Brick Breaker Game Design

Bingyao Shi (bs3119)
Rui Chen(rc3205)
Shao-Fu Wu(sw3385)
Dajing Xu (dx2178)

1. Introduction (project overview & game rules)

We will implement a simple brick breaker game. We have layers of colored bricks and ball with which to break the layers. The player moves the paddle from left to right to keep the ball from falling. A life is used when the player fails to hit the ball.

The paddle doesn't bounce the ball like a mirror, although it does so when the ball hits right in the middle. The closer the bounce take place to the left end of the paddle, a more significant left turn is added to an expected mirror bouncing.

A regular brick disappears when it's hit by the ball, or breaks a little if it's a bulkier brick.
At the same time, there will be music outputs to match different game effects. The basic user interface may be similar to Figure 1. If time permits, we plan to add more features to the game, such as bricks that drop items to make the paddle longer, make the ball faster, or turn the ball on fire so that it burns every brick on the path.

We will build a scoring system that reflects how efficient the user is to clear the bricks. Number of total paddle hits and maximum brick hits per trip are two of the possible factors.

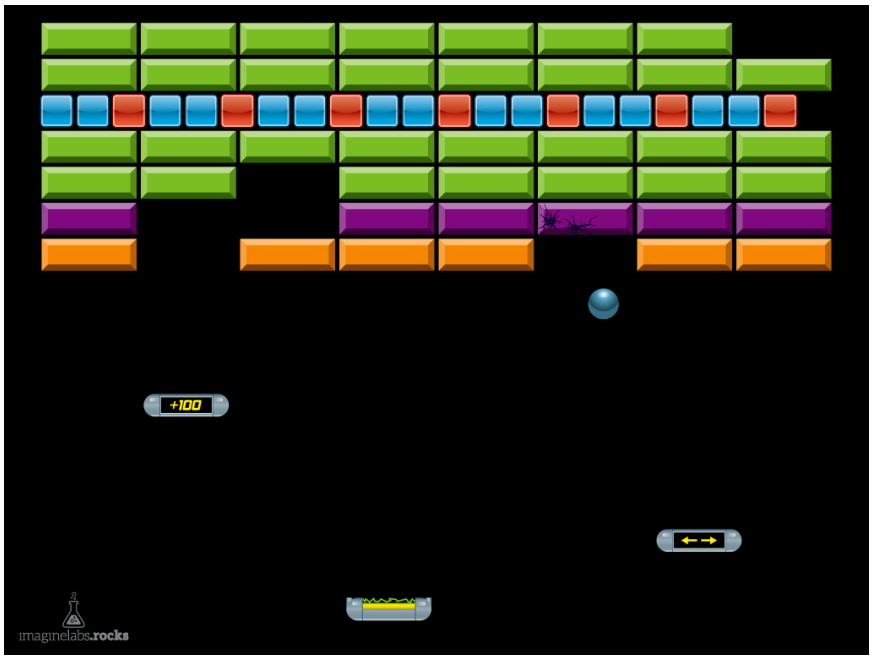

Fig.1 Sample interface from the internet[1]

2. Architecture

The below figure shows the architecture we are going to use for this project.

