CSEE W4840 Project: Shoot bubble video game

Team Name: Bubble Tea

Team Members:

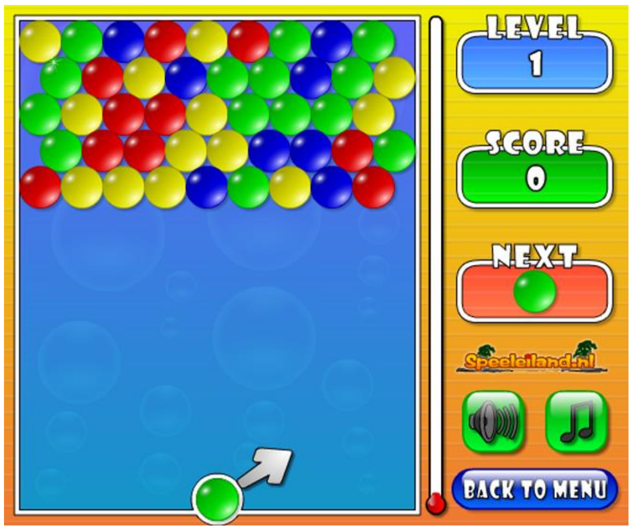
Jun Dai jd2968

Shuo-Shu Tsai st2786

Weichia Chen wgc2111

Michael Shone yh2567

1. Introduction

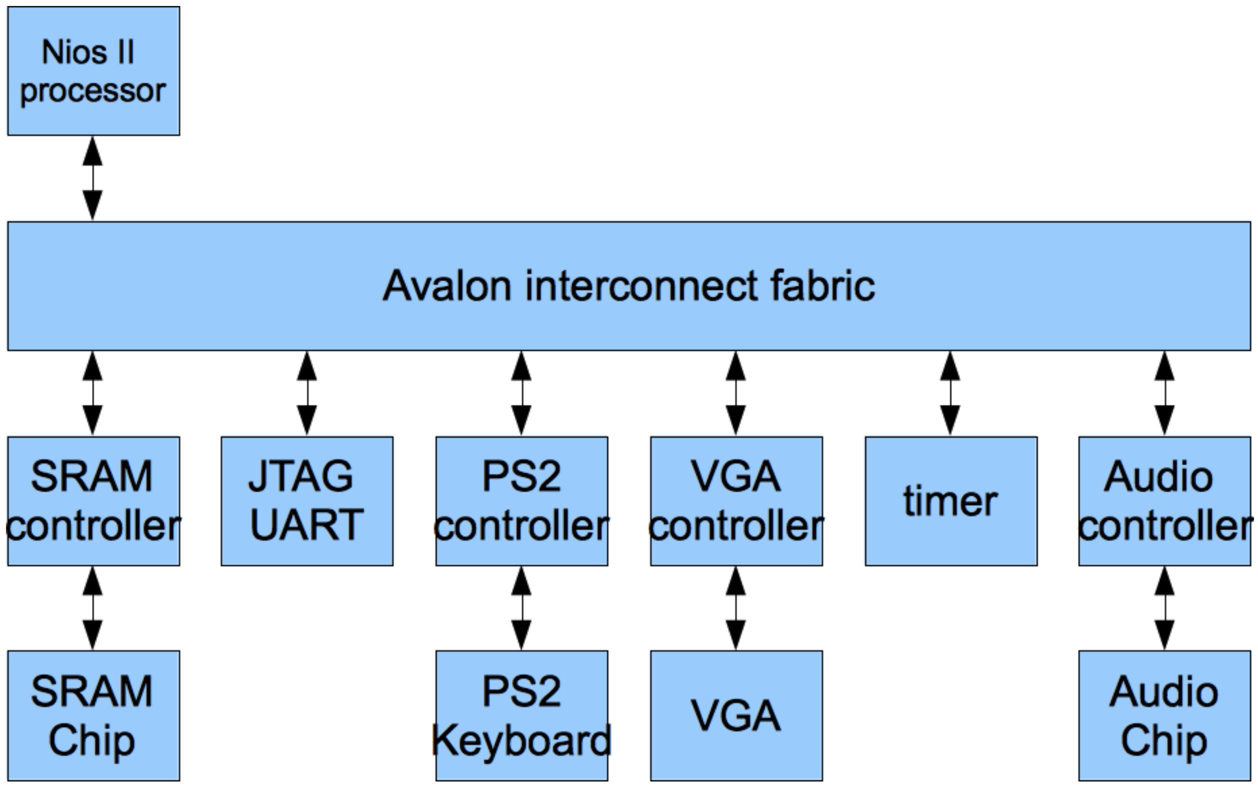


Shoot bubble Demo Figure

Our plan is to implement the video game called "shoot bubble". The basic idea shows on the figure. Once the player starts the game, on the upside of the screen, there are some bubbles with different colors holding together. In order to pass the game, player will choose the angle of the next bubble, and press a button to shoot it. If three or more bubbles are connected together, then they will disappear. At the time that there is no bubble in the screen, player wins the game.

2. Design Block

In order to realize our design, six major components are included: SRAM, JTAG/UART, PS2 keyboard, VGA, timer, and Audio chip. Following figure shows the connections between each other.



3. Software and Hardware Implementation

Whole Design split into two parts: software programming and hardware setup. Software program controls the game logic: path of the bubble, logic to remove the bubbles and decision of whether pass the game. Hardware generates the game background figure and deal with audio synthesize.

3.1. **Software**

