Design Document for Atari Breakout: Candy Edition

Jason Eriksen (jce2148)
Xurxo Riesco (xr2154)

Spring 2022

Table of Contents

1 Introduction

For our version of Atari Breakout, we plan to implement a variety of components. Several components will be modified versions of those completed in the prior labs (i.e. vga_ball), but we also develop several completely new components, like audio and an interface via joystick instead of keyboard. On the software side, we will utilize a driver similar to that of Lab 3, but with updated logic to reflect the rules of Atari Breakout, such as the shattering of bricks, bounces on the paddle, and the changes in speed of the ball. Also, we plan to include separate display modes for menu, victory, and defeat screens. Additionally, we will develop the aforementioned user input driver that maps joystick movements to in game actions. This will all be communicated to the hardware via the avalon bus. The hardware will include the video logic for displaying the ball, paddle, bricks, as well as the audio logic for the sound effects.

2 System Block Diagram

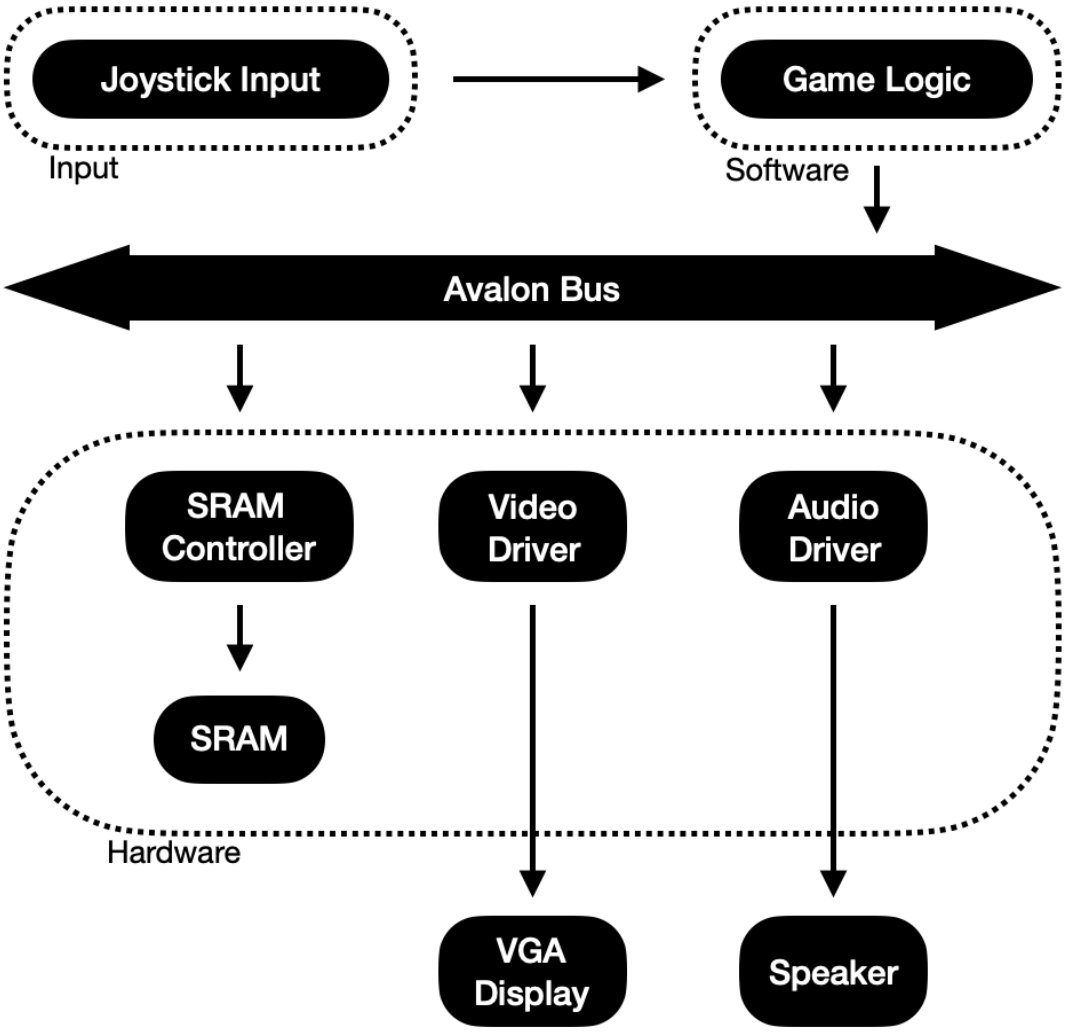


Figure 1: System Architecture

3 Algorithms

3.1 Software Algorithms

3.1.1 Game rules

The paddle is only allowed to be moved along the horizontal axis. Along the vertical axis, the rules are simple: if the top part of the ball touches the bottom part of a brick, the brick is destroyed, if the bottom part of the ball goes below the vertical position of the paddle while not making contact with it, the player loses a life. If the player is able to remove all the bricks before running out of lifes, the player wins the game, in contrast, if the player runs out of lives while there is one or more bricks in the screen, the player loses the game.

