

# **CSEE 4840 Embedded System Design**

---

## **Flappy Bird Video Game**

**Wei Zheng wz2299**

**YenHsi Lin yl3284**

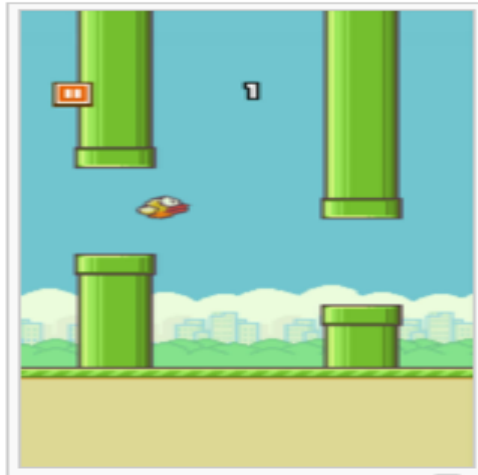
**Junhui Zhang jz2605**

**Gaoyuan Zhang gz2216**



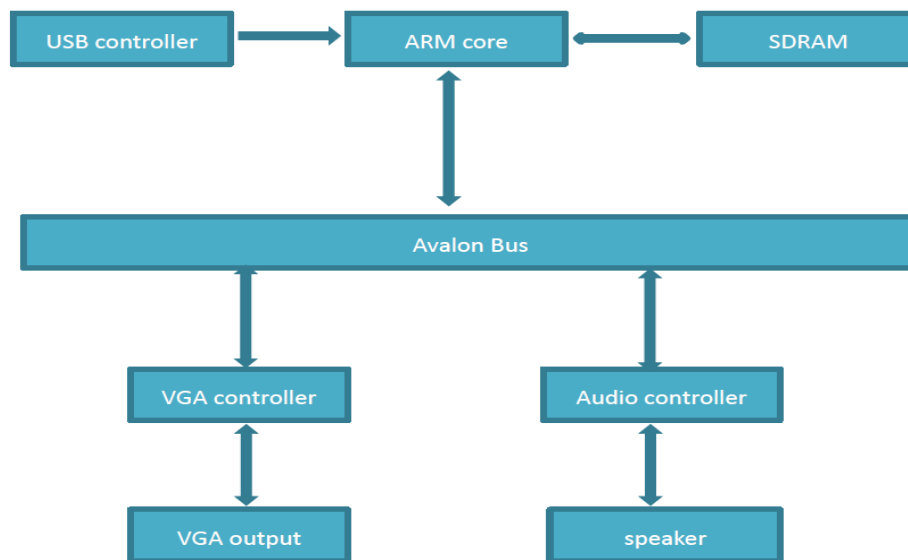
## 1. Introduction

We try to bring the popular game Flappy Bird, which is originally on Android platform, to the FPGA board. The game is a side-scroller where the player controls a bird, attempting to fly between rows of green pipes without coming into contact them. If the player touches the pipes, it ends the game. The bird briefly flaps upward each time the player taps the screen; if the screen is not tapped, the bird falls due to gravity.



## 2. Software and Hardware Components

The major components in our design includes the game controller, game logic, video and audio module.

