16 FEVEREIRO 2023

CALCULADORA

Projeto final

GRUPO B13:

Vitor Guedes - 221017130
Davy Viana - 211055559
Gabriela Rodrigues - 180120859

UnB | CIC

RESUMO

Neste projeto foi implementado uma calculadora com uma ULA que realiza operações de adição, subtração e multiplicação. Para as entradas utilizamos um teclado de membrana, que contém os números em decimal e as letras que correspondem as operações. Os resultados são exibidos nos LEDs e 2 Displays de 7 segmentos da placa FPGA, quando o resultado da subtração for um número negativo, o sinal também será exibido. Todo o projeto foi realizado no quartus II utilizando Verilog.

ABSTRACT

In this project was implemented a calculator with an ALU that performs addition, subtraction and multiplication operations. For the entries we use a membrane keyboard, which contains the numbers in decimal and the letters that correspond to the operations. The results are displayed on the LEDs and 2 7-segment displays of the FPGA board, when the subtraction result is a negative number, the sign will also be displayed. The entire project was carried out in quartus II using Verilog.

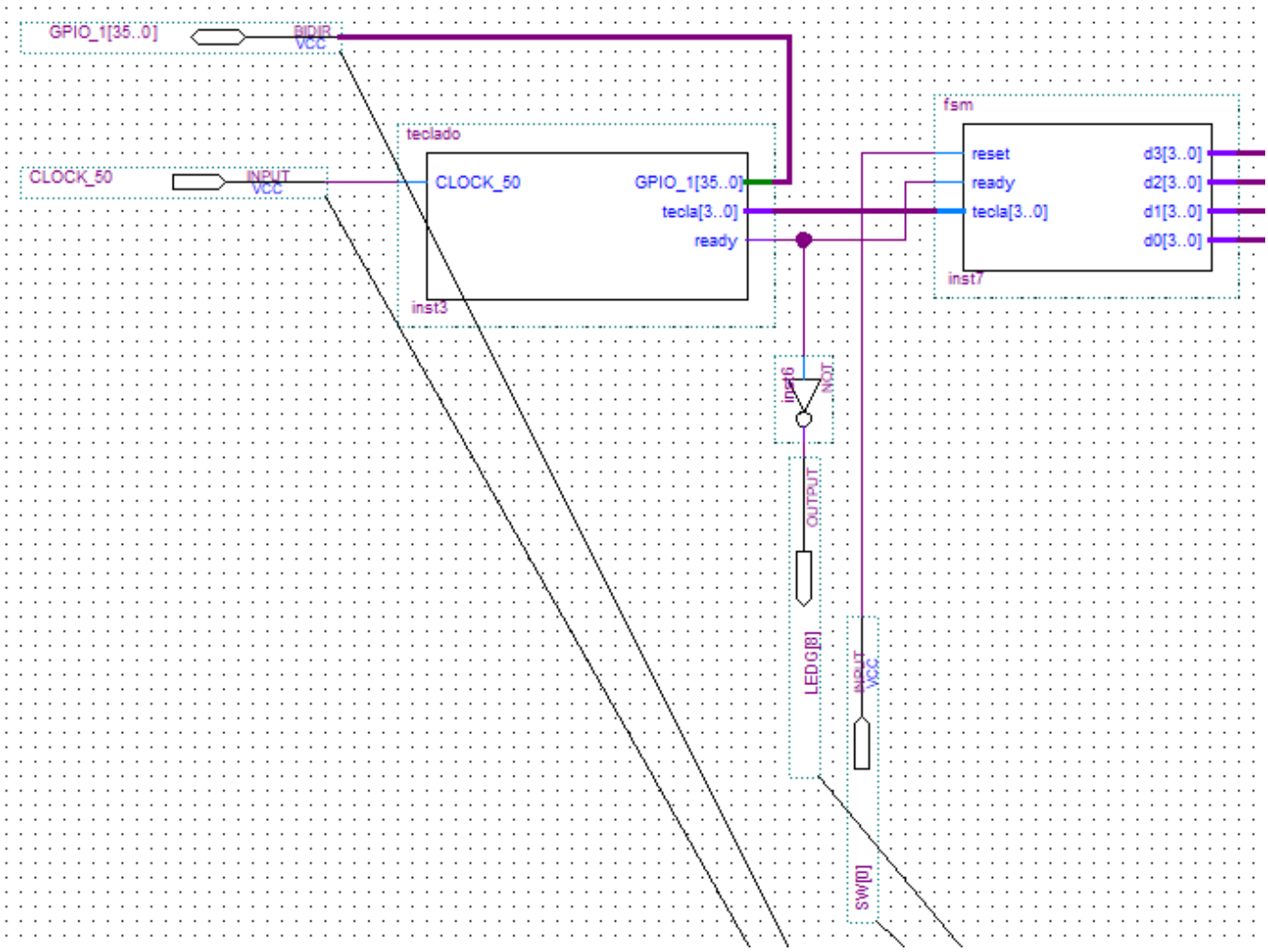
INTRODUÇÃO

As calculadoras são objetos imprescindíveis na vida de vários estudantes, engenheiros e cientistas e, para a construção da mesma, é necessário que se possua entre outros componentes, uma ULA (Unidade Lógico-Aritmética) capaz de realizar as operações. Neste projeto foi desenvolvido um sistema digital síncrono que implementa uma calculadora com uma ULA que realiza as operações de adição, subtração e multiplicação de números inteiros de um dígito decimal; então será lido um número em decimal de 0 a 9 do teclado, e ao pressionar uma tecla de operação (A, B ou C), uma determinada operação será realizada. A corresponde a multiplicação, B subtração e C a soma. O D corresponde ao =. Para a entrada, usaremos um teclado matricial 4x4, um mini-teclado de membrana que será implementado no projeto do circuito como "teclado" em Verilog. As saídas do controlador, neste caso "teclado", são os números de 4 bits que corresponde ao número pressionado no teclado de membrana. Todo o projeto foi desenvolvido em Verilog e o sistema implementado no chip FGPA. Além da ULA e do controlador do teclado, o sistema conta com decodificadores, o módulo responsável pelo sinal e o módulo do display. Os resultados da operações são exibidos em dois displays de 7 segmentos, e caso a operação de subtração gere um número negativo, o sinal também será exibido.

ENTRADAS

As entradas da calculadora são realizadas em um teclado de 4x4, onde são utilizados todos os dígitos em decimal como valores das operações, e a calculadora realiza operações de soma, subtração e multiplicação, onde as teclas: A= multiplicação, B= subtração, C= soma, D= resultado. A calculadora foi feita para receber os valores na seguinte ordem:

- 1° entrada A
- 2° operação
- 3° entrada B
- 4 ° autorização para mostrar resultado



TECLADO

O bloco do teclado recebe os sinais de entrada do teclado físico, e converte para um código de 4 bit para funcionamento da calculadora. O sinal que é gerado pelo teclado físico chega em 8 bits, e o bloco fará a conversão para 4 bits, seguindo a logica crescente em binário. Ao ser pressionada uma tecla no teclado, sai um sinal ready, que indica tecla pressionada, e desligara o LEDG8, que se ligado, indica estar pronto para receber uma nova entrada.

