Universidade Federal de Viçosa

CAMPUS FLORESTAL

TRABALHO PRATICO

CONVERSOR DE CODIGO MORSE

Disciplina: CCF 251F - INTRODUÇÃO AOS SISTEMAS LÓGICOS DIGITAIS

Roniel Nunes Baborsa	3464
Pablo Miranda Batista	3482
Lucas Antonio de Souza Lima	3481
Cleidimar Lacerda dos Passos	3473

FLORESTAL – MG
OUTUBRO 2018

1. Sumário

2. Introdução

O trabalho pratico baseou-se na implementação de um codificador Morse, capaz de converter números de 0 a 9 em sinais Morse. Sendo desenvolvido usando a linguagem de descrição de hardware Verilog e GTKWave para simulações dos resultados.

Inicialmente foi utilizada tabela da verdade e mapa de Karnaugh para encontrar as funções booleanas, depois foi implementado as funções no programa Logisim para vermos se as logicas estão corretas, depois da confirmação das funções iniciamos a implementação no verilog.

3. Softwares usados

Foi utilizado no desenvolvimento do TP os softwares: Logisim, Icarus Verilog, GTKWave, IDE Visual Studio.

4. Etapas do desenvolvimento:

I. Tabela da Verdade

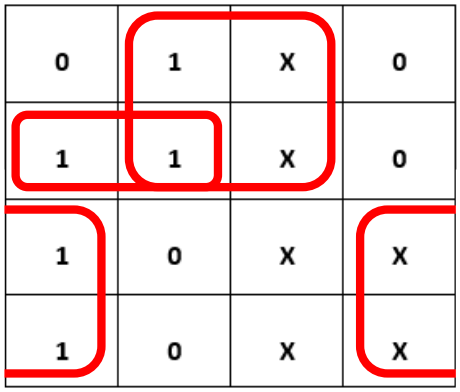
Entradas: A, B, C e D.

Saídas: X1, X2, X3 e X4.

A	B	C	D	X1	X2	X3	X4	X5
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	1	1	0	0	0
0	0	1	1	1	1	1	0	0
0	1	0	0	1	1	1	1	0
0	1	0	1	1	1	1	1	1
0	1	1	0	0	1	1	1	1
0	1	1	1	0	0	1	1	1
1	0	0	0	0	0	0	1	1
1	0	0	1	0	0	0	0	1
1	0	1	0	X	X	X	X	X
1	0	1	1	X	X	X	X	X
1	1	0	0	X	X	X	X	X
1	1	0	1	X	X	X	X	X
1	1	1	0	X	X	X	X	X
1	1	1	1	X	X	X	X	X

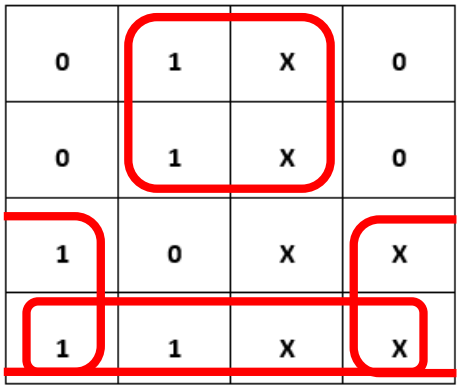
II. *Simplificação das equações booleanas utilizando mapas de Karnaugh*

Mapa de Karnaugh para saída X1:



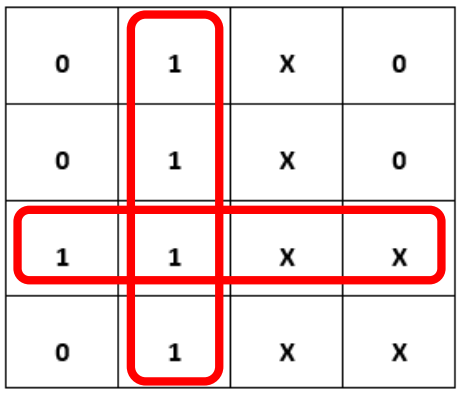
Equação X1: (BC') + (B'C) + (A'C'D)

