Pah! Pah! Pah!
A voice activated video game

Hao Hu(hh2506), Kezhen Liu(kl2688), Shaobo He(sh3156), Sheng Zheng(sz2372), Yi Su(ys2646)

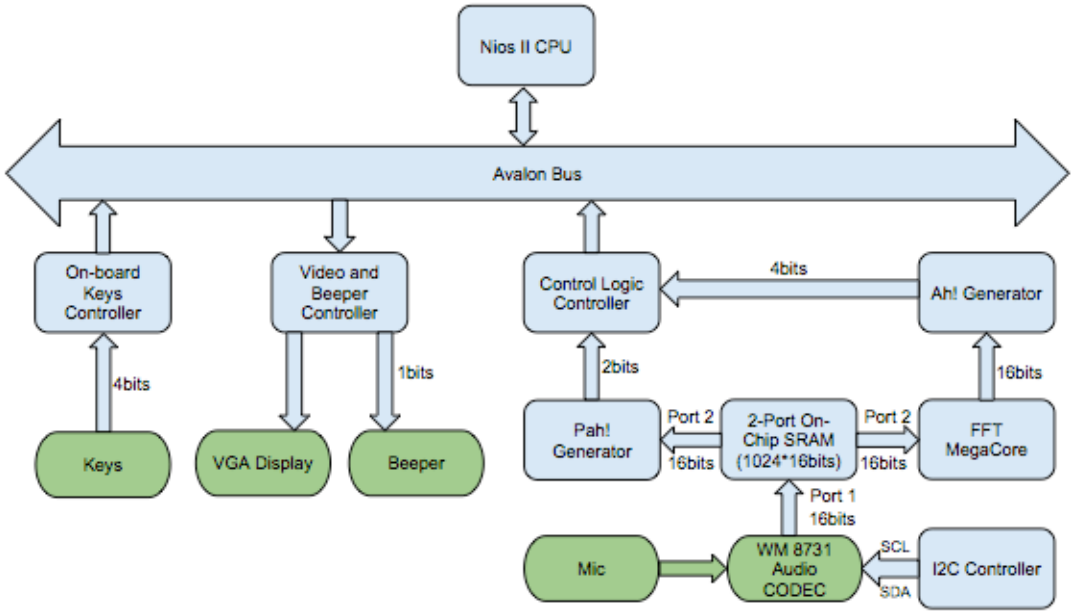
Abstract

The goal of our project is to port a voice activated iPhone game called Pah to Altera DE2 Board. This project utilizes both the hardware and software capabilities of the Altera DE2 Board. The hardware designs first presented along with a high level block diagram of the project. Each hardware component and peripheral is described in detail. The software design is then presented and its modules are described in details. References used in the design construction are presented at the end of the paper.

General Description of the Game

When the game started, an animation scene will be displayed on the VGA screen. Besides, a list of choice will also be displayed through which the user could choose the mode under which they want to play. Basically we plan to provide the user with two modes: keyboard-control mode and voice-control mode. Obviously, if the keyboard-control mode was chosen, then the game is played by keyboard otherwise the game is played by voice. To make the game more interesting and energetic, we are going to add inspiring music when the user started the game by choosing one of the two modes.

Now suppose the user has started the game by choosing the voice-control mode we mentioned above, a spaceship will appear on the screen with dynamic animations. After that, some meteorites will be displayed on the screen. The player's task is to avoid colliding with these meteorites. To simulate the real situation in space, the positions of these meteorites will be randomly place on the screen and the number of these stones is intended to be four. To make the spaceship move up vertically, the user should make a sound so that the sound could be detected by the MIC on the board. And if the player wants to make the plane keep moving upward steadily, he should make his tone become higher. This is mainly because the 'up' movement of the spaceship is controlled by the tone. Apart from just avoiding, the user could shoot the stone by suddenly making his voice high, which could generate a high frequency audio signal and will be captured by our FFT module. Thus the spaceship will fire correspondingly. So the player could play this game through these basic elements. In addition to these basic elements, the player will also have their status (i.e. scores, lives, time etc.) recorded and displayed on the top of the screen. Besides, we also plan to add more entertainment elements to the game. Some stone will be marked by some special colors. For example, if 3 of the regular meteorites are white or gray, then the special one will be red or yellow. When the player hit this special stone with bullet, the stone will be exploded. The explosion of the special meteorite will spawn some rewards to the player. If the player got this reward by hit it with the body of the spaceship, then the ship's status (i.e. weapon, lives, time) will get improved.