FSM

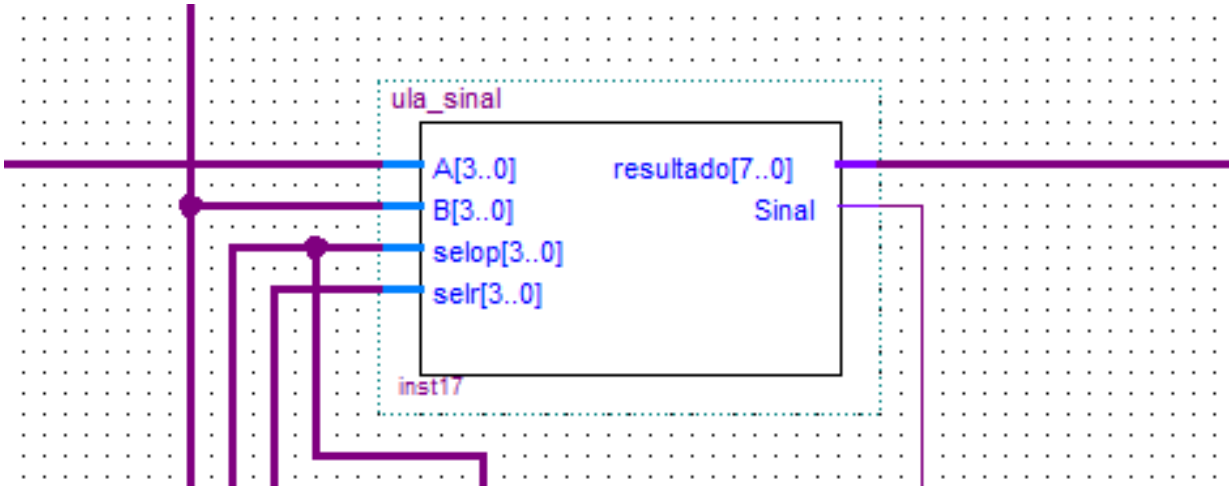
O bloco FSM tem a função de receber as entras já convertidas para 4 bits, guardar e ordenar os valores em registradores. FSM reconhece cada tipo de entrada esta recebendo de acordo com a ordem de inserção especificada na calculadora, armazenada cada valor em um registrador e retorna nas saídas d3,d2,d1,d0 os respectivos valores para uso na calculadora. Caso queira que seja reiniciado todos os valores e operações na calculadora, o FSM tem a entrada reset, que é ativa pela porta sw[0], que caso acionada desempenha a função de reiniciar a calculadora.

```
module   fsm(
            input logic reset,
            input logic ready,
            input logic [3:0] tecla,
            output  logic [3:0] d3, d2, d1, d0
         );

   typedef enum logic [1:0] {DISPLAY_A, DISPLAY_B, DISPLAY_C, DISPLAY_D} statetype;
   statetype state, nextstate;

   always_ff @(posedge ready, posedge reset)
      if (reset) state <= DISPLAY_A;
      else state <= nextstate;

   always_comb
         case (state)

            DISPLAY_A: if (tecla<16) begin
                     nextstate = DISPLAY_B;
                end
               else nextstate = DISPLAY_A;

