Design Document

Natalie Hughes, Qian Zhao, Shiyan Wang

Spring 2024

Contents

1 Introduction

This design document outlines the development of a Bomberman video game implemented on an FPGA platform. The game will utilize VGA for video output, a keyboard for user input, and FPGA-based hardware for game logic and processing. Bomber man is a 2D single or multi-player game which was developed by Hudon Soft and has been distributed in many versions. In this game, the players use bombs to clear the obstacles and 'kill' other players. The last player who survives from the bombs will win. It involves the player navigating through the maze, using bombs to eliminate obstacles and enemies.
The USB joystick connects to the user space program of the game logic via the USB protocol and the libusb library. The game logic, in turn, communicates with the FPGA hardware through the vga ball device driver and the Avalon bus interface. This facilitates control over the VGA monitor hardware peripheral via the vga ball module, which is integrated into the FPGA.
The software components comprise the primary game logic contained within a .c file, responsible for managing character movement, bomb placement, explosions, scoring, and other game mechanics. Another .c file, dedicated to USB functionality, will interpret inputs from USB joystick controllers to facilitate the execution of the game logic. Lastly, the VGA device driver implemented in the vga.c file will interact with the FPGA to update the graphics displayed on the VGA monitor according to the game's logic.

2 System Block Diagram

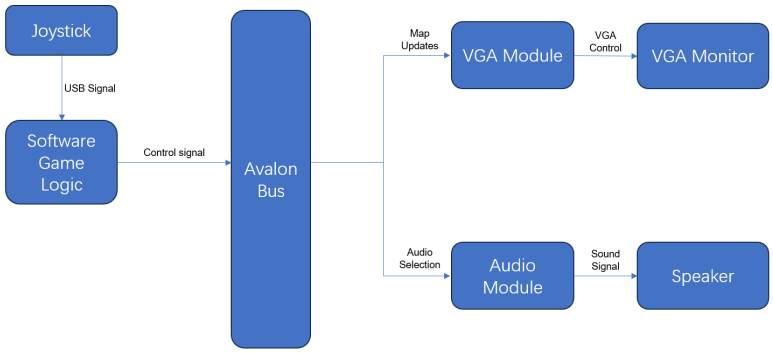

Figure 1: System Diagram

The system diagram is shown in Fig.1. The software game logic will first process USB input from the user, update the game-play logic and send corresponding

control signal to the Avalon bus using iowrite16. The control signal will write 18 * 16 bit memory space which corresponds to 18 16-bit wide registers in main.sv. The first 4 registers will be used to denote the status of two players. Then the next 4 registers will represent the explosion effect happening in this game tick. The following 2 registers will represent the bombs placed in this game tick. And the last 8 registers will represent the boxes destroyed in this tick (because the game logic allows 2 explosion in one tick and each explosion can remove up to 4 boxes). Also, when there is a placement or a explosion effect. The software game logic will be in charge of all the game rules, such as collision, explosion, movement of players and game over. The signal from the Avalon Bus will then be processed by main.sv to output RGB signals and audio selection signals to output the image and the audio.

2.1 VGA module

Fig.2 shows the block diagram of the VGA module. The map is represented in map.v, which is equivalent to a 1200 * 3 bit RAM block. 1200 rows mean that there are 40 * 30 blocks with size 16 * 16 pixels within a 640 * 480 resolution map. 3 bits means that there are up to 8 elements in the map, which are background, non-removable bricks, removable boxes, bombs, explosion effect and other props. These blocks are stored in ROM files, with 16 * 16 * 2 bits representing the graph of the block, which means that each block has at most 4 colors. There are at most 256 colors in the game in total, which means that the color map is 4 * 8 bits. The VGA module will first convert the row number and column number of pixel to the type of the block and the position within the block. Then it will use the type to find the specific ROM that represents the block, use the position info to index into the graph to get the 2-bit color and at the same time translate the 2-bit color into 8-bit color using the color map in the same ROM. Finally it will use these 8-bit color to index into the color ROM to find the RGB values that the 8-bit color corresponds to. To display the player, the procedure is similar except that the position is the center of the player in pixel location, and there are four facings for a single player and two players, so players need 4 * 16 * 16 * 2 graphs and 4 * 8 * 2 color map. There are two layers in the map, the top layer is the player and the bottom layer is the map.

