Design Document for Breakout game project

Youfeng Chen (yc3999), Angzi Xu(ax2157),

Zheyuan Song(zs2527), Wang Chen(wc2794)

Spring 2022

1. Introduction

Our goal is to clone the famous Breakout game and let it run on our DE1-SOC board. Our design includes hardware and software parts, and they will communicate using several communication pathways. The hardware part will be responsible for having essential elements ready, like a ball, blocks, and paddle, and software will control the position or show/not show those elements. Something that does not need to be communicated between software and hardware, like game background and ball hit determination, is internally processed and will not be sent through paths.

For the hardware part, we plan to draw a ball, a few blocks, a paddle, and a game background and have them ready to be controlled by our software, except for the background, which will be static. Then the hardware part will determine the VGA signal to send to the monitor to have our game correctly displayed, using the method that we have practiced a lot in our previous labs. On the other hand, we will have four different sound pieces ready in our hardware which will be played when the ball hits a brick, the paddle catches the ball, win, and game over. The software also controls the time to use sound effects. We plan to use the 3.5mm audio output to connect a speaker.

Our software is responsible for managing almost everything about the game. It will take and process the player's input, determine the game logic, and send out the control command. First, it will take and translate keyboard input from the USB port, like what we have done in Lab 2, and send the input information to the main logic. The main reasoning will determine the ball's real-time position, paddle's position, which brick should be taken out, determine if the ball hits something, display some texts if necessary, decide if some sound needs to be made, based on the player's real-time input and internal computing result. Then those results will be sent to the driver, and the driver will send them to the hardware part.

