UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC
CENTRO DE ENGENHARIAS DE MOBILIDADE

Wellington Rodrigo Gallo

Implementação do jogo '*Air Hockey Game*-Table' para FPGA com VGA

Joinville
2015

Wellington Rodrigo Gallo

Implementação do jogo '*Air Hockey Game*-Table' para FPGA com VGA

Relatório técnico apresentado como requisito parcial para obtenção de aprovação na disciplina Hardware para Sistemas Embarcados, no Curso de Engenharia Mecatrônica, na Universidade Federal de Santa Catarina.

Prof. Dr. Anderson Wedderhoff Spengler

Joinville

2015

O Jogo

“Hóquei de ar ou Air-Hockey é um esporte em que dois jogadores oponentes rebatem um disco que flutua sobre uma camada de ar frio no intento de marcar pontos inserindo-o na meta adversária.”[1]

O jogo é comumente jogado apenas em duas pessoas, porém o mesmo também pode ser jogado em duplas, aqui vemos uma mesa oficial para o jogo em dupla:

Figura 1 – Air Hockey Table para Duplas

Neste projeto foi desenvolvido a mesa para apenas dois jogadores, onde cada um controlando o seu próprio “paddle” tenta acertar a bola rebatendo pela mesa e acertar o gol do oponente, como podemos ver este exemplo de uma mesa de hóquei de ar virtual:

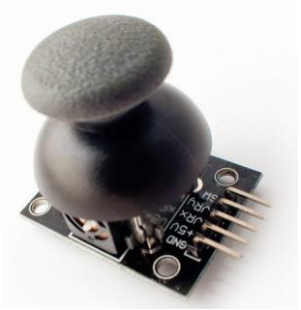
Figura 2 – Exemplo de um Hóquei de Ar Virtual para Dois Jogadores

Neste projeto busca-se desenvolver um jogo semelhante, porém com a mesa virtual sendo mostrada de cima de forma que os gols fiquem nos cantos horizontais da tela, onde os paddles sejam movimentáveis através de *joysticks*, sendo um joystick para cada paddle.

Os Joysticks

Para este projeto foram usados dois analógicos da marca Keyes_Sjoys, que são comumente vendidos em kits de desenvolvimento para arduino.

Figura 3 – Analógico Usado no Projeto



Como se pode ver o dispositivo usa saídas analógicas, visto que ele funciona através de potenciômetros que regulam as saídas de tensão Rx e Ry baseados nos movimentos do analógico tendo como referencial as tensões de entrada (Vcc e Gnd), desta forma fora utilizado o microcontrolador TM4C123GH6PM, para realizar as quatro

conversões analógicas, duas para cada analógico, e analisar quando houve alguma movimentação e identificar os eixos em que houve movimentação levando em consideração uma pequena zona morta para ignorar qualquer movimento dentro daquela área, e assim enviar um sinal alto identificando que houve movimentação em tal eixo e então em outro pino um sinal alto é enviado identificando que o eixo foi puxado ao seu extremo valor ou então um sinal baixo é enviado identificando que o eixo foi puxado ao seu mínimo valor, desta forma temos os seguintes pinos que necessitam de conversão analógica para digital:

- Paddle0_Rx;
- Paddle0_Ry;
- Paddle1_Rx;
- Paddle1_Ry;

E depois de feita a conversão analógica para digital e analisado o seu comportamento para cada eixo, o microcontrolador é responsável por enviar os seguintes sinais:

- Paddle0_Rx_EN;
    - Informa que houve movimentação em pad0_Rx;
- Paddle0_Rx;
    - Informa qual o sentido de movimentação que houve;
- Paddle0_Ry_EN;
    - Informa que houve movimentação em pad0_Ry;
- Paddle0_Ry;
    - Informa qual o sentido de movimentação que houve;
- Paddle1_Rx_EN;
    - Informa que houve movimentação em pad0_Ry;
- Paddle1_Rx;
    - Informa qual o sentido de movimentação que houve;
- Paddle1_Ry_EN;
    - Informa que houve movimentação em pad1_Ry;
- Paddle1_Ry;
    - Informa qual o sentido de movimentação que houve;

Desta forma cada analógico conecta ao todo 4 pinos cada no microcontrolador, e o microcontrolador retorna outros 4 pinos para cada analógico informando informações

de movimento que são utilizadas para representar a movimentação do paddle na mesa virtual.

