Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Computação e Automação
Engenharia de Computação
Orientador: Kennedy Reurison Lopes

Discentes:
César Zaqueu Fernandes de Medeiros
Jhonat Heberson Avelino de Souza
Vinicius de Azevedo Menezes

Minicalculadora de 4 bits em FPGA

Natal

08 de março de 2019

Sumário

1 INTRODUÇÃO 2
1.1 Resumo do problema 2
1.2 Metodologia utilizada 3
2 APRESENTAÇÃO DA SOLUÇÃO 4
3 DESENVOLVIMENTO E RESULTADOS 6
4 CONCLUSÃO 18

1 Introdução

Este presente trabalho tem como objetivo apresentar, debater e principalmente aplicar os conhecimentos visto em sala de aula da disciplina DCA0212.1 – Circuitos Digitais – Laboratório, lecionada pelo professor Kennedy Reurison Lopes.

De início começaremos apresentando o nosso problema que se trata em projetar uma minicalculadora em VHDL e apresentá-la em uma placa FPGA. Apresentaremos a metodologia que foi utilizada durante a construção do nosso trabalho. Em seguida, iniciaremos a fase de conclusão do trabalho, onde apresentaremos como iremos solucionar o problema, dividindo essa apresentação em desenvolvimento, resultados e conclusão.

1.1 Resumo do problema

Este projeto no qual estamos aqui construindo, tem como objetivo a criação de uma minicalculadora feita em VHDL que deve implementar cinco operações básicas: Adição, subtração, maiorQue, menorQue e inversão. A entrada do projeto serão duas sequências de 4 bits.

A escolha da operação será realizada através de 3 chaves de comando, e adicionalmente um botão liga/desliga, que desabilita todas as operações. Selecionada a operação, o circuito deverá efetuá-la de maneira correta, direcionando a resposta para um display de 7 segmentos, mostrando assim para o usuário o resultado da operação que ele selecionou nas chaves de comando.

Posteriormente, explicaremos mais detalhadamente como funcionará cada parte específica do projeto. Tendo em vista, que estamos apenas no resumo do projeto.

1.2 Metodologia utilizada

Como pode-se observar na capa do trabalho, nosso grupo é formado por três integrantes. Com isso, para maior facilidade e que todos ficassem comprometidos com uma parte do projeto, dividimos o mesmo em três partes iguais, onde ficou da seguinte maneira:

Um componente ficou com as 5 funções de operações básicas, outro com o multiplexador e o último com o Conversor Display e a criação do relatório. Optamos por essa metodologia, pelo fato de que como basicamente os três blocos são "independentes" (dependendo apenas da saída do anterior), poderíamos assim cada um ficar com uma parte, e no final juntar todas e termos o trabalho completo e bem dividido, sem que nenhum ficasse sobrecarregado.

Cada componente teve total responsabilidade pela parte na qual ficou comprometido. Porém, todos tivemos o mesmo método de solução para o problema: usamos dos instrumentos utilizados em sala de aula, no qual já tínhamos utilizado, os códigos disponibilizados pelo professor, e total orientação do mesmo, que teve suma importância para criação deste trabalho, tendo em vista que o mesmo sempre se disponibilizou para ajuda seja por meio de e-mail, seja pessoalmente.

Então, com a ajuda do professor e com o material que tínhamos antes, conseguimos desenvolver a nossa minicalculadora, na qual a mesma consegue realizar as operações desejadas. E fornecer para o usuário o seu resultado em um display de 7 segmentos.

2 Apresentação da solução

De início, iremos apresentar a imagem disponibilizada pelo professor, para início do projeto. A partir dela, poderemos explicar como irá funcionar cada método do projeto, em seguida apresentar os respectivos códigos, e por fim o projeto final.

A figura abaixo, representa o diagram geral da nossa minicalculadora (Figura 2). Vamos anilisá-la detalhadamente e explicar detalhadamente como funcionará a mesma.

