Design Document:
ScreamJump

Ananya Maan Singh (am6542)
Sharwari Bhosale (sb5032)
Kamala Vennela Vasireddy (kv2446)

Contents:

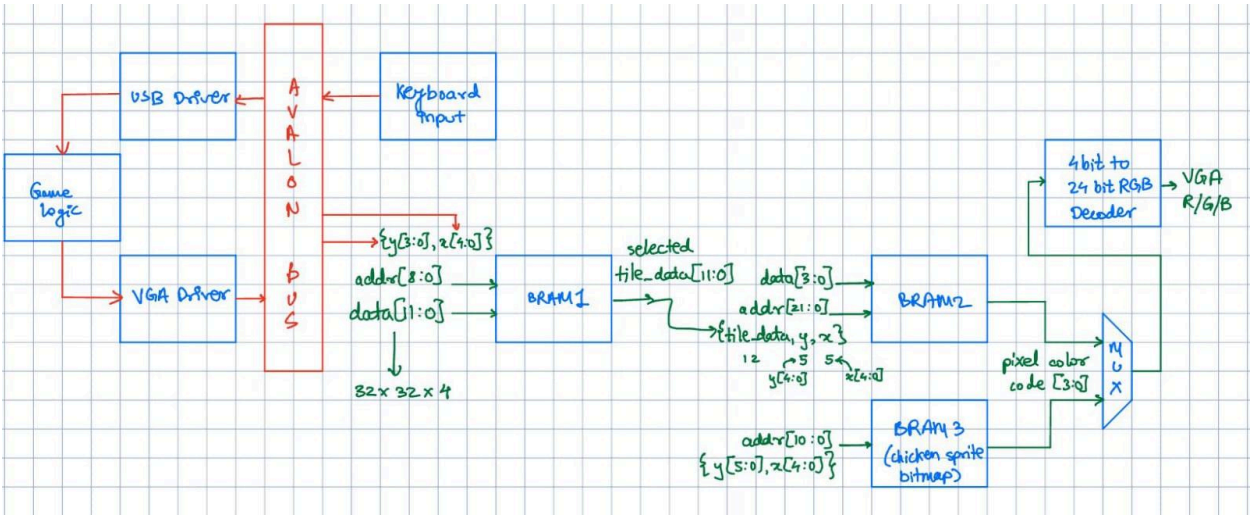
Introduction

ScreamJump is a game we aim to implement on the DE1-SoC FPGA board, where players take on the role of a quirky chicken character. The primary objective is to skillfully navigate through a world filled with various obstacles and platforms by tapping a button to control the chicken's jumps. Players must time their jumps precisely to soar over hurdles, all while avoiding a dangerous fall to the abyss below. As they progress, players will encounter increasingly difficult levels that require quick reflexes and strategic timing to keep the chicken soaring through the air.

Our goals:

- Ensure the game mechanics are verified, focusing on jumping, barriers, fall detection, and scoring through keyboard inputs.
- Set up the VGA rendering system along with sprite management.

Block Diagram



Inputs:

<u>USB Controller:</u>

- <u>Key Press Monitoring:</u>
  The input controller module monitors key presses from a USB-connected controller, specifically capturing the Up Arrow for jumping.
  The inputs are decoded into internal control signals representing a jump.

- <u>Control Signal Generation:</u>
  Based on the keycode received, the module outputs the following:
  - Jump = 1 when the Up Arrow is pressed.

  These binary signals are then packed and sent to the USB Driver for further communication with the game engine.

- <u>Debouncing and Edge Detection:</u>
  To ensure consistent behavior, debouncing logic and edge detectors will be used to recognize a single "press" rather than a held key being registered multiple times.

USB Driver:

- HID Keycode Decoding:
  The USB Driver receives keypress events from the connected USB controller and translates USB HID keycodes into standard control signals.
  The driver maps these to a compact control format understandable by the Game Logic Module.

- Avalon Bus Communication:
  The USB Driver interfaces with the rest of the system via a memory-mapped Avalon bus interface. It writes control signals (jump/move) to a predefined register accessible by the game module.
  This decouples the USB hardware specifics from the game logic and ensures a clean separation between I/O handling and gameplay decisions.

Game Logic:

- User Control: The user controls the movement of the chicken (constrained to vertical movements only) by using inputs from the controller. The user must press the input button based on the platforms viewed on the screen such that the chicken can land on top of the platform safely. Additionally, the chicken constantly falls downward (unless there is an object under it) if the player does not continuously supply the input.

- Platforms: There will be a bunch of platforms that the user needs to navigate through. These involve varying platforms that the chicken can rest on until it receives the next input. The “ground” and platforms will be constantly moving from the right side to the left side of the screen at a speed set by us.

- Adaptive difficulty: Once the user successfully earns a certain number of points, say, 15, a more difficult level ensues where the user now has to face a faster progression of platforms.

- Scoring system: The user scores points by successfully passing each platform. This will be displayed and constantly updated on the screen. The game ends when the chicken falls off of platform and falls to the abyss.

- Game over: The game ends when the chicken falls into the abyss. The final score will be displayed on the screen when the game ends, and the user has the option to navigate back to the menu.

