Design Document for Crazy Arcade Game

Yuqi Lin (yl5334), Yelin Mao (ym3000), Hongkuan Yu (hy2819)

Spring 2024

Contents

1 Overview

1.1 Short Introduction

The goal is to develop a "Crazy Arcade" game (shown in Figure 1 below) on an FPGA board, leveraging SystemVerilog for hardware implementation and C for the game's algorithm. The game features two players competing in real-time, navigating through a maze, and placing bombs to destroy another player. Bombs will explode after a certain time, with the affected distance as a "+" (up, down, left, and right).

Figure 1: Crazy Arcade game

1.2 Game Rules

At the beginning of each game, players will spawn on opposite corners of the screen, surrounded by permanent obstacles. There are also many paths without obstacles. Players can reach each other through different paths and place bombs, exploding after a set of times, to block another's path. The bomb has an affected distance, and within this distance, the explosion will kill nearby players, including the bomber (suicide). So it is obvious that if obstacles and a bomb stuck the path of a player, then he has no chance but only waits for the explosion and gets killed!

If one player is killed, the surviving player wins the game. If both players are killed simultaneously, the game ends in a draw. After each game concludes, players

respawn at their starting positions with a new initial map with the same layout of permanent obstacles.

2 System Block Diagram

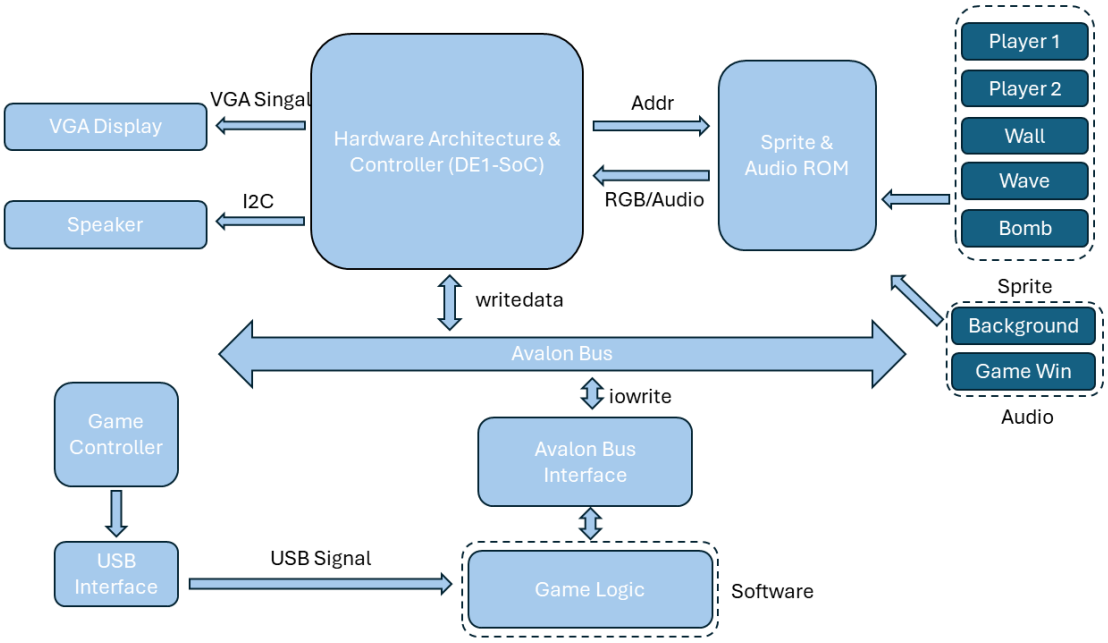

Figure 2: System Block Diagram

3 Algorithms

3.1 Hardware

The primary hardware function involves processing the graphics for display and audio functions. For the graphics, we plan to use the hierarchy the same as what we did in Lab 3. The audio functions may need some extra connections managed in Platform Designer. The hardware parts will all be fulfilled using SystemVerilog.

We will utilize the sprite-tile graphics, rendering the static map using tiles. Each position in the map will correspond to a specific tile, with the pattern name table storing the memory address of each relevant tile, and the pattern generator table housing the pixel data in the VROM of the Altera FPGA. Additionally, preloaded sprites like the characters and bomb will reside in the sprite generator table within the ROM. The sprite attribute table will store both the memory addresses of the graphics and the display positions of the sprites. Sprite display occurs line by line, where the horizontal position is loaded in, and the counter ticks down by the count until it reaches zero, signifying the drawing of the sprite pixel by pixel on the display. The sprite drawing process concludes based on the size of the sprite. Following our resource assessments conducted in the subsequent section, we do not foresee exceeding the FPGA's ROM capacity, thus eliminating the need for additional memory access or the utilization of extra logic to optimize memory usage, such as a color lookup table.

3.2 Software

3.2.1 Player Control

Each character will be controlled via two keyboards or game controllers (depending on if we can borrow them from the Professor or TAs). User input will be controlled through two keyboard inputs: W, A, S, and D are used to move characters; the Spacebar is used for placing bombs at the current positions of characters. Or via two separate game controllers, with $\uparrow, \longleftarrow, \downarrow, \longrightarrow$ for movement and X for placing bombs.