As configurações de conexão dos analógicos no TIVA segue da seguinte forma:

- pad0_gnd => GND
- pad0_+5V => 3v3
- pad0VRx => GPIO_E[3]
- pad0VRy => GPIO_E[2]
- pad0SW => (Não usado neste projeto)
- pad1_gnd => GND
- pad1_+5V => 3v3
- pad1VRx => GPIO_D[3]
- pad1VRy => GPIO_D[2]
- pad1SW => (Não usado neste projeto)

As configurações de conexão do TIVA com a placa altera DE2 segue da seguinte forma:

Notação na placa altera DE2:

- GPIO_0 é o mesmo que JP1
- GPIO_1 é o mesmo que JP2

Conexão (TIVA => Altera DE2):

- GPIO_C[4] => GPIO_1[33]
- GPIO_C[5] => GPIO_1[31]
- GPIO_C[6] => GPIO_1[29]
- GPIO_C[7] => GPIO_1[27]
- GPIO_E[4] => GPIO_0[34]
- GPIO_E[5] => GPIO_0[35]
- GPIO_A[6] => GPIO_1[34]
- GPIO_A[7] => GPIO_1[35]

O código utilizado no Code Composer Studio para programar o TIVA para este projeto está em conjunto com o arquivo “.pdf” deste relatório, e para poder configurar os pinos de saída do TIVA para uso no altera DE2 define-se um arquivo “*cvs” com os seguintes pinos à serem importados para o projeto:

```
To,Location
pad0_Ry_EN      ,PIN_U21
pad0_Ry         ,PIN_T19
pad0_Rx_EN      ,PIN_V23
pad0_Rx         ,PIN_V25

pad1_Ry_EN      ,PIN_L25
pad1_Ry         ,PIN_L19
pad1_Rx_EN      ,PIN_W25
pad1_Rx         ,PIN_W23
```

O Layout da Mesa

Para poder representar os elementos na mesa, antes de mais nada algumas imagens são preparadas na forma de matrizes de cores, escrita de uma forma que a linguagem utilizada na FPGA Altera DE2 possa entender, o que no caso é a linguagem VHDL, para fazer isso as imagens são preparadas e então modificadas com um editor de imagem para assim poder analisar a imagem e diminuir ao máximo a quantidade de cores existente em cada imagem, após isso fora desenvolvido um código em Python utilizando algumas funções das bibliotecas “scipy”, “PIL” e “pylab” para assim ler as imagens finais e então criar um arquivo de texto no formato “*txt” com uma matriz de cores, com a quantidade de linhas da matriz correspondente a quantidade de pixel verticais da imagem, e a quantidade de colunas da matriz correspondente a quantidade de pixel horizontais da imagem, em cada elemento dessa matriz há um número inteiro, assim a matriz poderia ser representada com o seguinte tipo na linguagem VHDL, onde bastaria então abrir o código VHDL onde a imagem seria colocada e colar os dados da matriz da imagem da seguinte forma :

```
type linha_img is array( 0 to COLUNAS ) of integer;
type matriz_img is array( 0 to LINHAS ) of linhas_img;
constant img_paddle0: matriz_img := (colar os dados aqui)
```

Porém ainda não foi falado sobre o que estes números da matriz significam, para reduzir informações no arquivo de texto, cada número representa um índice para um vetor de cores RGB que contém todas as cores que existiam na imagem, assim cada posição da matriz indica qual cor que é usada naquele pixel, desta forma poderíamos aproveitar a informação num código escrito na seguinte forma:

```
type cor_RGB is array(0 to 3) of integer range 0 to 255;
type vetor_cores is array(0 to N_Cores) of cor_RGB;
constant img_paddle0_cores: vetor_cores := (colar os dados aqui)
```

Desta forma fora possível adicionar todas as imagens ao projeto, porém no caso da imagem da mesa, fora utilizado uma imagem bem menor que o tamanho apresentado na tela, e então os pixels são repetidos através de uma função de ajuste que melhor identifica o pixel a ser representado na tela, essas funções e tipos comuns são declarados dentro do arquivo “work.vhd” que nada mais é que uma biblioteca definida para este projeto.

O código em python para preparo das imagens também estão em conjunto com este relatório.

Figura 4 – Layout da Mesa Virtual de Air Hockey Utilizada no Projeto

Fora testado imagens melhores e maiores no projeto, funcionaram bem, mas o tamanho da imagem e a quantidade de cores influi diretamente no tempo de compilação do projeto, como também na memória utilizada pelo mesmo, por exemplo uma imagem de 160x240 com 30 cores fez a compilação do projeto ultrapassar 40min.

