UNIVERSIDADE FEDERAL DE SÃO CARLOS
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Arquiteturas de Alto Desempenho

Implementação de jogo arcade em processador ARM

Guilherme Gomes Arcencio - 769731 - Engenharia de Computação
Guilherme Silva Castro - 769763 - Ciência da Computação

1 Introdução

Neste experimento, foi implementado um jogo arcade no processador ARM de uma FPGA (Field-Programmable Gate Array), com a saída visualizada através de uma interface VGA (Video Graphics Array). O jogo escolhido para implementação é uma réplica do jogo Flappy Bird [1].

2 Desenvolvimento teórico

Flappy Bird é um jogo eletrônico de estilo arcade para dispositivos móveis que alcançou notoriedade global por sua jogabilidade extremamente simples, porém altamente desafiadora. O objetivo do jogador é guiar um pássaro através de um cenário de rolagem lateral, fazendo-o passar por uma série de vãos entre canos verdes. Há apenas um único botão de interação do jogador, que faz o pássaro bater as asas e ganhar uma pequena impulsão vertical, combatendo a força constante da gravidade que o puxa para baixo. A pontuação é calculada de forma direta, com o jogador ganhando um ponto para cada par de canos ultrapassado com sucesso.

O jogo foi implementado no processador ARM de uma placa FPGA DE10-Standard, da Intel, utilizando a ferramenta Intel FPGA Monitor Program. Os botões da placa foram utilizados para o controle do jogador, e a sua interface VGA foi usada pra exibir o jogo em um monitor.