2. System Block Diagram

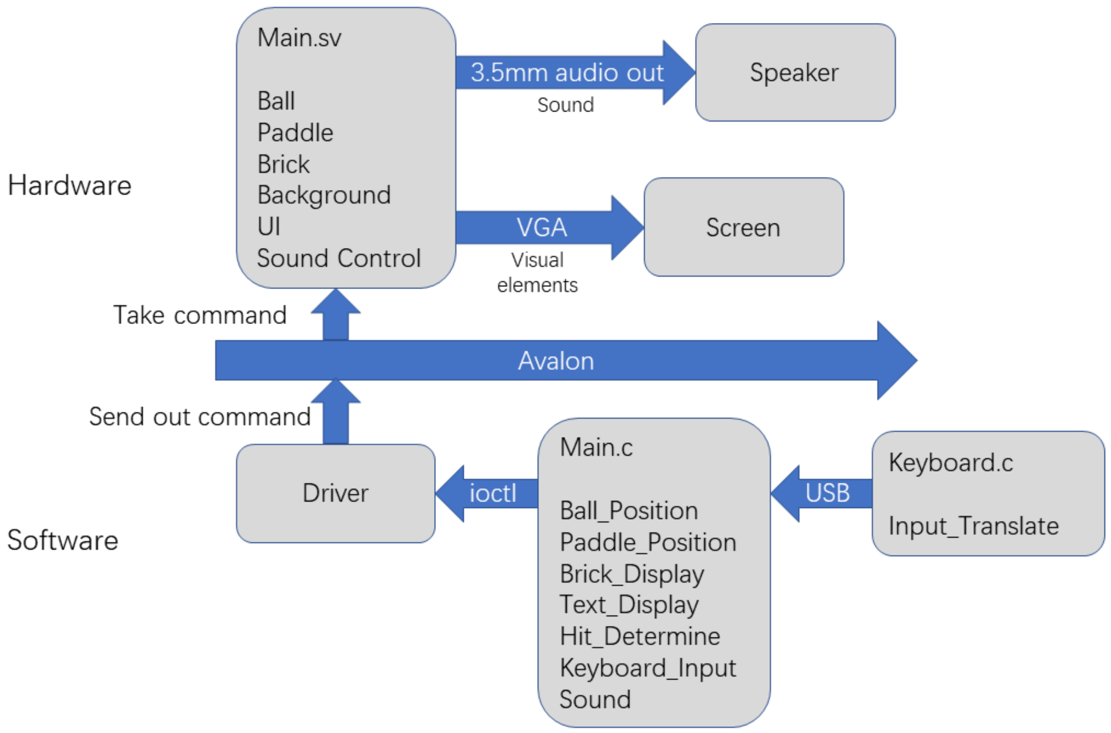


Figure 1

3. Algorithms

3.1 Game rule

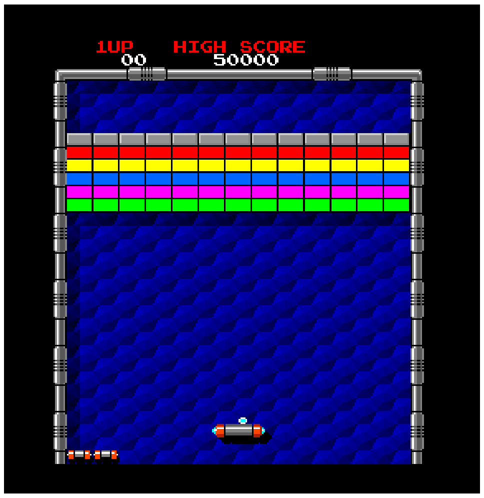


Figure 2

As shown in figure 2, the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally. If the player could destroy all bricks, the player wins. If the player cannot catch the ball with the paddle, the game is over. In particular, there are several conditions.

(1) When the ball hits the bricks, the brick breaks, and the ball bounces.

(2) When the ball hits the wall or the paddle, the ball bounces.

(3) When the ball passes the paddle and falls out of the screen from the bottom, the game is over.

3.2 Hardware Algorithms

There are two essential parts, including the screen and speaker.

(1) Screen part: based on the ball's position, paddle, and block matrix, we will initialize the position of the state of the ball, paddle, and blocks on the screen.

(2) Speaker part: four sound effects will be ready to be played, which are the sounds of the ball hitting a brick, the paddle catching the ball, wins, and the game is over. The software decides when to play those sounds.

3.3 Software Algorithms

As shown in Figure 2, there are several steps.

Step 1: Ball, Paddle, and Block Matrix Initialization.

In this step, we will initialize the position of the state of the ball, paddle, and blocks.

Step 2: Draw Block, ball, blocks, and paddle.

Based on the ball's position, paddle, and blocks, we can visualize the ball, paddle, and blocks.

Step 3: Ball Move.

Based on the signal from the keyboard, we can move the ball and paddle.

Step 4: Ball and Block Crash Detection.

Based on the ball's position, we can check whether there is a crash between blocks and the ball. Then, we update the block matrix.

