

## Car Racing Game

-- Project Design

### 1 Introduction

For this Car Racing Game, we would like to accomplish a video game imitating the existing game showed as Figure 1 with a projective view. The theme of our game is to compete with the other 5 opponents that are controlled by computer in a racing tournament, the player's goal is to get to the destination as soon as possible while trying to avoid bumping to other cars or road object, the final score will be posted according to the finishing position, numbers of bumps and the time.

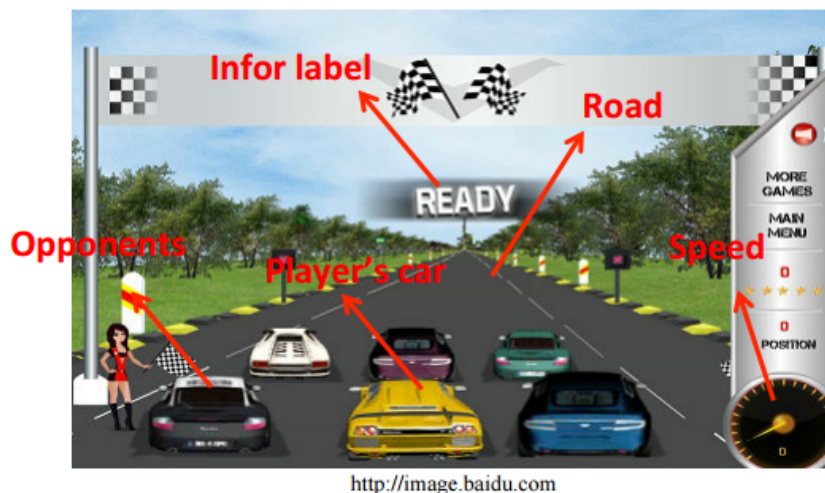


Figure 1 The Car Racing Game

This game consists of three major modules. First part is the Game Logic Generator which calculate the logic of this game, such as to detect bumps to obstacle, speed control based on keyboard input, opponents control and road generation, and this module is based on software. The second part is the Screen Rendering Module, we adopt the Sprite Graphics technique to decompose the display screen into 7 layers, which will be explained in details in later section. The last part is the Audio Module which generate the proper sound under the control of game logic. The overall design of the game is demonstrated in figure 2. The following content of this paper explain each modules in details.