            DISPLAY_B: if (tecla<16) begin
                     nextstate = DISPLAY_C;
                end
               else nextstate = DISPLAY_B;

            DISPLAY_C: if (tecla<16) begin
                     nextstate = DISPLAY_D;
                end
               else nextstate = DISPLAY_C;

            DISPLAY_D: if (tecla<16) begin
                     nextstate = DISPLAY_A;
                end
               else nextstate = DISPLAY_D;

            default: nextstate = DISPLAY_A;

         endcase
```

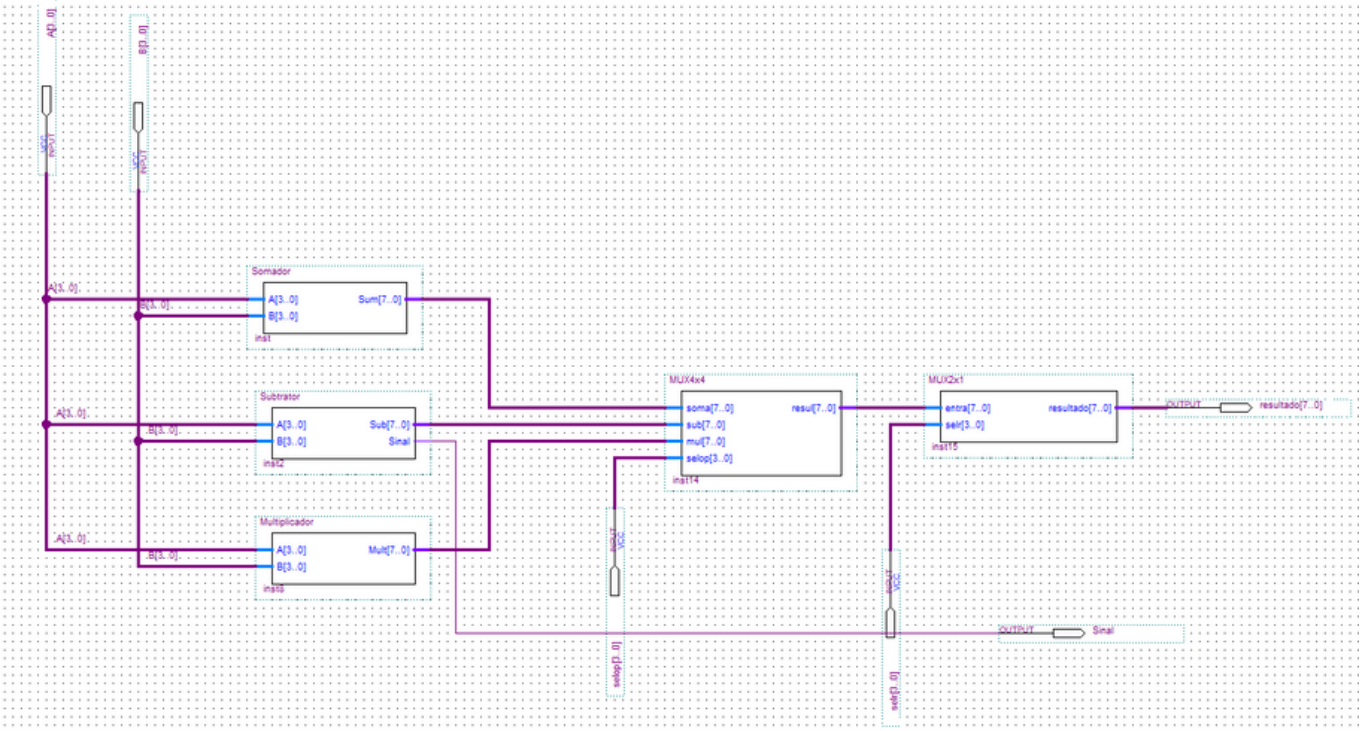
```
   always_ff @(negedge ready, posedge reset)
         if(reset) begin
               d3 = '0;
               d2 = '0;
               d1 = '0;
               d0 = '0;
         end
         else
            case (state)

               DISPLAY_A: if(tecla<16) d0 = tecla;

               DISPLAY_B: if(tecla<16) d1 = tecla;

               DISPLAY_C: if(tecla<16) d2 = tecla;

               DISPLAY_D: if(tecla<16) d3 = tecla;

               default: begin
                     d3 = '0;
                     d2 = '0;
                     d1 = '0;
                     d0 = '0;
               end

            endcase
endmodule
```