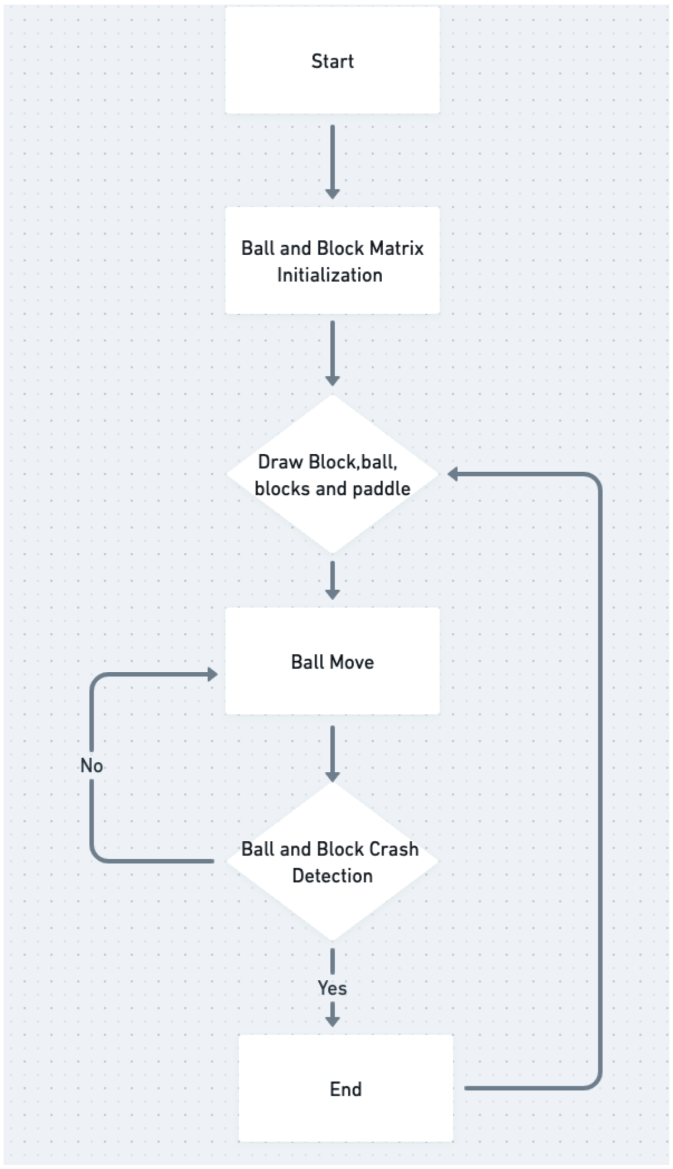

Start
Ball and Block Matrix Initialization
Draw Block,ball, blocks and paddle
Ball Move
No
Ball and Block Crash Detection
Yes
End

Figure 3 Algorithms of the game

The definition of class will be shown as follow.

```
struct Ball
{
   int   x;
   int   y;
} ball;

struct Paddle
{
   int   x;
   int   y;
} paddle;

struct Blocks
{
  int positions[100][200]; // 0 is empty 1 is block
} block;

static void paddle_move(paddle *p, int add_x, int add_y);
static void ball_move(ball *b, int add_x, int add_y);
static void crash_detection(ball *b, block *my_block);
static long game_ioctl(struct file *f, unsigned int cmd, unsigned long
arg);
static void write_background(ball *b, block *my_block, paddle *p);
```

4. Resource Budgets

our project needs to do a budget estimate of the resources. Both graphics images and sampled audio will consume some memory resources, which should be less than half a megabyte in total. That will be a big challenge, so we need to plan the memory resources before doing the project.