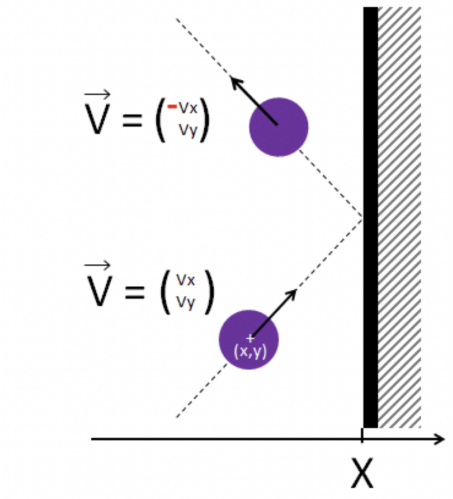
3.1.2 Bouncing Ball

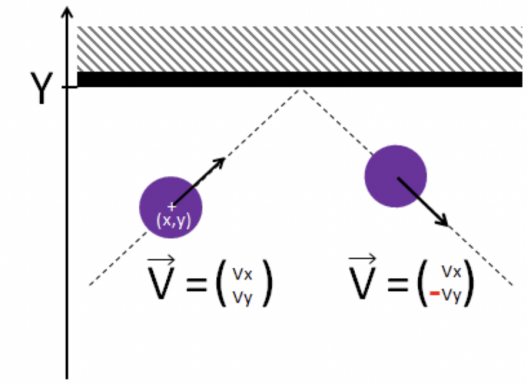
The movement of the ball is determined by the position of the ball along both axes, (x,y), and the velocity of the ball (v_x, v_y) in the following manner.

Position at t_0: (x, y)

Position at t_1: ($x+v_x$, $y+v_y$)

When the ball contacts the paddle or the bricks as defined in the Game Rules (3.1.1), the v_y component of the velocity is reversed via negation. Similarly, if the leftmost or rightmost point of the ball makes contact with the left or right wall, the v_x component of the velocity is reversed via negation. The contact with the right wall and brick are portrayed below, with contact with the left wall and the paddle being direct opposites.





3.1 Hardware Algorithms

3.1.1 Graphics

The hardware just receives the center position of the ball and center position of the paddle, with the software being responsible for all the calculations. In addition to that the software receives a bitfield containing the blocks per row and is then responsible for displaying said blocks in the screen, giving each of the blocks in a given row a particular design. The design for the blocks, the ball, and the paddle will be stored as sprites in the memory of the FPGA as well as the necessary game information, such as the lives. The hardware is then responsible for bringing the correct graphic at the positions determined by the software.

3.1.2 Audio

The hardware will also be responsible for storing 4 different sounds, ball bouncing off the paddle, ball bouncing off the wall, brick being destroyed, life being lost. The software will determine when each sound needs to be played and convey said information to the hardware while it is handling the game logic.

4 Resource Budget

4.1 Video

Image	Dimensions	# of Sprites	Size (bits)
Blocks	64 x 32	4	8,192
Ball	16 x 16	1	256
Paddle	128 x 16	1	2,048
Lives	32 x 32	1	1,024
Start/Win/Loss	64 x 64	3	12,288
		Total bits:	23,808

4.2 Audio

	Life Lost	Brick Destroyed	Wall Hit	Paddle Hit
time(s)	1.5	0.25	0.25	0.25
f_s(kHz)	8	8	8	8
memory	196,608	32,768	32,768	32,768
			Total bits:	294,912

4.3 Total bits

The total memory budget adds up to 318,720 bits.

We are well below the memory budget available, so it should be no issue implementing components of the aforementioned sizes.

5 The Hardware/Software Interface

The total dimensions of the screen are 640 x 480. Therefore, 19 bits are needed to encode any given position. 10 bits for x component ($640 < 2^{10}$) and 9 bits for the y component ($480 < 2^{9}$)

Register 1: Ball X Position

Stores the lowest 8 bits of the x component of the ball position.

Register 2: Ball Y Position

Stores the lowest 8 bits of the x component of the ball position.

Register 3: Paddle Position

Stores the lowest 8 bits of the x component of the paddle position. Position in y is fixed.

Register 4: Offsets + Lives

Bits 0-1: Remaining upper 2 bits of the x component of the ball position.

Bit 2: Remaining upper bit of the y component of the ball position.

Bit 3-4: Remaining upper 2 bits of the x component of the paddle position.

Bit 5-7: Each bit represents one of the lives.

Registers 5: Bricks Row 1

Bitfield representing the status of each of the 8 blocks in row 1. (1 exist, 0 broken)

Register 6-8: Brick Rows 2-4

Same as Register 5 for the remaining rows.

Register 9: Game Status & Control

Bits 0-1: Represent game status. (0 not started, 1 playing, 2 win, 3 loss)

Bits 2-3: Represent audio to play (4 sounds)

Bits 4-7: Unused

6 References

- https://www.101computing.net/bouncing-algorithm/
- http://www.cs.columbia.edu/~sedwards/classes/2022/4840-spring/designs/Breakout.pdf
- http://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/designs/BrickBreaker.pdf