Ah-Ah-Piu

Spring 2012 CSEE 4840 Embedded System Design

Department of Electrical Engineering
School of Engineering and Applied Science,
Columbia University in the City of New York

Ji Pei
jp3242@columbia.edu
Xiaolong Jiang
xj2137@columbia.edu
Junlin Lu
Jl3925@columbia.edu
Nan Li
nl2411@columbia.edu
Hongsen Yu
Yh2340@columbia.edu

March 2012

Overview

The goal of our project is to build an old-fashioned voice-controlled video game as suggested in the picture, in which a FPGA, a voice sensor and a monitor might be used.

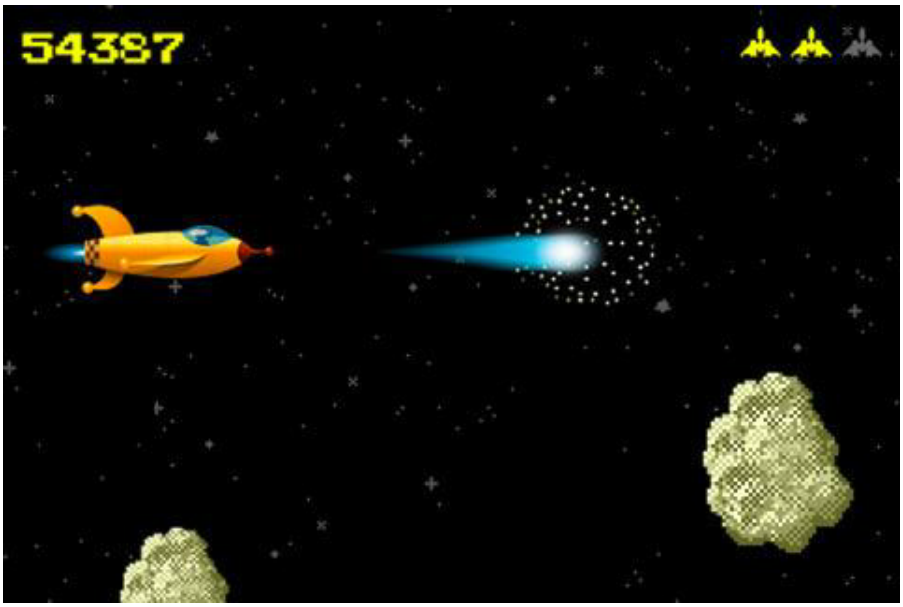

Figure1 The Ah-Ah-Piu Game Scene

Description

The whole idea of the game is about one character (maybe us, or like the aircraft in the picture) traveling through all the incoming enemies (probably all the amazing lab sessions and fabulous codes assigned by be-loved Edwards). The movement of the character is controlled by the voice of the player. As the name suggested, the player either makes continues noise like ”Ah…Ah” to keep the character rising up or be quiet to drop down the character a little bit. Besides just avoiding the collision with enemies, the player can shout out “Piu” to fire a bullet to destroy the incoming enemies. During the game, all the enemies travel towards the character in a pre-programmed way while the character just moving up and down alongside the left edge of the screen.

To be specific, our system requires a VGA monitor output to display the progressing game, and a microphone to take in the voice instruction of

the player. The received audio data is preprocessed on the hardware to determine what action should be taken by the character on the screen, our hardware logic will examine the input audio signal (after A/D convert) periodically on a base of 500ms, if it decides it is a long voice (longer than 500ms), a 11 flag will be sent to software, if there is no input, the flag is 00, besides, if it is a short burst, 01 will be sent. Then the software will invoke the C-coded program accordingly which enable the character to go up, drop down or fire a shot. Moreover, the software also have to handle the test of whether a collision occurs, calculate the score, and keep track of the player's game status. On-board memory space might be needed to store the models of our character, the background picture and the source C code.

