Design Document: Screaming Bird

Yiran Hu (yh3639)

Yuesheng Ma (ym2976)

Yang Li (yl5456)

Chenyang Zhou(cz2791)

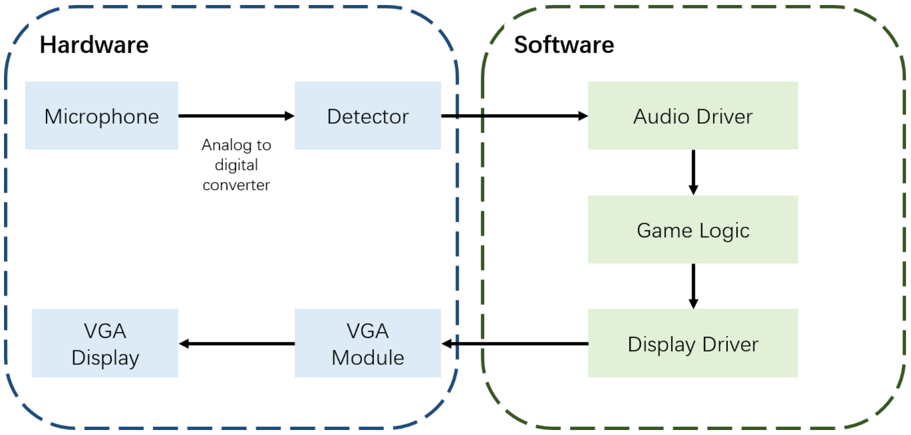
Table of Contents

Introduction

Screaming Bird is a voice control game based on Flappy Bird, where players only need to use one finger to control the flight status of the bird, allowing it to avoid uneven pipes along the way. In our design, we will replace finger manipulation with sound and use sound to control the rise and fall of birds. When a sound exceeding the threshold is detected, the bird will rise, otherwise it will naturally fall.

Block Diagram

The complete system takes in voice commands to control our game displayed on an VGA display. The system is divided into three components: audio module, game logic module, and display module.



Audio Module

Audio Detector

1. Microphone Input
   - Interfaces with the device's microphone to capture audio signals.
   - The microphone input is sampled at a specific sampling rate to convert the analog audio signals into digital samples.
2. Sound Level Threshold Detection
   - Continuously monitors the amplitude of the captured audio samples.
   - Calculates the average amplitude over a short window of time to account for background noise.
   - When the amplitude exceeds a predefined threshold level, indicating a significant sound event (such as a flap), the detector triggers a flap command.
3. Flap Command Generation
   - Upon detecting a flap event, the audio detector generates a flap command signal to instruct the bird to flap its wings.
   - The flap command signal is sent to the audio driver for further processing and integration into the game logic.

Audio Driver

1. Flap Command Synchronization
   - Receives flap commands from the audio detector and synchronizes them with the game's frame rate.
   - Ensures that flap commands are processed and applied to the bird's movement at the appropriate time within each game frame.
2. Integration with Game Logic

- The audio driver communicates with the game logic module to provide input for the bird's flapping action.
- Flap commands generated by the audio detector are translated into corresponding actions within the game, causing the bird to flap its wings.

3. Latency Optimization
    - Communicates with the game logic module to provide input for the bird's flapping action.
    - Flap commands are translated into corresponding actions within the game.

Game Logic Module

Game Mechanics

1. Player Control
    - The player controls the bird's altitude by making sound. Whenever the volume of the sound made by the user reaches a threshold relative to the ambient sound, the bird will flip its wings and ascend a short distance.
    - Gravity constantly pulls the bird downward, causing it to descend if the player does not tap the screen.
    - The player must time their sounds carefully to navigate the bird through openings between the pipes.
2. Obstacles
    - Green pipes serve as obstacles for the bird to navigate through. They are positioned at different heights, with gaps between them for the bird to fly through.
    - Pipes move from right to left across the screen at a constant speed.
    - If the bird collides with any part of a pipe or the ground, the game ends.
3. Scorings
    - The player earns points by successfully navigating the bird through the gaps between pipes.
    - Each successful passage through a pair of pipes earns one point.
    - The player's score is displayed on the screen during gameplay.
4. Difficulty
    - The game becomes progressively more challenging as the player's score increases.
    - The speed of the moving pipes increases when the user's score reaches a certain value (such as 10), making it harder to navigate through the gaps.
5. Game Over
    - The game ends when the bird collides with a pipe or the ground.
    - Upon game over, the player's final score is displayed on the screen.
    - The player can choose to restart the game to try again.