Mapa de Karnaugh para saída X2:



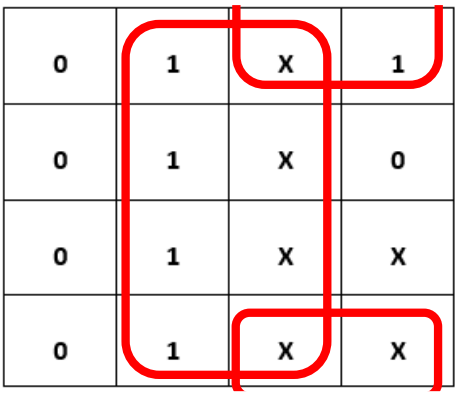
Equação X2: (BC') + (CD') + (B'^C)

Mapa de Karnaugh para saída X3:



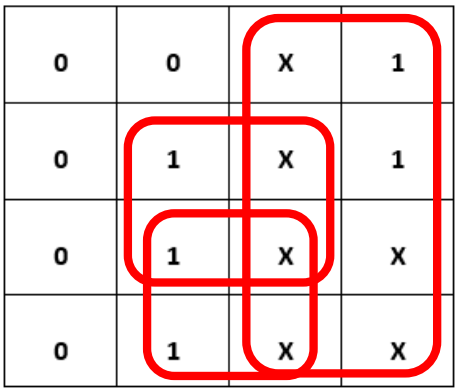
Equação X3: (A'B) + (CD)

Mapa de Karnaugh para saída X4:



Equação X4: B + (AD')

Mapa de Karnaugh para saída X5:



Equação X5: A + (BD) + (BC)

III. Formas canônicas

Mintermo(s):

X1 (A,B,C,D) = Σm (1,2,3,4,5) + Σd (10,11,12,13,14,15)

X2 (A,B,C,D) = Σm (2,3,4,5,6) + Σd (10,11,12,13,14,15)

X3 (A,B,C,D) = Σm (3,4,5,6,7) + Σd (10,11,12,13,14,15)

X4 (A,B,C,D) = Σm (4,5,6,7,8) + Σd (10,11,12,13,14,15)

X5 (A,B,C,D) = Σm (5,6,7,8,9) + Σd (10,11,12,13,14,15)

Maxtermo (s):

X1 (A,B,C,D) = πM (0,6,7,8,9) + πD (10,11,12,13,14,15)

X2 (A,B,C,D) = π M (0,1,7,8,9) + πD (10,11,12,13,14,15)

X3 (A,B,C,D) = π M (0,1,2,8,9) + πD (10,11,12,13,14,15)

X4 (A,B,C,D) = π M (0,1,2,3,9) + πD (10,11,12,13,14,15)

X5 (A,B,C,D) = π M (0,1,2,3,4) + πD (10,11,12,13,14,15)

IV. Mintermo e Maxtermo das saídas

Mintermo(s):

X1 = (A'B'C'D) + (A'B'CD') + (A'B'CD) + (A'BC'D') + (A'BC'D)

X2 = (A'B'CD') + (A'B'CD) + (A'BC'D') + (A'BC'D) + (A'BCD')

X3= (A'B'CD) + (A'BC'D') + (A'BC'D) + (A'BCD') + (A'BCD)

X4= (A'BC'D') + (A'BC'D) + (A'BCD') + (A'BCD) + (AB'C'D')

X5= (A'BC'D) + (A'BCD') + (A'BCD) + (AB'C'D')+(AB'C'D)

Maxtermo(s):

X1 = (A'+B'+C'+D') (A'+B+C+D') (A'+B+C+D) (A+B'+C'+D') (A+B'+C'+D)

X2 = (A'+B'+C'+D') (A'+B'+C'+D) (A'+B+C+D) (A+B'+C'+D') (A+B'+C'+D)