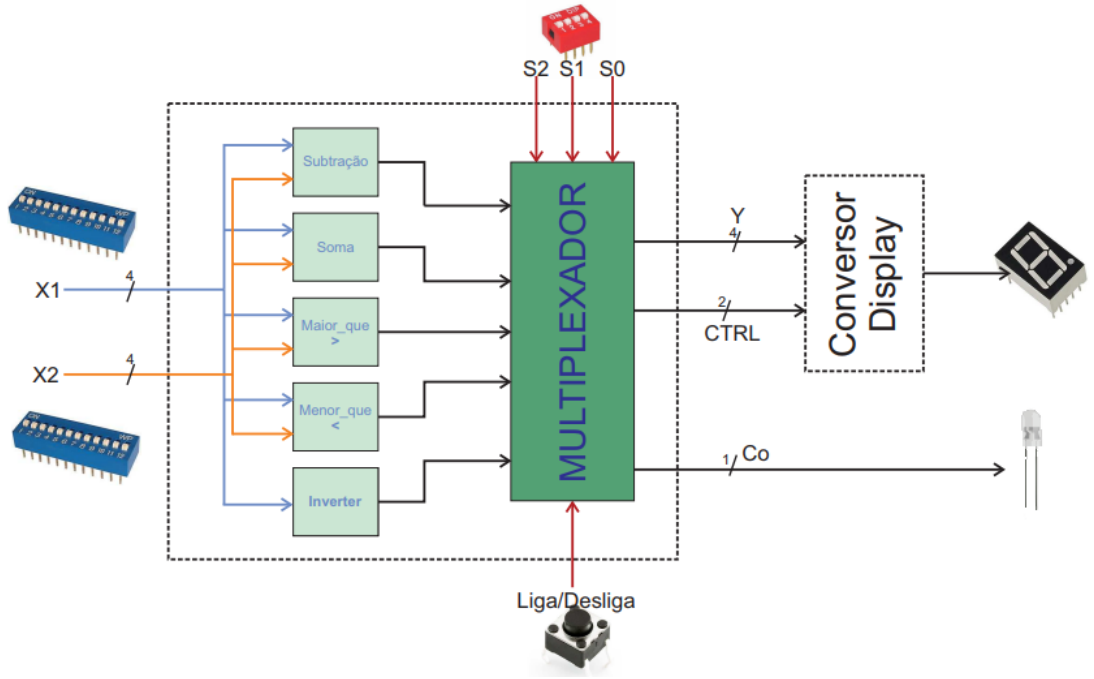

Figura 1 – Diagrama geral da Minicalculadora

Como foi destacado no resumo do trabalho, temos aqui o nosso diagrama geral com todos os problemas que teremos que solucionar.

Analisando a Figura 2, podemos iniciar nossa análise pelas nossas entradas. Como podemos ver, as entradas X1 e X2 são ambas entradas simples de 4 bits. Em

seguida, nossas 5 operações, na qual apenas a inverter só recebe uma das entradas. Cada operação funciona da seguinte maneira:

- Subtração -> Retorna um número de 4 bits, resultado da subtração entre X1 e X2. E apresenta erro caso o resultado seja inesperado ;
- Soma -> Retorna um número de 4 bits, resultado da soma entre X1 e X2. E acende o LED Co caso a soma ultrapasse o valor 15, e exibindo o valor no display;
- MaiorQue -> O LED Co é aceso caso X1 seja maior que X2;
- MenorQue -> O LED Co é aceso caso X1 seja menor que X2;
- Inverter -> Inverte o valor de X1. Ou seja, retorna seu complemento.

Após as nossas funções, vem o nosso multiplexador. O multiplexador é onde o usuário seleciona a função que deseja realizar. Ou seja, por meio das chaves seletoras S0, S1 e S2, o usuário escolhe uma das funções. Ao escolher uma das funções, automaticamente o resultado será repassado para o Conversor Display, que é a última parte do nosso projeto.

Como podemos ver o multiplexador tem 3 saídas. Onde duas vão para o Conversor e uma para o LED. A saída Co do LED é para sinalizar o resultado de algumas funções, e só tem um bit, que é aceso ou apagado. Já as outras saídas são a Y, que é o resultado das funções e CTRL, que é para ter o controle do que será exibido no display. Então, o conversor recebe essas duas entradas, e dependendo do resultado da variável CTRL exibe no conversor display o resultado da operação selecionada pelo usuário, ou caso tenha dado algum problema, o símbolo de erro.