3 Desenvolvimento prático

Nesta seção, é explicado o código do jogo, exibido a seguir.

```
1  #include "address_map_arm.h"
2
3  void video_text(int, int, char *);
4  void video_box(int, int, int, int, short);
5
6  void my_srand(unsigned int seed);
7  unsigned int my_rand(void);
8  void format_score(char* buf, int score_val);
9
10 #define SCREEN_WIDTH 320
11 #define SCREEN_HEIGHT 240
12
13 #define BLACK 0x0000
```

```
14 #define WHITE 0xFFFF
15 #define YELLOW 0xFFE0
16 #define GREEN 0x07E0
17 #define SKY_BLUE 0x5CF9
18
19 #define BIRD_WIDTH 10
20 #define BIRD_HEIGHT 10
21 #define PIPE_WIDTH 20
22 #define PIPE_GAP 90
23
24 #define GRAVITY 1
25 #define JUMP_STRENGTH -5
26
27 typedef struct {
28     int x;
29     int y;
30     int vy;
31 } Bird;
32
33 typedef struct {
34     int x;
35     int gap_y;
36     int scored;
37 } Pipe;
38
39 volatile int * KEY_ptr = (int *)KEY_BASE;
40 volatile int * SW_ptr = (int *)SW_BASE;
41 volatile int * pixel_ctrl_ptr = (int *)PIXEL_BUF_CTRL_BASE;
42 int pixel_buffer_start;
43 Bird bird;
44 Pipe pipes[2];
45 int score;
46 int game_over;
47 static unsigned int random_seed;
48
49 void setup();
50 void loop();
51 void initialize_game();
52 void draw_bird();
```

```
53 void erase_bird();
54 void update_bird();
55 void initialize_pipes();
56 void draw_pipes();
57 void erase_pipes();
58 void update_pipes();
59 void check_collision();
60 void wait_for_vsync();
61 void clear_screen();
62 void draw_score();
63 void clear_text();
64
65 int main(void) {
66     setup();
67     while (1) {
68         loop();
69     }
70     return 0;
71 }
72
73 void setup() {
74     my_srand(*SW_ptr);
75
76     *(pixel_ctrl_ptr + 1) = FPGA_PIXEL_BUF_BASE;
77     wait_for_vsync();
78     pixel_buffer_start = *pixel_ctrl_ptr;
79     clear_screen();
80
81     *(pixel_ctrl_ptr + 1) = FPGA_PIXEL_BUF_BASE;
82     pixel_buffer_start = *(pixel_ctrl_ptr + 1);
83     clear_screen();
84
85     initialize_game();
86 }
87
88 void loop() {
89     int prev_bird_y = bird.y;
90
91     if ((*KEY_ptr & 0b0001) != 0) {
```

```
92          if (!game_over) {
93              bird.vy = JUMP_STRENGTH;
94          }
95      }
96
97      if ((*KEY_ptr & 0b1000) != 0) {
98          if (game_over) {
99              initialize_game();
100         }
101     }
102
103     if (!game_over) {
104         erase_bird(prev_bird_y);
105         erase_pipes();
106
107         update_bird();
108         update_pipes();
109         check_collision();
110
111         draw_pipes();
112         draw_bird();
113         draw_score();
114     } else {
115         char final_score_text[20];
116         video_text(35, 29, "Game Over");
117         format_score(final_score_text, score);
118         video_text(35, 31, final_score_text);
119         video_text(28, 33, "Press KEY3 to restart");
120     }
121
122     wait_for_vsync();
123     pixel_buffer_start = *(pixel_ctrl_ptr + 1);
124 }
125
126 void my_srand(unsigned int seed) {
127     random_seed = seed;
128 }
129
130 unsigned int my_rand(void) {
```

```
    random_seed = 1664525 * random_seed + 1013904223;
    return random_seed;
}


char* int_to_str(int n, char* buf) {
    char temp_buf[10];
    int i = 0;

    if (n == 0) {
        *buf++ = '0';
        *buf = '\0';
        return buf;
    }

    while (n > 0) {
        temp_buf[i++] = (n % 10) + '0';
        n /= 10;
    }

    while (i > 0) {
        *buf++ = temp_buf[--i];
    }
    *buf = '\0';
    return buf;
}

void format_score(char* buf, int score_val) {
    char* prefix = "Score: ";
    while (*prefix) {
        *buf++ = *prefix++;
    }

    int_to_str(score_val, buf);
}

void initialize_game() {
    clear_screen();
    bird.x = 50;
```

```
170     bird.y = SCREEN_HEIGHT / 2;
171     bird.vy = JUMP_STRENGTH;
172     score = 0;
173     game_over = 0;
174     initialize_pipes();
175 }
176
177 void draw_bird() {
178     video_box(bird.x, bird.y, bird.x + BIRD_WIDTH - 1, bird.y +
        ↪  BIRD_HEIGHT - 1, YELLOW);
179 }
180
181 void erase_bird(int y) {
182     video_box(bird.x, y, bird.x + BIRD_WIDTH - 1, y + BIRD_HEIGHT -
        ↪  1, SKY_BLUE);
183 }
184
185 void update_bird() {
186     bird.vy += GRAVITY;
187     if (bird.vy > 5) {
188         bird.vy = 5;
189     }
190
191     bird.y += bird.vy;
192
193     if (bird.y < 0) {
194         bird.y = 0;
195         bird.vy = 0;
196     }
197 }
198
199 void initialize_pipes() {
200     pipes[0].x = SCREEN_WIDTH;
201     pipes[0].gap_y = (my_rand() % (SCREEN_HEIGHT - PIPE_GAP)) +
        ↪  PIPE_GAP / 2;
202     pipes[0].scored = 0;
203
204     pipes[1].x = SCREEN_WIDTH + (SCREEN_WIDTH / 2) + (PIPE_WIDTH /
        ↪  2);
```

```
205     pipes[1].gap_y = (my_rand() % (SCREEN_HEIGHT - PIPE_GAP)) +
        ↪ PIPE_GAP / 2;
206     pipes[1].scored = 0;
207 }
208
209 void draw_pipes() {
210     int i;
211     for (i = 0; i < 2; ++i) {
212         video_box(pipes[i].x, 0, pipes[i].x + PIPE_WIDTH - 1,
            ↪ pipes[i].gap_y - (PIPE_GAP / 2), GREEN);
213         video_box(pipes[i].x, pipes[i].gap_y + (PIPE_GAP / 2),
            ↪ pipes[i].x + PIPE_WIDTH - 1, SCREEN_HEIGHT - 1, GREEN);
214     }
215 }
216
217 void erase_pipes() {
218     int i;
219     for (i = 0; i < 2; ++i) {
220         video_box(pipes[i].x, 0, pipes[i].x + PIPE_WIDTH - 1,
            ↪ pipes[i].gap_y - (PIPE_GAP / 2), SKY_BLUE);
221         video_box(pipes[i].x, pipes[i].gap_y + (PIPE_GAP / 2),
            ↪ pipes[i].x + PIPE_WIDTH - 1, SCREEN_HEIGHT - 1,
            ↪ SKY_BLUE);
222     }
223 }
224
225 void update_pipes() {
226     int i;
227     for (i = 0; i < 2; ++i) {
228         pipes[i].x -= 1;
229
230         if (pipes[i].x + PIPE_WIDTH < 0) {
231             pipes[i].x = SCREEN_WIDTH;
232             pipes[i].gap_y = (my_rand() % (SCREEN_HEIGHT - PIPE_GAP))
                ↪ + PIPE_GAP / 2;
233             pipes[i].scored = 0;
234         }
235
236         if (!pipes[i].scored && pipes[i].x + PIPE_WIDTH < bird.x) {
```

```
237             score++;
238             pipes[i].scored = 1;
239         }
240     }
241 }
242
243 void check_collision() {
244     int i;
245     if (bird.y + BIRD_HEIGHT > SCREEN_HEIGHT) {
246         game_over = 1;
247         return;
248     }
249
250     for (i = 0; i < 2; ++i) {
251         if (bird.x + BIRD_WIDTH > pipes[i].x && bird.x < pipes[i].x +
        ↪  PIPE_WIDTH) {
252             if (bird.y < pipes[i].gap_y - (PIPE_GAP / 2) || bird.y +
            ↪  BIRD_HEIGHT > pipes[i].gap_y + (PIPE_GAP / 2)) {
253                 game_over = 1;
254                 return;
255             }
256         }
257     }
258 }
259
260 void wait_for_vsync() {
261     *pixel_ctrl_ptr = 1;
262     while ((*(pixel_ctrl_ptr + 3) & 1) != 0);
263 }
264
265 void video_box(int x1, int y1, int x2, int y2, short pixel_color) {
266     int row, col;
267
268     if (x1 < 0) x1 = 0;
269     if (y1 < 0) y1 = 0;
270     if (x2 >= SCREEN_WIDTH) x2 = SCREEN_WIDTH - 1;
271     if (y2 >= SCREEN_HEIGHT) y2 = SCREEN_HEIGHT - 1;
272
273     for (row = y1; row <= y2; row++) {
```

```
274         for (col = x1; col <= x2; ++col) {
275             volatile short *pixel_ptr = (short *)(pixel_buffer_start
                ↪  + (row << 10) + (col << 1));
276             *pixel_ptr = pixel_color;
277         }
278     }
279 }
280 
281 void clear_screen() {
282     clear_text();
283     video_box(0, 0, SCREEN_WIDTH - 1, SCREEN_HEIGHT - 1, SKY_BLUE);
284 }
285 
286 void draw_score() {
287     char score_text[15];
288     format_score(score_text, score);
289     video_text(2, 2, score_text);
290 }
291 
292 void video_text(int x, int y, char * text_ptr) {
293     int offset;
294     volatile char * character_buffer = (char *)FPGA_CHAR_BASE;
295 
296     offset = (y << 7) + x;
297     while (*(text_ptr)) {
298         *(character_buffer + offset) = *(text_ptr);
299         ++text_ptr;
300         ++offset;
301     }
302 }
303 
304 void clear_text() {
305     int x, y;
306     for (x = 0; x < 80; x++) {
307         for (y = 0; y < 60; y++) {
308             video_text(x, y, " ");
309         }
310     }
```

```
311 }
```