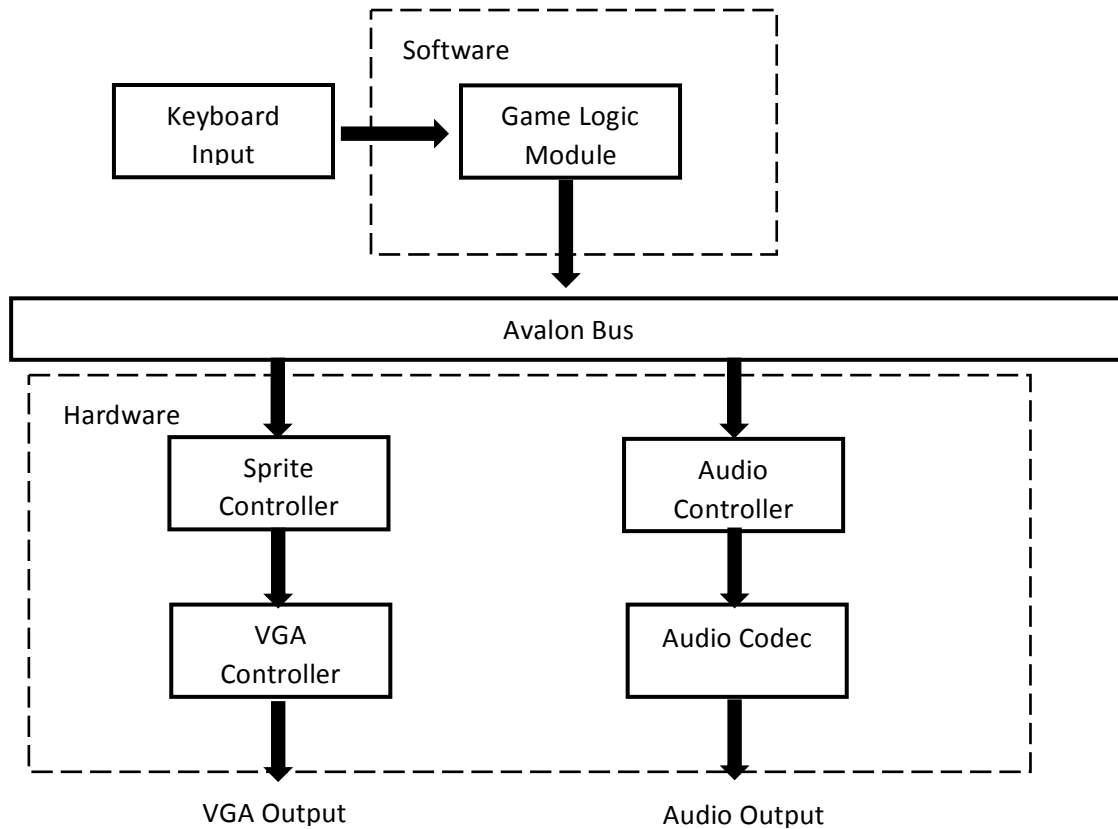


Figure 2 The overall design of Car Racing Game including Software and Hardware.

## 2 Game Logic Module (Software)

Game Logic Module will be completed in C language with two major driver – Sprite driver and Audio driver. Game Logic Module control the Sprite and Audio through Avalon Bus using the driver interface function. The main function of the Game Logic is list as following table.

Generate the position (coordinate) of object which is the control input of Sprite Controller, according to the game logic	Road
	Trees
	Clouds
	Cars
	Speed
	Position
	Score
	Timer
Generate the Audio control signal	Sounds like bump, break, accelerate etc.

The Game Logic Module generates all the motion control signals of all the objects in this game according to various algorithms and feed the signal to Sprite and Audio modules through the Avalon Bus using Driver Function implemented.

For player's car, there are two behaviors: move forward and change direction.

Move forward:

To implement the action of moving forward, we first make the end of the road at the center of the screen to make sure the player's focus is on the road. Then we create stripes of two different patterns, one is light and the other is dark. We draw the stripes on specified positions shown in Figure 2. We keep switching positions of two kinds of stripes to generate the effect of moving the road. We use the up arrow on the keyboard to speed up. When the up arrow is pressed, we raise the switching frequency to make the road move faster. When the player's car hit the back of another racing car, it slows down and we lower the switching frequency. When the player's car passes another car, we simply move the car that gets passed from its current position down to the bottom of the screen and then disappear. By moving the car, we will also keep scaling the car. The similar algorithm applies when the player's car gets passed; we just move the passing car from the bottom of the screen up and keep scaling the car as it moves.

Change direction:

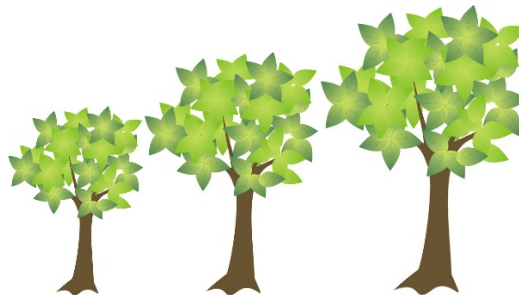
We use the left and right key to control the left and right movement of player's car. We will keep track of the position of the car. When left or right key is pressed, we move the position of the car to the left or right lane and draw the car based on the position.

### 3 Sprite Control Module (Hardware)

According to Figure 2, the Sprite Control Module's job is to generate the 640\*480 pixels RGB value for VGA Controller based on the control signal provided by the Game Logic Module. The Game Logic Module provides the X and Y coordinates for different objects, like trees, clouds, cars, scores, speed, which is also called Sprites. These Sprites are like label pictures store in ROM, and can be pasted to the screen according to the X and Y coordinates. We list a few Sprites we will use in our games.