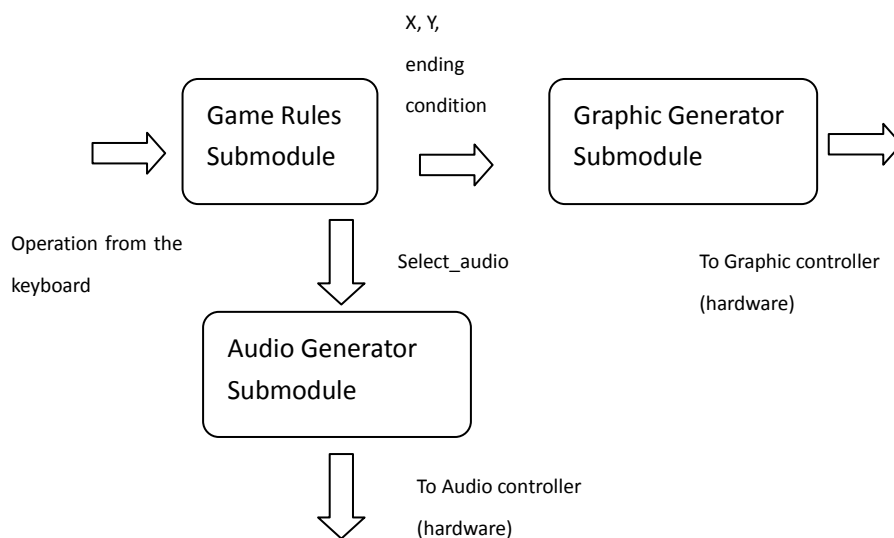


*Figure. High level software and hardware design components*

Game logic controller(software)

We implement the game logic by using C programming language. The game logic controller should realize the functions which are indicated below: deriving the location of the flappy bird from the keyboard, implementing the game rule (whether the game is over or not, computing how many pillars the bird has passed), generating

the appropriate audio in terms of the game rule, and controlling the generation of graphics. Based on the functions given above, there should be 3 submodules for the game logic controller, the figure of which is shown below:



1. Game rules: This is the core submodule of the game logic controller which interfaces with all of the other submodules, instructing them what to do based on the game rules. It should constantly update the screen by supplying the graphic generator with the location, the number of pillars that the bird has passed, as well as the judgment whether the game is over.

To be specific, there should be several functions that could monitor the game:

1. A function that keeps track of the movement of the bird by the previous location, instructions from the keyboard and also a subfunction that controls the automatic fall of the bird. Once the control button is pressed, the bird would be granted with a vertical upward speed  $v$ . Otherwise the bird would be doing a free-fall, Since the horizontal location of the bird is unchanged, we only care about the vertical location of the bird, which should be  $Y_{\text{bird}} = v * t - \frac{1}{2} * g * t^2$

2. A function that calculates the real-time score. The function would compare the X coordinate of the bird with that of the pillars. Since the speed of the pillars is constant, we can use a counter to calculate the score. As long as the game is not over, the counter would accumulate every time period ( depending on the distance between every adjacent pillar, and the speed of the pillar). The score would appear on the top of the screen.

3. A function that determines whether the game is over or not. The function is implemented by comparing the coordinates of the bird with that of the pillars, the ceiling and the ground, and checking if there is any overlapping between them. If so, it will force the bird to drop directly to the ground and then the submodule will send a message to the graphic controller, to generate an final interface, indicating how many scores are achieved, and a "play again" button.

4. A function that generates the X and Y coordinates of the pillars that has already

appeared on the screen, as well as the length of the upcoming pillar that is going to appear from the right side of the screen. Since the X coordinate of the bird remains invariant, the location of the pillars would be keep moving leftwards. The length of the pillar should be random, as long as the distance between the pillars is constant.

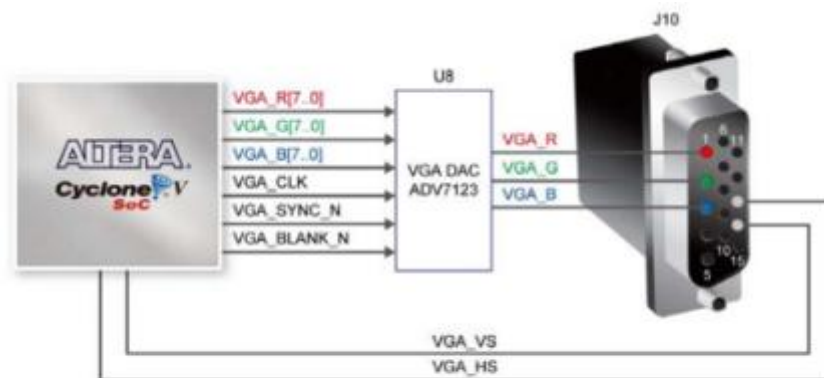
2. Graphic generator: This submodule received all the data required for the generation of graphic, including the coordinates of the bird and the pillars, the identification whether the game is over or not, and so on. These coordinates are stored in memory, updated according to the game logic. The memory is accessed by the graphic controller through address, which then displays the necessary graphic on the screen.

