Save Edwards

CSEE 4840 Embedded System Design

WEI WEI WW2313

ZEYANG YANG ZY2171

ZHE CAO ZC2237

GE ZHAO GZ2196

March 26th, 2013

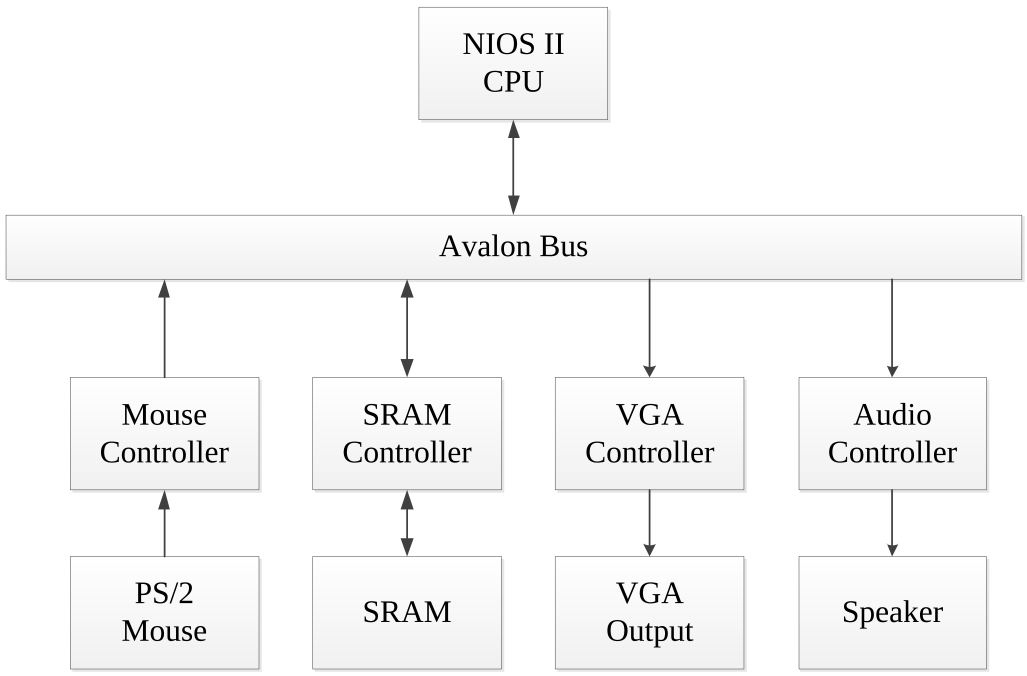
Project Introduction

In this project, we intend to implement a tower defense video game. Basically, the player needs to set up defense facilities beside the path, in order to kill the attackers before they get through the path to the destination. Any attackers who survive from our towers and arrive at the destination will do certain amount of damage to our boss, prof. Edwards, and the game will be over if our professor runs out of health points. Otherwise, if the professor is still alive when all the attackers are killed by the towers, you win and can head to next scene! We will have several facilities with different features which need to be selected each time you build a tower, and of course it has different price. We have a unique money earning and paying system which could better control the speed and the difficulty of this game. What's more, the defense facilities could be upgraded to have a stronger power.

There are some key points of this project with which we need to handle carefully. First is the VGA display, since there will be lots of different figures, special effects and complex background, we need to use proper methods to implement the dynamic and complicated display of the game. Second is the algorithm issue. For the attackers, we need to set them walking along the designed path and think of the different behavior after being attacked by different defense facilities. For the defense facilities, we need to design different characteristics according to different types. And how the tower judges whether there are any attackers in its attack range and decide which one to attack need to be considered carefully as well. The third one is the operation of mouse, which was briefly mentioned in lab2 and need to be revised for this specific project. The last key point is how we deal with the sound effect using the FPGA. We have been in touch with the audio problem in lab3 already, and in this project we need to play some certain sound waveform in specific cases.

A high-level hardware structure

After consideration, we initially draw a high-level block diagram as below:



As in the figure, we have a NIOS II CPU and four major peripherals: VGA display, mouse, storage part and audio output module. All of above are connected to the Avalon bus so that they could communicate with the NIOS II CPU. For instance, we move the mouse and left click to generate an input signal in the mouse controller and transfer it to the CPU through the bus. The CPU recognizes the input, does the operation and passes the control signal to corresponding peripherals. And after the control signal dealt with in the controllers, the peripherals generate output, such as showing figures on the screen or generate sound effects from the speaker.