To incorporate various character statuses, both stationary and in motion, a minimum of eight states (four for movement, four for standing) will be required and selected from sprites. The keyboard will be programmed to recognize different inputs from two keyboards without any conflict. Also, two game controllers will be programmed to simultaneously monitor inputs from both.

3.2.2 Player Status

The software will manage and monitor each character's current status, including parameters such as the maximum number of bombs allowed to be placed, bomb exploding range (the affected distance), character speed, and immunity status. Some functions are advanced functions and may only be done if time permits, please refer to the **Advanced Function** section.

3.2.3 Obstacles

The software will govern the permissible movement areas for each character, delineating between accessible and restricted zones. Non-passable obstacles include non-destructible obstacles, destructible obstacles (if time permits), and active bombs. Characters will have free passage through other elements such as special items (if time permits) and opponents. Obstacle detection will occur each time a movement key is pressed for a character.

3.2.4 Bomb Explosion Detection

Bomb Explosion detection will occur each time a bomb is exploding.

If the affected distance in some directions for an explosion includes a permeant obstacle, then the explosion cannot destroy the obstacle, and the explosion should only affect between the bomb and the obstacle.

(If time permits) If the affected distance in some directions for an explosion includes a destructible obstacle, then the explosion can destroy the obstacle but cannot tunnel through the obstacle even if the affected distance is very long. In other words, the destructible obstacle can protect the character behind it only once.

If the affected distance in some directions for an explosion includes a character, then the explosion can kill the character and trigger the "end of game" command.

3.2.5 Map Configuration

The game will adhere to one static map layout, with all non-destructible obstacles. If time permits, we may use a randomly generated map with non-destructible and destructible obstacles. Upon the start of each new game, the same map layout will be generated.

4 Resource Budgets

Each pixel is set to be 24 bits, and the two audio sources use 8 bits per sample. Therefore, the total memory usage is 262.464 kB, which is less than the size of ROM on the FPGA board.

Video			
Image	Pixel Size	# of Items	Total bits
	24*24	1	24*24*24 = 13824
	24*24	8	24*24*8*24 = 110592
	24*24	1	24*24*24 = 13824
	24*24	1	24*24*24 = 13824
	24*24	1	24*24*24 = 13824
	24*24	1	24*24*24 = 13824
Audio			
	Time	Rate	Total bits
BGM	15 s	8 kHz	8k*8*15= 960000
Game Win	15 s	8 kHz	8k*8*15= 960000
Total			**2099712 (bits) = 262464 (bytes)**

Figure 3: Resource Budget

5 The Hardware/Software Interface

Our hardware-software interface will function as follows: The essential sprite data will reside in the video ROM on the FPGA. The static graphics representing the map walls will be presented in a predetermined grid format using tiles. Memory addresses corresponding to these tiles will be stored in a pattern name table. Byte-addressable memory locations will map to both the pattern name table and the sprite attribute table. Altering the positions of sprites will be accomplished via software by writing data to the appropriate byte address in the ROM (passing the coordinates from the software to the hardware).

Given the frequent updates required for sprite positions during gameplay, we aim to simplify this process from the software perspective. This involves passing only the address to be written to and the new coordinates when updating the displayed elements and their locations. To facilitate this, we will ensure that the size of the data passed to the VGA module is 32 bits. This allows us to include all necessary information, such as the new positions for both tanks and the position of any bullets. The 32-bit value will be dissected into its respective components and then written to the corresponding addresses in the VROM.

5.1 Register file

Register1: Player 1 x position

Register2: Player 1 y position

Register3: Player 2 x position

Register4: Player 2 y position

Register5: Player 1 bomb x position

Register6: Player 1 bomb y position

Register7: Player 2 bomb x position

Register8: Player 2 bomb y position

Register9: Offset of script

Register10: Game status

Register file: position of the bricks

6 Advanced Functions

If time permits, we plan to add more uncertainties to the map, including:
1. Destructible obstacles and randomly generated maps
2. Special power-up items
3. Special sound effects for the bomb explosion

For destructible obstacles and randomly generated maps, we plan to write a random coordinate generator in software to tell the hardware which coordinates should be implemented with the destructible obstacles.

Special power-up items are hidden randomly within destructible obstacles, offering players both advantages and disadvantages. These items can have either temporary or permanent effects throughout the game. Below is a list of potential items we're considering implementing.

The buff items we are considering:
1. Permanent increase in the affected distance after the bomb exploded
2. Permanent increase the maximum number of bombs placed (originally each player could only place one bomb)
3. Permanent increase in player's moving speed
4. Temporary ability to place bomb which has full length and full width affected distance after it exploded
5. Temporary immunity against bombs

The de-buff items we are considering:
1. Temporary decrease in player's moving speed
2. Temporary decrease in the affected distance after the bomb exploded
3. Temporary reverse the direction control

7 Milestones

7.1 Milestone 1

In our first milestone, we want to implement the game controller to at least control 1 player to move in the game. The pattern of the 1 player can work well. We will try to add the background audio in this milestone.

7.2 Milestone 2

In Milestone 2, we will ensure the player can place the bomb, and the bomb can explode successfully. We also need to add some sound effects in the second milestone, such as the bomb explosion sound.

7.3 Milestone 3

In Milestone 3, we will add another player to the game. The 2 players need to work together without misinformation. We also need to design the player that can be killed by the bomb. The ultimate goal is that the game can work smoothly.