User Interface

1. Game Screen
    - Display: Shows the bird, pipes, and current score.
2. Game Over Screen

- Final Score: Displays the player's score from the most recent game session.

Display Module

Display Driver

1. Graphic Buffer
   - Maintains a graphic buffer that stores the current frame's graphical data, including the bird, pipes, background, and score.
   - Is updated continuously as the game progresses, reflecting changes in the game state.
2. VGA Signal Generation
   - Generates VGA signals, including horizontal sync (HS), vertical sync (VS), and color signals (RGB), to communicate with the VGA module and display device.
   - Synchronizes the timing of these signals to ensure proper display rendering without artifacts or flickering.
3. Frame Rendering
   - Retrieves graphical data from the graphic buffer and converts it into the appropriate format for display.
   - Maps the game's objects to specific pixel positions on the display screen based on their coordinates and dimensions.

VGA Module

1. Signal Conversion
   - Converts digital VGA signals generated by the display driver into analog signals to the VGA display.
   - Includes circuitry for generating the appropriate voltage levels corresponding to the RGB color components and synchronization signals.
2. Timing Control
   - Synchronizes its timing with the display driver to ensure proper signal transmission and display rendering.
   - Adjusts the timing parameters as needed to match the requirements of the connected display device.

VGA Display

1. Analog Signal Reception
   - The analog VGA signals are transmitted through the VGA cable.
   - Amplifies and processes these signals to generate the corresponding pixel colors and synchronize the display's scanning process.
2. Pixel Rendering
   - Renders the received pixel data onto its screen, generating the visual representation of the game.
   - Refreshes the screen at a specific frame rate determined by the VGA signals' timing parameters.

Algorithms

Sound Threshold Detection Relative to Ambient Sound

Continuous Short-Time Fourier Transform (STFT) Approach

1. Continuous Audio Sampling
   - Perform STFT on short, overlapping time windows of the audio sample to capture the frequency spectrum over time.
   - Utilize a frequency range suitable for the game's audio input requirements (e.g., 1200 - 3600 Hz as indicated).
2. Noise Reduction
   - Generate a "fingerprint" of the ambient noise level using the initial STFT data at game start or during quiet periods.
   - Apply bandpass filtering to isolate the frequency band of interest and reduce the effect of noise outside this band.
3. Reconstruct Audio
   - Apply inverse STFT (ISTFT), which reconstructs the time-domain signal from the modified time-frequency representation.
4. Thresholding
   - Define a threshold value for the amplitude. Whenever the amplitude exceeds this threshold, it generates a command signal (e.g., 1) to make the Flappy Bird "flap".

Pseudocode

```
# Initialization
audio_stream = pyaudio.PyAudio().open(...)  # Configure the audio stream
ambient_noise_level = calibrate_ambient_noise(audio_stream)
# Function to calibrate the ambient noise level
def calibrate_ambient_noise(stream):
    # Record audio for calibration
    data = stream.read(...)
    frequencies, times, spectrogram = scipy.signal.stft(data, ...)
    # Calculate the ambient noise level
    return np.mean(spectrogram)
while game_is_running:
    data = audio_stream.read(...)
    frequencies, times, spectrogram = scipy.signal.stft(data, ...)
    # Noise Reduction
    filtered_frequencies =noise_reduction(spectrogram, ambient_noise_level)
   # Reconstruct signal
   flappy_sound=scipy.signal.istft( filtered_frequencies , ...)
```

```
    # Thresholding and game logic
    if flappy_sound>threshold:
        flappy_command=1;
    else:
        flappy_command=0;
```

Physics Simulation

1. Gravity Effect
   - Apply a constant downward acceleration (gravity) to the bird's vertical speed.
   - Update the bird's position based on its speed at each frame.
2. Rise on Command
   - When a "scream" is detected, temporarily apply an upward force that counteracts gravity, simulating a rise.
   - This could be a sudden decrease in vertical speed or setting it to a negative value for a moment.
3. Collision Detection
   - Check if the bird collides with obstacles (pipes) or the ground.
   - Implement game over logic if a collision is detected.