Diagram

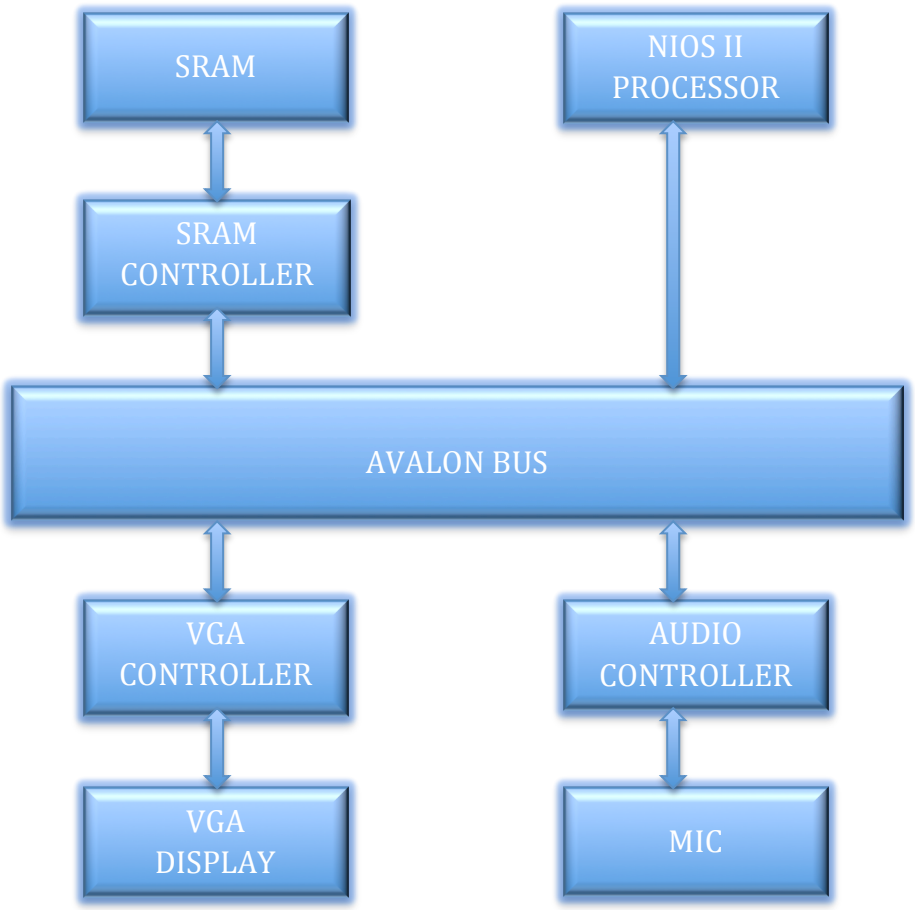

Figure2 The Design Diagram

Hardware Peripheral

Our system is mainly based on the Altera DE2 board, it takes a VGA monitor for display, and a microphone for game control input. Besides, audio components within the DE2 board as well as memory are also needed. The diagram shows above give a general idea about the interconnection logic of our design. As we can see in here, the NIOS II processor has the control over the whole system, while the Avalon bus serves as the communicating medium between different hardware components. Further detail about each hardware components and the memory space they require are as follows.

1. VGA controller

The VGA raster component within the VGA controller communicates with the Avalon bus and display the video on the monitor during the whole process. Within our design, the video content continuously appears on the screen is generated picture by picture along the time. The resolution of the monitor is 640*480, as a compromise of the limited on-board memory space and a better appearance, we decide to divide the screen into basic block of 4*4 pixels, thus there will be 160 blocks horizontally and 120 blocks vertically. Each colorful pixel has a 3-bit RGB vector.

RED	GREEN	BLUE	YELLOW	CYAN	WHITE	BLACK
100	010	001	110	011	111	000

Table1 RGB Vector

There are three image modules in total should be stored within the Flash memory: firstly, our character is 10*10 blocks figure, which will take up approximately 10*10*2*2*2/8/1024=0.1KB; secondly, the obstacles moving towards the character also is with the same size; thirdly, the BOSS which will show up at the end of the game enlarge to 40*40 blocks, so it needs 6.4KB to store in the Flash. Besides the modules, a background picture and a GameOver scene will also be kept in the Flash memory, where each will use 20KB space.

As the game goes on, the hardware logic will send an instruction to the software every 500ms, in order to make the game display looks

seamlessly, 30 pictures will be drawn to the screen within the interval. So a 20KB buffer will be required in the SRAM.

