Design Document for Snake Game

Cristopher Marte (cjm2301), Somya Mehta (sm5169),

Rohan Rabbi (mr4280), Saher Iqbal (si2443)

CSEE4840 Embedded System Spring 2024

Content

Introduction

Our project aims to create a 2D grid-based game with dimensions of 625 pixels by 625 pixels. The game will be controlled using a joystick and will output visuals on a VGA display, refreshing at a rate of 60 frames per second (60fps). The software's primary functionality involves updating the positions of the snake, food, and score on the grid in real time, ensuring smooth gameplay and accurate feedback to the player. https://en.wikipedia.org/wiki/Snake_(video_game_genre)

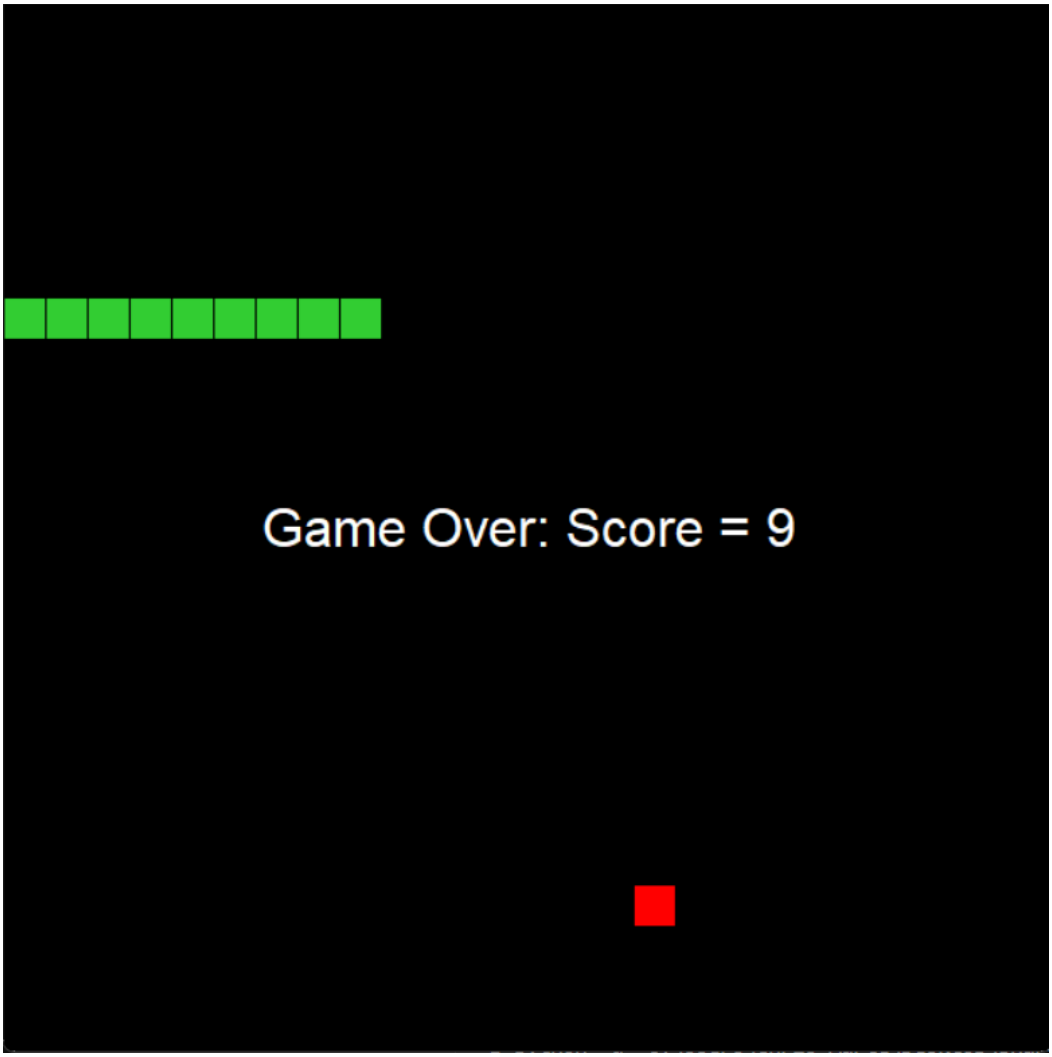

Figure 1: Python Code (prototype)

