Projeto Aplicativo
Racing Game

Abdullah Zaiter, 15/0089392
Ian Moura Alexandre, 15/0129661
Lukas Lorenz, 16/0135109

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116351 - Circuitos Digitais - Turma A

{abdullah.zaiter,ianzeba,lukaslorenzandrade}@gmail.com

Abstract. *Development of a racing game using the concepts learned on Digital Circuits.*

Resumo. *Implementação de um jogo de corrida usando os conceitos vistos na diciplina de Circuitos Digitais, usando-se de portas lógicas, software de simulação Quartus e circuitos de memória e a linguagem de circuito verilog.*

Leia este relatório ao som dessa música: ***(Youtube)***. Realmente vale a pena.

1. Introdução

Desde a década de 1970, os video games vem encantando gerações. O desenvolvimento dos jogos eletrônicos e o crescimento de seu mercado - principalmente no Brasil, mesmo em meio a crise link - transcende faixa etária quando se trata de um dos maiores tipos de entreterimento, atualmente.

Um dos principais tipos de jogos eletrônicos são os *Racing Games*. Pode-se citar diversos exemplos, como Need for Speed***(Wikipedia)***, Top Gear ***(Wikipedia)*** e Enduro ***(Wikipedia)***, o qual foi um dos últimos jogos lançados em 1983 pela empresa Activision para o console Atari 2600 antes de anunciar falência. Este jogo é implementado neste projeto usando apenas lógicas combinacionais e sequênciais (registradores, flip-flops e portas lógicas).

O Enduro é um jogo na qual o jogador controla um carrinho - para os lados acelera e freia - em uma pista e sua função é percorrer a maior distância possível sem que o automóvel se choque contra os obstáculos. A medida que o tempo passa, ou seja a distância percorrida aumenta, o carro fica mais rápido e o cenário muda de forma a aumentar o nivel de dificuldade do jogo.

Contudo, nesse projeto, o jogo contará apenas com controles de deslocamento lateral e velocidades, além de um botão de pause e *restart*. Os clássicos *joysticks* foram substituidos por dois potenciômetros que regulam a velocidade e a posição do carrinho dentro da pista que, ao invés do monitor da televisão, será representada por uma matriz de LED 8x8. Toda a lógica do sistema será implementado na FPGA *(Field Programmable Gate Array)*.

Para a execução deste *Racing Game*, usou-se do embasamento teórico obtidos na disciplina Circuitos Digitais para a arquiquetura e implementação daquele. Além

dos conceitos vistos nos laboratórios de CD, como circuitos sequenciais** e combinacionais**, contadores e registradores, divisores de frequências, codificadores e montagens de circuitos com portas lógicas, tanto na *protoboard* quanto no software Quartus II, trabalhou-se com novos conceitos e materiais como a Matriz de LEDs 8x8, Amplificadores Operacionais, Conversores AD.

A matriz de LED's 8x8 é um quadrado com 8 linhas e 8 colunas, onde os LEDs de mesma coluna são acionados quando se tem 1 como entrada numa linha e coluna, que são conectadas na FPGA. Já os potenciômetros, são conectados aos aplificadores e vários resistores, que formam um conversor analógico digital o qual converterá tensão de entrada em um código digital.

** Circuitos sequenciais são aqueles que usam a lógica de latches, Flip-Flops e Registradores, por exemplo. Já os combinacionais, são os que usam de portas lógicas como AND, OR e NOT, por exemplo.

1.1. Objetivos

O projeto realizado tem como objetivo aplicar os conhecimentos ensinados na disciplina de Circuitos Digitais pata a implementação de um Racing Game.