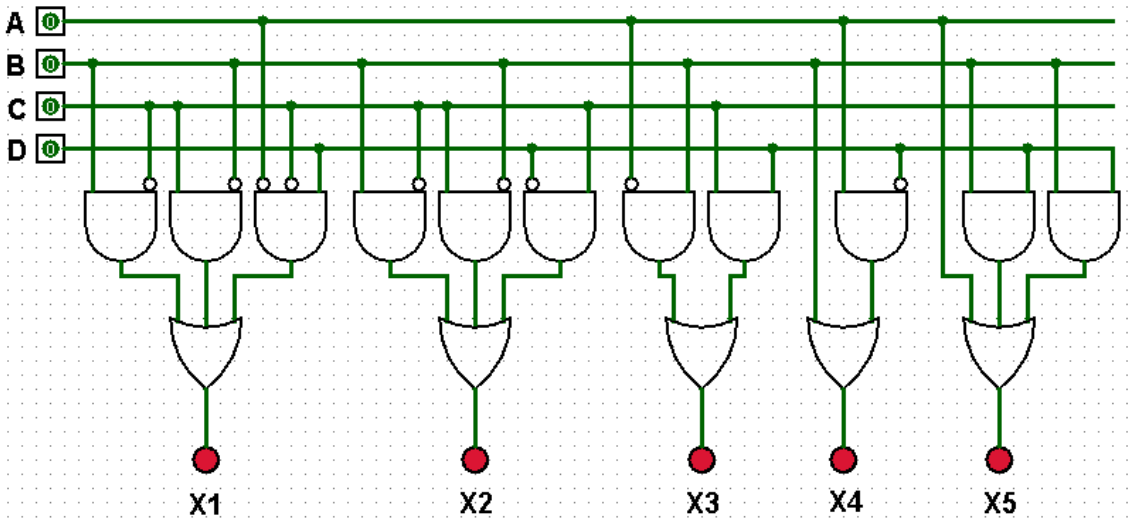
X3 = (A'+B'+C'+D') (A'+B'+C'+D) (A'+B'+C+D') (A+B'+C'+D') (A+B'+C'+D)

X4 = (A'+B'+C'+D') (A'+B'+C'+D) (A'+B'+C+D') (A'+B'+C+D) (A+B'+C'+D)

X5 = (A'+B'+C'+D') (A'+B'+C'+D) (A'+B'+C+D') (A'+B'+C+D) (A'+B'+C+D)

V. *Circuito simplificado com portas lógicas no software Logisim*

Utilizamos o programa Logisim para demonstrar o sistema de portas logicas tirado da tabela da verdade utilizando mapa de karnaugh. Como podemos ver foi utilizado bolhas de inversão nas entradas do ANDs para representar as negações. Nas saídas os leds vão ficar verde quando a saída for 1 e vermelho quando for 0.



5. Implementação em Verilog

Na implementação em verilog utilizamos as entradas como sendo wire (fios), pois as entradas vão ser recebidas constantemente.

As saídas foram definidas como reg (registrador), pois preciso que a saída seja mantida armazenada até que o simulador a imprima na tela ou no gráfico.

O bloco Always Reset só será executado quando a entrada Reset for 1, quando isso acontecer o programa deixara todas as saídas com 0.

O bloco Always Ready só será executado quando a entrada Ready for 1 e Reset for 0, quando isso acontecer ele executara as logicas das funções booleanas.

```
1  module codificador (A, B, C, D, reset, ready, X1, X2, X3, X4, X5);
2
3      input wire A, B, C, D, reset, ready;
4      output reg X1, X2, X3, X4, X5;
5
6      always @ (reset) begin
7          X1 = 0;
8          X2 = 0;
9          X3 = 0;
10         X4 = 0;
11         X5 = 0;
12     end
13
14     always @ (ready) begin
15         if (~reset)begin
16             X1 <= (B & ~C) || (~B & C) || (~A & ~C & D);
17             X2 <= (B & ~C) || (C & ~D) || (~B & C);
18             X3 <= (~A & B) || (C & D);
19             X4 <= B || (A & ~D);
20             X5 <= A || (B & D) || (B & C);
21         end
22     end
23 endmodule
```

A seguir criamos um arquivo que e responsável por fazer a simulação da logica implementada no arquivo anterior.