ULA

A unidade Lógica e Aritmética (ULA) compões uma parte fundamental de processadores e microprocessadores e consiste em um circuito digital capaz de realizar operações lógicas e aritméticas. Neste projeto, a ULA é capaz de realizar operações de soma, subtração e multiplicação. As entradas são os valores que irão compor as operações, sendo estes A, B e os seletores. A saída será o resultado da operação e o bit do sinal, para exibição caso seja negativo.



Nesta ULA todas as operações são realizadas simultaneamente e através do seletor no primeiro MUX selecionamos qual resultado iremos exibir ao usuário.



Cada componente nesta ULA, tanto a soma, subtração quanto a multiplicação foi implementado com o uso de Verilog. As operações são simples e com o uso do Verilog poucas linhas foram necessárias para a descrição das operações.

Adição

O somador recebe as entradas A e B e realiza a operação de soma, que sai no output Sum, como demonstrado na figura a seguir.

```
1   module Somador(
2       input [3:0] A,
3       input [3:0] B,
4       output [7:0] Sum
5   );
6
7   assign Sum = A + B;
8
9   endmodule
```

Subtração

O subtrator ao receber os valores de A e B, verifica qual é o maior valor, se A for maior, então A - B, caso contrário B - A. Também é verificado através do maior valor, qual será o o bit do sinal para o elemento, então há 2 outputs, o resultado da subtração e o bit do sinal.

```
1   module Subtrator(
2   input [3:0] A,
3   input [3:0] B,
4   output [7:0] Sub,
5   output Sinal
6   );
7
8   assign Sub = (A >= B)? A - B: B - A;
9   assign Sinal = (A >= B)? 0: 1;
10
11  endmodule
```

Multiplicação

De maneira análoga as anteriores , em verilog, a operação de multiplicação é realizada. Com os inputs A e B, e output o resultado da multiplicação.

```
module Multiplicador(
input [3:0] A,
input [3:0] B,
output [7:0] Mult
);

assign Mult = A * B;

endmodule
```