1.2. Materiais

Neste experimento foram utilizados os seguintes materiais e equipamentos:

- Kit de equipamentos DE2;
- *protoboard*;
- Fios (*jumpers*);
- 24 Resistores de 1k 1/4W;
- 2 potenciômetros rotativos de 10k;
- 1 Matriz de LEDs 8x8;
- Software Quartus-II v.13.0;
- 4 CIs LM324 (Amp-Ops);

2. Montagem

A implementação deste *Racing Game* foi separado em - partes: Montagem dos potenciômetros para controlar a posição e velocidade do carrinho com os conversores AD, a apresentação do carrinho e da pista na matriz 8x8 e a impressão da velocidade percorrida. Todas as implementações e montagens estão dispostos no repositório do github, disponível neste link: ***Link do repositório***

2.1. Montagem dos potenciômetros

Para o usuário poder controlar a posição e a velocidade do carrinho, usou-se os dois potenciômetros rotativos de 10kΩ, ligados no circuito mostrado nas figuras 1 e 2.

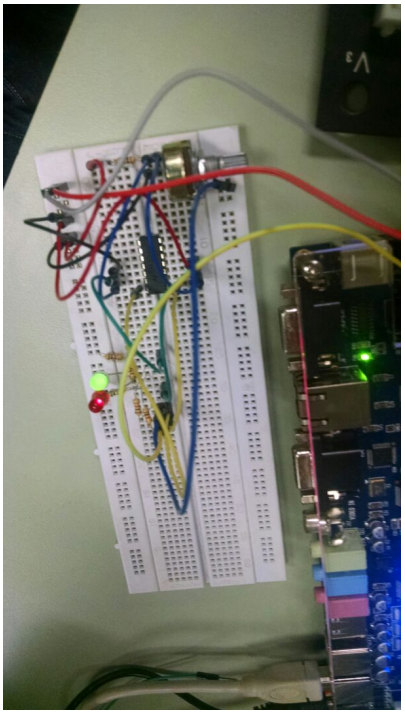

Figura 1. Foto do controlador da posição do carrinho

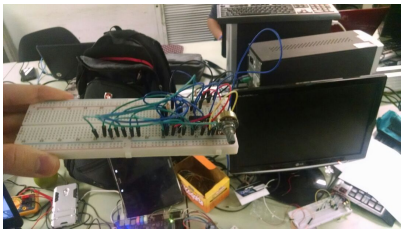

Figura 2. Foto do controlador da velocidade do carrinho

Para a montagem do controlador de posição, utilizou-se além do potenciômetro, um circuito com 4 amp-ops e 6 resistores de 1kΩ.
Com a escada de resistores pôde-se definir as condições do circuito. Com três estados possíveis (Virar para a direita, virar para a esquerda e continuar reto), as condições de posição do carrinho pôde ser determinada com base na Tabela 1.

Tabela 1. Tabela verdade - potenciômetro de posição

A	B	Esq	Dir
0	0	0	1
1	1	1	0
1	0	0	0

De acordo com a tabela, foi-se definido que o carrinho move uma posição à direita caso no caso Esq/Dir = 01, move-se à esquerda caso as saídas Esq/Dir = 10 e nos

quando as duas saídas são iguais (00), o carrinho continua na mesma posição.

Fazendo-se os ciclos de Kernough, viu-se que $Dir = \bar{A}$ e $Esq = B$. Sendo um circuito simples, ao invés de ser feito um bloco de conversor, apenas foi-se substituido os valores nos circuitos.

Já para o controlador de velocidade, fez-se o conversor AD, o qual separa a tensão variada pelo potenciômetro conforme o fundo de escala do conversor, ou seja, a cada nível, ou resistor, a tensão diminui proporcionalmente e esta é selecionada no terminal dos *jumpers* verdes. Na saída destes colocou-se um codificador para que a cada nível de tensão, obtendo-se um número em binário proporcional a esse, conforme mostrado na Figura (3).