Three sizes of Cloud (32\*32, 48\*48, 64\*64)



Three sizes of Tree (16\*32, 24\*48, 32\*64)





Three sizes of Car of different directions (32\*32, 48\*48, 64\*64)

The following table list all the Sprites that we need in this game.

Sprite	Amount	Pixel Size	Total ROM (Bytes)
Cloud	3	32*32, 48*48, 64*64	22272
Tree	3	16*32, 24*48, 32*64	11136
Car	9	32*32*3, 48*48*3, 64*64*3	66816
Background (including road, land and sky)	1	640*480	921600
Number	10	32*32	30720
Speed	1	48*48	6912
Total	27	353152	1059456

The estimated total ROM size = 1.02 MB , each Pixel is defined by 8-bit R + 8-bit G + 8-bit B.

We break the game display figure into 7 layer with different priority. Sprite with higher priority will be displayed above that with lower priority. The diagram for the Sprite Controller Module is demonstrated in Figure 3.

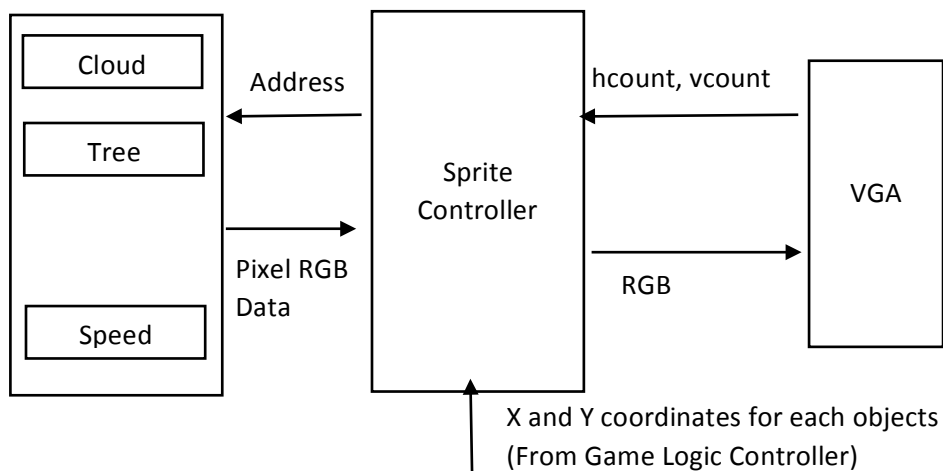


Figure 3 The Sprite Controller Module

The VGA module is the same as Lab 3. The Sprite Controller Module merges the different Sprites to generate the final RGB information for displaying on the screen. The priority of different layers is showed in Figure 4.

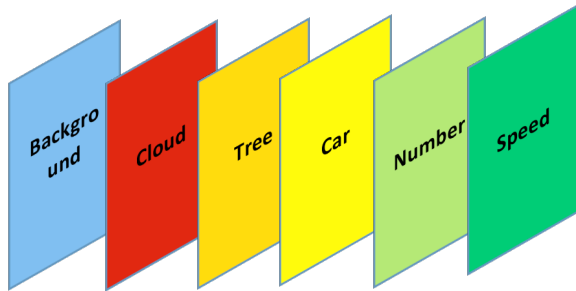


Figure 4 Sprite Merging (the priority ranks from low to high from left to right)

## 4 The Background Generation Algorithm

In our design, the background containing road, land and sky is generated different from other Sprites, because it can't be simply be pasted to the screen according to X and Y coordinates of Sprites. Therefore we design a special module for Background Generation. The background is showed in Figure 5.