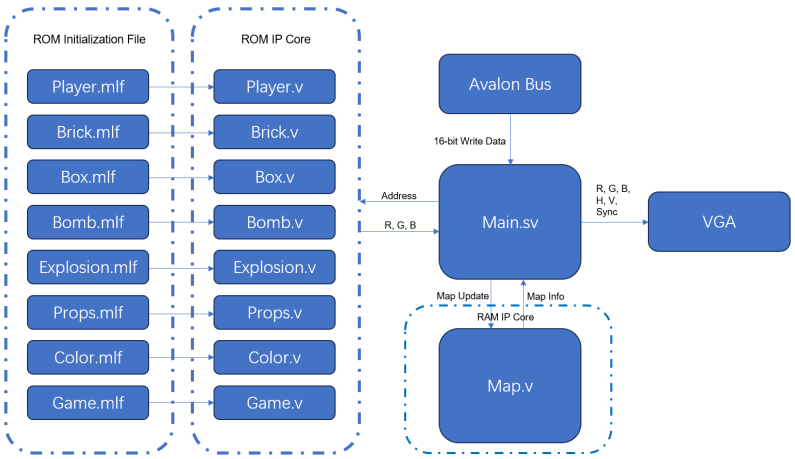

Figure 2: VGA module Diagram

2.2 Audio module

Whenever the bomb placement and explosion register is non-empty, or one player is dead, the main.sv will send a sound select signal to the audio-generation module to start the sound effect. If more than two of three effects are triggered, the priority sequence is game-over, explosion, placement. The data is then read from ROM in FIFO order and then fed into the audio IP core and then will be translated to DAC input signal of WM8731 chip. The config.v will hard-code the configurations such as volume and transmit it to WM8731 chip using I2C protocol. The DAC will then output the sound to the speaker. The explosion effect is 0.5 second, the placement is 0.3 second and the game-over is 1 second. Assuming a sample rate of 40kHz and sample depth of 8 bits, this 1.8s audio will occupy 576000 * 2 bit which is about 144 kB. If the sound effect is too large, reducing sampling rate or deducing the length can be considered.

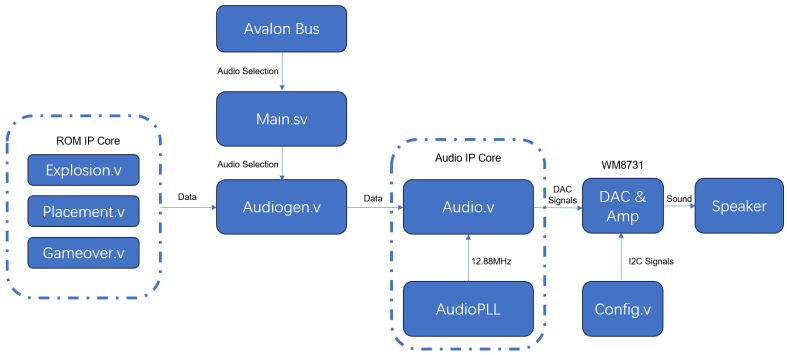

Figure 3: Audio module Diagram

3 Algorithms

3.1 Hardware

The logic to display graphics.

- Sprites: All sprites in this game, such as players, bombs, bricks of the maze and explosion effects, will be stored in ROM. There will be indexes to read sprites from ROM.
- Screen: The screen is going to be divided into 40×30 tiles. Each tile has a size of 16×16 pixels.
- Maze: The map will be fixed and static. The information about the maze will be stored in ROM.
- Ending: Use text to represent the final winner on screen. This can be coded in the hardware codes.

The logic about audio.

- Audio data is read from ROM in FIFO order as needed. This data may include explosion sound effects, placement sound effects, and game over sound effects. The audio data read from the ROM is input to the audio IP core. The audio IP core may use digital signal processing (DSP) algorithms to process the audio data, such as volume control or audio effect enhancement. The processed audio data is converted into an input signal for a digital-to-analog converter (DAC), which converts the digital signal into an analog voltage signal. The analog voltage signal is output to the speaker via the WM8731 chip.

3.2 Software

- Generate player: 2 players per game.
- Player Movement: The characters are only allowed to move inside the maze map and are not allowed to pass through the walls. The characters are allowed to move in 4 directions (up, down, left and right). The algorithm is going to track the positions of the characters as well as the position of the bombs.
- Overlap: Players are allowed to overlap, while players can not walk through walls, obstacles and bombs.
- Bomb placement and explosions: Characters can place bombs in the maze. Once a bomb is placed, it will automatically boom in 3 seconds and the players and the obstacles within the bomb location and its surroundings will be 'killed' or removed.
- End of game: The player who is 'killed' first will lose the game and the other player will win.

4 Resource Budgets

Category	**Size(bits)**	**Number of Images**	**Total(bytes)**
Block(map obstacle)	16*16	2	128
Map	40*30	1	450
Player	16*16	2	64
Bomb	16*16	2	64
Explosion	16*16	2	64
End Screen	32 * 16 * 16 * 4 + 4 * 8	1	4100
Color	256	24	768
Props	16 * 16	2	64
Sound			144000

The table outlines various game element categories, including blocks for map obstacles, the map itself, players, bombs, explosions, and the end screen. The exact number of block may be increased later if time and memory space permits. If the sound is too large, the sample rate and duration can further be cut.

5 The Hardware/Software Interface

The game's hardware interface consists of multiple key registers with different purposes. The 11-bit Position Register holds logarithmic coordinates describing the positions of blocks and players on the map, while the 3-bit Map Data Register identifies the kind of block present in the

map. Each player has 26 bits in the Player State Register, which contains fields for their X and Y coordinates, facing direction, explosion radius, number of bombs left, and player status. Moreover, graph and colormap displays can be enabled or disabled using the 2-bit Display Control Register. While Status Registers communicate game status information, such as game over or bomb detonation, the 1-bit Bomb Placement Register starts the placement of bombs based on the player's facing direction. Lastly, the transmission of updated block data for communication with other hardware components is made easier by the 14-bit Avalon Bus Register.