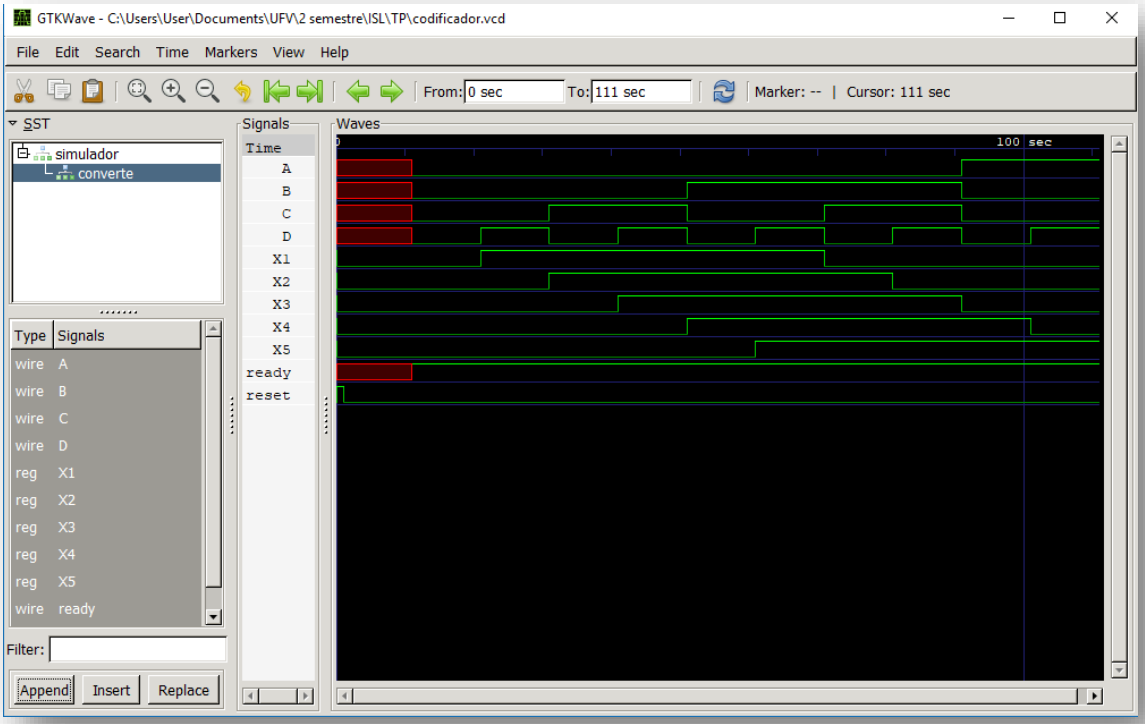
Inicialmente chama-se o arquivo da logica e atribui onde será armazenado cada valor recebido pela logica.

Depois definimos como será feito a impressão na pront de comando utilizando os comandos display e monitor.

E para finalizar definimos quais valores seriam atribuídos as entradas em cada instante de tempo.