Hardware System Overview

The above figure is the micro-architecture of out hardware system.

Detailed Block Description

Video and Beeper controller:

1.There are overall 14 visual elements should be considered in this part. All these elements will receive signals about whether they will display or not and the coordinate from NIOS.

(1). Start screen

When we start the game, this screen will appear including the name of our game and some simple pictures.

(2). Different modes

Because our game has two modes: key controlled and voice controlled, an arrow will direct to different modes according to player's selection. This part will be displayed at the bottom of beginning screen and changing according to signal received.

(3). Platform

After we start the game, the beginning screen will disappear and the spaceship will stopped on this platform. Then the platform will fall and game will start.

(4). Background

In order to obtain a vivid effect of flying in the universe, the background is required. We will draw some simple white dots or small stars.

(5). Spaceship

We will use array of 0 and 1 to build the shape of the spaceship. An example of our model is showed in the following.

```
"0000000000000000000000000000000000000000000000000000000000000000",
"0000111111111000000000000000000000000000000000000000000000000000",
"0000011111111100000000000000000000000000000000000000000000000000",
"0000000011111111100000000000000000000000000000000000000000000000",
"0000000001111111110000000000000000000000000000000000000000000000",
"0000000000011111111100000000000000000000000000000000000000000000",
"0000000000001111111110000000000000000000000000000000000000000000",
"0000000000000111111111100000111111111000000000000000000000000000",
"0000000000000000111111111111111111111111111100000000000000000000",
"0000000000000000011111111111111111111111111111000000000000000000",
"0000000000000000011111111111111111111111111111111000000000000000",
"0000000000000001111111111111111111111111111111111111000000000000",
"0000000000111111111111111111111111111111111111111111111000000000",
"0011111111111111111111111111111111111111111111111111111100000000",
"0111111111111111111111111111111111111111111111111111111111111110",
"1111111111111111111111111111111111111111111111111111111111111111",
"0111111111111111111111111111111111111111111111111111111111111110",
"0011111111111111111111111111111111111111111111111111111100000000",
"0000000000111111111111111111111111111111111111111111111000000000",
"0000000000000001111111111111111111111111111111111111000000000000",
"0000000000000000011111111111111111111111111111111000000000000000",
"0000000000000000011111111111111111111111111111000000000000000000",
"0000000000000000111111111111111111111111111100000000000000000000",
"0000000000000111111111100000111111111000000000000000000000000000",
"0000000000001111111110000000000000000000000000000000000000000000",
"0000000000011111111100000000000000000000000000000000000000000000",
"0000000001111111110000000000000000000000000000000000000000000000",
"0000000011111111100000000000000000000000000000000000000000000000",
"0000011111111100000000000000000000000000000000000000000000000000",
"0000111111111000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000"
```

The spaceship has 4 colors. The cabin is white and the body is yellow and the nose is brown and the tail is red. It will move up and down according to the signal from NIOS.

(6). Meteorolite

They are implemented in the same way with spaceship and they have two colors. There are four meteorolites to be handled in this game. They will moving from right to left by default. If the meteorolite is hit by spaceship or a missile, it will disappear according to the receiving signal and restart from the right of the screen.

(7). Missile

There are two missiles could be displayed on the screen at one time at most. When the AV controller gets the signal of launching, the missile will flying from the nose of spaceship horizontally to the right until it hits a metorolite.

(8). Life

The player has 3 lives by default at first. When the spaceship hit a meteorolite, the player will lose one life. This function will be displayed at the top right corner and using different number of heart-shaped image to show lives. The number of heart will change according to signals from NIOS.

(9). Score

When a missile hit a meteorolite, the score will increase by 10 points.

(10). Simple meteorolite broken animation

When a meteorolite is hit, this animation will appear according to signals from NIOS.

(11). Simple spaceship broken animation

When a spaceship is hit, this animation will appear according to signals from NIOS.

