Super Frogger

CSEE4840 Project Design Document

March 20, 2012

Department of Computer Science,

School of Engineering and Applied Science,

Columbia University in the City of New York

Group Member:

Ziyao Xu zx2127

Xin Zhang xz2270

Shengzhen Li sl3356

Chenxi Liu cl2985

Abstract

In this project, we plan to design a video game called. The project involves both software design and hardware design. For software design, we mainly focus on the dynamic image signal processing. For hardware part, we adopt Altera Cyclone II FPGA board which has a video port, an audio port and a keyboard interface.

Introduction

Frogger is an arcade game developed by Konami in 1981. The object of the game is to direct frogs to their home one by one. To do this, each frog must avoid cars while crossing a busy road and navigate a river full of hazards. The game is regarded as a classic from the golden age of video arcade games and was noted for its novel gameplay and theme. Frogger is still popular and versions can be found on many Internet game sites.

To implement the Frogger, the project will involve both hardware set up and software programming. In our opinion, the most difficult part lies on the display of the game. We need to display frogs, different types of cars, logs etc. What's more, we hope to make the image of different items move vividly, which means each item might contain two or more display image.

Another challenge is how to control different items move correctly and how to determine crash of items by software.
The game play is very simple. The player starts with three frogs (lives). The player guides a frog starting at the bottom of the screen and uses the keyboard to control it move right, left or up, finally let the frog arrive at the top area of the screen.

Hardware Implementation

This project will be implemented with FPGA boards connected with following hardware components: VGA for video display, PS2 keyboard for game control, and audio blocks inside the DE2 board. The interconnection block diagram is shown below:

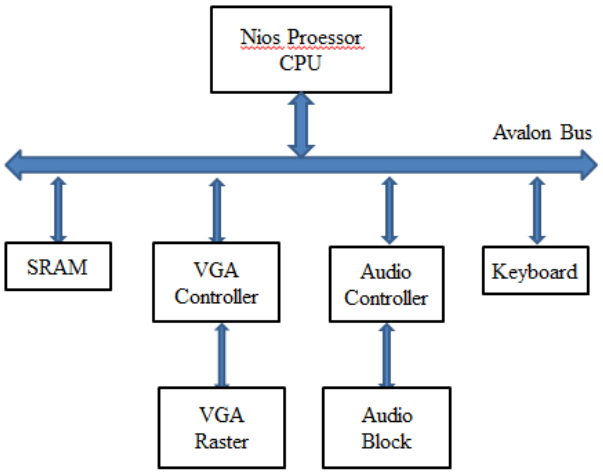

Figure1. Project Architecture

Note that the Avalon bus is the communication medium between different hardware components. The nios processor is controlling the bus as the master of the nios system. The other slaves such as VGA controller, keyboard controller and audio blocks are thus working under processor's instruction, and also through the operation controlled by C software. Here we list the implementation of the different hardware:

1. VGA controller

   VGA controller contains the VGA raster component as a top level communicating with Avalon bus. With the bus signal transmitted through VGA controller, it finally controls the VGA raster to display video on the display screen. The design specification is explained below:

   Here is the detailed information about our design of Super Frogger. Since the resolution of the screen is 640*480 pixels, we decide to divide the screen into basic matrix, or block, of 32*32 pixels. Thus, there will be 20 blocks in horizontal dimension and 15 in vertical dimension.

   On the bottom and top side of the screen, there will be a half-block wide blank which will be filled with black. On the right side of the screen, there will be a 4-block wide board to display important information about the game like lives left, high scores and time left.

   The left side of the screen, which is the main part of the game, is divided into 16 blocks in horizontal dimension and 14 blocks in vertical dimension. On the bottom, there will be a one and half wide zone where the fogger appears at the start. Then above, there will be a five-block wide road where different vehicles like cars, motors and trucks come. Frogger has to jump across the road without being killed by vehicles.

After crossing the road, there will be a little break as a reward for the frogger. There is a median zone that has a width of 1 block. Then, the frogger will face more dangers. There will be a five-block width river lying before the frogger in which crocodiles will appear. But do not be panic, there will also be logs on which the frogger can stand safe. At last, there will be five homes on the top of the screen. You have to guide five froggers safely to each of the homes to pass the game.