3. Audio generator: The audio sounds needed in the game, including the background music, the flappy sound of the bird, as well as the sound when the bird hits the pillar, are encoded inside the audio generator. The submodule should instruct the audio controller which one to play, based on the game logic.

### 3. VGA Block

#### *vga controller*

SOckitboard includes a 15-pin D-SUB connector for VGA output. The VGA synchronization signals are provided directly from the Cyclone V SoC FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC (only the higher 8-bits are used) is used to produce the analog data signals (red, green, and blue). The following figure gives the associated schematic.



#### *sprite controller*

The input of the sprite controller is X and Y coordinate of different elements and then the sprite controller will send the RGB values of each pixel to the VGA controller.

In our game, we are going to use six types of elements that will be displayed on the screen.







- a) alive bird
- b) dead bird
- c) pipe

- d) sky scene and ground scene
- e) score indicator
- f) start and pause button

Sky scene and ground scene are the background of the game. Pipes are the barrier of the bird. They have different length and distance from each other. When the bird touches any of the pipes, it should change to a dead bird pattern and the game is over.

In order to display each kind of basic elements on the screen, we can update each element in the form of sprites. There are two levels of sprites shown on the screen, top level and bottom level. The figures on the top level are alive bird, dead bird and pipes. The background scene on the bottom level will remain the same.

### ***Memory Budget***

Element	Number of sprites	Pixel size	Size	Example
Ground scene	1	64*80	5KB	
Sky scene	1	128*400	50KB	
Bird	2	80*60	4.69KB	
Pipe	10	64*80 64*100 64*120 64*140 64*160	5KB 6.25KB 7.5KB 8.75KB 10KB	
Score	10	30*40	1.17KB	
Button	2	60*25	1.46KB	

## 4. Audio interface in FPGA

### Brief introduction

The Cyclone V SocKit board provides high-quality 24-bit audio via the analog devices chip SSM2603 audio CODEC (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports as shown in figure. We are going to store our sound effect and music in our SocKit board memory so we don't need inputs microphone-in and line-in. What we want just output our background music and sound effects.

SSM2603 supports a sampling rate adjustable from 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz which we often use is 44.1kHz. From Nyquist theory, we know it can be used to sample at most 22kHz voice matching human's acceptable frequency range.