3 Desenvolvimento e Resultados

Após ter sido apresentado na seção anterior, como funciona cada um dos elementos que formam o nosso projeto, vamos agora para a parte da apresentação de como desenvolvemos os códigos para conseguirmos solucionar os problemas desse trabalho.

Seguiremos a ordem de como foram apresentados aqui no trabalho, primeiro mostraremos os códigos das cinco operações, em seguida o multiplexador e por final o conversor display.

A seguir serão apresentadas as cinco funções responsáveis pelas operações básicas nas quais foram submetidas o projeto. Que são: 1) Subtração, 2) Soma, 3) MaiorQue, 4) MenorQue e 5) Inverter. Os códigos serão apresentados nessa mesma ordem abaixo:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

-- declaring entity
entity subtra_4bits is
port(
        a,b: in std_logic_vector(3 downto 0); -- vectors input the 4 bits
        s  : out std_logic_vector(3 downto 0); -- vectors output the 4 bits or result subtracion
        c4 : out std_logic -- logic output or error
        );
end subtra_4bits;

-- declaring architecture
architecture estrutural of subtra_4bits is
        signal t, result, resultSub, aux:std_logic_vector(3 downto 0);
        signal queue, agtb : std_logic;

        -- include component lessThan
        component lessThan is
        port (
                A, B: in std_logic_vector(3 downto 0); -- vectors input the 4 bits
                AgtB : out STD_LOGIC --logic output
                );
        end component;

        -- include component inverter
        component inverter is
        port (
                A: in std_logic_vector(3 downto 0); -- vectors input the 4 bits
                S: out std_logic_vector(3 downto 0) -- vectors output the 4 bits
                        );
        end component;

        -- include component somador_4bits
        component somador_4bits is
        port(
                a,b: in std_logic_vector(3 downto 0); -- vectors input the 4 bits
                c0 : in std_logic; --logic input
                s  : out std_logic_vector(3 downto 0); -- vectors output the 4 bits
                c4 : out std_logic --logic output
                );
        end component;

        begin -- técnica do eleva dois
                inv_b0: inverter port map(b, t); -- inveter o numero a ser subtraído
                sub: somador_4bits port map(a,t, queue, result); -- soma com o numero maior que o que invertir
                sub2: somador_4bits port map(result, "0001", queue, resultSub); -- soma com 1 o resulta da soma anterior
                trat: lessThan port map(a,b, agtb); -- fazendo tratamento para ver se b >a
                aux(0) <= agtb; -- fazendo 4 bits para ser o erro
                aux(1) <= agtb;
                aux(2) <= agtb;
                aux(3) <= agtb;
                s <= (resultSub and not(aux)); -- se aux for 1, logo da erro, não é possível fazer a subtração
                c4 <= agtb;
        end estrutural;
```

Figura 2 – Código da função subtração

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

-- declaring entity
entity somador_4bits is
port(
        a,b: in std_logic_vector(3 downto 0); -- vectors input the 4 bits
        c0 : in std_logic; -- logic output
        s  : out std_logic_vector(3 downto 0); -- vectors input the 4 bits, which is the sum
        c4 : out std_logic -- logic output, elevates one to make adder with more bits
        );
end somador_4bits;

-- declaring architecture
architecture estrutural of somador_4bits is
        signal vai_um : std_logic_vector(0 to 2); -- goes one bit by bit

        -- include component somador_completo
        component somador_completo is
        port(
                a,b     : in std_logic; -- logic inputs
                cin     : in std_logic; -- logic input
                s       : out std_logic; -- logic output
                cout    : out std_logic -- logic output
        );
        end component;

        begin -- bit bit adder
                s1: somador_completo port map(a(0), b(0), c0, s(0), vai_um(0)); -- using the complete adder component to add bit by bit
                s2: somador_completo port map(a(1), b(1), vai_um(0), s(1), vai_um(1)); -- using the complete adder component to add bit by bit
                s3: somador_completo port map(a(2), b(2), vai_um(1), s(2), vai_um(2)); -- using the complete adder component to add bit by bit
                s4: somador_completo port map(a(3), b(3), vai_um(2), s(3), c4); -- using the complete adder component to add bit by bit
        end estrutural;
```