Já o dado fornecido pelo potenciômetro controlador da velocidade da pista deve-se passar por um codificador de 7 entradas e 3 saídas, Figura (**??**) o qual foi feito na linguagem Verilog, seguindo o padrão apresentado pela tabela verdade mostrada na figura 3:

Entrada analógica	Saídas dos comparadores							Saídas digitais		
V_A	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C	B	A
0 - 0.625V	1	1	1	1	1	1	1	0	0	0
0.625 - 1.25V	0	1	1	1	1	1	1	0	0	1
1.25 - 1.875V	0	0	1	1	1	1	1	0	1	0
1.875 - 2.5V	0	0	0	1	1	1	1	0	1	1
2.5 - 3.125V	0	0	0	0	1	1	1	1	0	0
3.125 - 3.75V	0	0	0	0	0	1	1	1	0	1
3.75 - 4.375V	0	0	0	0	0	0	1	1	1	0
> 4.375V	0	0	0	0	0	0	0	1	1	1

Figura 3. Tabela verdade - potenciômetro da velocidade

À partir da tabela mostrada acima, implementou-se um codificador em Verilog:

```
1  module CodfDePrio(
2     output wire [2:0] one,
3     input wire [6:0] in
4  );
5
6  assign one = 3'b111 - (in[6]+in[5]+in[4]+in[3]+in[2]+in[1]+in[0]) ;
7
8  endmodule
```

Figura 4. Código em Verilog do codificador de prioridades

O código leva em conta que a saída é igual ao resultado em binário do tamanho do vetor (no caso 7) subtraído do número de 1's encontrados no vetor de entrada.

Assim, consegue-se converter o valor analógico dado pelo conversor AD para um valor em binário que serão usados para controlar a posição do carrinho e a velocidade.

2.2. Jogo mostrado na Matriz 8x8

Para a visualização do jogo, foi necessário montar a estrutura da pista, a lógica do carrinho, unir ambos para o funcionamento na matriz de LEDs, juntando à parte implementada na Figura(2.1) e, por último, estabelecer o critério de colisão.

Para a elaboração da estrutura da pista, implementou-se um código em Verilog com 64 linhas que determinava cada estado da pista. Cada linha dessa representava os 8 bits de

uma linha da Matriz, onde os zeros (0's) determinavam o LED que estava apagado (não há obstáculo) e os uns (1's) determivam o LED ligado (presença de obstáculo). Dessa forma, com uma pista pré carregada, poderia-se jogar com o carrinho nela. Ao final dos 64 estados, a pista retornaria ao seu primeiro estado (pista[0]).

Neste bloco, ou código, tem-se ligado uma entrada RESET e um CLOCK. O reset indica que quando esta entrada for 1, a nossa pista volta ao valor inicial.Já o CLOCK é definido pelo seletor de velocidades do potênciometro e pela entrada da FPGA de 50MHz, que, com o valor analógico codificado - Figura (4), fornece a onda do clock por meio do divisor de frequência de 50 kHz.

Como mostrado na Figura (6), este circuito divide a frequência do CLOCK em diferentes valores (dependendo do valor do codificador de prioridades). O divisor usa um DeMux, vários divisores de frequência que seguem a lógica mostrada na Figura (5), só que com alguns valores, e um MUX. Este CLOCK será compartilhado entre a pista e o movimento do carrinho. Entretanto, este CLOCK só é fornecido para os circuitos caso a entrada de PAUSE seja negativa. A chave de PAUSE tem por intuito parar o jogo quando acionada, ou seja, permanecesse no último estado encontrado até que a chave fosse desligada (nível zero).

```
module fdiv_1 (clkin,clkout);
input clkin;
output reg clkout;

integer cont;

initial
   cont=0;
  |
always @(posedge clkin)
  begin
   if (cont==25000000)
      begin
         cont<=0;
         clkout<=~clkout;
      end
      else
      begin
         cont<=cont+1;
         clkout<=clkout;
      end
   end
endmodule
```

Figura 5. Divisor de frequencia de 2

Tendo isso pronto, elaborou-se a lógica do movimento do carrinho. Analisando como o carrinho translada pela pista, a cada pulso de CLOCK o bit 1 é deslocado para direita ou para esquerda (dependendo do valor fornecido pelo potenciômetro de posição), onde a lógica se assemelha a um contador em anel reversível.

Entretanto deve-se considerar que nas extremidades o carrinho permaneça na mesma posição, caso o jogador deseje mover o carrinho para fora da pista (matriz) do jogo. O circuito que define tal movimento é mostrada na Figura (7), onde este é composto por vários MUX, Flip-Flops tipo D e portas lógicas.