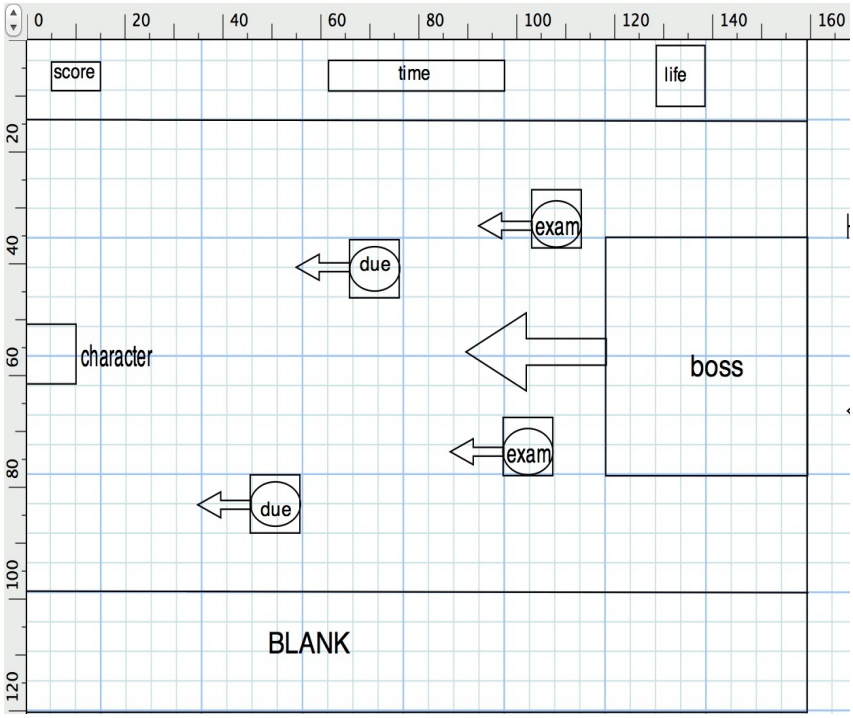

Figure3 VGA Display Distribution

2. Audio Controller

2.1 Audio Output:

As we learned in lab3, the WM8731 Audio CODEC provided on the DE2 board, including ADC and DAC parts, can be used to analyze or change the scales of tunes. In our design, however, we just need it to load and play the music we stored on the board. Our games need five different tunes:

Tune1	Background audio
Tune2	Fire a shot
Tune3	Collision
Tune4	Bullet explore an obstacle
Tune5	Boss laugh

Table2 On-board Stored Music

Since the expensive SRAM memory space should be saved for real-time

video storage which will be accessed more frequently, we plan to store the audio data in the Flash.

2.2 Audio Input

Within our design, both microphone-in and line-out ports will be used for player's voice input. Here The Wolfson WM8731 audio CODEC is configured in the master mode, where the audio CODEC generates AD/DA serial bit clock and the left/right channel clock automatically. To set the sample rate (typically at 48kHz) the WM8731 is controlled by a serial I2C bus interface. We will choose Microphone that collect data with frequency from 30hz to 16khz, which is tailored for human voice. The board input has +6dB to -34dB volume level adjustment. In the case of large white noise, we will then filter the audio input by a volume threshold.

Software Peripheral

For the purpose to run the game more fluently, we should carry out most calculation on the hardware side. As discussed above in the description, after the hardware logic make the decision about the input audio signal, a flag will be sent to the software to instruct what it should do. At normal case, the software should generate the distribution of obstacles on the screen in real time, which will appear likes moving towards the character in a configured lane. Besides, the software will test whether a collision between the character and an obstacle occurs instantly. It will be the same logic to test whether a fired bullet will eliminate an incoming obstacle. Every time a collision or an explosion happens, The software should conduct the hardware to play the according tune. Further more, during the game, the player's score and status will be shown on top of the screen, these figures will be also calculated by related codes.

Critical Path Of Design

Our critical path is when our character collides with obstacles. Since that when this situation occurs, at first, the software have to invoke the images about collision then hardware have to transfer tune 3 instead the background sound from flash. In other words, the hardware should wait for the signal from software to make a change. It's reading data from memory and put a signal into bus that limit the speed.

Milestones

Milestone1:
1.Create and edit image for different models
2.Build sprite structure

Milestone2:
1.Create and edit various sound effect and BGM
2.Complete the component for collection of sound data
3.Finish game logic design (going up/down, enemy engaging, fire, collision)
4.Display the graphic into screen

Milestone3:
1.Finish noise filtering
2.Create menu and miscellaneous object in the screen
3.Add sound effect to the game