Universidade Federal do Rio de Janeiro

UFRJ

Sistemas Digitais – EEL 480

Projeto de Unidade Lógica Aritmética

Maria Fernanda Debatin de Magalhães

Pedro Francisco Ignácio Achcar

Thauã Melo Gomes

Professor: Roberto Pacheco

:

Rio de Janeiro
Julho 2023

1. INTRODUÇÃO

Este trabalho tem como objetivo a proposta de desenvolvimento de uma Unidade Lógica Aritmética (ULA) com algumas de suas funções. Ao longo do texto, o documento descreve o projeto de uma ULA, seu funcionamento e a implementação do circuito digital que a compõe, realizando operações lógicas e aritméticas sobre números binários.

A estrutura interna de um computador pode ser dividida em quatro unidades lógicas: a unidade central de processamento (ou processador), a memória, os circuitos de entrada e saída (ou E/S) e os programas (software).

Dentro da unidade central de processamento encontra-se a ULA, ela é responsável pelas operações lógicas e aritméticas básicas num processador. Ela é formada com uma rede combinacional, que implementa uma função de suas entradas com base em operações lógicas ou aritméticas.

As operações aritméticas tipicamente realizadas por uma ULA são adição, subtração, incremento e decremento. Dentre as operações lógicas citam-se o E, o OU, identidade e complemento. A Figura 1, apresenta um diagrama funcional de um computador digital.

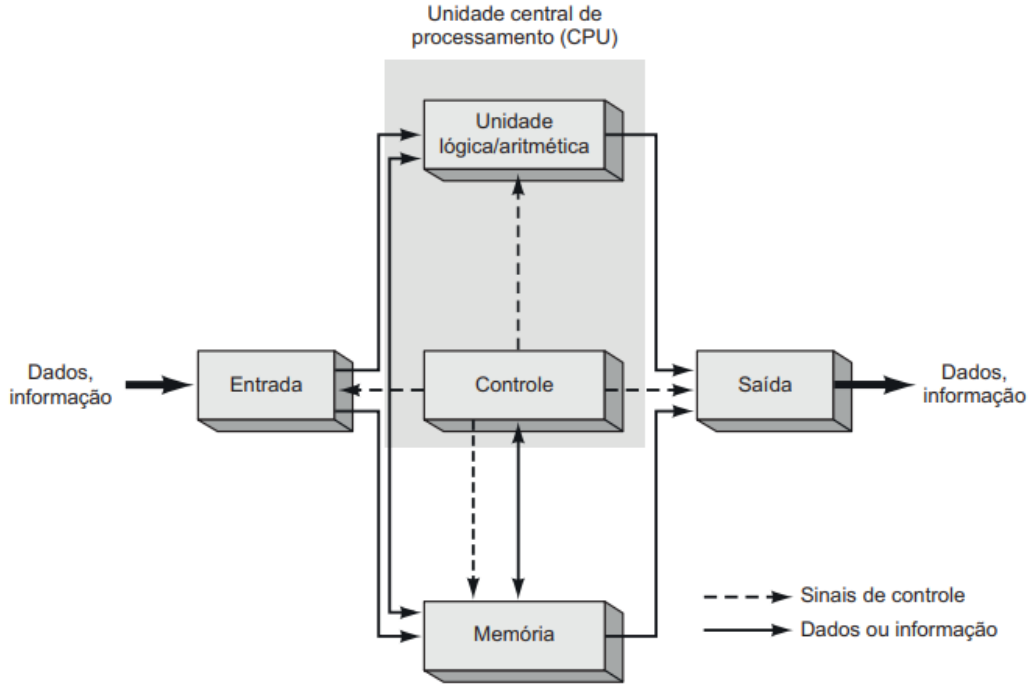


Figura 1: Diagrama funcional de um computador digital (Fonte: Tocci [1])

A ULA proposta neste trabalho é capaz de realizar operações lógicas e aritméticas com números de 4 bits. Estas operações são selecionadas através de três entradas de controle, seus os resultados são mostrados a posteriori em displays de 7 segmentos. Além

disso, possui flags de sinalização, que indicam os casos de overflow, número zero, número negativo e existência de carry out. Cabe ressaltar que a ULA foi projetada utilizando-se portas lógicas primitivas.

