of the game.
Apple Sprites: These are the food items the snake is supposed to eat to grow in length.
Score Display: This score starts from 0 and increases by one every time the snake consumes an Apple.

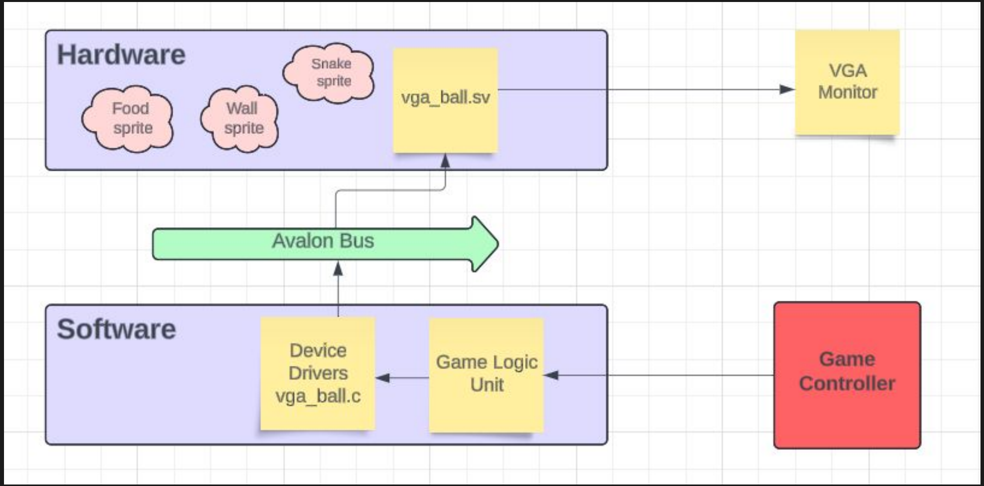
Game Rules and Logic



System Overview

Hardware part of the project includes the design and implementation of modules for VGA image display, control input via the controller. Specifically, the game logic unit transmits data concerning each object's position and status to the hardware via the device drivers. The hardware peripherals, upon receiving this data, process and decode it to produce the appropriate visual outputs.

The software portion consists of the device drivers and the game logic unit. The game logic manages several key functions, including the display of Apple sprites, Snake sprites, score calculation, food eating, and wall collision detection. The device drivers serve as the bridge connecting the hardware components—such as the controller and VGA monitor—to the logic control unit.

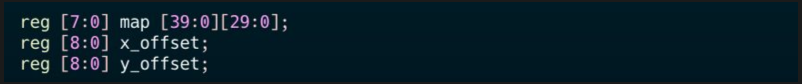


Hardware

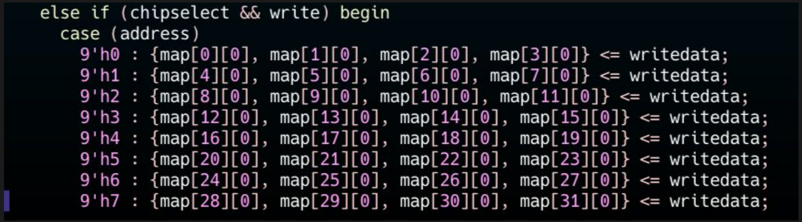
The way the hardware displays the sprites is by communicating the address as a 32 bit packet write data. This 32 bit write data is given in the form of four 8-bit packets. We have created a reg [7:0] map [39:0][29:0] and made 300 different registers in a way that is shown in the figure.

By organizing the sprite data into this grid of 300 registers, we enable a direct and efficient mapping of sprite information to specific locations on the screen, facilitating rapid updates and visual outputs.

We have divided our VGA Monitor by dividing it into 40x30 tiles such that each tile is 16x16.