Pseudocode

```
gravity = 9.81
bird_y_velocity = 0
bird_position = [x_initial, y_initial]

# Function to update the bird's position based on physics
def update_physics(time_step):
    global bird_y_velocity, bird_position
    bird_y_velocity -= gravity * time_step
    bird_position[1] += bird_y_velocity * time_step

# Function to simulate rise on scream detection
def trigger_rise():
    global bird_y_velocity
    # Apply an upward force or change in velocity
    bird_y_velocity = -some_upward_velocity
```

Control Algorithm

1. Listen for Commands:
   - Continuously monitor the microphone for sound exceeding the threshold.
   - Trigger a "rise" action whenever such sound is detected.
2. Update Game Physics:
   - At each frame, update the bird's physics based on gravity and any active "rise" commands.

- Adjust the bird's position and handle collisions accordingly.

3. Game Flow Control:
    - Manage game states (start, playing, game over) and transitions between them.
    - Reset game parameters at the start or after a game over to prepare for a new game.

Pseudocode

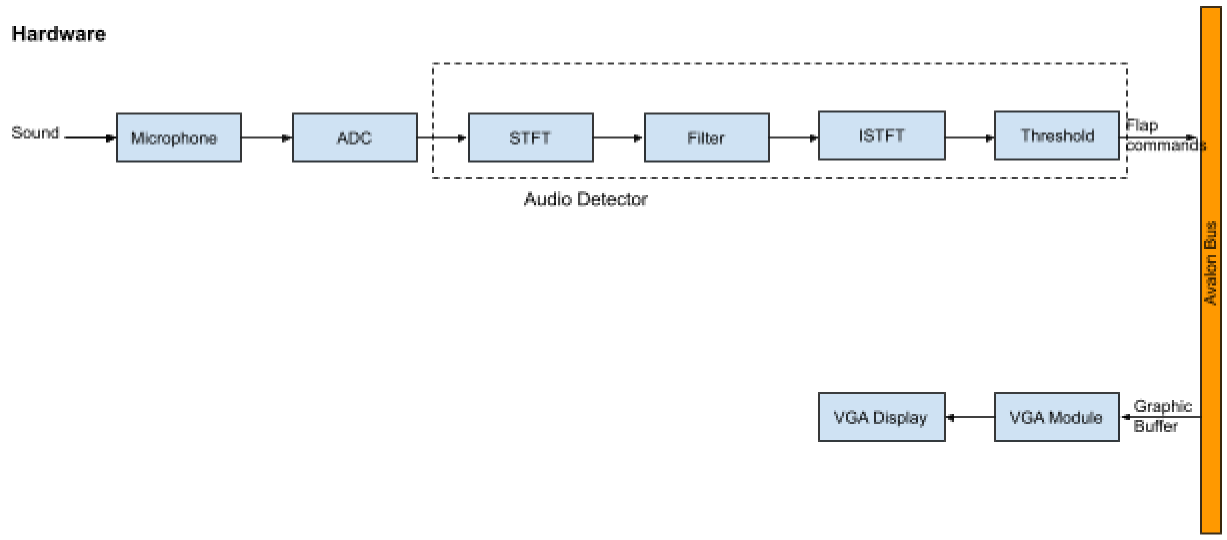
```
while game_is_running:
    # Listen for scream command (pseudo-function representing threshold detection)
    if is_scream_detected(audio_stream):
        trigger_rise()
    update_physics(time_step)
    if check_collision(bird_position, obstacles):
        handle_game_over()
    update_game_state()
    # Render the frame (pseudo-function representing graphics rendering)
    render_frame(bird_position, obstacles)
```