Figura 3 – Código da função soma

```vhdl
library ieee;
use ieee.std_logic_1164.all;

-- declaring entity
entity somador_completo is
port(
        a, b    : in std_logic; -- logic input
        cin     : in std_logic; -- logic input
        s               : out std_logic; -- logic output
        cout : out std_logic -- logic output
);
end somador_completo;

-- declaring architecture
architecture dataflow of somador_completo is -- addition of a bit
begin
        s <= a xor b xor cin; -- sum of one bit
        cout <= (a and b) or (a and cin) or (b and cin); -- goes one of the sum
end dataflow;
```

Figura 4 – Código da função soma completa (Subrtração e Adição)

```vhdl
library ieee ;
use ieee.std_logic_1164.all ;

-- declaring entity
entity biggerThen is
port (
        A, B: in std_logic_vector(3 downto 0); -- vectors input
        AgtB : out STD_LOGIC -- logic output
        );
end biggerThen ;

-- declaring architecture
architecture Behavior of biggerThen is -- biggerThen
begin
AgtB <= (A(3) and not(B(3))) or ((A(3) xnor B(3)) and (A(2)and not B(2)))
 or ((A(3) xnor B(3)) and (A(2) xnor B(2)) and (A(1)and not B(1)))
 or ((A(3) xnor B(3)) and (A(2) xnor B(2)) and(A(1) xnor B(1)) and (A(0)and not B(0)));
end Behavior ;
```

Figura 5 – Código da função maior que

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4  -- declaring entity
5  entity lessThan is
6  port (
7          A, B: in std_logic_vector(3 downto 0); -- vectors input
8          AgtB : out STD_LOGIC  -- logic output
9          );
10 end lessThan;
11
12 -- declaring architecture
13 architecture Behavior of lessThan is
14
15 -- include component bigger Then
16 component biggerThen is
17 port (
18         A, B: in std_logic_vector(3 downto 0);
19         AgtB : out STD_LOGIC
20         );
21 end component ;
22
23
24 begin -- less Than
25         menor: biggerThen port map(B, A ,AgtB); -- I made the smaller one from the larger one, inverting inputs
26 end Behavior ;
```

Figura 6 – Código da função menor que

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3  use ieee.std_logic_arith.all ;
4
5  -- declaring entity
6  entity inverter is
7  port (
8          A: in std_logic_vector(3 downto 0); -- vector input
9          S: out std_logic_vector(3 downto 0) -- vector output
10         );
11 end inverter;
12
13 -- declaring architecture
14 architecture Behavior of inverter is --bitwise inverter
15 begin
16 S(0) <= not (A(0)); -- inversing A(0)
17 S(1) <= not (A(1)); -- inversing A(1)
18 S(2) <= not (A(2)); -- inversing A(2)
19 S(3) <= not (A(3)); -- inversing A(3)
20 end Behavior ;
```

Figura 7 – Código da função inverter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity selecionador is -- entidade do selecionador que será o multiplexador
        port(
           soma, subt, invt : in std_logic_vector (3 downto 0);
                carrySoma, carrySub, maq, mnq, enable: in std_logic;
                S : in std_logic_vector (2 downto 0);
                Y: out std_logic_vector(3 DOWNTO 0);
                ctrl : out std_logic_vector (1 downto 0);
                Co: out std_logic);
end selecionador;

architecture archSelecionador of selecionador is
begin

   y(0) <= (enable and (not S(0) and not S(1) and not S(2)) and (subt(0))) or (enable and (S(0) and not S(1) and not S(2)) and
        (soma(0))) or (enable and (S(0) and S(1) and S(2)) and (invt(0)));

        y(1) <= (enable and (not S(0) and not S(1) and not S(2)) and (subt(1))) or (enable and (S(0) and not S(1) and not S(2)) and (soma(1))) or
        (enable and (S(0) and S(1) and S(2)) and (invt(1)));

        y(2) <= (enable and (not S(0) and not S(1) and not S(2)) and (subt(2))) or (enable and (S(0) and not S(1) and not S(2)) and (soma(2))) or
        (enable and (S(0) and S(1) and S(2)) and (invt(2)));

        y(3) <= (enable and (not S(0) and not S(1) and not S(2)) and (subt(3))) or (enable and (S(0) and not S(1) and not S(2)) and (soma(3))) or
        (enable and (S(0) and S(1) and S(2)) and (invt(3)));