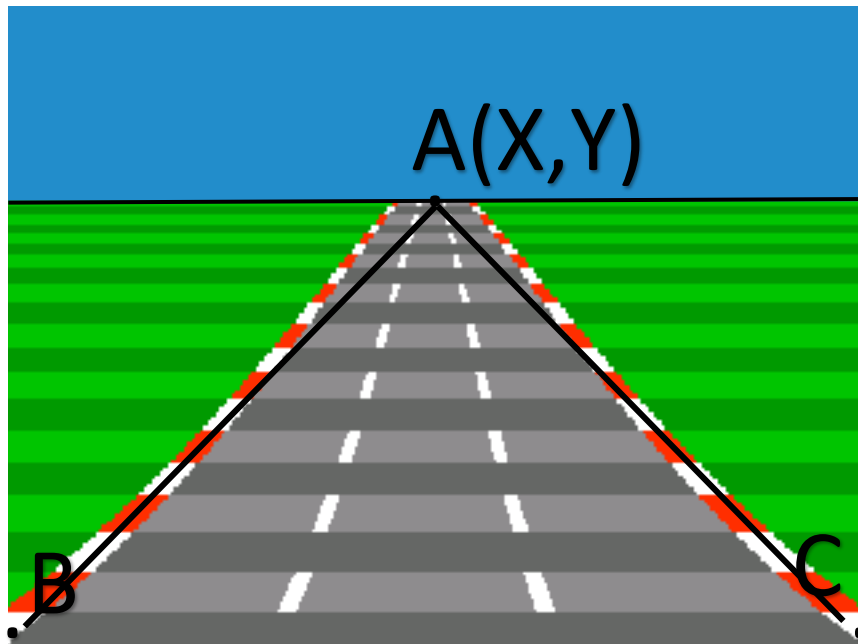


Figure 5 The Background Sprite

The rendering algorithm of this background is based on the coordinate of A (X, Y). Every pixel above Y horizon is set to Blue because it is Sky Region. All pixels within Triangle ABC is set to Gray, because they belong to the Road Region. The rest pixels are set to Green, because they belong to the Grass Region.

## 5 Audio Design

Audio Generator Module is designed to send instructions to the Audio Controller Module in order to play appropriate sound effect according to the game processes. For example, the background music is always

on during the period when game is on. And special sound effects such as vehicle acceleration, braking, and collisions are integrated when the specific situations happen. At the beginning of the game, the software extracts the background sound sample previously stored in the SDRAM and load it into the circular queue in the Audio Controller Module, which will continuously send signals to the SSM2603 Audio Codec to play the background sound sample until the player exit the game. During the car racing, the software continuously checks whether the player hits the acceleration or braking button, or if there is any collision happens. Whenever a specific situation is detected, the software will instruct the DRAM to load the corresponding special sound effect into FIFO, and instruct the SSM2603 Audio Codec to play the special sound effect.

The Sockit board we are using in the lab provides the Analog Devices SSM2603 audio CODEC (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8 kHz to 96 kHz. The SSM2603 is controlled via a serial I2C bus interface, which is connected to pins on the CycloneV SoC FPGA. The schematic diagram of the audio circuitry is shown in Figure 6.

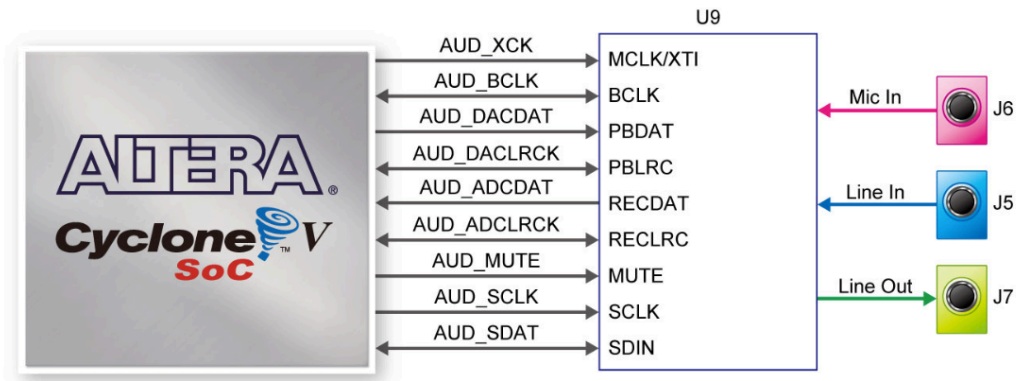


Figure 6. Audio Circuitry in Sockit Board