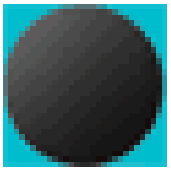
Figura 5 – Layout do Paddle0

Para não representar uma imagem quadrada, o bloco que trata de cada paddle só envia informações do paddle quando este identifica que a cor a ser enviada não é a cor de fundo da imagem do paddle e nenhuma cor parecida demais com a cor de fundo (mesmo diminuindo ao máximo o número de cores na imagem com um editor, este acaba deixando algumas cores inúteis na imagem que dariam muito trabalho para serem removidas dentro do código feito em Python, então foi preferido apenas marcá-las para serem ignoradas).

Figura 6 – Layout do Paddle1

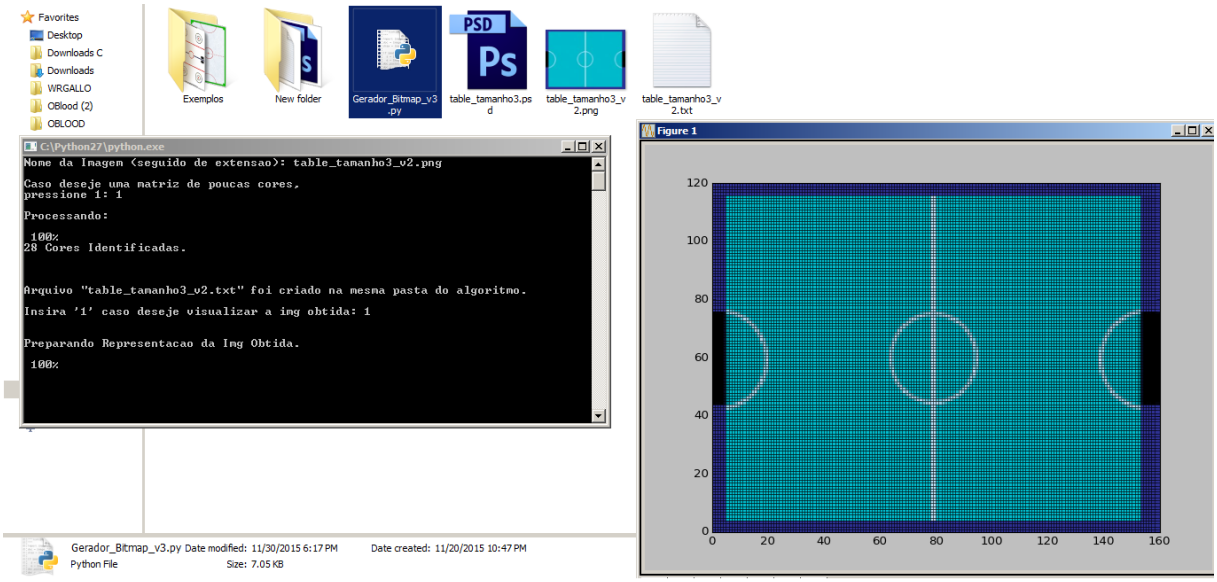
Cada paddle possui uma resolução de 61x61 pixels e com 30 cores cada e sua imagem é representada no tamanho original na tela, sem ampliação ou deformação.

Figura 7 – Layout do Puck



O puck é uma imagem de 30x30 pixels e com 30 cores, e também é representado em seu tamanho original na tela.

Figura 8 – Exemplo de Uso do Aplicativo em Python para Gerar Arquivos de Texto



O aplicativo fora feito de forma genérica, capaz de gerar bitmaps (imagens de apenas 2 cores, preto e branco), dados para imagens RGB_1D imagens onde todas as cores são representas por apenas um número entre 0 e 255, e finalmente a que foi utilizada neste projeto as imagens RGB_3D onde cada cor é representada em um número de 8 bits, e há a possibilidade de fazer uma enorme matriz onde em cada elemento é representado um vetor de 3 posições com as 3 cores ou então como utilizei neste projeto, apenas uma matriz de índices de cores e um vetor de cores.