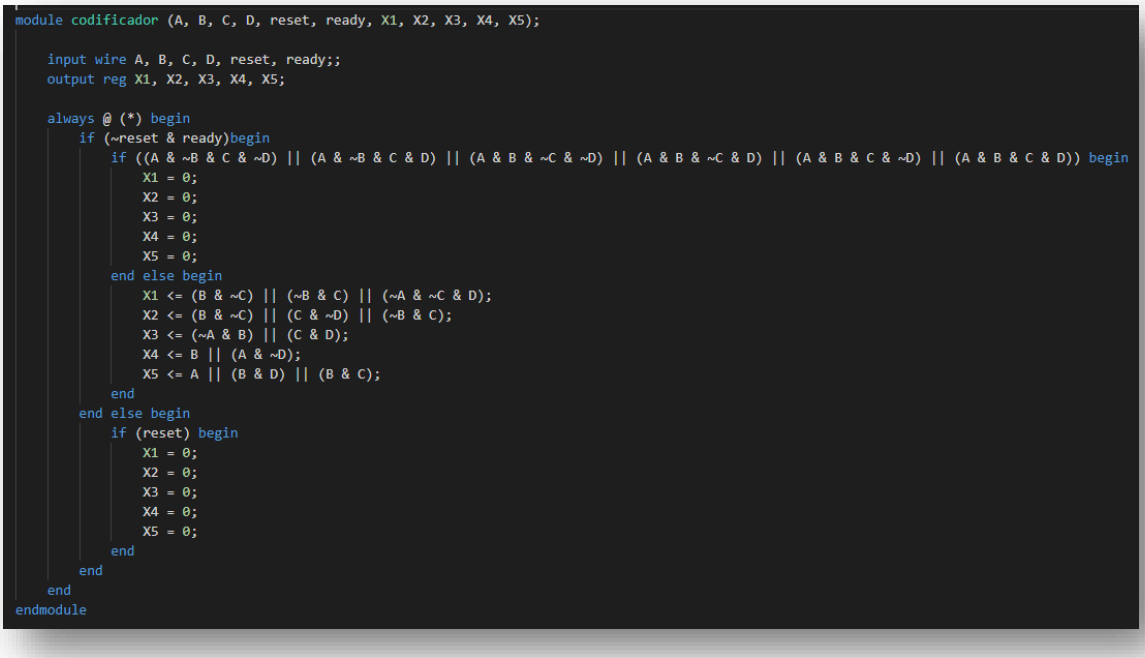
```
1
2   `include "codificador.v"
3
4   module simulador ();
5       reg A, B, C, D, reset, ready;
6       wire X1, X2, X3, X4, X5;
7
8       codificador converte (.A(A), .B(B), .C(C), .D(D), .reset(reset), .ready(ready), .X1(X1), .X2(X2), .X3(X3), .X4(X4), .X5(X5));
9       initial begin
10        $dumpfile ("codificador.vcd");
11        $dumpvars (0, simulador);
12        $display ("\nCodificador de Binario para codigo morse\n\nENTRADA   SAIDA");
13        $monitor ("%b%b%b%b    = %b%b%b%b%b", A, B, C, D, X1, X2, X3, X4, X5);
14      end
15
16      initial begin
17              reset = 1;
18        #1; reset = 0;
19        #1; ready = 0; A = 0; B = 0; C = 0; D = 0; ready = 1;
20        #1; ready = 0; A = 0; B = 0; C = 0; D = 1; ready = 1;
21        #1; ready = 0; A = 0; B = 0; C = 1; D = 0; ready = 1;
22        #1; ready = 0; A = 0; B = 0; C = 1; D = 1; ready = 1;
23        #1; ready = 0; A = 0; B = 1; C = 0; D = 0; ready = 1;
24        #1; ready = 0; A = 0; B = 1; C = 0; D = 1; ready = 1;
25        #1; ready = 0; A = 0; B = 1; C = 1; D = 0; ready = 1;
26        #1; ready = 0; A = 0; B = 1; C = 1; D = 1; ready = 1;
27        #1; ready = 0; A = 1; B = 0; C = 0; D = 0; ready = 1;
28        #1; ready = 0; A = 1; B = 0; C = 0; D = 1; ready = 1;
29        #1; ready = 0; A = 1; B = 0; C = 0; D = 1; ready = 1;
30        $finish;
31      end
32  endmodule
```

A seguir estará como ficou o gráfico de estado do projeto, utilizando GTKWave.



6. Implementação na FPGA

Para desenvolver essa parte do projeto tivemos que fazer alterações no modulo de codificação, devido a conflitos que a IDE Quartus assinalou durante o debug.