(12). Special meteorolite

The special meteorolite is a red one, while the ordinary ones are white with purple pits. When it is hit, it will break and leave some reward.

(13). Reward

When a special meteorolite is hit, it will appear.

(14). Game over

When the player loses the last life, the game is over and this screen appears.

2. There are 6 audio elements will be handled in this game. They are controlled whether to be played or not from signals from NIOS.

(1). Selection sound

At the beginning screen, when player selects the mode, this sound will be played.

(2). Beginning music

After the mode is decided, the game will start with this music.

(3). Background music

During the game, this music is played.

(4). Spaceship explosion sound

When the spaceship is hit, this sound will be played.

(5). Meteorolite broken sound

When a meteorolite is hit, this sound will be played.

(6). Sound of launching a missile

When a missile is launched, this sound will be played.

3. Interface of this module

```
reset_n      : in std_logic;
clk          : in std_logic;
read         : in   std_logic;
write        : in   std_logic;
chipselect   : in   std_logic;
address      : in   std_logic_vector(4
downto 0);
readdata     : out std_logic_vector(15
downto 0);
writedata    : in   std_logic_vector(15
downto 0);
VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK,
VGA_SYNC : out std_logic;
VGA_R, VGA_G, VGA_B : out
std_logic_vector(9 downto 0);
```