System Block Diagram

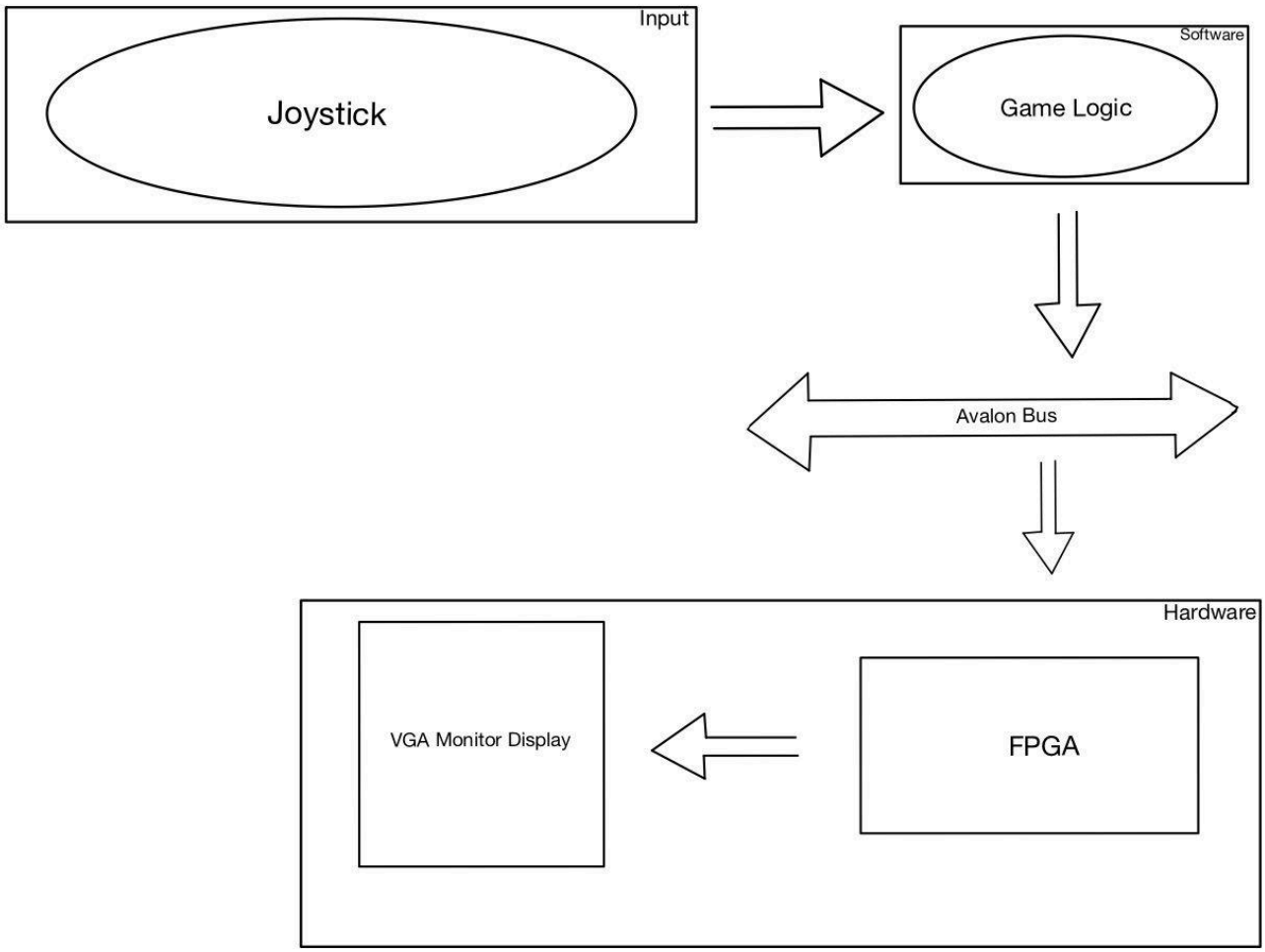

Figure 2: Block diagram

Project Algorithm

The game will simultaneously wait for input from the Joystick to change direction while still updating the position of the snake from the current velocities.