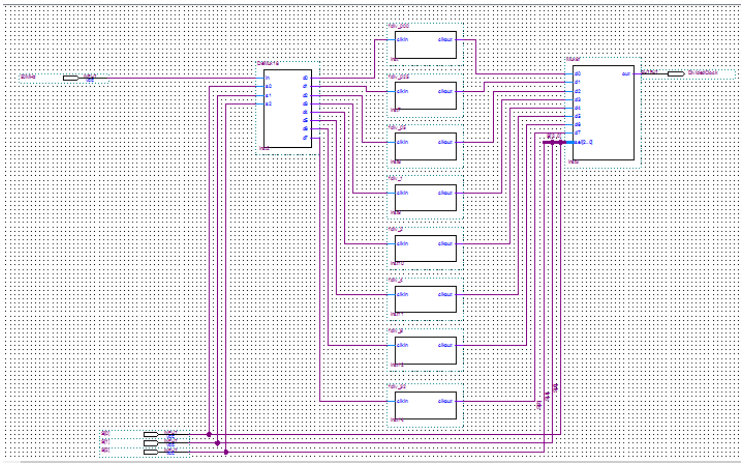

Figura 6. Divisor de frequencia de 50

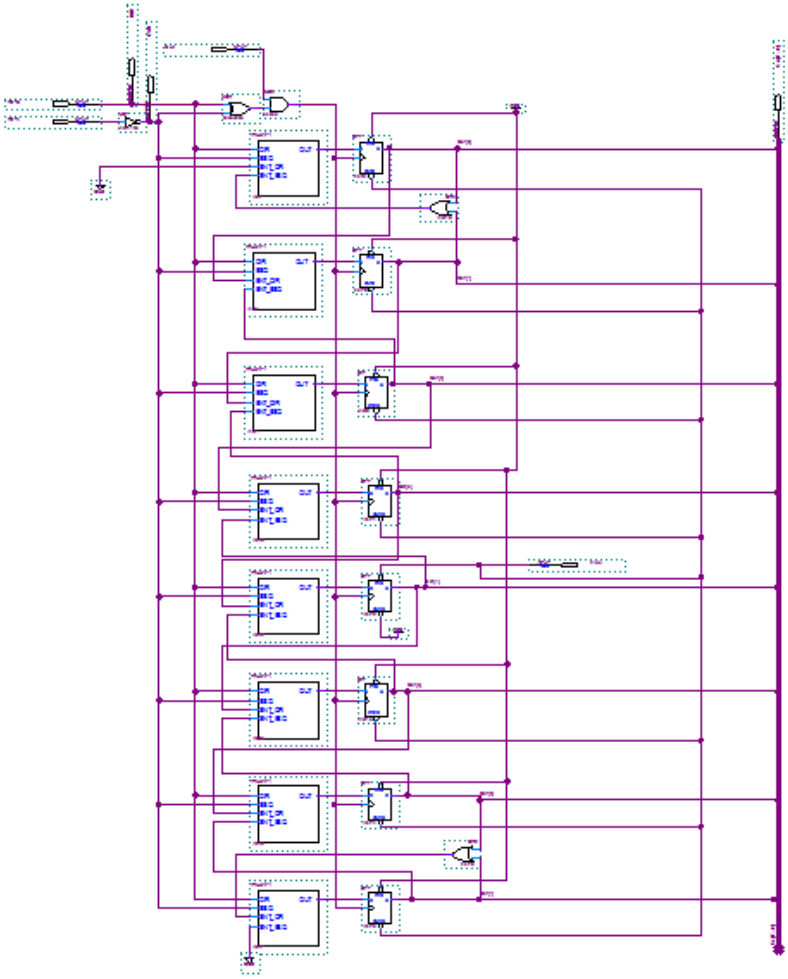

Figura 7. Circuito que define o deslocamento lateral do carrinho

Como pode-se ver, a entrada de cada Flip-Flop D, que define cada bit da pista, é dada por um MUX que tem como seleções as entradas Dir e Esq - referentes a seção

(montagem dos potenciômetros) no deslocamento do carrinho. Se o movimento for para a direita, o próximo Flip-Flop recebe o valor do carrinho, enquanto se for para esquerda, o anterior é que recebe tal valor.