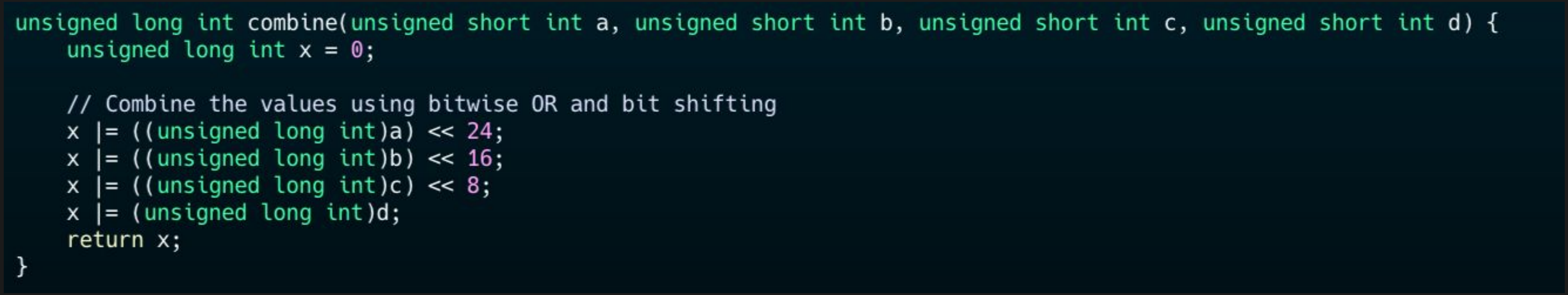
```
reg [7:0] map [39:0][29:0];
reg [8:0] x_offset;
reg [8:0] y_offset;
```



```
  else if (chipselect && write) begin
    case (address)
      9'h0 : {map[0][0], map[1][0], map[2][0], map[3][0]} <= writedata;
      9'h1 : {map[4][0], map[5][0], map[6][0], map[7][0]} <= writedata;
      9'h2 : {map[8][0], map[9][0], map[10][0], map[11][0]} <= writedata;
      9'h3 : {map[12][0], map[13][0], map[14][0], map[15][0]} <= writedata;
      9'h4 : {map[16][0], map[17][0], map[18][0], map[19][0]} <= writedata;
      9'h5 : {map[20][0], map[21][0], map[22][0], map[23][0]} <= writedata;
      9'h6 : {map[24][0], map[25][0], map[26][0], map[27][0]} <= writedata;
      9'h7 : {map[28][0], map[29][0], map[30][0], map[31][0]} <= writedata;
```

Software

Software portion of our game plays an important role in moving the snake sprites in the direction as desired by the game player. In order to implement our software logic, we have defined a combine function as shown below:



```c
unsigned long int combine(unsigned short int a, unsigned short int b, unsigned short int c, unsigned short int d) {
    unsigned long int x = 0;

    // Combine the values using bitwise OR and bit shifting
    x |= ((unsigned long int)a) << 24;
    x |= ((unsigned long int)b) << 16;
    x |= ((unsigned long int)c) << 8;
    x |= (unsigned long int)d;
    return x;
}
```

Software

This function has been used by us in the code to position the snake sprites as a full snake, we have mapped the value of each snake sprites in terms of numbers such that:

0	**→ Background**
1	**→ Apple Sprite**
2	**→ Snake Head Looking Up**
3	**→ Snake Head Looking Down**
4	**→ Snake Head Looking Left**
5	**→ Snake Head Looking Right**
6	**→ Body Vertical**
7	**→ Body Horizontal**
8	**→ Snake Body Corner 1**
9	**→ Snake Body Corner 1**
10	**→ Snake Body Corner 1**
11	**→ Snake Body Corner 1**
12	**→ Snake Tail Up**
13	**→ Snake Tail Down**
14	**→ Snake Tail Left**
15	**→ Snake Tail Right**

The way we use the combine function is like:

vla.grid.data = combine(0,0,14,5); // Snake head_right and tail_left placed of the first two columns of the corresponding row

or

vla.grid.data = combine(0,14,7,5); // Snake head_right, horizontal snake body and tail_left placed of the first two columns of the corresponding row

Hardware - Software Interfacing

The hardware of the DE1-SoC board interacts with the software logic to achieve its functionality. Writing driver or interface code, controlling memory access, and setting up peripherals are all part of this. Following the computation of the game logic, the software modifies the coordinates of the apple, snake, and body length and sends them via a kernel program to the DE1-SoC board's registers. The VGA display is updated by the hardware once it reads the data from the registers.

Conclusion

Goals achieved:

1. Displaying the Apple, Snake and the Wall sprites successfully.
2. Making the vga_ball.sv file successfully.
3. Connecting the controller successfully to move the snake.
4. Interfacing the Hardware-Software correctly.

In progress - Game Design for moving the snake in different directions.

THANKS