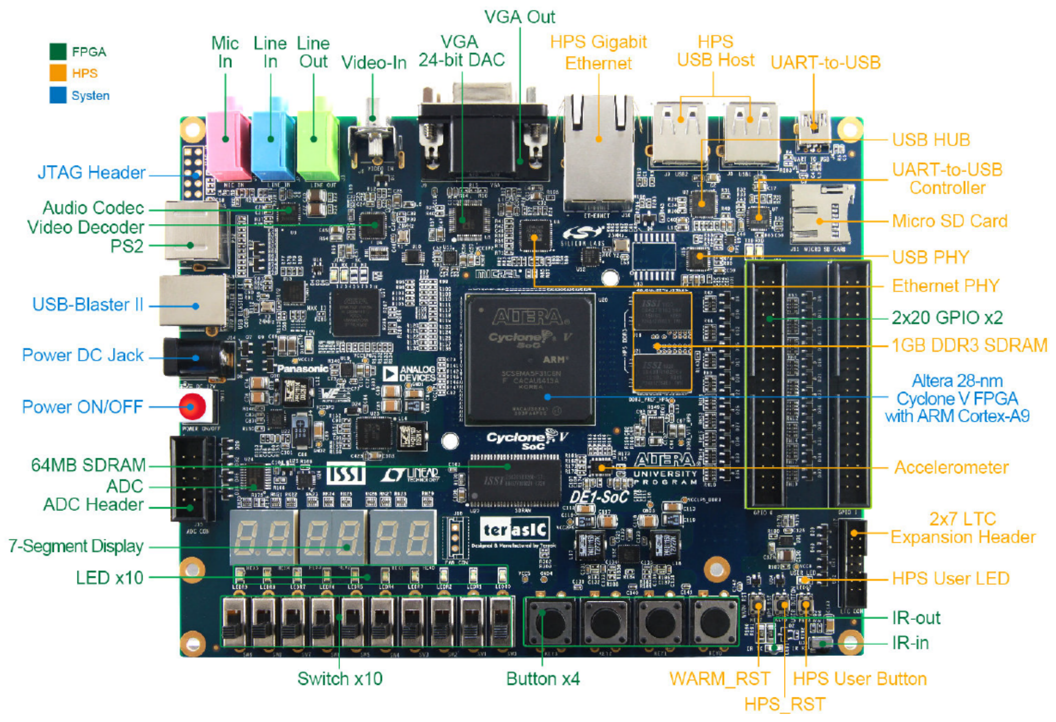


Figure 4-1 Block diagram of DE1-SoC board

The DE1-SoC board with hard processor ARM Cortex-A9 Dual-Core offers these memory devices: 1GB(2*256M*16) DDR3 SDRAM on HPS and 64MB(32M*16) SDRAM on FPGA and Micro SD Card Socket on HPS. These are the memory resources provided to our project. This FPGA Device includes: Cyclone V SoC 5CSEMA5F31C6 Device; Dual-core ARM Cortex-A9 (HPS); 85K Programmable Logic Elements; 4,450 Kbits embedded memory; 6 Fractional PLLs; 2 Hard Memory Controllers. Therefore, the biggest concern for the project is that the embedded memory in the DE1-SoC board is 4,450 Kbits, which limits the maximum size of graphics and audio storage.

·Hardware Graphic Memory Budget

The sprites we use have different sizes: 64*32; 32*32; 100*32; 45*60 pixels. Because the primary colors, Red, Green, and Blue, have 8 bits each, each pixel requires 24 bits of memory to store. As shown in the detailed calculation below, the total image storage memory required for Breakout Game is 313632 bits. This memory size is less than 4,450 Kbits, so the solution is feasible.

Classification	Graphics	Pixels	Size(bits)
Brick1		64*32	49152
Brick2		64*32	49152
Brick3		64*32	49152
Ball		32*32	24576
Paddel		100*32	76800
Game End		45*60	64800

Figure 4-2 Graph Memory Budget

·Hardware Audio Memory Budget

Put the digitized audio data into the audio interface and then store the sound data in the SDRAM, assuming that the sound data per second requires 120KB of memory. After that, Breakout Game will use two effect music and one game over background music in the game. Sound Effects: Ball hitting paddle (0.3 seconds), ball hitting bricks (0.3 seconds), and game over music (3 seconds). As shown in the detailed calculation in the figure below, the total sound storage memory required for Breakout Game is

432000bits. This memory size is less than 4,450 Kbits, so the solution is feasible.

Classification	Brick hit	Wall hit	End game bgm
Time(s)	0.30	0.30	3.00
Memory(bits)	36000	36000	360000

Figure 4-3 Audio Memory Budget

5. The Hardware/Software Interface

I. Hardware

In this game, we can use the hardware control similar to what we have done in Lab 3, which lets the software send directions to the hardware to control those pre-made sound/visual elements.

1. Register datawrite[X:0]

   An X-bit register for commands from the software, X depends on how much data the hardware needs to receive.

2. Register address[3:0]

   A 3-bit register for address, helping the hardware to know how it should react to data from datawrite.

3. Register VGA_R/G/B[7:0] & other 1-bit VGA_x control signals

   Registers for VGA display output.

4. Register hori_ball[10:0]

    Bit [10:0] indicates the horizontal coordinate of the ball, ranging from 0 to 1280.

5. Register vert_ball[8:0]

    Bit [8:0] indicates the vertical coordinate of the ball, ranging from 0 to 480.