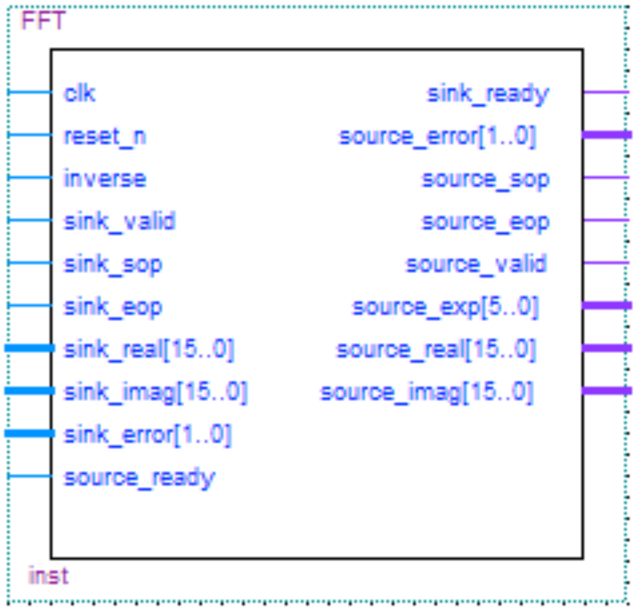
4. Control signal and according address

```
signal beginning_display: std_logic;
        --store data from address 0000000
signal modes_display: std_logic;
        -- store data from address00 00001
signal modes_x: unsigned(9 downto 0);
        -- store data from address 0000011
signal modes_y: unsigned(9 downto 0);
        -- store data from address 0000100
signal platform_display: std_logic;
        -- store data from address 0000101
signal platform_x: unsigned(9 downto 0);
        -- store data from address 0000110
signal platform_y: unsigned(9 downto 0);
```

```
        -- store data from address 0000111
signal background_display: std_logic;
        -- store data from address 0001000
signal spaceship_display: std_logic;
        -- store data from address 0001001
signal spaceship _x: unsigned(9 downto 0);
        -- store data from address 0001010
signal spaceship _y: unsigned(9 downto 0);
        -- store data from address 0001011
signal meteorolite_1_display: std_logic;
        -- store data from address 0001100
signal meteorolite_1_x: unsigned(9 downto
0);     -- store data from address 0001101
signal meteorolite_1_y: unsigned(9 downto
0);     -- store data from address 0001110
signal meteorolite_2_display: std_logic;
        -- store data from address 0001111
signal meteorolite_2_x: unsigned(9 downto
0);     -- store data from address 0010000
signal meteorolite_2_y: unsigned(9 downto
0);     -- store data from address 0010001
signal meteorolite_3_display: std_logic;
        -- store data from address 0010010
signal meteorolite_3_x: unsigned(9 downto
0);     -- store data from address 0010011
signal meteorolite_3_y: unsigned(9 downto
0);     -- store data from address 0010100
signal meteorolite_4_display: std_logic;
        -- store data from address 0010101
signal meteorolite_4_x: unsigned(9 downto 0)
        -- store data from address 0010110
signal meteorolite_4_y: unsigned(9 downto
0);     -- store data from address 0010111
signal missile_1_display: std_logic;
        -- store data from address 0011000
signal missile_1_x: unsigned(9 downto 0);
        -- store data from address 0011001
signal missile_1_y: unsigned(9 downto 0);
        -- store data from address 0011010
signal missile_2_display: std_logic;
        -- store data from address 0011011
signal missile_2_x: unsigned(9 downto 0);
        -- store data from address 0011100
signal missile_2_y: unsigned(9 downto 0);
        -- store data from address 0011101
signal life_1_display: std_logic;
        -- store data from address 0011110
signal life_1_x: unsigned(9 downto 0);
        -- store data from address 0011111
signal life_1_y: unsigned(9 downto 0);
        -- store data from address 0100000
signal life_2_display: std_logic;
        -- store data from address 0100001
signal life_2_x: unsigned(9 downto 0);
        -- store data from address 0100010
signal life_2_y: unsigned(9 downto 0);
        -- store data from address 0100011
signal life_3_display: std_logic;
        -- store data from address 0100100
signal life_3_x: unsigned(9 downto 0);
        -- store data from address 0100101
signal life_3_y: unsigned(9 downto 0);
        -- store data from address 0100110
signal score_1_display: std_logic;
        -- store data from address 0100111
signal score_1_num: std_logic;
        -- store data from address 0101000
signal score_1 _x: unsigned(9 downto 0);
        -- store data from address 0101001
signal score _1_y: unsigned(9 downto 0);
        -- store data from address 0101010
signal score_2_display: std_logic;
        -- store data from address 0101011
signal score_2_num: std_logic;
        -- store data from address 0101100
signal score_2_x: unsigned(9 downto 0);
        -- store data from address 0101101
signal score _2_y: unsigned(9 downto 0);
        -- store data from address 0101110
signal score_3_display: std_logic;
        -- store data from address 0101111
signal score_3_num: std_logic;
        -- store data from address 0110000
signal score_3 _x: unsigned(9 downto 0);
        -- store data from address 0110001
signal score _3_y: unsigned(9 downto 0);
        -- store data from address 0110010
signal spaceship _explosion_1: std_logic;
        -- store data from address 0110011
signal spaceship _explosion_2: std_logic;
        -- store data from address 0110100
signal meteorolite_1 _broken_1: std_logic;
        -- store data from address 0110101
signal meteorolite_1 _broken_2: std_logic;
        -- store data from address 0110110
signal meteorolite_2 _broken_1: std_logic;
```

```
        -- store data from address 0110111
signal meteorolite_2 _broken_2: std_logic;
        -- store data from address 0111000
signal meteorolite_3 _broken_1: std_logic;
        -- store data from address 0111001
signal meteorolite_3 _broken_2: std_logic;
        -- store data from address 0111010
signal meteorolite_4 _broken_1: std_logic;
        -- store data from address 0111011
signal meteorolite_4 _broken_2: std_logic;
        -- store data from address 0111100
signal reward_display: std_logic;
        -- store data from address 0111101
signal reward _x: unsigned(9 downto 0);
        -- store data from address 0111110
signal reward _y: unsigned(9 downto 0);
        -- store data from address 0111111
signal GameOver_display: std_logic;
        -- store data from address 1000000
```

FFT Megafunction:



We will use the FFT IP core in Quartus. The IP core uses the N complex vector in the time domain as the input, and outputs the complex vector in the frequency domain in the natural order. The inputs include 1 bit clk, 1bit reset_n signal, 1bit inverse bit to determine the order of output, 1 bit sink valid to determine the valid bit of the input, 1 bit start of packet signal, 1bit end of packet signal, one 16 bits real part of signal, one 16 bits imaginary part of signal, 2 bit error control bit, and 1 bit source ready signal. The output is very similar with the input. 1bit ready signal, 2 bit error control bit, 1 bit sop and 1bit eop. 1 bit valid signal, 6 bits source_exp, 16 bits real part of output and 16 bits imaginary part of the output.