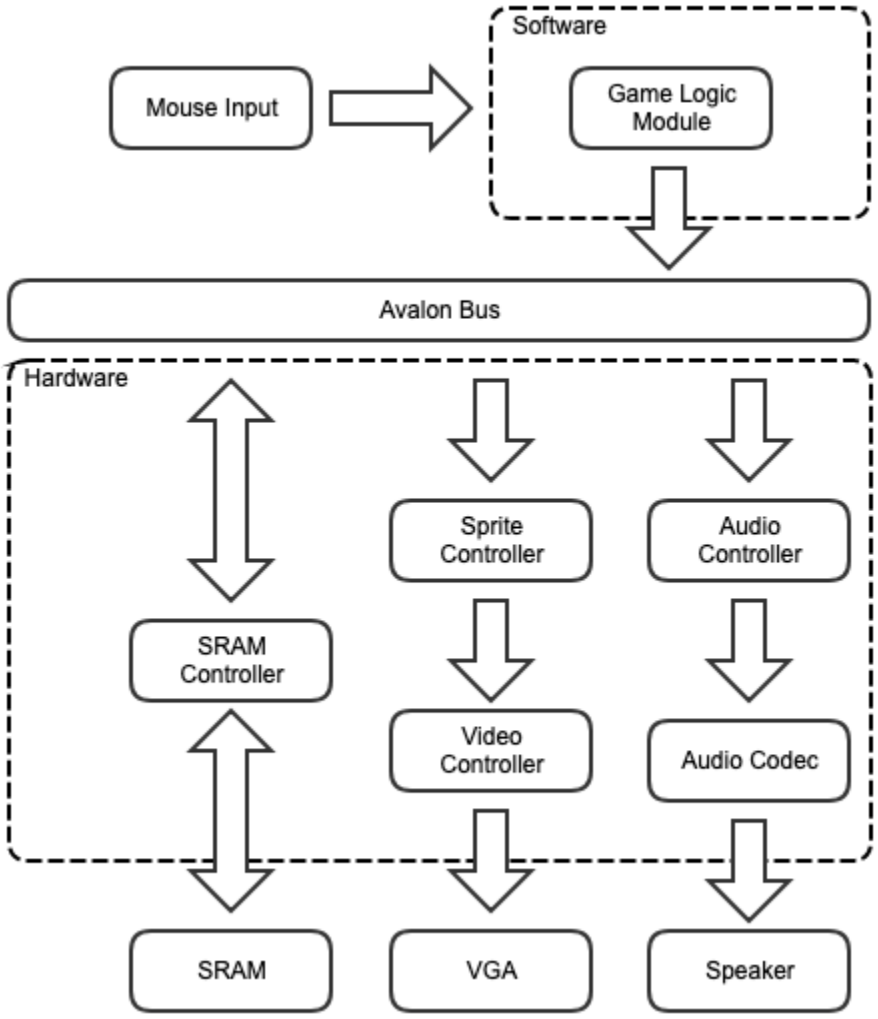

Fig.2 System architecture

3. Hardware Implementation

A. VGA Block

From a design point of view, we plan to include 4 layers of graphics. At the bottom, we will display the background picture. In the middle 1 layer, there will be different kinds of destroyable bricks, the bouncing ball and the paddle. The middle 2 layer displays bonus effects falling off.The top layer will include score counting, current game level and how many lives left for this level.

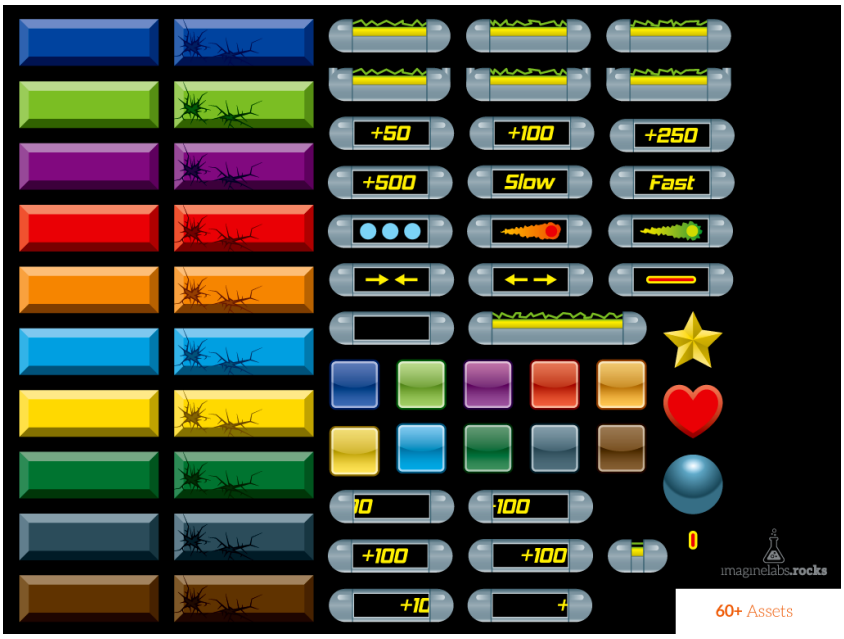

Fig 3. Open-source sprites[1]