        Co <= (enable and (not S(0) and not S(1) and not S(2)) and carrySub) or (enable and (S(0) and not S(1) and not S(2)) and carrySoma) or
        (enable and (not S(0) and S(1) and not S(2)) and maq) or (enable and (S(0) and S(1) and not S(2)) and mnq);

        ctrl(0) <=(carrySub and (not S(0) and not S(1) and not S(2)))or
                  (S(0) and ((S(1) and S(2))or (not S(1)and S(2)))) or
                                (not S(0)and S(1)and S(2));

        ctrl(1) <= not enable or (not S(0) and S(1) and not S(2)) or (S(0) and S(1) and not S(2));

end archSelecionador;
```

Figura 8 – Código da função 'Selecionador'

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Led is -- entidade do Led
5          port(
6             S : in std_logic_vector (2 downto 0);
7                  Carry : in std_logic;
8                  enable : in std_logic;
9                  s_maq,s_mnq : in std_logic;
10                 Co: out std_logic
11 );
12 end Led;
13
14 architecture archLed of Led is -- arquitetura do Led
15
16
17         begin
18    process(S)
19         begin
20
21
22         if (enable = '1') then -- Quando o botão de ligar estiver desligado, o Led também estará
23                 Co <= '1';
24
25         elsif (S = "001") and (Carry > '0') then -- Led irá ascender quando a soma tiver um carry
26                 Co <= '0';
27         elsif (s_maq='1') then -- Led irá ascender quando X1>X2
28                 Co <= '0';
29         elsif (s_mnq='1') then  -- Led irá ascender quando X1<X2
30                 Co <= '0';
31
32         end if;
33
34         end process;
35 end archLed;
```

Figura 9 – Código da função responsável pelo LED Co

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MAIN is -- entidade principal
5          port(
6                  X1, X2 : in std_logic_vector(3 DOWNTO 0);
7                  enable : in std_logic;
8                  S : in std_logic_vector (2 downto 0);
9                  display: out std_logic_vector(6 downto 0);
10                 Co, s_maq, s_mnq : out std_logic);
11 end MAIN;
12
13 architecture archPrincipal of MAIN is -- arquitetura responsável por chamar todas as outras entidades
14
15 component subtra_4bits is
16 port(
17         a,b: in std_logic_vector(3 downto 0);
18         c0 : in std_logic;
19         s  : out std_logic_vector(3 downto 0);
20         c4 : out std_logic
21         );
22 end component;
23
24 component somador_4bits is
25 port(
26         a,b: in std_logic_vector(3 downto 0);
27         c0 : in std_logic;
28         s  : out std_logic_vector(3 downto 0);
29         c4 : out std_logic
30         );
31 end component;
32
```

Figura 10 – Código da função Main - Parte 1

```
33  component biggerThen is
34  port (
35          A, B: in std_logic_vector(3 downto 0);
36          AgtB : out STD_LOGIC
37          );
38  end component;
39
40  component lessThan is
41  port (
42          A, B: in std_logic_vector(3 downto 0);
43          AgtB : out STD_LOGIC
44          );
45  end component;
46
47  component inverter is
48  port (
49          A: in std_logic_vector(3 downto 0);
50          S: out std_logic_vector(3 downto 0)
51          );
52  end component;
53
54  component selecionador is
55          port(
56             soma, subt, invt : in std_logic_vector (3 downto 0);
57                  carrySoma, carrySub, maq, mnq, enable: in std_logic;
58                  S : in std_logic_vector (2 downto 0);
59                  Y: out std_logic_vector(3 DOWNTO 0);
60                  ctrl : out std_logic_vector (1 downto 0);
61                  Co: out std_logic);
62  end component;
```

Figura 11 – Código da função Main - Parte 2

