**Duck Hunt '22**

Design Document

Kristen Shaker (kls2243)
Alex Yao (awy2108)
Bryce Natter (bdn2113)

We are recreating the classic NES game “Duck Hunt.” The premise of the game is that ducks are introduced from the bottom of the screen and fly in a randomized pattern. The player has three shots to hit the ducks. After shooting all three bullets, the state resets and new ducks are introduced. We will use a Wii remote to control the position of a crosshair on the screen displayed by the VGA. If a player successfully positions the crosshair over a duck and clicks the trigger, they have successfully shot the duck. The player score will be computed from the number of ducks shot in a single game instance, and we will incorporate varying game difficulties.

We will be using tile and sprite graphics to display our visuals. The game logic and input will be handled by software. We will prioritize making a functional Wii remote controller for our game, which will communicate to the FPGA via Bluetooth. The software will communicate coordinates and game state to the hardware for display on a screen.
# HW/SW Interface

## Register Table

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>[1:0]</td>
<td>GameState</td>
<td>Controls background for displaying start screen (0), game (1), and game over (2).</td>
</tr>
<tr>
<td>0x020</td>
<td>[31:0]</td>
<td>Score</td>
<td>Current Score</td>
</tr>
<tr>
<td>0x040</td>
<td>[31:0]</td>
<td>Bullets</td>
<td>Number of Remaining Bullets</td>
</tr>
<tr>
<td>0x060</td>
<td>[31:0]</td>
<td>Round</td>
<td>Current Round</td>
</tr>
<tr>
<td>0x080</td>
<td>[19:10]</td>
<td>Crosshair X</td>
<td>X and Y location of crosshair</td>
</tr>
<tr>
<td></td>
<td>[9:0]</td>
<td>Crosshair Y</td>
<td></td>
</tr>
<tr>
<td>0x0A0</td>
<td>[24:20]</td>
<td>D1 Sprite</td>
<td>Control parameters for duck 1</td>
</tr>
<tr>
<td></td>
<td>[19:10]</td>
<td>D1 X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[9:0]</td>
<td>D1 Y</td>
<td></td>
</tr>
<tr>
<td>0x0D0</td>
<td>[24:20]</td>
<td>D2 Sprite</td>
<td>Control parameters for duck 2</td>
</tr>
<tr>
<td></td>
<td>[19:10]</td>
<td>D2 X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[9:0]</td>
<td>D2 Y</td>
<td></td>
</tr>
</tbody>
</table>
Algorithms

The core game logic and game state will be updated through control algorithms on the software side. The main states we keep track of during gameplay are:

- Game state (score, round, bullets remaining, ducks hit)
- Duck states (position, velocity vector, sprite option)
- Crosshair position

**Game state** - The main game loop will be responsible for tracking the gameplay in one instance. On game over, the loop will reset all states for a new playthrough. The game state we track during gameplay are the scores (increasing by multiples of 500 as ducks are shot), bullets remaining (from 3), round (increasing after each set of ducks), and total ducks hit (used for display). The score displays are communicated over to hardware for display.

**Duck states** - Each duck state encapsulates the information that should be communicated to the hardware during each frame (position, sprite option) as well as some meta information about the velocity of the duck. This provides sufficient information to determine duck position at the next frame. E.g. If the duck has been shot, decrease Y position and display “dead” sprite.

**Crosshair position** - This information is polled from the Wii remote controller during each pass and checked for collision with ducks that are not dead or inactive. We note that there is required setup before the main game loop to sync the Wii controller for accurate positions.

We have produced the following pseudocode which captures the general interfaces, components, and main function logic of our program:
enum duck_state { flap_up, flap_down, dead, inactive };

typedef struct {
    unsigned int x, y;
    unsigned int vx, vy;
    unsigned int value;
    duck_state state;
} duck_config_t;

typedef struct {
    unsigned int x, y;
} duck_coord_t;

typedef struct {
    unsigned int bullets, score, round;
} game_config_t;

// Calculates hit-square associated with the x,y position of the duck. returns 1 if 
// wii controller x, y coordinates are in the hit-square.
int shoot(duck_coord_t * coord);

// called during game initialization. set number of bullets to three, score to zero 
// and round to 1.
int set_up_config(game_config_t * config);

// called when a player shoots or when a round is over.
int update_game_config(game_config_t * config);

// uses ioctl calls to tell our driver to update the coordinate of the duck.
int write_duck_coord(duck_config_t * config);

// uses ioctl calls to tell our driver to update the state of the duck.
int write_duck_state(duck_config_t * config);

int poll_wii_controller();

int main() {
    // sync Wii remote locations

    while(1){
        play_game();
    }
}

int play_game(){
    // setup logic, send stuff to hardware
duck_config_t duck;
game_config_t game;
set_up_config(&game);
while (1) {
    // poll for x,y values and trigger press
    poll_wii();
    if (is_game_over())
    {
        return;
    }
    if (trigger_pressed)
    {
        if (shoot_duck())
        {
            update_duck_state(); // change duck state to dead;
            game.score += duck.value;
        }
        game.num_bullets -= 1;
    }
    // Note: ducks below location Y on screen are hidden behind bush sprites
    // compute location of ducks
    move_duck(&duck);

    update_game_config(&game); // ioctl calls to update hardware
    write_duck_coord(&duck);

} return 0;
# Resource Budget

<table>
<thead>
<tr>
<th>Category</th>
<th>Image</th>
<th>Size (pixels)</th>
<th>Variants</th>
<th>Total Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duck</td>
<td><img src="image" alt="Duck Image" /></td>
<td>40 x 34</td>
<td>3</td>
<td>$3 \times 40 \times 34 \times 4 = 16,320$ bits</td>
</tr>
<tr>
<td>Bullet</td>
<td><img src="image" alt="Bullet Image" /></td>
<td>16 x 16</td>
<td>1</td>
<td>$1 \times 16 \times 16 \times 4 = 1024$ bits</td>
</tr>
<tr>
<td>Background</td>
<td><img src="image" alt="Background Image" /></td>
<td>16 x 16</td>
<td>8</td>
<td>$8 \times 16 \times 16 \times 4 = 8192$ bits</td>
</tr>
<tr>
<td>Score (numbers)</td>
<td><img src="image" alt="Score Number Image" /></td>
<td>16 x 16</td>
<td>10</td>
<td>$10 \times 16 \times 16 \times 4 = 10240$ bits</td>
</tr>
<tr>
<td>Ducks Counter</td>
<td><img src="image" alt="Ducks Counter Image" /></td>
<td>16 x 16</td>
<td>1</td>
<td>$1 \times 16 \times 16 \times 4 = 1024$ bits</td>
</tr>
<tr>
<td>Color Pallet</td>
<td><img src="image" alt="Color Pallet Image" /></td>
<td></td>
<td></td>
<td>$10 \times 24 = 240$ bits</td>
</tr>
</tbody>
</table>

Total Memory: 37040bits = 4.63 kB