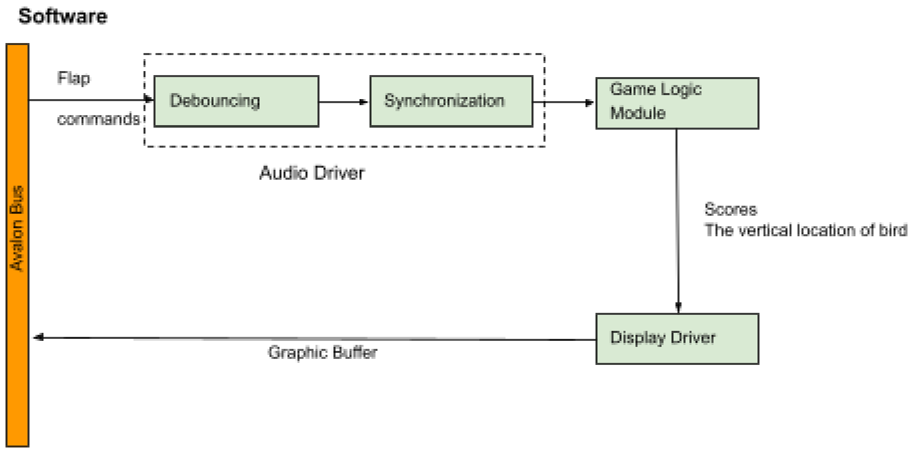
Resource Budget

The cyclone V has two types of BRAMs, MLABs and M10Ks. According to the document, we have 138 Kb MlAB memory and 1400 Kb M10K memory.
The approximate memory required is listed as shown in the table below.

Element	Number of sprites	Pixel size	Size
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

Dataflow





Hardware/Software Interface

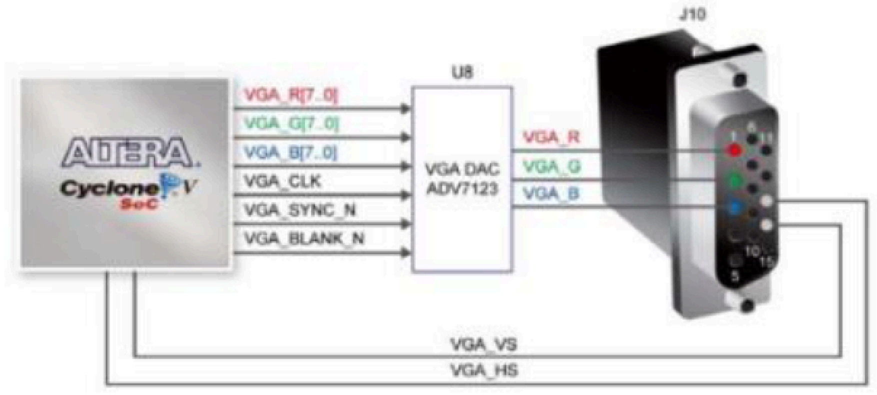
Address	Register name	Description	
0	bird_position_y	We only need the position of y because x position is fixed.	
1	background_color_R	R background color	
2	background_color_G	G background color	
3	background_color_B	B background color	
4	voice_command	If we receive voices, the voice_command will be	

		1, otherwise it will be 0.	
5	score	The bird score.	
6	dead	If the bird touches the pipe, the dead will be 1, otherwise it will be 0.	
7	pipe_position_x	Get the pipe position x.	
8	pipe_up_down	The pipe is up or down in the screen	
9	pipe_height	How long of the pipe.	

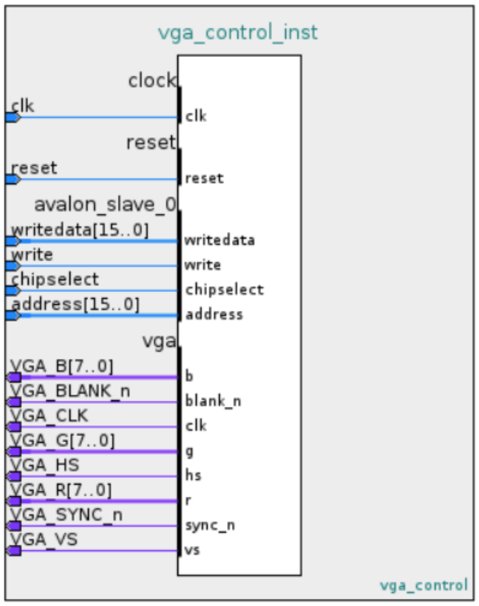
From peripheral to bus, it reads the audio data from the hardware part. Then, the software will do some operations on the data. It also reads the background color, bird position and some other environment information. After the game logic and display module. The software part will write the display module information like bird position and the environment information to the hardware. Then the hardware will display it on the monitor.

VGA

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 AnalogDevices 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 figure below shows the associated schematic.



VGA Controller



The VGA controller receives an address and writes data according to our software-to-hardware data packet. The 16 addresses will store all sprite attribute tables and display them on the screen accordingly.