Control logic controller and On-board keys controller

The function of these two blocks are similar which converts the signal from either the 4 keys or the audio status signals to the Avalon bus.

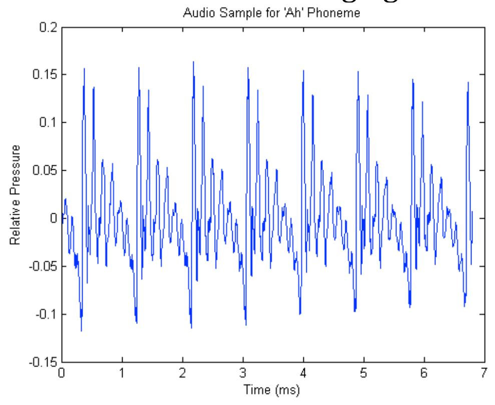
Pah! Generator:

The Pah! generator is basically an amplitude detector. When the amplitude of input in the time domain is above a very large threshold, it will pull 1 bit clear_screen bit high. When the amplitude of input in the time domain is between the large threshold and second large threshold, it will generate one 1 bit missile bit. When it is below the second large threshold, it will pull the clear_screen and missile bit to 0. The pseudo code for this part is as below:

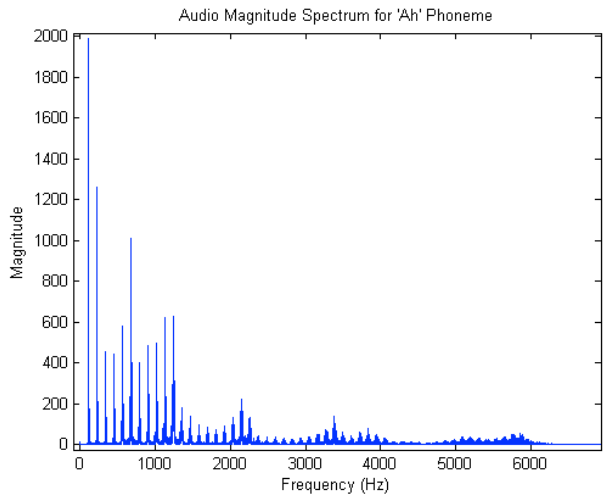
```
If (amplitude > large threshold)
        clear_screen <= 1;
else if (amplitude > second threshold)
        missile <= 1;
else
        clear_screen <= 0;
        missile <= 0;
```

Ah! Generator:

The function of this block is to compare the fundamental frequency of the voice received. The audio sample of the sound of ‘Ah’ is showed in the following figure.



The FFT output of the signal is the showed in the following figure.



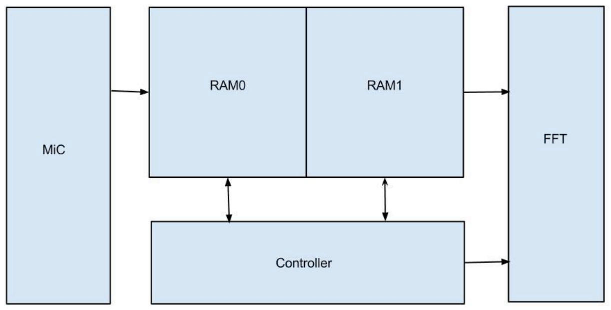
We can see from the figure that there will be a frequency component that has the largest amplitude in spectrum. We use this frequency as the fundamental frequency. And will be stored and compared with the fundamental frequency of the previous time period. Since different people have different fundamental frequency, we only use the difference across time.

There would be a certain threshold of magnitude of fundamental frequency. If the magnitude is too small, this frequency component will be discarded. Also, since there will be some tremble of human voice in small scale if want to hold our pitch when pronouncing 'Ah', there should be a threshold to arbitrate the 'stable' of fundamental frequency.