```
63
64
65  component conversondisplay is
66  port(
67    ctrl      : in std_logic_vector(1 downto 0);
68    Y      : in std_logic_vector(3 downto 0);
69    display              : out std_logic_vector(6 downto 0));
70  end component;
71
72  signal ctrl : std_logic_vector(1 downto 0);
73  signal Rsub, Rsoma, Rinv, Rdisp: std_logic_vector(3 DOWNTO 0);
74  signal zero, Csub, Csoma, Cmaq, Cmnq: std_logic;
75
76  begin
77  zero <= '0';
78
79          SUB : subtra_4bits port map (a=>X1, b=>X2, c0=>zero, s=>Rsub, c4=>Csub);
80
81          SOMA: somador_4bits port map (a=>X1, b=>X2, c0=>zero, s=>Rsoma, c4=>Csoma);
82
83          MAIOR: biggerThen port map (A=>X1, B=>X2, AgtB=>Cmaq);
84
85          MENOR: lessThan port map (A=>X1, B=>X2, AgtB=>Cmnq);
86
87          INV : inverter port map (A=>X1, S=>Rinv);
88
89          SEL : selecionador port map(soma=>Rsoma, subt=>Rsub, invt=>Rinv, carrySoma=>Csoma, carrySub=>Csub,
90          maq=>Cmaq, mnq=>Cmnq, enable=>enable, S=>S, Y=>Rdisp, Co=>Co, ctrl=>ctrl);
91
92          CONV: conversondisplay port map (ctrl=> ctrl, Y=>Rdisp, display=>display);
93
94
95  end archPrincipal;
```

Figura 12 – Código da função Main - Parte 3

```
library ieee;
use ieee.std_logic_1164.all;

entity conversondisplay is -- entidade do conversor display
port(
  CTRL      : in std_logic_vector(1 downto 0); -- CTRL tem 2 bits
  Y      : in std_logic_vector(3 downto 0); -- Y tem 4 bits (resultado da operação)
  DISPLAY               : out std_logic_vector(6 downto 0)); -- Display tem 7 bits (Saída de 7 segmentos)
end conversondisplay;

architecture archConversor of conversondisplay is -- arquitetura do conversor

signal display2 : std_logic_vector(6 downto 0); -- sinal para auxiliar o resultado final
begin
        WITH Y SELECT -- nesse WITH SELECT usamos o sinal para "receber" e converter o Y para 7 segmentos

    display2 <=  not("1111110") WHEN "0000", -- 0
                 not("0110000") WHEN "0001", -- 1
                 not("1101101") WHEN "0010", -- 2
                 not("1111001") WHEN "0011", -- 3
                 not("0110011") WHEN "0100", -- 4
                 not("1011011") WHEN "0101", -- 5
                 not("1011111") WHEN "0110", -- 6
                 not("1110000") WHEN "0111", -- 7
                 not("1111111") WHEN "1000", -- 8
                 not("1111011") WHEN "1001", -- 9
                                    not("1110111") WHEN "1010", -- A
                                    not("0011111") WHEN "1011", -- B
                                    not("1001110") WHEN "1100", -- C
                                    not("0111101") WHEN "1101", -- D
                                    not("1001111") WHEN "1110", -- E
                                    not("1000111") WHEN "1111", -- F
                                    not("0000000") WHEN OTHERS;

               WITH CTRL SELECT -- nesse segundo WITH SELECT trabalhamos com a variável controle e a nossa saída DISPLAY

                        DISPLAY <= display2              WHEN "00", -- quando CTRL for 00, DISPLAY receberá display2
                                                  not("1001001") WHEN "01", -- quando CTRL for 01, DISPLAY receberá ERRO
                                                  not("0000000") WHEN "10", -- quando CTRL for 10, DISPLAY receberá DESLIGADO
                                                  not("0000000") WHEN OTHERS;

end archConversor;
```

Figura 13 – Código do Conversor Display

4 Conclusão

Com base no que foi visto em sala de aula, e repassado pelo professor, conseguimos concluir este projeto no qual é resposável pela nota da primeira unidade.

Encontramos algumas dificuldades ao decorrer do projeto, porém todas foram solucionadas durante a execução do mesmo, fazendo com que chegassemos no final do o trabalho executando todas suas funcionalidades.

No demais, deixamos aqui nossos agradecimentos ao professor que não mediu esforços para auxiliar na criação deste trabalho.