In the VGA module, we plan to use the frame buffer to display different sprites. The reason we choose to use the sprite graphic display is that we have many small pictures such attackers and towers, and their behavior is mutually independent. So we can better control them by laying them on different sprites. So there will be one sprite control part. Meanwhile, we will also design a normal VGA control module, with which we can show every pixel on the screen as in the lab3.

For the mouse control, we need to do the similar job as we did in lab2. The difference is that we change the PS/2 peripheral from keyboard to mouse. The header file is given in lab2, so we need to write the C file of the mouse control according to the lecture and some other materials.

Next is the storage module, after the initial estimation, we decide to use the 512KB SRAM on the board to store the pictures and short sound effect waveform. To read from and write into the SRAM, we need to design a SRAM controller, which is the key point in this module.

Finally comes to the audio module. In this project, we just need some short music as the sound effects in some specific cases such as attacking the attackers, the attackers die, winning or losing the game at the end and etc. We can refer to the FM Sound Synthesizer part of lab3 and do some revision according to the practical situation.

Memory requirement

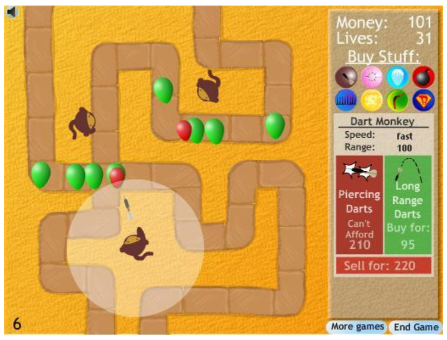
For a video framebuffer: each pixel have 10 * 3 = 30 bits. The screen 640 * 480 is divided into 20 * 15 = 300 areas each contains 32 * 32 pixels. The background image is formed by tiling each 32 * 32 s quares. Two kinds of tiles are designed, one for path and the other for tower positions. Three types of towers are in 32 * 32 size. Three kinds of monsters are in 32 * 32, and one boss in 40 * 40 size. Some words(26 letters) and numbers (10 digits) indicating money, life are also in images with size of 20 * 10 each. The total memory for images is
30 * (32 * 32 * (2 + 3 + 3) + 40*40 + 20 * 10 *36) = 509760 bits = 63.7 KBytes.
These data are stored in SRAM.
For sound waveforms: two types of sounds are used, one is background music and the other is sound effects such as attacking from the tower and yelling while dead. The sampling rate of 8K and 16 bits per sample is good enough in this game. The background music is a clip about 10s playing over and over again. So it is about 160 Kbytes, and it is stored in SRAM. These data are stored in SRAM. Sound effects are from 0.5s to 3s and the total sound effects is no more than 100 Kbytes. These data are stored in on-chip block RAMs.

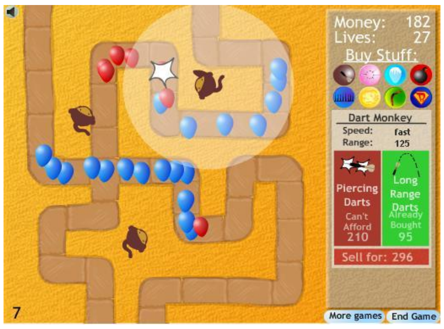
Implement details: VGA Display

There are 5 kinds of elements to be shown

1. Four kinds of monsters(each has different appearance)
2. Three kinds of attacking effects(for each of those towers)
3. Three different towers(each has different appearance)
4. HP bar of each monster should be displayed on the top of them
5. Background which is also the map of the game
6. The score and the game money should be shown on the top of the screen

All those elements can be stored as fixed figures in the SRAM on the board as it save the resource on the FPGA. To realize the attacking action display dynamically, we plan to display all the elements by refreshing the sprite frequently. The total screen is 640x480 pixels, so that it would be reasonable to design each attacking and defensing elements and attacking effect element to size of 32x32.