2. CONCEPÇÃO E IMPLEMENTAÇÃO DO PROJETO

A ULA desenvolvida neste trabalho realiza operações com números de 4 bits, sendo 4 operações aritméticas, a saber: soma, subtração em complemento de 2, incremento +1 e troca de sinal, e quatro operações lógicas, como: troca de sinal, and bit a bit, or bit a bit e comparação de magnitude, com indicação de maior, menor e igual. Tais operações são selecionadas através de três entradas de controle denominadas por: f_0, f_1 e f_2.
A Figura 2 apresenta o diagrama em blocos da ULA projetada.

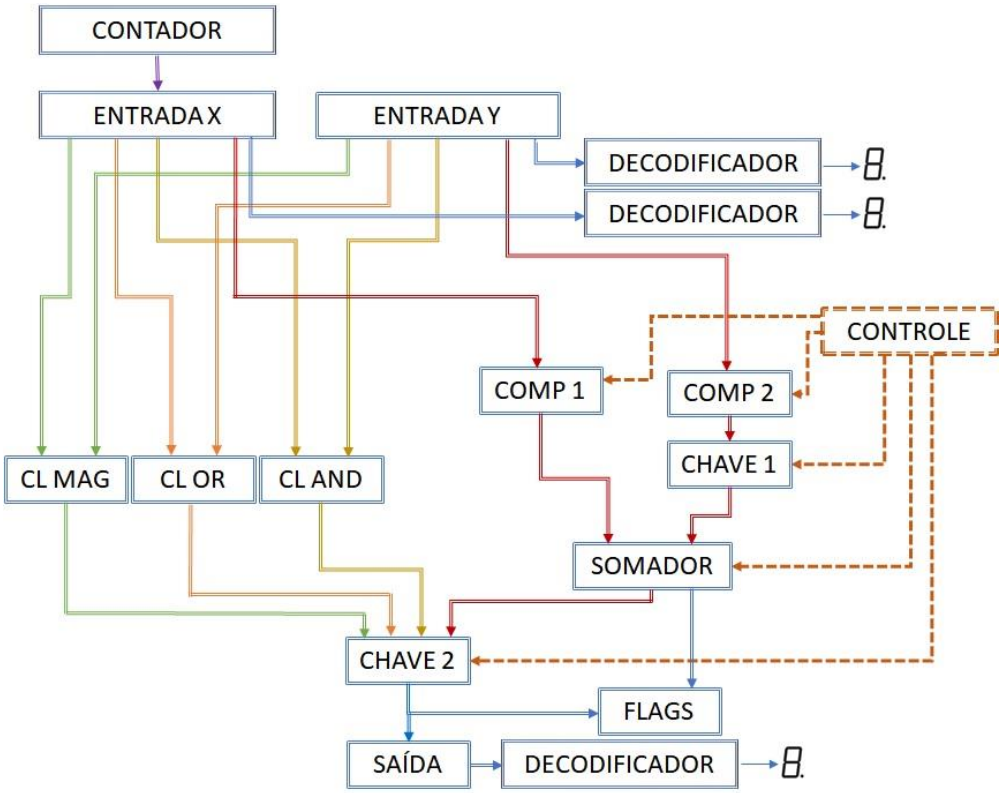


Figura 2: Diagrama em Blocos da ULA (Fonte: Autor)

O bloco ENTRADA X e ENTRADA Y correspondem, respectivamente às entradas dos números de 4 bits, os blocos COMP 1 e COMP 2 são compostos por circuitos lógicos para a operação de complemento, a CHAVE 1 comuta suas entradas para deixar fluir a entrada Y ou zeros. O módulo SOMADOR é o responsável pela operação de soma,

o CL MAG é composto pelo circuito lógico de definição da magnitude, o CL OR é o circuito lógico de OR bit a bit, o CL AND é o circuito lógico de AND bit a bit, o CONTROLE é possui o circuito lógico para geração dos sinais de controle da ULA, a CHAVE 2 é usada para selecionar a saída de acordo com os sinais de controle da entrada da ULA e por fim, o módulo SAÍDA apresenta os resultados da operação selecionada. Acrescenta-se o módulo FLAGS para indicação dos flags da Unidade Lógica Aritmética.

2.1. MÓDULO CONTROLE

A unidade de controle (UC) é responsável pelo movimento dos dados dentro da ULA, ela se encarrega de fazer com que todas as partes da CPU recebam os dados corretos e executem as instruções corretas em todos os momentos. Ela também é responsável pela interpretação de um comando e acionar a ULA para executá-lo.

Para uma mais simples implementação, que certamente não é a mais econômica, adota-se individualmente a execução de uma operação para posteriormente tal operação ser selecionada por um multiplexador.