Após as três operações serem realizadas, prosseguimos para a seleção dos resultados através do seletor. No primeiro MUX selecionamos qual dos resultados das operações seguirá para uma futura exibição, o segundo MUX é responsável por receber o =, quando este é pressionado, e prosseguir com o resultado. Foram implementados 2 MUX, um 4:4 e um 2:1, o resultado das operações realizadas são escolhidas através do operador selecionado, A = Multiplicação, B = Subtração, C = Soma e D = (=), o resultados da operação.

```
module MUX4x4(soma, sub, mul, selop, resul);
   input [7:0]soma;
   input [7:0]sub;
   input [7:0] mul;
   input [3:0]selop;
   output reg [7:0]resul;

   always @ (soma or sub or mul or selop)
      begin : MUX
       case(selop)
          4'b1010 : resul = mul;
          4'b1011 : resul = sub;
          4'b1100 : resul = soma;
          default: resul = 8'b00000000;
       endcase
      end

endmodule
```

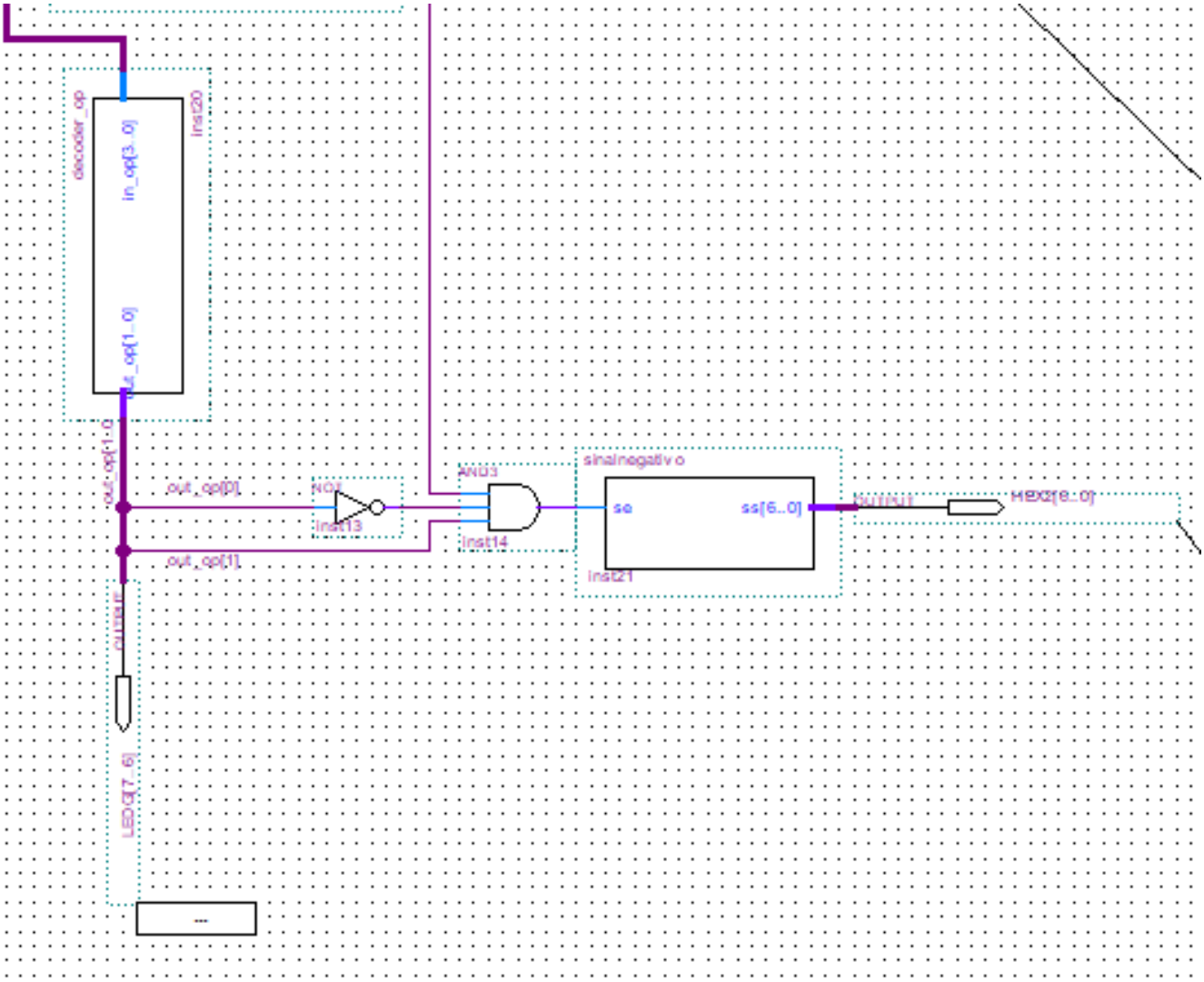
MUX 4:1

```
module MUX2x1( entra,selr, resultado);
   input [7:0]entra;
   input [3:0]selr;
   output reg [7:0]resultado;

   always @ (selr)
      begin : MUX
       case(selr)
          4'b1101 : resultado = entra;
          default: resultado = 8'b00000000;
       endcase
      end
endmodule
```