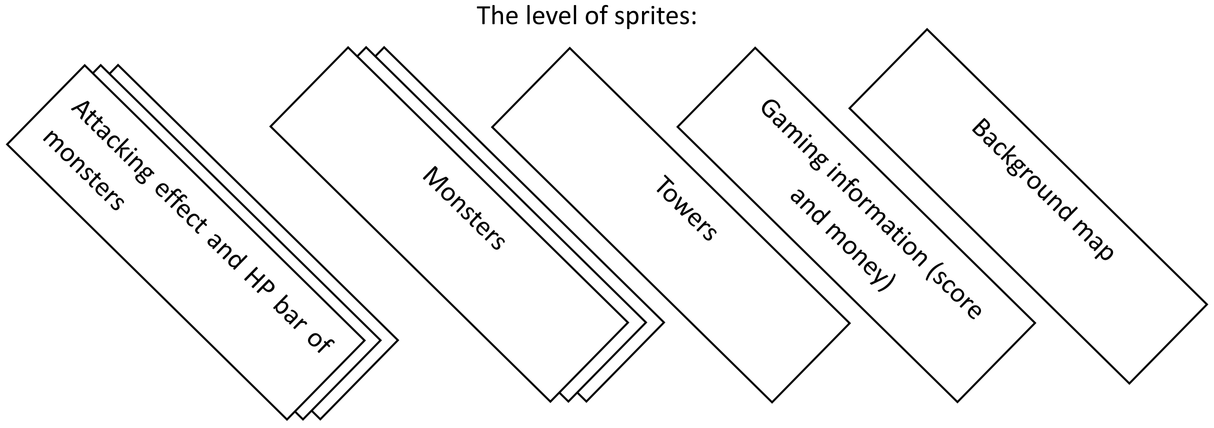




The difficulty in the graph display is that the monsters are generated continually, and those elements may overlap each other. The tower and the attacking effect are also continually generating. The generation of each monster and each attacking effect leads to a new sprite because their motion should be treat individually (some of them may be slow down by a decelerate tower or may be killed). The tower and the gaming information each occupies a sprite with new elements generate on the sprite. The attacking effect has the top level and then the monster, following the tower and the last would be the game information and background. Considering there won't be more than 15 monsters in the screen, so that the number of sprite for monsters is 15, and same as the attacking effect. So that in total we have 33 sprites. This is shown in the following figure.

The level of sprites:

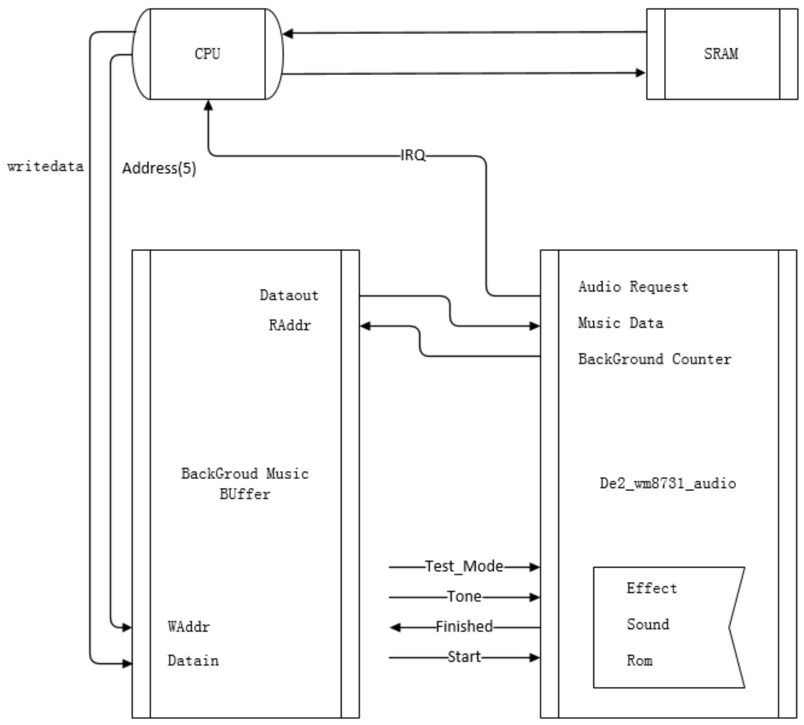
The level of sprites:


Attacking effect and HP bar of monsters
Monsters
Towers
Gaming information (score and money)
Background map

The VGA control unit would be very important in this section. It provides interface with Avalon bus, and the software is able to control what and where to display a sprite.