Microphone

The microphone sensor module has 4 pins VCC, GND, Digital Out, and Analog Out. We here use the AO pin as an output for analog reading and connect it to the ADC header in the DE1-SoC board.

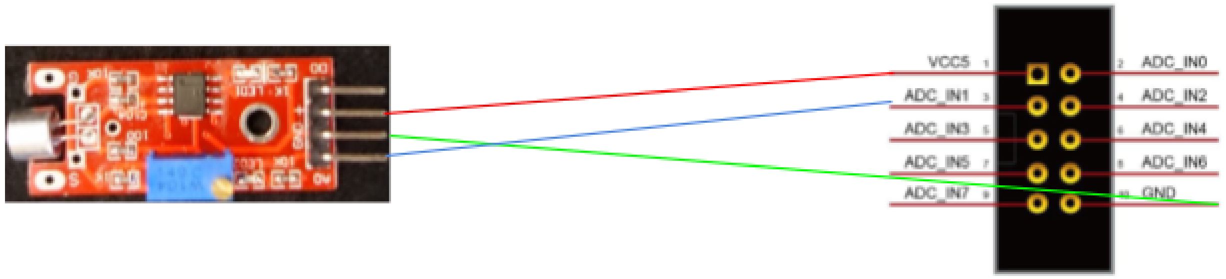
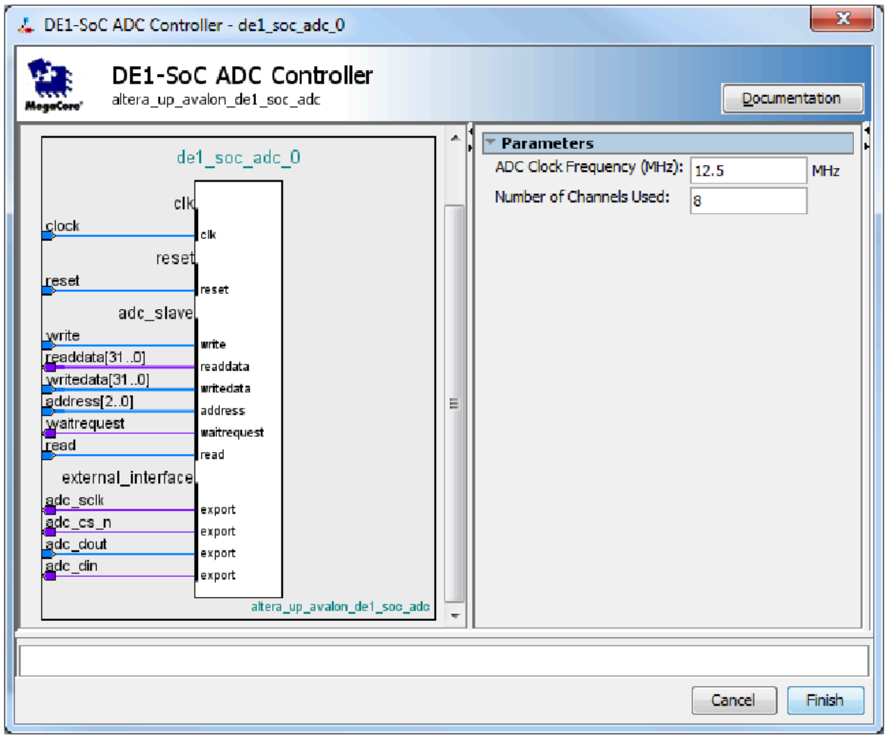


Figure 3-29 Pin distribution of the 2x5 Header

ADC Controller

The DE1-SoC ADC Controller IP Core provides access to all 8 input channels of the AD7928 Analog-to-Digital Converter. We can use the core's configuration wizard to specify the number of channels to be read by the ADC Controller as shown in the figure below. (The *Number of Channels Used* would be 1)



ADC Clock Frequency (MHz): 12.5 MHz

Number of Channels Used: 8

Milestones

Milestone 1

1. Implement audio interface on the FPGA board.
2. Implement display interface on the FPGA board.

Milestone 2

1. Software driver interface with hardware
2. Implement the Game logic

Milestone 3

1. Complete Graphics
2. Finish the project

References

1. http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/de1-soc_user_manual.pdf
2. https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/DE1-SoC_ADC_Controller.pdf
3. https://www.intel.com/content/www/us/en/docs/programmable/683694/current/maximum-resources-66420.html