GridBrawl Game

Siyao Yu(sy3342), Sitao Zhang(sz3419), Anyongyong Zhao(az2932)

Spring 2026

1. Introduction

Our team plans to design a two-player, tile-based, competitive game implemented on the DE1 SoC FPGA platform. The game takes place on a closed map filled with hard walls and soft walls. Two players compete on the map by running across tiles to capture territory. Players can place bombs to destroy soft walls, create paths, and attack the opponent. If a player is hit by a bomb explosion, the player dies, and all tiles previously captured by that player become empty. The first player who captures 100 tiles wins the game.

2. System Block Diagram

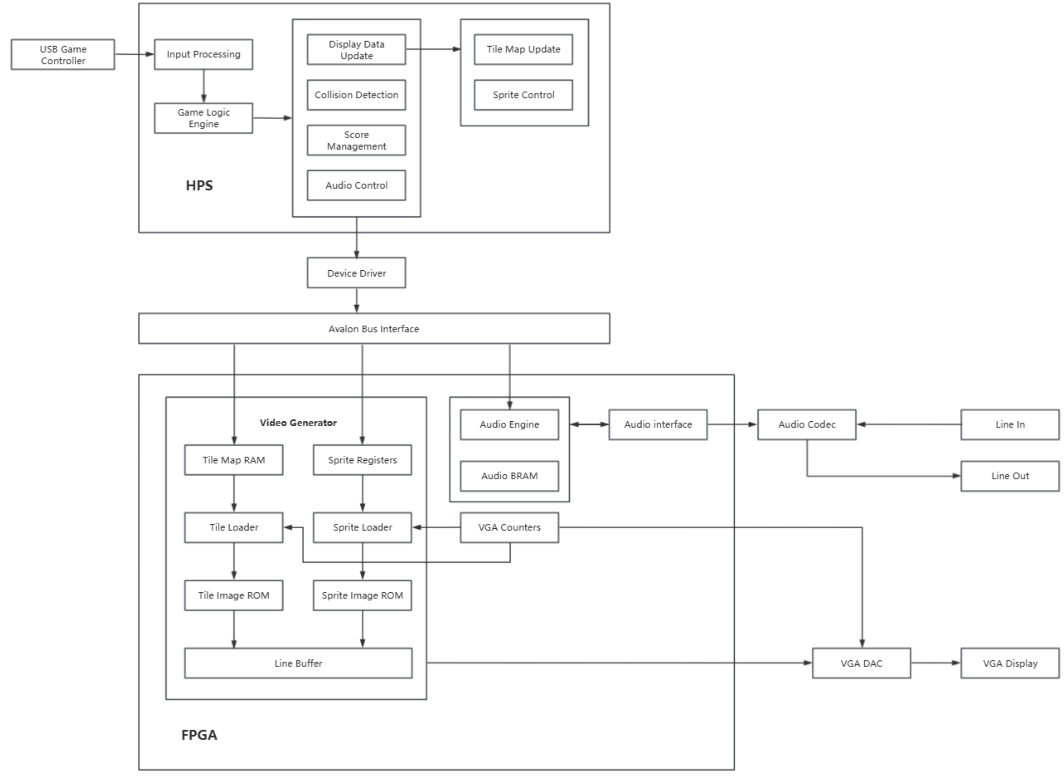

Figure 1. Hardware/Software System Block Diagram for GridBrawl on the DE1-SoC

3. Algorithm

3.1 Hardware

- **Graphics Display**

  Tile-based rendering: The display is implemented using a tile-based architecture on a 640 × 480 VGA screen, organized into a 20 × 15 grid with 32 × 32-pixel tiles.
  Tile and map storage: The game map is stored in Tile Map RAM, where each entry corresponds to the tile type. The corresponding pixel data for each tile is stored in Tile Image ROM and accessed during rendering.
  Sprite handling: Dynamic objects such as players, bombs, and explosions are implemented as sprites. Their states and positions are stored in Sprite Registers, and their pixel patterns are stored in Sprite Image ROM.
  Line-by-line rendering: The system uses a line buffer to generate one scanline at a time. For each line, background tile pixels are first written into the line buffer, followed by sprite pixels to achieve proper layering.
  VGA output: VGA Counters generate pixel coordinates and synchronization signals. The video generator uses these coordinates to fetch tile and sprite data and produces RGB values, which are sent through the VGA DAC to drive the display.

- **Audio Output**

  Audio control: The FPGA audio module receives control signals from the HPS and selects appropriate sound effects based on game events such as explosions, bomb placement, and game over.
  Audio playback: Audio samples are read from memory and managed by the audio controller, which sends the digital audio stream to the WM8731 audio CODEC for digital-to-analog conversion and output.
  Parallel operation: The audio and video subsystems operate concurrently, allowing real-time, graphics rendering, and synchronized sound playback.

3.2 Software

- **Tile Coloring System**

  When a player moves onto an empty tile, the tile changes to that player's color, and the player's score increases by 1. If a player moves onto the opponent's tile, the tile changes to the current player's color. The current player's score increases by 1, while the opponent's score decreases by 1.

- **Map Element**

  The map consists of three types of tiles:
  Empty Tiles: Can be captured by players and change to the player's color.
  Soft Walls: Can be destroyed by bomb explosions. Once destroyed, they become empty tiles that can be captured.
  Hard Walls: Cannot be destroyed or captured, and act as permanent obstacles and map boundaries.