O programa é estruturado em torno de um laço infinito que assegura a sua execução contínua. A lógica é organizada principalmente nas funções `setup()` e `loop()`. A função `setup()` é executada uma única vez no início para inicializar o sistema, configurando o controlador de vídeo para apontar para o buffer de pixels na memória. Em seguida, a função `initialize_game()` é chamada para definir o estado inicial do jogo, posicionando o pássaro, zerando a pontuação e inicializando os canos fora da tela, além de zerar a variável de controle `game_over`.

A função `loop()` contém a lógica central executada a cada quadro, gerenciando o estado do jogo através da flag `game_over`. Enquanto o jogo está ativo, o código processa o salto do pássaro, atualiza as posições de todos os elementos, verifica colisões e redesenha a tela. Se o jogo termina, uma tela de "Game Over"é exibida com a pontuação final, e o sistema aguarda que o jogador pressione o botão `KEY3` para reiniciar. Para organizar os dados, foram utilizadas estruturas que representam os elementos principais: `Bird`, que armazena a posição e velocidade vertical do pássaro, e `Pipe`, que guarda a posição de um par de canos, a localização de seu vão e um indicador para controle de pontuação.

As mecânicas de jogo são controladas por funções específicas. A cada quadro, a velocidade vertical do pássaro é incrementada para simular a gravidade. Quando o jogador pressiona `KEY0`, a velocidade recebe um valor negativo (`JUMP_STRENGTH`), impulsionando o pássaro para cima. Os canos se movem horizontalmente da direita para a esquerda com velocidade constante. Ao saírem da tela, são reposicionados no lado direito com uma nova altura de vão, calculada aleatoriamente para manter o desafio. A pontuação é incrementada quando o pássaro ultrapassa um cano, e uma flag interna impede a pontuação múltipla no mesmo obstáculo. A colisão é detectada se o pássaro toca a borda inferior da tela ou se suas coordenadas se sobrepõem às de um cano, ativando o estado de "Game Over".