MUX 2:1

BIT DE SINAL

Em casos onde a segunda entrada da operação for maior que a primeira, as operações de subtração serão negativas, consequentemente a calculadora mostrará no display HEX2 um traço indicando que o resultado é menor que zero, caso contrario, o display fica constante em zero. Para implementação de tal, foi usado um circuito que só mostrara o sinal de negativo, caso seja feita uma operação de subtração, e a ULA indicar ser negativo o resultado da operação. Para separar tais informações foram criados 2 blocos, um decodificador de 4 bit que recebe o tipo de operação selecionada, e um codificador que caso acionado, retorna o código de ativação do display HEX2.



O decodificador de operações, recebe o tipo de operação em binário de 4 bits, sendo: 1010,1011,1100 e retornando (11,10, e 01) multiplicação, subtração, e soma respectivamente, se a entrada for diferente de algum desses casos, retorna 0000. O respectivo resultado do decodificador em 2 bit é mostrado nos LEDG[7..6], indicando em binário para o usuário qual o tipo de operação realizada. Caso a operação seja de subtração e o resultado for negativo, a Nand de entradas retorna 1 e o decodificador do sinal de negativo é ativo, manifestando para o usuário o sinal de negativo no display de 7 segmentos.

```
module decoder_op(
   input reg [3:0] in_op,
   output reg [1:0] out_op
);

   always @(in_op)
      begin
         case(in_op)
            4'b1010 : out_op = 2'b11;
            4'b1011 : out_op = 2'b10;
            4'b1100 : out_op = 2'b01;
            default : out_op = 2'b00;
         endcase
      end
endmodule
```

O decodificador de display, caso sejam satisfeitos todos os requisitos para sua ativação, ela retornara o código em binário de 7 bits que corresponde ao mapa do símbolo de negativo no display de 7 segmentos(0111111), caso contrario permanece o símbolo zero(1000000) .

```
module sinalnegativo(
   input reg se,
   output reg [6:0]ss
);

   always @(se)
      begin
         case(se)
            1'b1 : ss = 7'b0111111;
            default: ss =7'b1000000;
         endcase
      end

endmodule
```

Decoder 7bit

O processo do Decoder se trata de pegar os números de 4 bits de entrada e mostrar nos LEDs de 7 segmentos os números que o usuário apertou de entrada.

O circuito pega o numero em 4 bits que o usuário entrou e verifica qual seria o relativo desse numero nos LEDs de 7 segmentos.

```
1  // Decodificador  de binário para
2  // display 7 segmentos
3  module decoder7 (
4      input [3:0] In,
5      output reg [6:0] Out
6
7  );
8
9      always @(*)
10         case (In)
11             4'b0000 : Out = ~7'b0111111;
12             4'b0001 : Out = ~7'b0000110;
13             4'b0010 : Out = ~7'b1011011;
14             4'b0011 : Out = ~7'b1001111;
15
16             4'b0100 : Out = ~7'b1100110;
17             4'b0101 : Out = ~7'b1101101;
18             4'b0110 : Out = ~7'b1111101;
19             4'b0111 : Out = ~7'b0000111;
20
21             4'b1000 : Out = ~7'b1111111;
22             4'b1001 : Out = ~7'b1101111;
23             4'b1010 : Out = ~7'b1110111;
24             4'b1011 : Out      = ~7'b1111100;
25
26             4'b1100 : Out = ~7'b0111001;
27             4'b1101 : Out = ~7'b1011110;
28             4'b1110 : Out = ~7'b1111001;
29             4'b1111 : Out = ~7'b1110001;
30
31             default : Out = ~7'b0000000;
32         endcase
33
34 endmodule
35
36
```