6. Register hori_peddle[10:0]

    Bit [10:0] indicates the horizontal position of our peddle, ranging from 0 to 1280.

We will also add a score controller, which consists of two parts. One automatic adder will let the score plus one whenever the ball hits a brick. And the other one is a MUX, and it will send the display signal to the HEX.

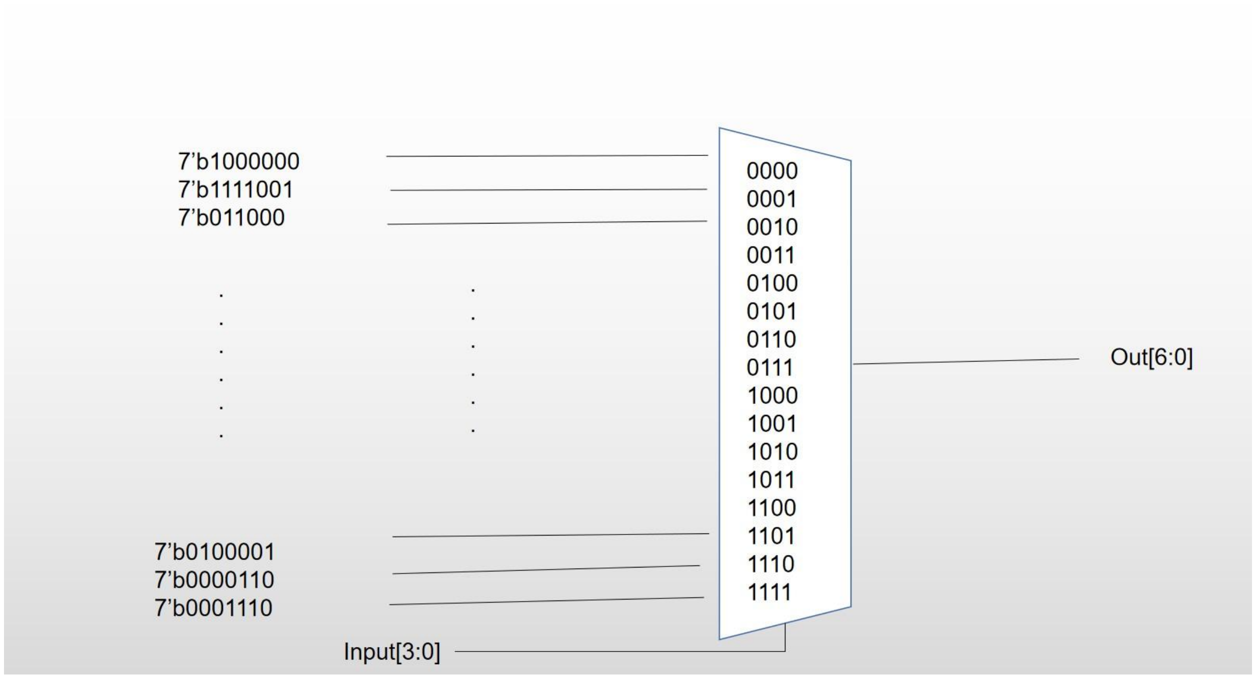
The MUX has 32 cases. 4-bit SEL inputs, and a 7-bit output.

7. Register score_input [4:0]

    Bit [4:0] indicates the score number, ranging from 0 to 32. It is also the SEL input of the MUX

8. Register score_output[13:0]

    An 14-bit MUX output to control the MUX display. Bit [13:7] controls the left digit, and bit [6:0] controls the right digit (two digits to be displayed in total).



II. Software

First of all, for the control of the bouncing ball, we can learn from the method in lab3 to control the position of the bouncing ball and do a similar thing to control the position of the peddle and control the removal of bricks.

For the software side, we need to know which kind of object is hit by the ball, either peddle, brick, or wall. We plan to let the ball bounce first when it hits something and then judge what is hit. If it is a brick, the software will change the state of the hit brick to let it disappear. If the item is the peddle or a wall, the status of the hit item will remain unchanged.