Já nas extremidade, o Flip-Flop recebe o valor que estava em sua saída (o estado seguinte) ou o valor anterior (estado atual), caso o carrinho mude de posição na matriz ou transgrida seus limites. Caso contrário, ele recebe zero.

Tendo a pista e o movimento do carrinho implementado, precisa-se unir esses blocos na matriz de LEDs e definir a condição de colisão - vidas a serem perdidas. O bloco de controle da matriz é definido assim:

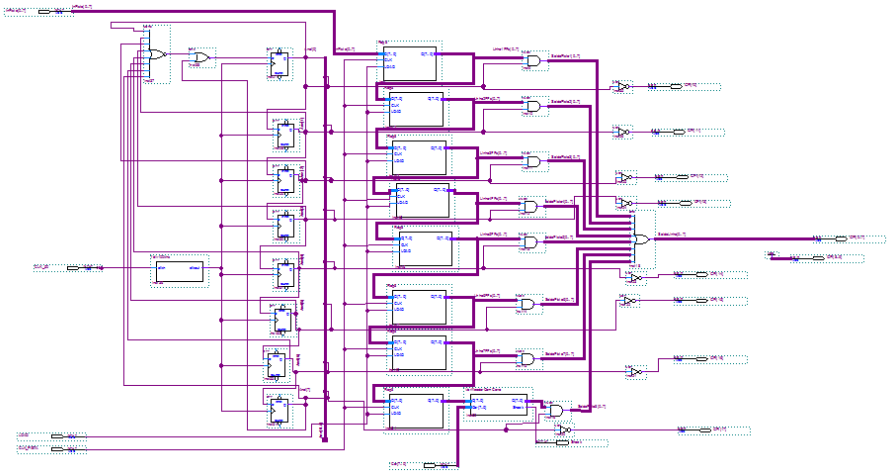

Figura 8. Circuito que define o controle da matriz de LEDS

```
module Reg8 (
    input [7:0] D,
    output reg [7:0] Q,
    input CLK,
    input LOAD
);
initial
    Q = 8'b 10000001;
always @(posedge CLK or posedge LOAD)
    if (LOAD)
        Q=8'b 10000001;
    else
        Q=D;

endmodule
```

Figura 9. Registrador de * bits

O registrador mostrado na Figura (9) indica o estado atual de cada linha da pista.

Caso o LOAD seja positivo, ele retorna o valor inicial, se não vai retornando o anterior. Sabendo-se que a matriz precisa funcionar com um clock de frequência extremamente alta, de forma com que o olho humano seja incapaz de perceber tal variação, pode-se contadores, entre outras formas, para que a matriz possa ser deslocada de forma uniforme.

Neste caso usou-se um contador em anel com frequência elevada e 8 registradores de 8 bits, de forma que cada registrador armazena uma linha da matriz. Por conseguinte, a cada transição de borda de subida do CLOCK, a informação da linha superior de registradores é passada para a inferior, e a de cima recebe o conteúdo de uma nova. Assim, o bit 1 estaria sendo deslocado a todo momento de forma imperceptível.

No caso em que foi implementado, tem-se que o obstáculo representa um valor 1 na matriz da pista, assim como o carrinho representa o valor 1 no registrador de 8 bits de posição. Dessa forma, elaborou-se a condição de colisão quando o bit 1 dos dois registradores se encontram provocando a perda de uma a vida do jogador, como será explicado adiante.

2.3. Contador de velocidades e vidas

Com o jogo já funcionando, implementou-se dois contadores. Esses tem por finalidade imprimir a distancia percorrida e a quantidade de vidas restantes.