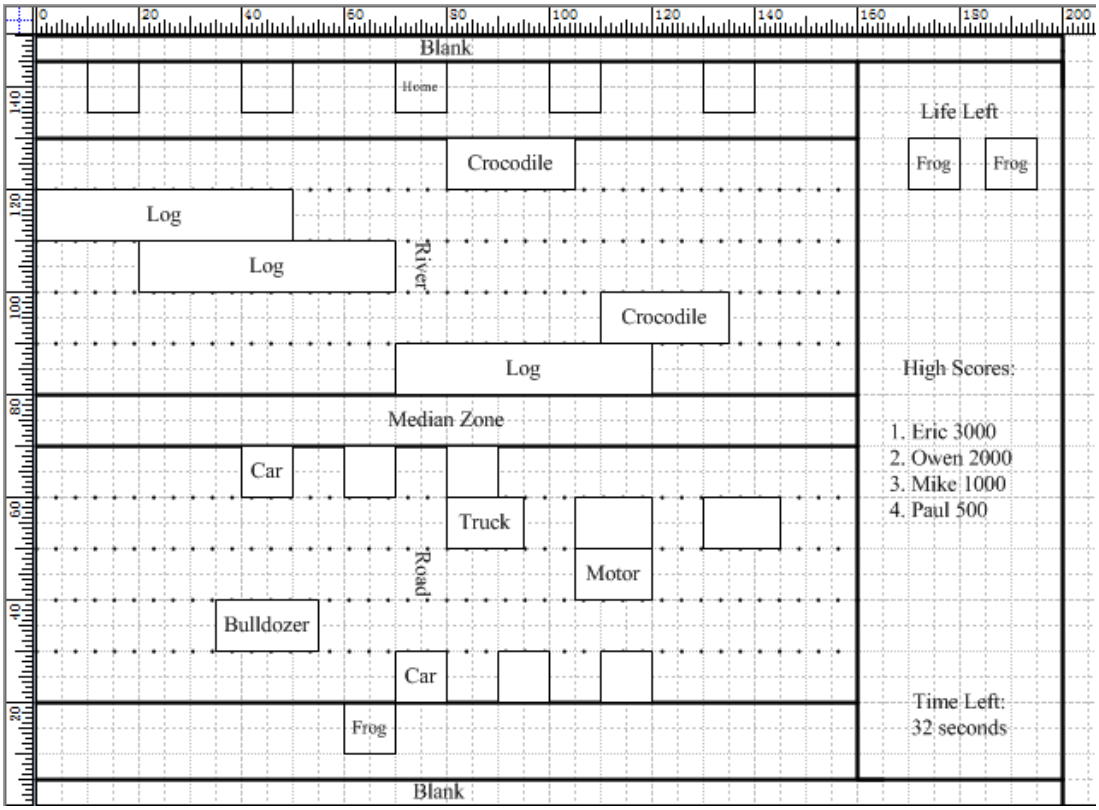

Figure2. VGA display distribution

There are also some specifications that we need to stress. The basic unit for our game is block which is 32*32 pixels. However, to avoid being too crowded on the screen, the moving unit in the game like frogger, vehicles and logs will have a real dimension of 28*28 pixels and so is the "collision dimension" that will be set in the codes. On the right side board, since the width of the board is 4*32 = 128 pixels, and we know a character owns a dimension of 8 *16 pixels, there is only places for 128/8=16 characters. Hence, the input name after each game will be limited to 6 characters and the scores will have a maximum of 5 characters. Other spaces are left to guarantee the board without being too crowded. The time left function will display the time left for each frogger to find their home. When time is used up and frogger is still on the way, you lose. This part may be revised into a progress bar that gradually decreases on the implementation.

2. Keyboard Block

In this project, the keyboard also plays an important role which is responsible for control the frogger and open/close menus. With SOPC builder, we can code the keyboard program in C language learning from lab 2 so that the data on the keyboard can be easily read via the Avalon bus.

The keyboard only activates 4 buttons in this project. Up, down, left, right arrows are responsible for to moving accordingly for the frog. Every press will be monitored by nios C program to control the VGA controller to display only one step away from the original position.

3. Audio blocks

We use WM8731 Audio CODEC, including ADC and DAC parts, to implement the sound output in this game. As learned in Lab 3, this WM8731 Audio CODEC is provided by the DE2 board. However, in this design, we do not need to analyze or change the scales of the tune, but just load the music which is saved on this DE2 board. We choose 5 tunes for this game, each of which has a different priority and a unique signature to load, shown as follows.

Table1. Music Allocation

Tune 1	Background music, load all the time except there's no other tunes	Lowest priority
Tune 2	Sound of movement of the frog	Low priority
Tune 3	Sound of collision	Medium priority
Tune 4	Sound of failure	Highest priority
Tune 5	Sound of success	Highest priority

Tune 1 is the background music of this game, which needs to be read all the time. However, as soon as other tunes are loaded, tune 1 should stop to not disturb the loaded sound. Thus, tune 1 has the lowest priority among all the music. No other signals are needed to control the beginning and end of tune 1 except the loading sign of other tunes and the beginning and end signs of the game.

Tune 2 is the sound of the movement of the frog. Thus only the direction keys (up, down, left and right) can load this tune. Since this tune will be loaded most frequently other than the background music, it has a low priority which means whenever tune 3, tune 4 or tune 5 is loaded, tune 2 must stop.

Tune 3 is the sound of collision which is read only when the frog collides with some obstacles. Obviously, tune 4 should be loaded after tune 3 immediately. Tune 4 and tune 5 are mutually exclusive to each other, which both mean the end of the game. Thus these two tunes have the highest priority among all the music.

Another problem is that, where can we save these tunes. Due to the very large volume we need in the SRAM for image pixels, it seems better to save these tunes

somewhere else. However, if we write this audio source into the ROM, it becomes much slower to read any of them. Considering the frequency of each tune, we can save the first two in the SRAM and the other three into the ROM if needed.

Software

The software part of this project contains several parts. First of all, the vehicles and logs should move automatically under certain patterns and independently from each other. In our idea, vehicles are moving in its own pattern lane by lane. Each pattern has different distribution of vehicles and pattern length and is independent from each other. Second, the software should be able to recognize the signal from the keyboard correctly. For example, if the up-arrow key is pressed on the keyboard, the software should get the signal, change the position of the frog, let the frog move up and display this on the screen. What's more, the software needs to be awareness of the crash between frog and vehicle. When the crash happens, the program should be able to end the current round of game and let the audio part work with a special sound and then restart a new round of game. Besides, in this game each regular up-move would earn a certain amount of scores while move to a bonus position would earn more. The software needs to be able to do the calculations correctly. This part is combined with both move control part and position & display part.