1. **change_direction()**:
    a. This will handle input events to change the direction of the snake while updating the velocity in the x and y directions based on the input given.
2. **move_snake():**
    a. This function will update the position of the snake based on its current direction and will return if the game_over bool is set to true which can happen if the snake head collides with the window boundaries or if the snake head collides with its own body. This function will also update the score every time the snake head collides with a food item on the game grid while generating a new one randomly.
3. **Draw ():**
    a. The Draw function clears the game grid and then redraws the snake, food, and score based on the updated values. If the game is over, the game will display the final score.

```python
def move_snake():
  global snake, snake_body, food, game_over, game_score

  if(game_over): # Exit the function if the game is over(No need to move the snake after game over)
    return

  # Detect collision with the window boundaries
  if (snake.x < 0) or (snake.x >= WINDOW_WIDTH) or (snake.y < 0) or (snake.y >= WINDOW_HEIGHT):
    game_over = True
    return # Exit the function if the snake hits the window boundaries (Game Over)

  for tile in snake_body:
    if (snake.x == tile.x) and (snake.y == tile.y):
```

```
      game_over = True
      return # Exit the function if the snake hits itself (Game Over)

  # Detect collision with snake head and food
  if (snake.x == food.x) and (snake.y == food.y):
    snake_body.append(Tile(food.x, food.y)) # Add a new tile to the snake body
    food.x = random.randint(0, COLS-1) * TILE_SIZE # Random x position of the food
    food.y = random.randint(0, ROWS-1) * TILE_SIZE # Random y position of the food
    # result = generate_food(snake_body) # Generate a new food position
    game_score += 1

  # Move the snake body
  for i in range(len(snake_body)-1, -1, -1):
    tile = snake_body[i]

    if (i == 0): # Move the first tile of the snake body to the snake head's
position
      tile.x = snake.x
      tile.y = snake.y
    else: # Move the rest of the tiles to the previous tile's position
      tile.x = snake_body[i-1].x
      tile.y = snake_body[i-1].y

  snake.x += dx*TILE_SIZE # Move the snake by dx Tiles (625 px)
  snake.y += dy*TILE_SIZE # Move the snake by dy Tiles (625 px)
```

Figure 3: Snippet of Python Implementation

Resources

Our VGA monitor has a resolution of 625x625 with a refresh rate of 60fps. We assume the tile size to be taken as 8x8. The resource budget is as follows:

Category	Dimensions	Number of Bits
Snake Body	250 x 8 x 8	16000
Food	8 x 8 x 10 (10 food particles on screen at a time)	640
Score	16 x 16 x 3	768
Frame Buffer (8-bit color depth, single buffer)	625×625×8	3125000
Total		3142408 bits

Hardware/Software requirements

1. **Hardware**
    a. **DE1-SoC FPGA Board:**

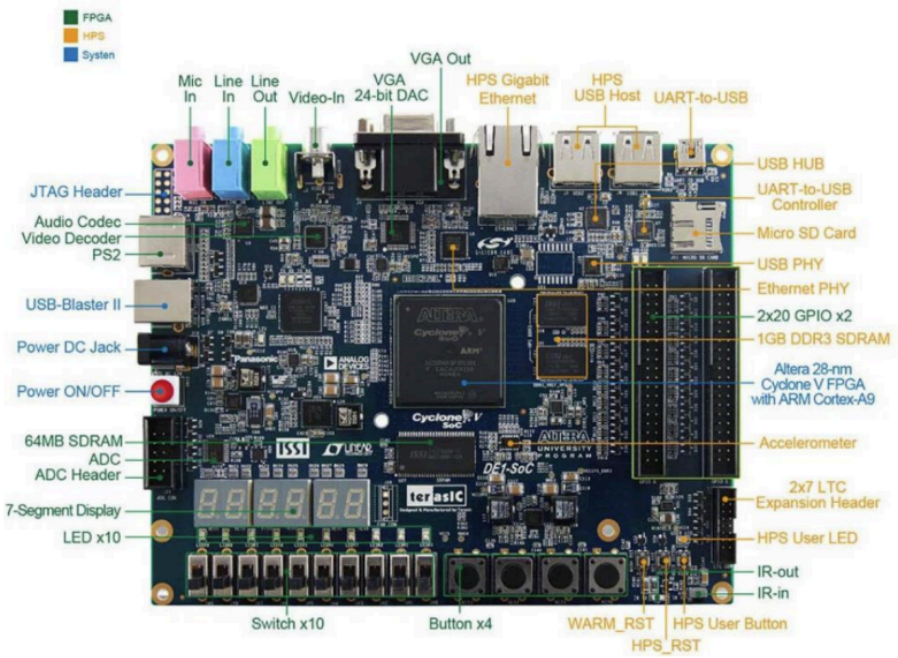

Figure 4: Pin Out for DE1-Soc Board

    b. **VGA Output Display**
        i. The visual setup will include a plain background color, with a contrasting shade (like blue) utilized to clearly mark the boundary walls, the snake will be depicted in a vibrant lime color, while the fruit (apple) will be represented in a striking red.
        ii. We will include a pause screen and a game-over screen as part of the GUI.
    c. **Joystick**
        i. We would only need the d-pad from any Xbox controller or PlayStation controller where the start would pause/unpause the game.

2. **Software**
    a. To create the game logic, we will use SystemVerilog, this will allow us to describe the digital circuitry of our game, including the processing units, memory elements, and input/output interfaces.