A renderização visual é realizada pela manipulação direta da memória de vídeo da FPGA, acessada através de ponteiros para endereços de hardware específicos, como os dos botões e do controlador de vídeo. A função `video_box()` é a primitiva usada para desenhar retângulos, formando o pássaro e os canos. A animação é criada ao "apagar"o objeto em sua posição antiga (desenhando-o com a cor de fundo) antes de desenhá-lo na nova posição. Texto, como a pontuação, é exibido pela função `video_text()`. Para evitar artefatos visuais como screen tearing, o sistema emprega double buffering, alternando entre dois buffers de imagem. A função `wait_for_vsync()` sincroniza as operações de desenho com o ciclo de atualização do monitor, pausando a execução até o fim da varredura vertical para então trocar os buffers, o que resulta em uma animação fluida e sem falhas.

4 Discussão dos resultados

O jogo em execução na placa, e exibido no monitor, é mostrado na Figura 1. Também é possível visualizar o jogo em execução no vídeo `https://youtu.be/BfRBzJd9EHc`.

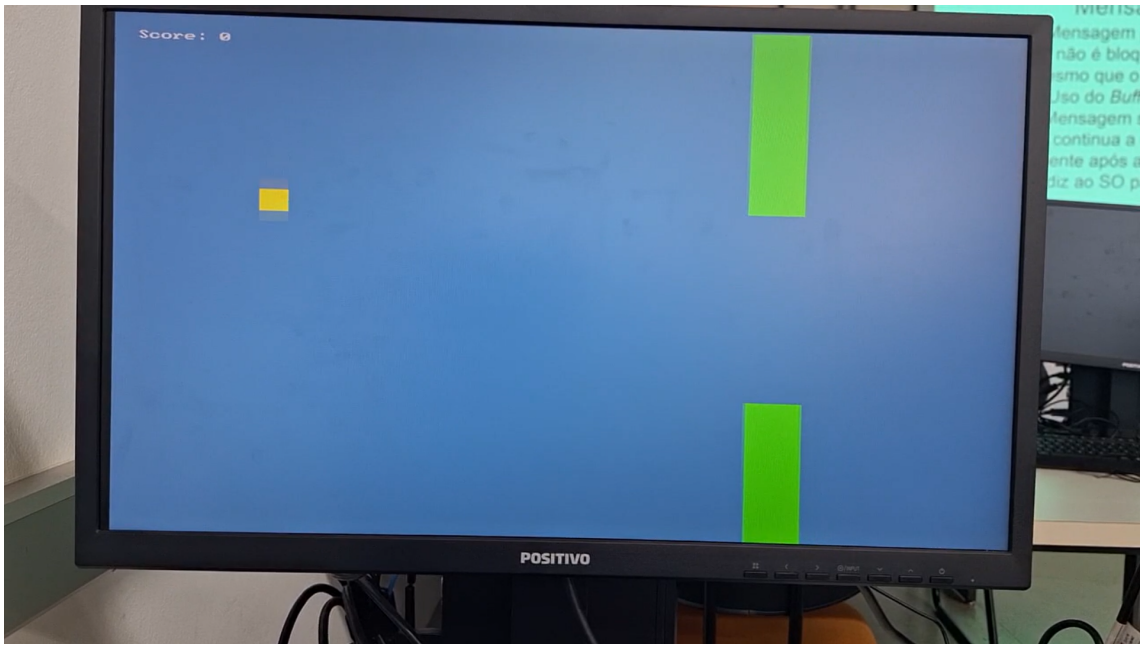

Figura 1: Réplica do Flappy Bird em execução na FPGA.

5 Conclusões

O desenvolvimento da réplica do jogo Flappy Bird no processador ARM de uma placa FPGA DE10-Standard cumpriu com sucesso os objetivos propostos, demonstrando a aplicação prática de conceitos de arquiteturas de alto desempenho. O projeto integrou a manipulação de hardware em baixo nível, como o controle de botões para entrada do jogador e a escrita direta na memória de vídeo para saída VGA, com a lógica de software necessária para a execução de um jogo interativo. A implementação bem-sucedida da mecânica do jogo, incluindo a simulação de gravidade, detecção de colisão, e a geração procedural de obstáculos, resultou em uma experiência funcional e fluida.

Referências

1 WILLIAMS, Rhiannon. **What is Flappy Bird? The game taking the App Store by storm**. 2014. Disponível em: <`https://www.telegraph.co.uk/`

technology/news/10604366/What-is-Flappy-Bird-The-game-taking-the-App-Store-by-storm.html>. Acesso em: 10 jul. 2025.