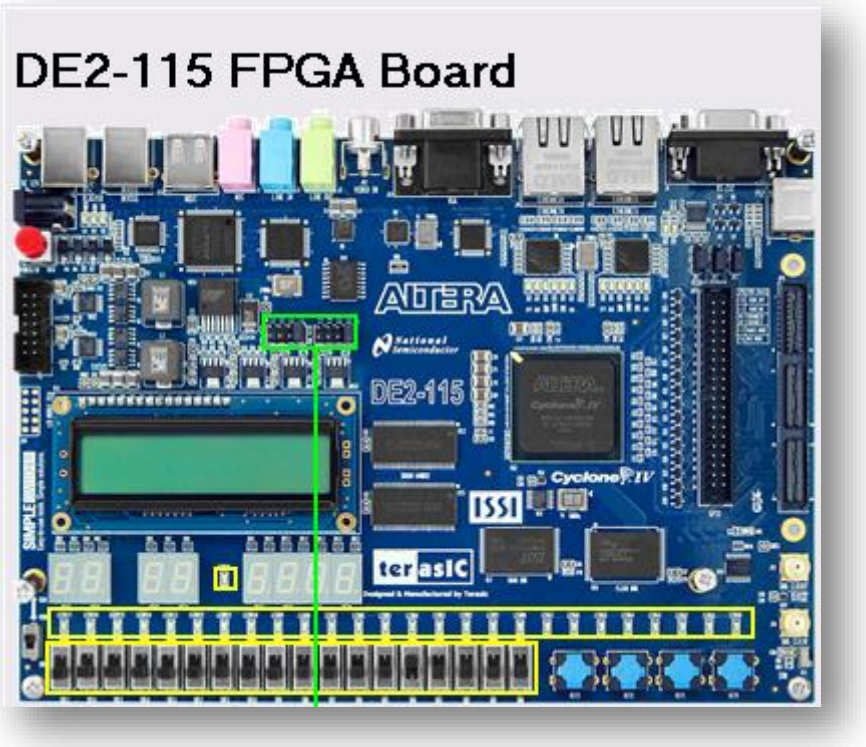


```
module codificador (A, B, C, D, reset, ready, X1, X2, X3, X4, X5);

    input wire A, B, C, D, reset, ready;;
    output reg X1, X2, X3, X4, X5;

    always @ (*) begin
        if (~reset & ready)begin
            if ((A & ~B & C & ~D) || (A & ~B & C & D) || (A & B & ~C & ~D) || (A & B & ~C & D) || (A & B & C & ~D) || (A & B & C & D)) begin
                X1 = 0;
                X2 = 0;
                X3 = 0;
                X4 = 0;
                X5 = 0;
            end else begin
                X1 <= (B & ~C) || (~B & C) || (~A & ~C & D);
                X2 <= (B & ~C) || (C & ~D) || (~B & C);
                X3 <= (~A & B) || (C & D);
                X4 <= B || (A & ~D);
                X5 <= A || (B & D) || (B & C);
            end
        end else begin
            if (reset) begin
                X1 = 0;
                X2 = 0;
                X3 = 0;
                X4 = 0;
                X5 = 0;
            end
        end
    end
endmodule
```

Para que nosso código funcionasse na FPGA tivemos que remover um dos dois Always existentes e integrar sua lógica dentro do outro, pois a FPGA só aceita ter um Always rodando ao mesmo tempo. E para que ficasse mais bem feito implementamos um if que impede que a FPGA represente as entradas a partir de 10, pois nosso projeto está voltado em codificar números de 0 a 9 em código Morse.



Para fazer a simulação estamos utilizando os switches 17, 16, 15 e 14 para receber o valor de entrada, os switches 1 e 0 para receber o valor do reset e ready. Para representar o resultado da codificação estamos usando os Led’s vermelhos 17, 16, 15, 14 e 13.

7. Conclusão:

Este trabalho foi muito importante para nosso conhecimento a compreensão do tema, pois permitiu-nos a melhorar competências de trabalho em equipe, organização e forçou o aprendizado da linguagem de descrição de hardware (Verilog) além da utilização de softwares de simulação e na FPGA.