Para o primeiro, precisava-se de um contador que, a cada pulso do CLOCK da pista, incrementasse uma unidade, imprimindo o número no display de 7 segmentos. No nosso caso, dispunha-se de um contador em hexadecimal, como no código em Verilog mostrado na figura (9). Dessa maneira, vai se incrementando no contador uma unidade a cada pulso de CLOCK, enquanto não houver um reset na pista. O *reset* é caso da chave referente a essa funcionalidade seja ativada pelo usuário ou em caso de colisão, fazendo o carro e a pista voltar para a posição inicial.

Na saída do contador de distância possuem 2 *decoder* de 3 entradas e 7 saídas (decodificador de 7 segmentos fornecido pelos professores no moodle), como visto na Figura 14.

```
module Contador(count, clk, rst_n);
  output reg [7:0] count;
  input clk;
  input rst_n;

  initial
    count = 0;
always @(posedge clk or posedge rst_n)
    if (rst_n)
      count = 0;
    else
      count = count + 1;

endmodule
```

Figura 10. Circuito correspondente ao contador da distancia

Na seção 2.2, foi introduzido a lógica por trás da contagem de vidas restantes. A cada vez que há uma colisão, o contador de vidas é decrescido uma unidade. Dessa maneira, pode-se ver que o contador de vidas é um contador decrescente que conta desde 7 até 0. Depois que o jogador perde a última vida restante dele, o jogo para,se ndo necessário que o jogador reinicie o jogo, sendo as vidas voltando para 7 automaticamente.

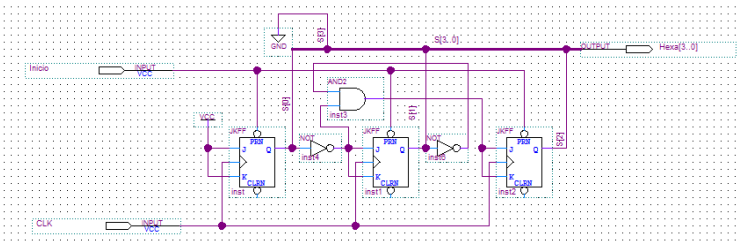

Figura 11. Circuito correspondente ao contador de vidas

3. Funcionamento

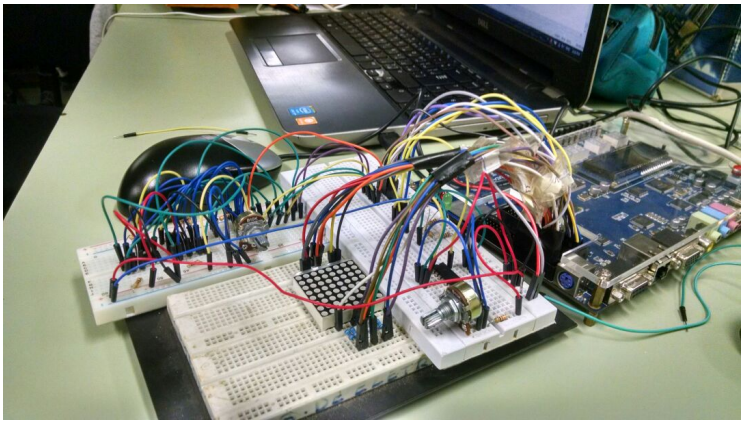

Figura 12. Foto do circuito completo (vista frontal)

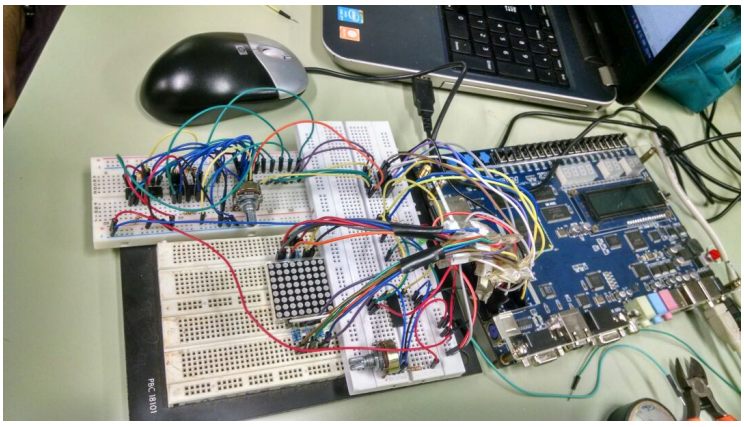

Figura 13. Foto do circuito completo (vista superior)