B. Audio interface in FPGA

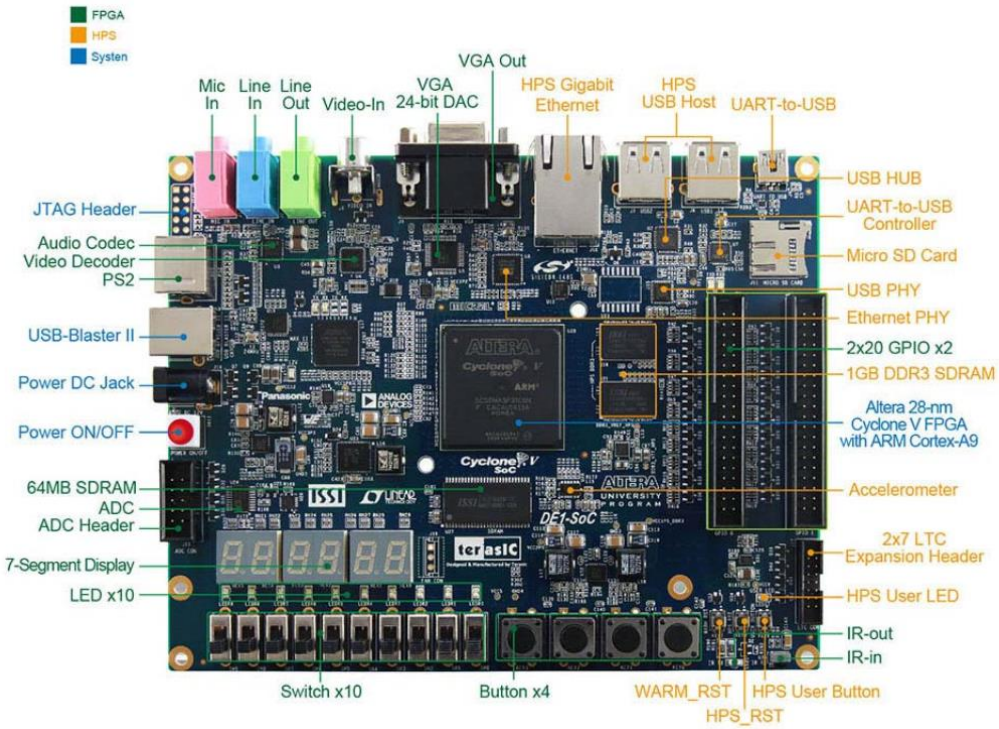

Fig.4 Block diagram of DE1-SoC board

The DE1-SoC board offers high-quality 24-bit audio via the Wolfson WM8731 audio CODEC (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports, with adjustable sample rate from 8 kHz to 96 kHz. The WM8731 is controlled via serial I2C bus, which is connected to HPS or Cyclone V SoC FPGA through an I2C multiplexer. The connection of the audio circuitry to the FPGA is shown in **fig.5**, and the associated

pin assignment to the FPGA is listed in **fig.6**.[2] In our case, a sample rate of approximately 40kHz should be applied which corresponds to human's hearing range.

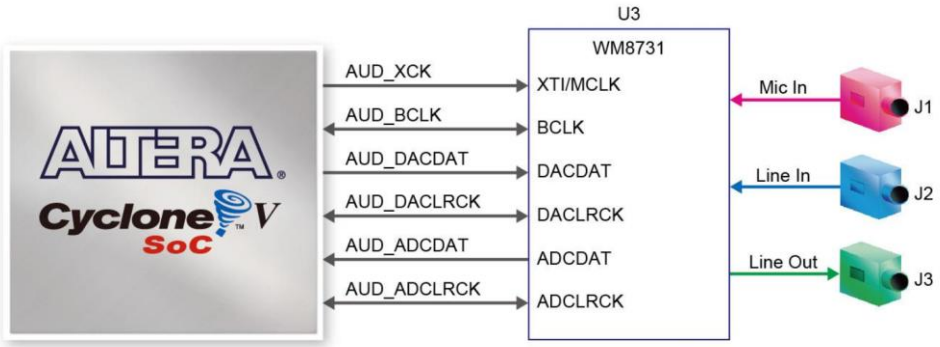

Fig.5 The connection of the audio circuitry to the FPGA

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_K8	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_K7	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_H8	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_J7	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_G7	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_H7	Audio CODEC Bit-stream Clock	3.3V
I2C_SCLK	PIN_J12 or PIN_E23	I2C Clock	3.3V
I2C_SDAT	PIN_K12 or PIN_C24	I2C Data	3.3V

Fig.6 associated pin assignment to the FPGA

Below is the block diagram of CODEC WM8731.

Fig.7 Block diagram of WM8731 CODEC

Configure the codec through the I2C interface. The I2C interface requires two signals: one for clock and another one for data. In the idle state both signal are high. By pulling the data line LOW while the clock line remains HIGH is the start condition. Then the master sends an address of a device that it wants to talk to. After a 7-bit address and a R/W bit are sent, the master releases the data line. The data line has a pull up resistor, thich pulls the line HIGH while it's state is not actively controlled. If an I2C device receives its address, it pulls the data line LOW at the next clock cycle. Then two byte of data is sent. After all the data are transmitted, a stop condition is generated by pulling the clock line High, while the data line is kept LOW.