Implement details: Audio Controller

CPU gets audio data from SRAM, computes and sends commands of playing audio to audio controller.



The audio controller consists of a RAM buffer for background music and Wolfson WM8371 audio codec. The buffer size is 32 * 16 bits. In the block diagram above, Waddr is the write address controlled by CPU, whose width is 5 bit, representing 32 entries. DataIn interface get data from CPU. Raddr is the read address controlled by audio codec counter. DataOut send data in the Raddr to the codec. When the last entry is read from the buffer, the codec sends a request to CPU ask for more data.

Sound effect audios are stored in on-chip block RAMs. Sound effect and background music can play at the same time or play only one at a time by test_mode controlled by CPU. If only sound effect plays, select tone to choose the sound effect ID and CPU sends start command to start playing. Once the sound is finished, a finished signal is returned and resume playing background music.

Implement details: Mouse

For this project, we will use the mouse to do the basic operation, such as move the pointer over the screen and left click to select. As we have finished the relative work on the keyboard, which uses the same PS/2 port as mouse, some header files and C files can be directly used after some slight revision. And here are some points that we want to clarify.

First, the resolution of the screen is 640x480, and we decide to divide into the basic tiles with the size of 16x16. So there will be 20x16=320 tiles in total. When move the mouse, there will be a pointer icon moving correspondingly with the mouse. And according to the position of the tip pixel of the pointer, we can decide which tile the mouse is hover on. In that case, once the CPU recognizes the input signal of a left click, which means a selection operation, the open land tile is selected by the mouse.

Once the land is selected, we design to pop up a frame, in which are the different towers the player can select. This can be implemented easily by the sprite graphic display module. What the player needs to do is to move the mouse and select the tower he or she wants to build. Of course, if there are three optional towers, the frame might occupy three basic tiles and the sprite of the frame should be at a higher level.

And here is the PS/2 mouse protocol, the host must send 0xF4 (enable data reporting) to make sure three bytes sent every time mouse moves or button clicked:

MSB							LSB
Y	X	Y	X	1	Middle	Right	Left
Overflow		Sign			Buttons		
X movement							
Y movement							

Movement values are since last transmission: 9-bit two’s-complement (signed) numbers.

Implement details: Algorithm

Here we want to briefly introduce our idea about the implementation algorithm of the behavior control of every entity, mainly the defense facilities and the attackers.

Regarding the attackers, they have two behaviors: move and die. When the attackers appear at the beginning of the path, they will follow the designed path at the initial speed. We plan to have three kinds of attackers, normal hp with normal speed, high hp with slow speed and low hp yet with fast speed. The different speed can be easily implemented by adding speed flag bits. However, there will be one kind of tower, which can slow down the speed of the attackers. This can be realized by a judgment, namely that if the attackers are first hit by this kind of tower, the speed slows one level by changing the speed flag bits and meanwhile set a time counter. Every time the attackers are attacked again, refresh the time counter. Once the time counter decreases to zero, the attackers will recover to the original speed by changing the speed flag bits back to the initial value. The other behavior is killed by the tower, which can be implemented by setting an hp counter. And when the hp counter decreases to zero, the attackers die.

For the tower, there are two behaviors: attacking and upgrading. Every tower has its own attack range, which can be 25 tiles or 36 tiles around it. Once the tower is built, the boundary is fixed. So if there are any attackers entering this boundary, the tower will attack. Similar with drawing a line, we can calculate the relative position according to the x and y coordinates of the attacker and the tower. But here we plan to use special effect of small circle instead of the line. The tower will follow the first attacker coming into its attack range until the attacker gets out of the boundary or dies. Then the tower will turn to attack the second one. This behavior can be implemented by numbering every attacker in the range from 1 to n, and once the first attacker goes far away or dies, every number decrease by 1, and the tower always attacks the number 1 attacker. The other behavior is upgrading. Every tower can be upgraded to achieve stronger power and larger attack range.