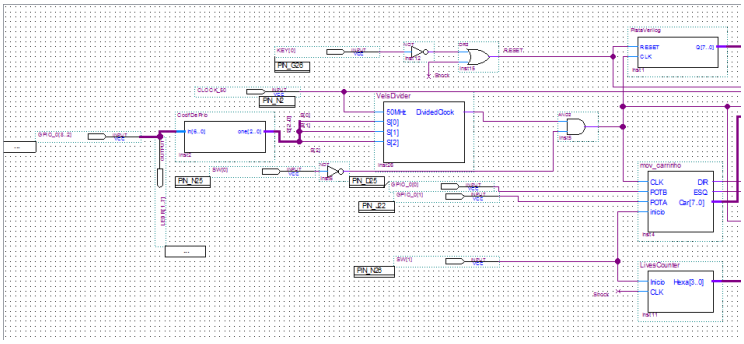

Figura 14. Circuito implementado no quartus para o jogo (parte 1)

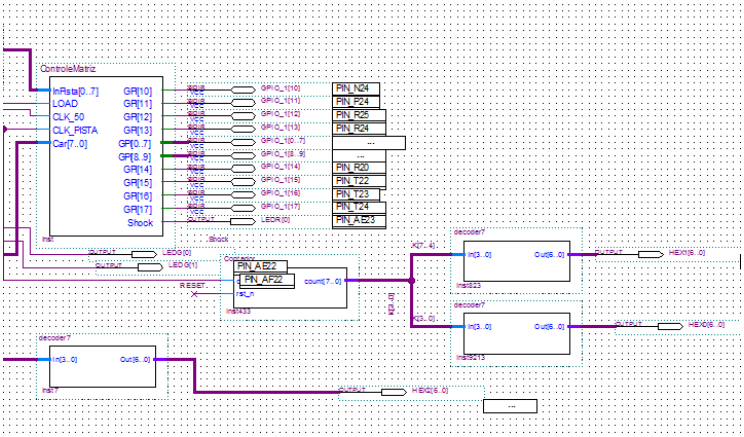

Figura 15. Circuito implementado no quartus para o jogo (parte 2)

Video do funcionamento do jogo completo
Video do funcionamento do visor 8x8 (pista e carrinho)

4. Conclusão

Considerando os resultados obtidos na seção 3, apesar dos pequenos erros perceptíveis, como o carrinho mover mais de uma casa para o lado ou o potenciômetro extremamente sensível aos movimentos, o projeto obteve os resultados esperados.

As funções implementadas e a visualização do jogo se mostraram corretas. Vale lembrar que quando problemas com *hardware*, ele sempre está sujeito à interferência do ambiente e limitado ao fato de que os materiais não são as condições ideais implantadas na teoria.

Dessa maneira, pôde-se utilizar dos conceitos de Circuitos Digitais, unida com as ferramentas e tecnologias dispostas, para se implantar um jogo bem divertido, mas que possui em seu desenvolvimento pequenas partes capazes de explicar o funcionamento de circuitos mais complexos e desenvolvidos, sendo assim um pequeno passo para o estudo de máquinas como os computadores.

Referências

Roteiro do jogo Racing Games.
Asic : www.asic-world.com

Stack Overflow : https://stackoverflow.com

5. Agradecimentos

Nós, Abdullah Zaiter, Ian Moura e Lukas Lorenz gostaríamos de agradecer profundamente ao profº Lamar e ao monitor Alexandre por fazerem com que esse projeto se tornasse possível.
Agradecemos também à profª Carla Koike e aos outros monitores da matéria de CD, Divino e Renan, pela base teórica e apoio nos laboratórios, fundamental no desenvolvimento deste experimento.