The WM8731 should be configured at I2S mode.

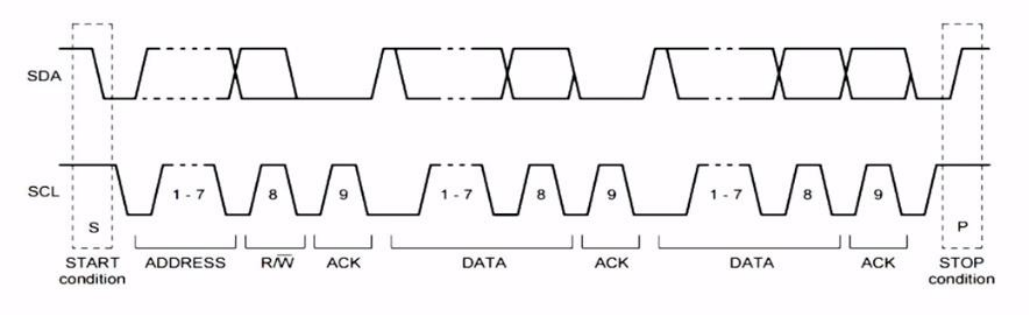

Fig.8 Timing of I2C interface

Feed the digitized audio to audio interface.we will store several sound data into the SDRAM, and an audio controller will call corresponding file according to different situations. The format of music might need to transfer to .mif format.

Assume the sampling rate is 40kHz, which is actually a worse case, we can decrease the sampling rate if the sound has low frequency. The data word consists of 24 bits. Then for each sampling point, one needs 24 bits, for 1 sec, one has 40000 sampling points. So for each second, one needs 40000*2byte=80KB.
We will apply sound effects and background music during the game.
Sound effect: ball hitting on edge and paddle(0.3 sec), ball hitting on bricks (0.3 sec), ball falling outside the paddle(0.3 sec) and game over music (1sec).
Background music: about 20 sec and repeat.
Then the total audio time is 0.3*3+1+20=21.9sec. Thus memory we need is 80*21.9=1752KB=1.7MB. (Actually, a 8kHz sampling rate is enough in most cases, which would give us roughly 350KB.)
For SoC-DE1 board, memory device include
• 64MB (32Mx16) SDRAM on FPGA
• 1GB (2x256Mx16) DDR3 SDRAM on HPS
• Micro SD card socket on HPS
In our case, we configure the audio complementation at I2C mode, so we can use 1GB DDR SDRAM on HPS to store the sound, which is apparently enough.

4. Software Implementation

A. Paddle control

The paddle will be controlled by a mouse. The mouse's moving to left or right will correspond to the paddle's movement. The moving speed will also relate to how fast we move the mouse. By clicking on the left key, the paddle can launch extra balls as bonus in higher level rounds.

Software will be used to keep track of the current status of the paddle, to see if it's under special status, like elongated, shortened, multi-ball launching, or fireball launching (fireball can destroy all bricks in its trajectory).

B. Bouncing ball

Software will be used to assigning new locations of the ball as the ball bounce around walls (edges of the screen), bricks and paddle. When the angle of incidence changes, the angle of reflection changes too.

We will also need a status tracker to see whether the ball is in regular status or fireball status. Also, we implement the game with player have multiple lives.

C. Bricks

For each brick, the software will have a status tracker. The tracker will record how many breaks needed to break a specific brick (different types of brick may need 1 or 2 or 3 times of breaking before they are destroyed), what special effect they have and when to release, and to display or not (remaining or destroyed).

D. Score counting

Software will count for the scores and calculate the bonus points gotten by how many bricks broken in a row and other effects. The score will be displayed on the up right corner of the screen.

5. Milestones Plan

Milestone 1:

Display bricks, paddle, and ball in sprites. The ball should be bouncing when there are no obstacles. The paddle should move left and right controlled by the keyboard. If time permits, attach a sound effect when the ball hits the wall.

Milestone 2:

Hardware part should be finished by then so we don't have to make those files every time. Bouncing should work for most bricks and paddle, while it could be tricky when the ball hits the corner. More sound effects will be added, making it a generally playable game.

Milestone 3:

While some of us trying to figures out the math in irregular bouncing scenarios on the paddle and bricks, others will be adding more features to make up a real game.

Reference

[1] https://opengameart.org/content/breakout-brick-breaker-tile-set-free

[2] DE1-SoC User Manual

[3]https://www.youtube.com/watch?v=zzIi7ErWhAA