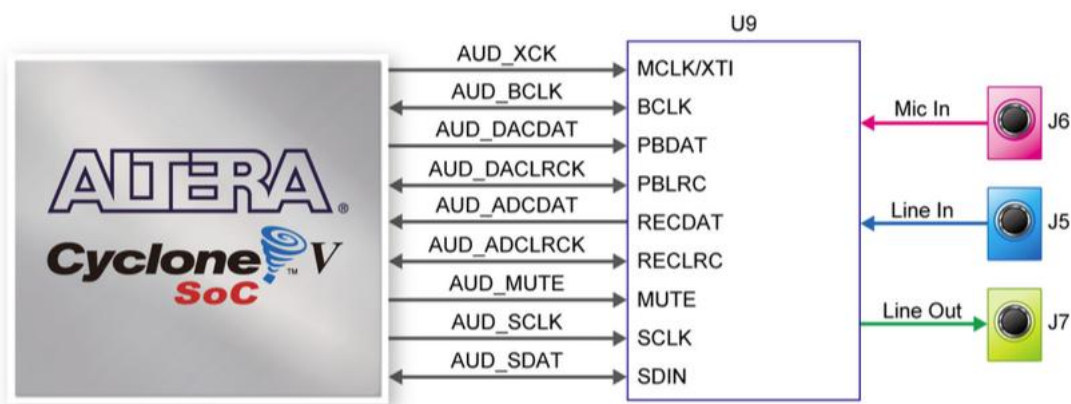


Figure. Connections between FPGA and Audio CODEC

Signal Name	FPGA Pin No.	Description	I/O Standard
AUD_ADCLK	PIN_AG30	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_AC27	Audio CODEC ADC Data	3.3V
AUD_DACLK	PIN_AH4	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_AG3	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_AC9	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_AE7	Audio CODEC Bit-Stream Clock	3.3V
AUD_I2C_SCLK	PIN_AH30	I2C Clock	3.3V
AUD_I2C_SDAT	PIN_AF30	I2C Data	3.3V
AUD_MUTE	PIN_AD26	DAC Output Mute, Active Low	3.3V

Figure. Pin assignment for audio CODEC

### IO protocol I2C and audio transmission interface I2S

I2C(Inter-Integrated circuit):

The SSM2603 is controlled via a serial I2C bus interface, which is connected to pins on the Cyclone V. I2C is an IO protocol used to communicate between master and slave devices. There are read/write modes for master and slave devices.

SDIN(Serial data in):Transmission line for address, data and state symbol.

SCLK(Serial clock):Global clock used for I2C buses

ACK: acknowledgement signal generate by receiver or slave

S/P: START and STOP state symbol.

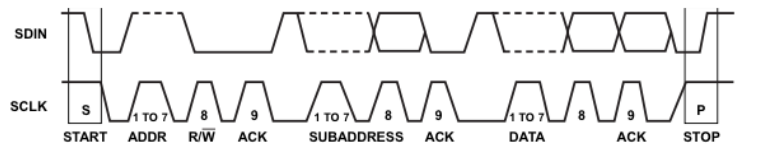


Figure 28. 2-Wire I<sup>2</sup>C Generalized Clocking Diagram

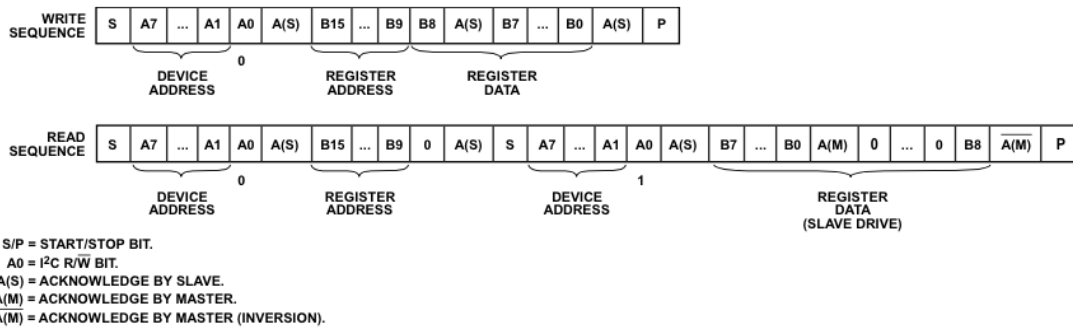


Figure. I<sup>2</sup>C Write and Read Sequences

I2S(Inter-IC sound):

I2S is an electrical serial bus interface standard used for connecting digital audio devices together. We use I2S mode for our SM2603 with 32bit ISA communicating with our sound device. There are also master and slaves modes. In our case, SM2603 is in master mode.

WS(Word select): WS=0 left side sound track; WS=1 right sound track.

SD(Serial Data):digital audio binary data.

SCK(Serial clock):global clock used for I2S bus.

Usually, ADC IC is master, and DAC is slave. However, it is not necessary. Master has to send WS and SCK while slave must receive WS and SCK.

In I2S,M SB is always first bit, and LSB is the last. Bit B15 to Bit B9 are the register map address, and Bit B8 to Bit B0 are register data for the associated register map.