Display

O processo do Display é parecido com o Decoder, ele recebe o resultado dos processos e verifica o sinal e o relativo do resultado nos LEDs de 7 segmentos

```
module display(
            bcd,
            seg,
        seg2
            );
    input [7:0] bcd;    //initializing bcd as an 4 bit input signal
    output[6:0] seg;    //initializing seg as an 8 bit output signal
   output [6:0] seg2;
    reg [6:0] seg;      //initializing bcd signal as registers
    reg [6:0] seg2;      //initializing bcd signal as registers

    always @ (bcd)      //using the always statement to indicate any change in the bcd signal results in evaluating the declared cases
    begin
        case(bcd)   //begining the case statement which evaluates the bcd value and assigns the appropriate contol signals to our 8bit output signal).
            0: seg = 7'b0111111;   //when bcd = 0
            1: seg = 7'b0000110;   //when bcd = 1
            2: seg = 7'b1011011;   //when bcd = 2
            3: seg = 7'b1001111;   //when bcd = 3
            4: seg = 7'b1100110;   //when bcd = 4
            5: seg = 7'b1101101;   //when bcd = 5
            6: seg = 7'b1111101;   //when bcd = 6
            7: seg = 7'b0000111;   //when bcd = 7
            8: seg = 7'b1111111;   //when bcd = 8
            9: seg = 7'b1101111;   //when bcd = 9

            10: seg = 7'b0111111;   //when bcd = 9
            11: seg = 7'b0000110;   //when bcd = 9
            12: seg = 7'b1011011;   //when bcd = 9
            13: seg = 7'b1001111;   //when bcd = 9
            14: seg = 7'b1100110;   //when bcd = 9
            15: seg = 7'b1101101;   //when bcd = 9
            16: seg = 7'b1111101;   //when bcd = 9
            17: seg = 7'b0000111;   //when bcd = 9
            18: seg = 7'b1111111;   //when bcd = 9
            19: seg = 7'b1101111;   //when bcd = 9

            20: seg = 7'b0111111;   //when bcd = 9
            21: seg = 7'b0000110;   //when bcd = 9
            22: seg = 7'b1011011;   //when bcd = 9
            23: seg = 7'b1001111;   //when bcd = 9
            24: seg = 7'b1100110;   //when bcd = 9
            25: seg = 7'b1101101;   //when bcd = 9
            26: seg = 7'b1111101;   //when bcd = 9
            27: seg = 7'b0000111;   //when bcd = 9
            25: seg = 7'b1101101;   //when bcd = 9
            26: seg = 7'b1111101;   //when bcd = 9
            27: seg = 7'b0000111;   //when bcd = 9
            28: seg = 7'b1111111;   //when bcd = 9
            29: seg = 7'b1101111;   //when bcd = 9

            30: seg = 7'b0111111;   //when bcd = 9
            31: seg = 7'b0000110;   //when bcd = 9
            32: seg = 7'b1011011;   //when bcd = 9
            33: seg = 7'b1001111;   //when bcd = 9
            34: seg = 7'b1100110;   //when bcd = 9
            35: seg = 7'b1101101;   //when bcd = 9
            36: seg = 7'b1111101;   //when bcd = 9
            37: seg = 7'b0000111;   //when bcd = 9
            38: seg = 7'b1111111;   //when bcd = 9
            39: seg = 7'b1101111;   //when bcd = 9

            40: seg = 7'b0111111;   //when bcd = 9
            41: seg = 7'b0000110;   //when bcd = 9
            42: seg = 7'b1011011;   //when bcd = 9
            43: seg = 7'b1001111;   //when bcd = 9
            44: seg = 7'b1100110;   //when bcd = 9
            45: seg = 7'b1101101;   //when bcd = 9
            46: seg = 7'b1111101;   //when bcd = 9
            47: seg = 7'b0000111;   //when bcd = 9
            48: seg = 7'b1111111;   //when bcd = 9
            49: seg = 7'b1101111;   //when bcd = 9

            50: seg = 7'b0111111;   //when bcd = 9
            51: seg = 7'b0000110;   //when bcd = 9
            52: seg = 7'b1011011;   //when bcd = 9
            53: seg = 7'b1001111;   //when bcd = 9
            54: seg = 7'b1100110;   //when bcd = 9
            55: seg = 7'b1101101;   //when bcd = 9
            56: seg = 7'b1111101;   //when bcd = 9
            57: seg = 7'b0000111;   //when bcd = 9
            58: seg = 7'b1111111;   //when bcd = 9
            59: seg = 7'b1101111;   //when bcd = 9

            60: seg = 7'b0111111;   //when bcd = 9
            61: seg = 7'b0000110;   //when bcd = 9
            62: seg = 7'b1011011;   //when bcd = 9
            63: seg = 7'b1001111;   //when bcd = 9
            64: seg = 7'b1100110;   //when bcd = 9
            default: seg=7'b0000000;    //any other value
        endcase
        case(bcd)   //begining the case statement which
            0: seg2 = 7'b0111111;   //when bcd = 0
            1: seg2 = 7'b0111111;   //when bcd = 1
            2: seg2 = 7'b0111111;   //when bcd = 2
            3: seg2 = 7'b0111111;   //when bcd = 3
            4: seg2 = 7'b0111111;   //when bcd = 4
            5: seg2 = 7'b0111111;   //when bcd = 5
            6: seg2 = 7'b0111111;   //when bcd = 6
            7: seg2 = 7'b0111111;   //when bcd = 7
            8: seg2 = 7'b0111111;   //when bcd = 8
            9: seg2 = 7'b0111111;   //when bcd = 9

            10: seg2 = 7'b0000110;   //when bcd = 9
            11: seg2 = 7'b0000110;   //when bcd = 9
            12: seg2 = 7'b0000110;   //when bcd = 9
            13: seg2 = 7'b0000110;   //when bcd = 9
            14: seg2 = 7'b0000110;   //when bcd = 9
            15: seg2 = 7'b0000110;   //when bcd = 9
            16: seg2 = 7'b0000110;   //when bcd = 9
            17: seg2 = 7'b0000110;   //when bcd = 9
            18: seg2 = 7'b0000110;   //when bcd = 9
            19: seg2 = 7'b0000110;   //when bcd = 9

            20: seg2 = 7'b1011011;   //when bcd = 9
            21: seg2 = 7'b1011011;   //when bcd = 9
            22: seg2 = 7'b1011011;   //when bcd = 9
            23: seg2 = 7'b1011011;   //when bcd = 9
            24: seg2 = 7'b1011011;   //when bcd = 9
            25: seg2 = 7'b1011011;   //when bcd = 9
            26: seg2 = 7'b1011011;   //when bcd = 9
            27: seg2 = 7'b1011011;   //when bcd = 9
            28: seg2 = 7'b1011011;   //when bcd = 9
            29: seg2 = 7'b1011011;   //when bcd = 9

            30: seg2 = 7'b1001111;   //when bcd = 9
            31: seg2 = 7'b1001111;   //when bcd = 9
            32: seg2 = 7'b1001111;   //when bcd = 9
            33: seg2 = 7'b1001111;   //when bcd = 9
            34: seg2 = 7'b1001111;   //when bcd = 9
            35: seg2 = 7'b1001111;   //when bcd = 9
            36: seg2 = 7'b1001111;   //when bcd = 9
            37: seg2 = 7'b1001111;   //when bcd = 9
```