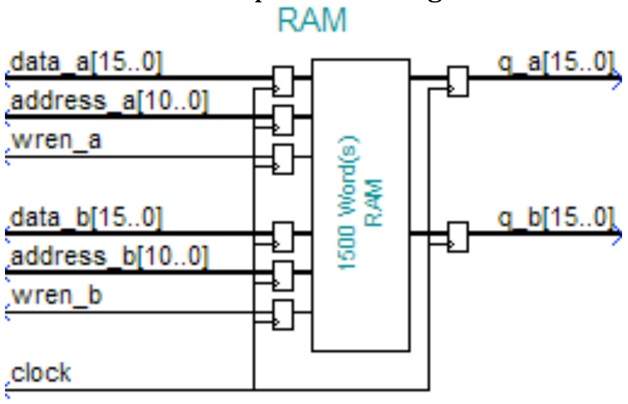
The logic will have 2-bit output. One bit represents the rising of fundamental frequency and the other bit represents the fundamental frequency is 'stable'.

Memory allocation:

The memory between MIC and FFT module mainly consists of two blocks. One is two synchronous RAMs with 10 bit address and 16 bit width and the other is memory controller for bridging the RAMs to both MIC and FFT module and performing Ping-Pong operation. We configure the depth of RAM to 1024 in order to support burst transmission to FFT since FFT requires both start and end signals to be asserted during one transmission, which is inefficient. And the reason why we use two RAMs is to avoid throttle the write operation into RAM when read operation is about to be performed. The scenario is when the MIC has written 1024 chucks of data into RAM, it can write data into the other RAM while FFT is fetching data from the one it just write. Memory controller is coordinating this process. Block diagram of this sub-module is shown as the following figure,



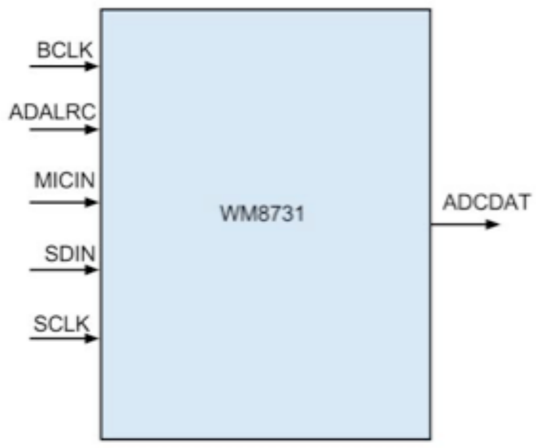
Using off-chip SRAM worked in principle, but it became increasingly difficult to arbitrate reading and writing. In addition, SRAM suffers from the inconvenience of setting up write enable and address a cycle before performing data operations. So we settled on, then, was to use dedicated on-chip storage elements via the two-port RAM Megafunction. This method enjoys the benefits of both previous methods, without their shortcomings. Specifically, the RAM Megafunction handles arbitration, so that the designer can read and write at the same time, but does not use up valuable logic elements.



In order to deal with the asynchronous problem caused by different speed of MIC and FFT, a counter is used to generate one clock cycle write enable signal even though the data is valid during multiple clock cycles. Obviously, the clock frequency of the RAM should be identical to that of the faster device.

The key components of controller is state machines and four counters, two for read and write RAM 0 and the other two for RAM1. Its state transaction diagram is shown as below.

Audio sampling:

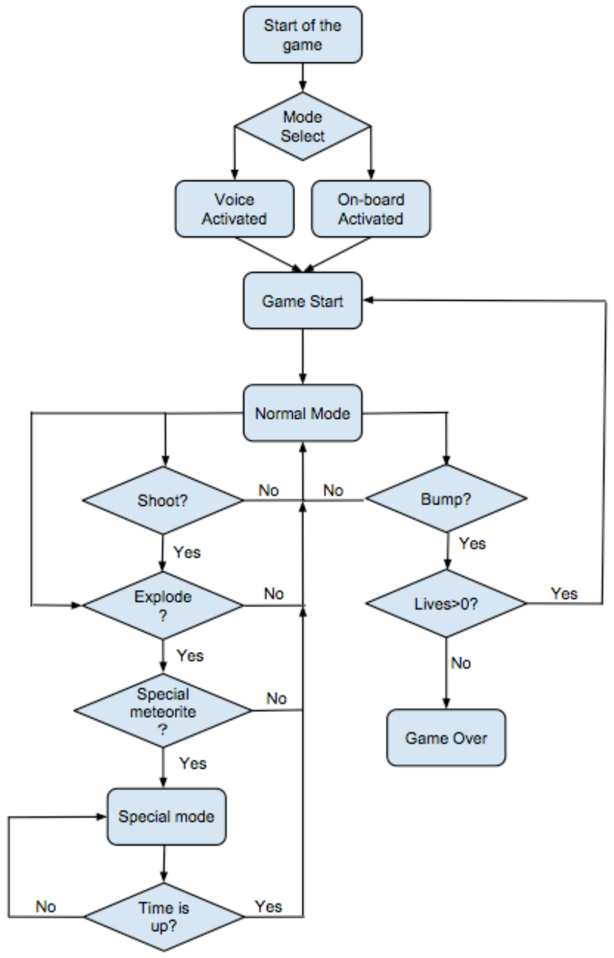


The WM8731 could do the ADC convert. It works at the clock of 18.432 MHz, sampling the audio at the rate of 48 KHz. We choose the data width of 16bits, working at the slave mode. There are five inputs of the chip: BCLK, ADCLRC, MICIN, SDIN and SCLK. BCLK is the 1 bit clock signal work at the frequency of 18.432 MHz. ADCLRC is the 1bit sample rate clock, that is, 48 KHz. MICIN the input from microphone. SCLK is the control clock and the SDIN the 16 bit control data. There is one 16 bit output ADCDAT, which is the digital signal sampled by the chip.

The chip's control signal is work under the I2C protocol, the output data transmitted between the chip and FPGA is work under the I2S protocol. The work flow is as below. The 15:9 bit on the SDIN is the address, that is, 6 bit address and 1 bit programmable select signal. The 8:0 bit on the SDIN is the control data. When master receive the correct address and the R/W bit is 0, which indicates writing, it begins to send data, or it will go to idle state. Then when the output 15:8 data is transmitted, the SDIN will pulled low. then bit 7:0 will be transmitted, then the SDIN pulled low again. After all the data transmitted, SDIN will pull from low to high, indicating the end of the transmitting.

Software System

Game flow diagram:



At first, the beginning screen appears and in this screen we can choose 2 different modes. They are voice activated and on-board key activated. After the mode is selected, the game will start and a simple piece of music will be played. The spaceship will leave a falling platform and start to flying in the universe. In the game, the shoot, explosion, bump actions will be detected and corresponding audio and video control signals will be transferred to av controller. Especially, if a special meteorolite is hit, which is the special mode, some rewards may be obtained by player. After some time, the special effect gained by reward will disappear, and game will return to normal. If the player loses the last life, game is over.

Video processing of X-Y coordinates of moving objects:

In general, Nios II processor will generate the X-Y coordinates of all moving objects including the spaceship, meteorite and the missile. After Nios acquire the audio status signal or onboard key status signal from Avalon bus it will change the Y coordinate of the spaceship by either hold or increasing and decide wither a missile should be launched or not. If the edge of spaceship and meteorite overlap, a explode effect of the spaceship will be generated and this will be achieved by sending control signal to VGA controller through Avalon bus. Similarly, if the edge of a missile and a meteorite overlap, a explode effect of the meteorite will be generated and send to Avalon bus.

The control of other moving objects are similar to what has been discussed above.

Milestones:

Milestone 1 (April 2):

- Make detailed hardware design specification
- Interface voice processing module to Avalon bus
- Configure VGA interface
- Build software skeleton

Milestone 2 (April 16):

- Implement voice processing module and validate it by writing testbench
- Build the game program and test it with designated stimulus

Milestone 3 (April 30):

- Put all together by connecting voice processing module to its interface to avalon bus
- Debug until it totally functions correctly

References:

1. FFT wiki link: http://en.wikipedia.org/wiki/Fast_Fourier_transform
2. Pah! 2.0 App Store link: https://itunes.apple.com/us/app/pah!- 2.0/id547013723?mt=8
3. FFT MageCore Function User Guide: http://www.altera.com/literature/ug/ug_fft.pdf
4. WM8731 datasheet: http://www.cs.columbia.edu/~sedwards/classes/2013/4840/Wolfson-WM8731-audio-CODEC.pdf