User Interface elements:

- Game Screen:
  The display shows the chicken, ground/platforms, and current score.

- Game Over Screen:
  Once the player loses, the final score is displayed as the player's score from the most recent game session. The player is also given an option to return to the menu.

Outputs:

VGA Driver:

- Avalon Bus Communication:
  The driver writes pixel position to the VGA_chick module via an ioctl interface, sending 20-bit values that encode: x,y position and enable.

- Driver Lifecycle:
  Implemented using the Linux platform device and miscdevice framework, the driver manages resource mapping and ensures safe interaction with the hardware module.

VGA Chicken:

- Timing Signal Generation:
  The VGA_counter module generates VGA-compatible timing signals such as horizontal (hcount) and vertical (vcount) counters, VGA_HS, VGA_VS, VGA_BLANK_n, and a derived 25 MHz VGA_CLK from the 50 MHz board clock.

- Tile Rendering Logic:
  Instead of using a framebuffer, we propose to use on-the-fly tile-based composition. The VGA module reads tile coordinates and uses BRAMs to select tile and fetch respective pixel data.

VGA Display - External VGA Display:

- Signal Reception:
  The monitor receives the analog VGA signals (VGA_R, VGA_G, VGA_B, VGA_HS, VGA_VS) generated by the VGA_chick module.

- Display Output:
  These signals are interpreted to display game elements like the chicken, platforms, score, and UI text. The rendering rate is synchronized at 60 FPS for smooth gameplay visualization.

Algorithms:

1. Jump & Gravity:

- On each frame tick (60 Hz):
  - If jump is high, set vertical velocity Vy = -JUMP_V.
  - Else, apply gravity: Vy = Vy + 1.
  - Update position: y = y + Vy.
  - Clamp Y to screen bounds.

2. Fall Detection:

   When chicken falls below a certain level (Y = 480) due to gravity, set game_over.

3. Scoring:

   When a platform passes behind the bird (X + width of obstacle < chicken X), increment score once per platform.

4. VGA Rendering:

- In each frame, the software/hardware writes a sprite packet for the chicken into a BRAM slot.
- The software gives the x, y coordinates of each pixel to be displayed, which are used to select the bitmap memory of the tile represented by the pixel (BRAM1).
- BRAM2 selects 4-bit colour code data for the respective pixel.
- This 4-bit code is encoded into a 24-bit RGB value, which is finally displayed on the VGA monitor.

Memory Allocation:

We plan to implement tile-based graphics rendering where each tile is of size 32*32. Each tile covers 1024 pixels. 34 unique tiles and their respective bitmaps will be stored, which will further be processed and selected by the BRAMs. The chick's graphic is implemented using sprite-based graphic rendering.

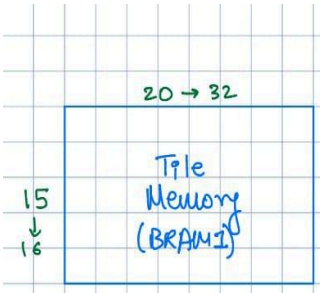
Element (tile)		**Number**	**Pixel**	**Size (Kb)**
Chick (Sprite)	**wings down**	**1**	**64*32**	**2**
	wings up	**1**	**64*32**	**2**
	dead	**1**	**64*32**	**2**
Landing blocks	**left**	**1**	**32*32**	**1**
	middle	**1**	**32*32**	**1**
	right	**1**	**32*32**	**1**
Ground	**left**	**1**	**32*32**	**1**
	middle	**1**	**32*32**	**1**
	right	**1**	**32*32**	**1**
Menu/Score characters	**letters**	**17**	**32*32**	**1**
	numbers	**10**	**32*32**	**1**
Background (sky)		**1**	**32*32**	**1**
Total		**34+3**		**15Kb**

Estimated total resource utilization is 157.5Kb (15 + 2 + 0.5 + 140)

Tile size: 32x32 pixels

sprite size: 64 x 32 pixels

32 x 64 = 2 kb



32 x 16 = 0.512 kb

Total tile memory

32x32x4x34 ← # of tiles

= 140 kb

Hardware/Software Interface:

So that software can interact with the hardware, we use a set of memory-mapped registers. These registers are located at fixed addresses and serve specific purposes in the game system:

1. Tile Data Register – Address 0x00
   This register allows the software to send information about the tiles to the hardware. When we need to draw or update a tile, we send the following values from the software that include:
   - 9 bits for the Y-position
   - 10 bits for the X-position
   - A 1-bit enable flag to control visibility.

2. Score Register – Address 0x04
   The score register is also 32 bits wide. The lower 16 bits store the player's current score, while the upper 16 bits can optionally store a high score from previous rounds. The software updates this register every time the score changes.

3. Control Register – Address 0x08
   This register lets the software send control commands to the game logic. Bit 0 is used to start a new game, bit 1 resets the game state, and bit 2 can be used to switch between different game modes or return to the main menu.

4. Status Register – Address 0x0C
   This is a read-only register that the software uses to check the current state of the game. Bit 0 indicates whether the game is over, and bit 1 indicates whether the chicken is currently allowed to jump again.