Link para o vídeo com o funcionamento do projeto: https://youtu.be/BH6ItjabLfU

CONCLUSÕES

Nesse projeto fizemos com sucesso uma calculadora com as operações de soma subtração e multiplicação de números inteiros.

Apesar de ser um projeto relativamente simples, foram necessárias algumas tentativas até chegarmos ao projeto final . Com este projeto fomos capazes de colocar em prático todos os conhecimentos acerca do conteúdo estudado ao longo do semestre. Após tentarmos a implementação com o uso de circuitos, este se mostrou mais demorado e visualmente longo que escrever em Verilog, desta maneira, utilizamos Verilog para toda a implementação da calculadora.

O resultados das operações é exibido em 2 displays de 7 segmentos, e caso o resultado da subtração gere um número negativo, o sinal também é exibido em um display de 7 segmentos.

Conseguimos através deste projeto implementar entre outras operações, Decodificadores, Multiplexadores, e também utilizar Verilog que se tornou muito útil em otimizar o espaço e o tempo destinado a construção dos circuitos no Quartus II.

REFERÊNCIAS

- Pedroni, V., Eletrônica Digital Moderna e VHDL, Campus, 2010
- Tocci, R. J. e Widmer, N. S, Sistemas digitais: princípios e aplicações, LTC, 2010
- Harris, D. M. e Harrys S. L. Digital design and computer architecture, Morgan Kaufmann, 2017