In the game, player uses arrow keys on PS2 keyboard to control the shooting direction of the bubble. Once the decision is made, player can press P button to shoot the bubble. Then program calculates the related shooting path of that bubble, and sends back the bubble position to VGA controller in order to display it on the monitor. There are two situations where the bubble stops: reach the boundary or touch the other bubble. After that, program logic makes the decision whether the bubble satisfies the condition to cancel out or just hold there. In addition, the program logic also required to judge whether the player win or lose the game. In the case that there is no bubble left in the screen, player wins the game. Another situation, the lowest bubble reaches the button line of the screen, player loses the game.

Since we include two players mode in the project, program should be fully optimized to accomplish the purpose of heavy data processing duty.

3.2. **Hardware**

Hardware is responsible for game background figure and audio synthesize. Sprites' rendering is the essential approach in our design. All the bubbles are stored as sprite arrays in the hardware. In order to make the design easy to process, each sprite is 32x32 pixels. Therefore we only have maximum 20x15=300 bubbles in the 640x480 screen. An array can be defined to store these bubbles. Our game also generates sound effects, like shooting sound, disappear sound, game starting music, and game ending music, etc.

4. **Keyboard**

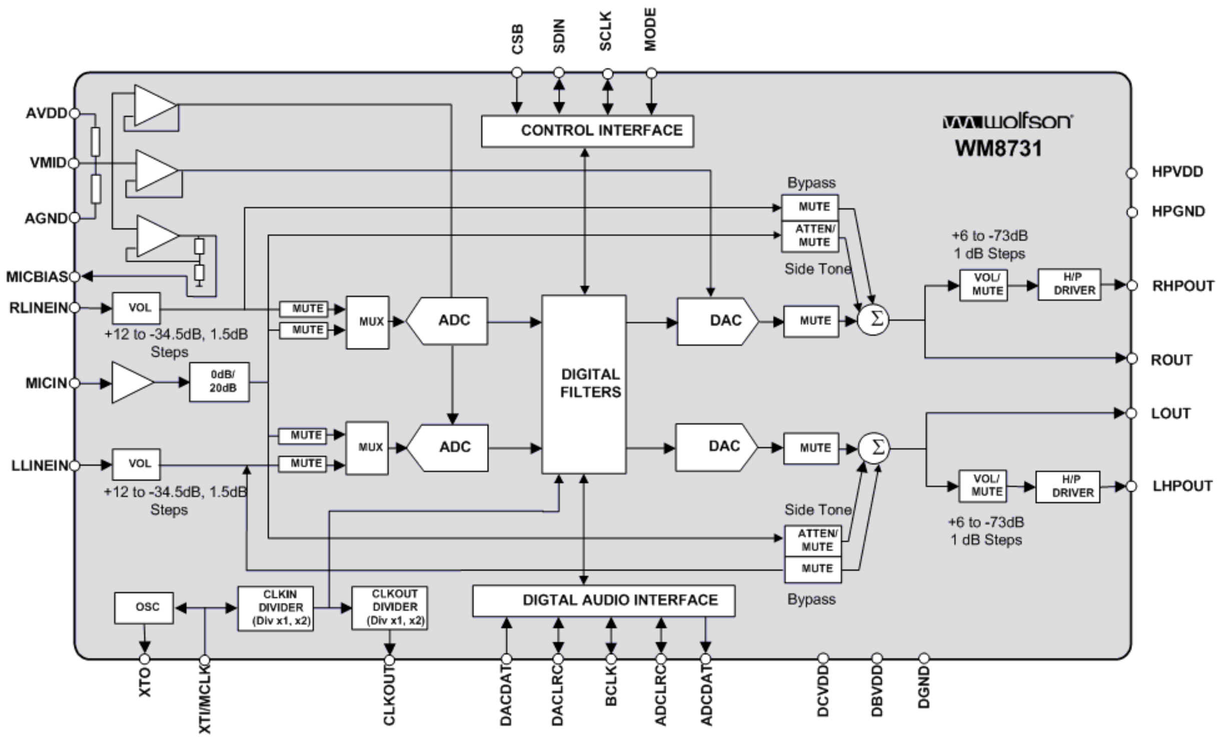
The keyboard in this project is where we get the input from the player to control the game. The keys and their functions shows in the following figure:

KEYS	**FUNCTIONs**
← , →	Player 1 control: Control the arrowhead which determine the direction for shooting
P	Player 1 Shoot!!
A,D	Player 2 control: Control the arrowhead which determine the direction for shooting
SPACE	Player 2 Shoot!!
↑, ↓	Using in menu for choosing mode
ESC	Returning to menu while playing

We have different functions for each shooting direction. While pressing right or left, logic change function from the original function to the specific one. In this way, controlling direction can be realized in the game.

5. **Audio:**

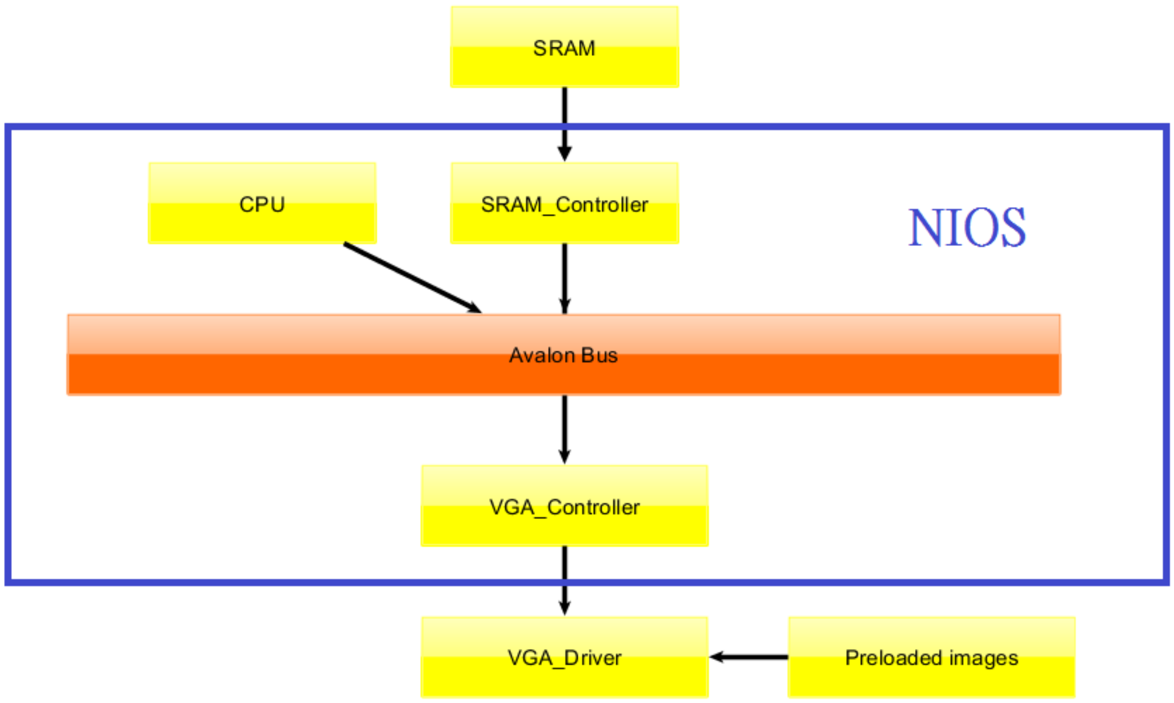
In our project, we would like to produce sound effect and background music when player is playing the game. We will use the high-quality 24-bit audio via the Wolfson WM8731 audio CODEC. The WM8731 is controlled by a serial I2C bus interface on the Cyclone II FPGA board we are using. It is designed with digital audio input word lengths ranging from 16-32 bits and sampling rates from 8kHz to 96kHz.