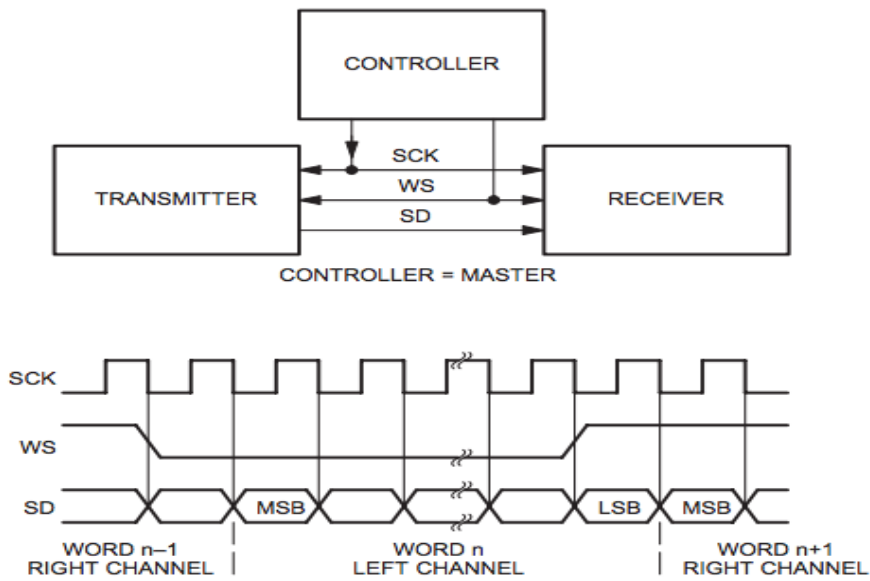


Figure. I<sup>2</sup>S Audio Input Mode

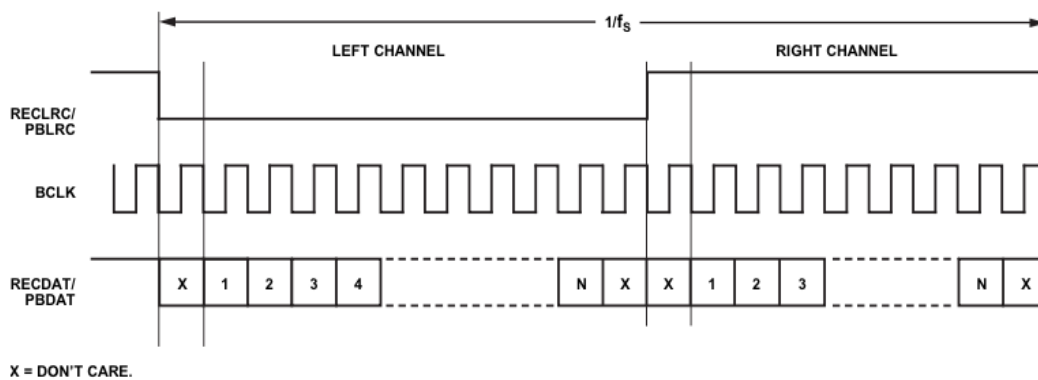


Figure. Simple System Configurations and Basic Interface Timing

### Music in our design

In our flappy bird game, we will apply sound effects and background music during the game.

Sound effect: Score counting (0.5 sec), bird jumping (0.5 sec), hitting on obstacle (0.5 sec) and game over (1sec).

Background music: about 30sec and repeat until the game is over.

### Memory calculation

In our case, we try to save our memory so we assume that our sampling rate is 8kHz. In I<sup>2</sup>S the data word consists of 16 bits that mean we use 16 bits to quantize one sampling point. According to these specification, we can get our sound effect is about 8 KB(0.5sec) and 16KB(1sec), and background music is 480KB. As we know there is a DDR3 1Gb SRAM in SocKit board, the music capacity is acceptable.



## **5. Milestone**

***Milestone1:*** implement video module.

***Milestone2:*** implement audio, game logic and game controller module.

***Milestone3:*** Integration of audio, video , game logic and game controller module.