- **Bomb System**

  Players can place bombs on the map. Each bomb produces a cross-shaped explosion covering five tiles, including the center tile and the four adjacent tiles in the up, down, left, and right directions. Bomb explosions can destroy soft walls and eliminate opponents within the blast radius. When an opponent is eliminated, all tiles previously captured by that opponent are cleared.

- **Win Condition**

  The game ends when one player captures 100 tiles. The player who first reaches 100 captured tiles wins.

4. Resource Budget

4.1 Video Resource Budget

Category	Graphics	Size (bits)	# of Images	Total Size (bits)
Player 1		32×32	4	32×32×4×24=98304
Player 2		32×32	4	32×32×4×24=98304
Bomb / Explosion		32×32	4	32×32×4×24=98304
Tiles		32×32	3	32×32×3×24=73728
Walls		32×32	2	32×32×2×24=49152
UI Assets		64×64	10	64×64×10×24=983040
Total	1400832 bits			

4.2 Audio Resource Budget

Category	Time (s)	Frequency (kHz)	# of Bits
Background Music	16	8	16×8000×16=2,048,000
Explosion	2	8	2×8000×16=256,000
Player Dies	1	8	1×8000×16=128,000
Victory	2	8	2×8000×16=256,000
Wall Breaks	1	8	1×8000×16=128,000
Total	2,816,000 bits		

5. Software/ Hardware Interface

The HPS communicates with the FPGA hardware through the Avalon memory-mapped (Avalon-MM) bus. The HPS acts as an Avalon-MM master, while the FPGA exposes memory-mapped slave regions for the Tile Map RAM, Sprite Registers, and Audio Controller.

5.1 Controller interface

The game is controlled through USB gamepads connected to the HPS. The HPS reads controller inputs using Linux USB drivers and translates them into player commands, including movement in four directions and bomb placement.

The HPS maintains the full game state in software. This includes player position updates, collision handling, tile-color updates, bomb placement and timing, explosion effects, player death handling, score tracking, and win-condition checking. Therefore, all high-level game logic is implemented on the HPS.

The FPGA serves as a hardware rendering and audio output engine. It does not process raw controller input directly or make game-logic decisions. Instead, the HPS communicates the current display and sound state to the FPGA through the Avalon-MM bus by writing tile map

data, sprite registers, and audio control registers. The FPGA continuously reads these memory-mapped values to generate VGA video output and audio playback signals in real time.

5.2 Address Mapping

Base Address	Offset	Meaning
0xFF200000	0x000 - 0x12B	Tile Map (300 total, 1 byte each)
0xFF201000	0x000 - 0x057	Sprite 0 – Sprite 21 (4 bytes each)
0xFF202000	0x000 - 0x007	Audio Controller

5.3 Detailed Register Definitions

All display-related states are stored in FPGA memory (Tile Map RAM and sprite registers) and updated by the HPS through the Avalon bus. The full game logic state is maintained in the HPS.

(1) Tile Map Memory

- Total tiles: $20 \times 15 = 300$
- Each tile: 3 bits

Total size (theoretical):

$$300 \times 3 = 900 \text{ bits} \approx 113 \text{ bytes}$$

Although only 3 bits are required, each tile is stored in 1 byte to simplify memory access and alignment.

Total size (actual):

$$300 \times 1 \text{ byte} = 300 \text{ bytes}$$

Tile Map RAM (0x000 - 0x12B)

Each byte in the Tile Map RAM corresponds to a 32x32-pixel grid cell on the 20x15 game map.

- Bits [2:0]: Tile ID (0: Empty, 1: Soft Wall, 2: Hard Wall, 3: Player 1 Color, 4: Player 2 Color).
- Bits [7:3]: Reserved.

(2) Sprite Registers

- Number of sprites: 22
- Each sprite: 32 bits

Total:

$$22 \times 32 = 704 \text{ bits} = 88 \text{ bytes}$$

Sprite Register Interface (0x000 - 0x057)

Dynamic display objects are controlled through memory-mapped sprite registers. Each sprite occupies one 32-bit register in the FPGA peripheral address space. The HPS updates these registers to control which sprites are visible and where they appear on the screen.

Each sprite register contains:

Field	Bits	Description
X position	[9:0]	0-639
Y position	[18:10]	0-479
Sprite ID	[23:19]	22 sprites
Enable	[24]	visible or not
Reserved	[31:25]	Unused

The total size is 25 bits, and each sprite entry is aligned to 32 bits.

The X and Y fields specify the upper-left pixel position of the sprite on the VGA display. The Sprite ID selects the sprite image stored in the sprite image ROM. The Enable bit determines whether the sprite is rendered. When Enable is 0, the sprite is ignored by the rendering hardware.

(3) Audio Controller (0x000 - 0x007)

- Control Register (Offset 0x0):
  Bit 0: Start_Play (Write 1 to trigger sound).
  Bit 1: Loop_Enable (1 for continuous background music).
- Sound Select Register (Offset 0x4):
  Bits [3:0]: Sound_ID (Selects explosion, wall breaks, or victory sound effects).
  Bits [31:4]: Reserved

The Sound_ID field selects the requested sound effect, such as bomb placement, explosion, or game-over audio. To play a sound, software first writes the desired Sound_ID to the Sound Select Register and then writes 1 to Start_Play in the Control Register. Loop_Enable may be used for continuously repeated background audio.

Write behavior: Writing to the Sound Select Register updates the requested sound effect. Writing 1 to Start_Play triggers playback of the selected sound.