Sound effect will be composed of different frequency sin wave that is similar with the FM sound synthesizer of the lab3. We will make 4 kinds of the background music in the VHDL part for the software to use in the video game: “Shoot”, “Move Arrow”, “Hit Bubble”, and “Completed Bubble”. Just change the modulating frequency and the modulation depth to make the musical notes we wanted.

Background music will be preloaded in the individual ROMs, and being played when the CPU gives the audio controller commands. We will record 3 kinds of sound from the original game as .wav file: “game start music”, “game playing music”, and “game over music”.

6. **VGA**



For the playing window region, we can divide it to 168 squares, each column contains 14 blocks and row contains 12 blocks as indicated below. We have six colors of balls for the most difficult level and three special bullets (shooting bubble). The sizes of bullets, balls and exploding images are 32x32 pixels. We can use a rotate function to generate the shot arrow in different angles so only need to store one image.

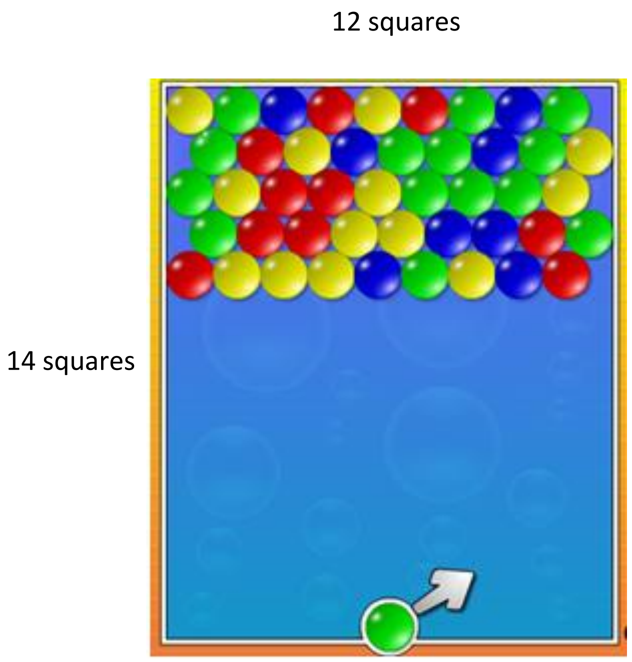
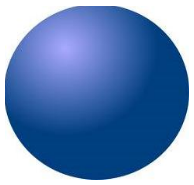


Image sample:

Green Ball

Blue Ball



shot arrow

exploding image

Image processing:

Due to the memory constrain, we don’t store sprites in 24-bitmap type. We use Matlab to covert images we search from websites to 8-bitmap; therefore, we can save sprites in SRAM.

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x6d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x00
0x00, 0xff, 0x92, 0x92, 0x92, 0x92, 0x6d, 0x6d, 0x6d, 0x6d, 0x6d, 0x6d
0x00, 0xff, 0x92, 0x92, 0x6d, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x92, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00
0x00, 0xff, 0x92, 0x00, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00, 0x00, 0x00
0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00, 0x00
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00, 0x00
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x92, 0x92, 0x6d, 0x00
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6d, 0x6d, 0x00, 0x00
```

Bitmap of 12x20 pixels shot arrow

7. **Conclusion**

For this project, setting up all the figure element configurations is the very first step in our creation. For the reason that our interface is simple, the space of SRAM is enough. After that, we decide to write most of the game logic in software part because it is easy to debug when hardware is good to go.

8. **Milestones**

Milestone 1:

1. Keyboard Interface
2. VGA Interface
3. Basic logic of the game

Milestone 2:

1. Score of the game
2. Multiple levels of the game
3. Special bubble(bomb bubble, rainbow bubble)

Milestone 3:

1. Two players mode
2. Sound effect
3. Background music

Extra function (if time permits)

1. Two players mode through Ethernet
2. Save game data