Estrutura do Projeto

O projeto se resume basicamente num arquivo de esquemático conectando os blocos, os blocos existentes são:

- vga_sync;
    - Trata das Informações de sincronização e é o responsável por informar os pixeis X e Y que estão sendo representados na tela;
- table;
    - Trata do layout da mesa virtual de air hockey, e informa a imagem ao video_mux com prioridade mínima (4), para qualquer outra imagem possa sobrepor ela;
- paddle0;

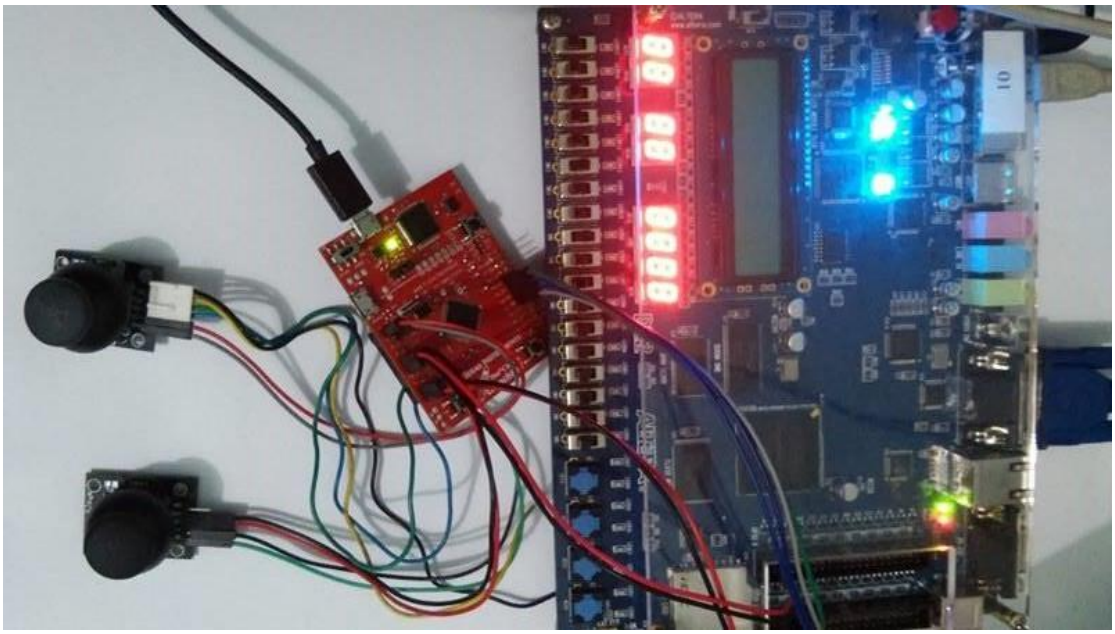
    - Trata do layout do paddle0, procedimento semelhante ao do bloco table, porém esta imagem tem maior prioridade que o bloco table, neste caso a prioridade é 1, inferior apenas a textos; neste bloco também trata-se da leitura e lógica para o paddle referente aos pinos informados pelo microcontrolador para este paddle;
- paddle1;
    - Trata do layout do paddle1 assim como o paddle0, porém tem prioridade inferior (2), porém dada regra de projeto um paddle é incapaz de atravessar para o campo inimigo, então um jamais irá se sobrepor sobre outro.
- puck;
    - Trata do layout do disco (puck) do jogo, tem prioridade de video = 3, ou seja, em caso de colisão com um paddle, o paddle iria passar por cima do disco; neste bloco toda a lógica de gols e direção em caso do disco ser rebatido em cada parede ou paddle é feita;
- debouncer;
    - Responsável por receber o push-button de reset e realizar o debouncer desse sinal e informar um tick correspondente ao sinal de entrada, porém quando o sinal for estabilizado;
- video_mux;
    - Recebe vários sinais de vídeo, sendo 2 para cada, um informa que este vídeo deve ser tratado agora, e outro informa a cor deste vídeo, assim cada vídeo é tratado conforme a prioridade e então a cor final é enviada para o conversor digital para analógico que irá enviar as cores para o conector VGA;
- refresh_pad;
    - Gera dois tiks, cada um em frequências diferentes, um serve para atualizar a posição do disco e outro para atualizar a posição dos paddles;
- score_gen;
    - Responsável pelos textos na tela, principalmente o score;

Resultados

Os analógicos funcionaram perfeitamente para modificar a posição dos paddles na tela, os paddles respeitam seus limites adequadamente, e o disco rebate nos paddles com precisão (ajustada para melhores resultados) e reconhece adequadamente onde deve ser rebatido (extremo da mesa ou entrar no gol), as imagens são representadas adequadamente ignorando a cor de fundo de cada uma delas.

Podemos observar aqui os analógicos conectados no microcontrolador e no altera DE2 Cyclone II.

Figura 9 – Joysticks conectados ao microcontrolador e ao altera DE2



Aqui podemos observar o jogo em perfeito funcionamento com um placar de 1x0 para o jogador do paddle azul.

Figura 10 – Jogo após um ponto do jogador do paddle azul.