A Tabela 1 apresenta a codificação das entradas de controle relacionadas as operações a serem efetuadas.

Tabela 1: Sinais de controle da Unidade de Controle

Entrada			Saída						Operação
f_2	f_1	f_0	K_X	K_Y	C_0	KM_0	KM_1	KM_2	
0	0	0	0	0	0	0	0	0	Soma (X + Y)
0	0	1	0	1	1	0	0	0	Subtração (X - Y)
0	1	0	1	x	1	1	0	0	Troca de Sinal de X
0	1	1	0	x	1	1	0	0	Incremento +1 em X
1	0	0	1	x	0	1	0	0	Complemento a 1 de X
1	0	1	x	x	x	x	0	1	AND bit a bit
1	1	0	x	x	x	x	1	0	OR bit a bit
1	1	1	x	x	x	x	1	1	Comparador de Magnitude

- Controle do módulo COMP1

$K_X = f_2' f_1 f_0' + f_2 f_1' f_0' = f_0'(f_2 \oplus f_1)$
- Controle do módulo COMP2

$K_Y = f_2' f_1' f_0$
- Controle do módulo SOMADOR

$C_0 = f_2' (f_1 + f_0)$
- Controle do módulo CHAVE 1

$KM_0 = f_2 \oplus f_1$
- Controles do módulo CHAVE 2

$KM_1 = f_2 f_1$

$KM_2 = f_2 f_0$

A Figura 3 apresenta a configuração interna da Unidade de Controle.

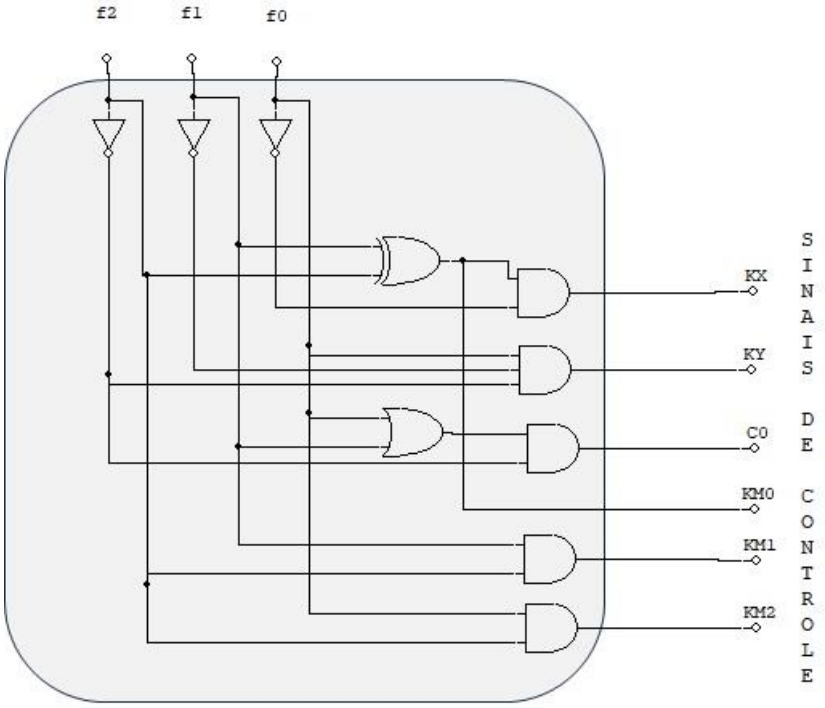


Figura 3: Configuração Interna da Unidade de Controle (Fonte: Autor)

2.1.1. CÓDIGO DO BLOCO CONTROLE

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

entity controle is
        port (f : in std_logic_vector (2 downto 0);
        KM0, KM1, KM2, C0, KX, KY: out std_logic ) ;
end controle;

architecture unidadecontrole of controle is
begin
        -- funcionamento da arquitetura do módulo de controle
         KM0 <= f(2) xor f(1);
         KM1 <= f(2) and f(1);
         KM2 <= f(2) and f(0);
         C0 <= not(f(2)) and (f(1) or f(0));
         KX <= not(f(0)) and (f(2) xor f(1));
         KY <= not(f(2)) and not(f(1)) and f(0);
end unidadecontrole;
```

2.2. MÓDULOS COMP 1 E COMP 2 (COMPLEMENTO A UM)

O circuito para realização de complemento a 1 é composto de portas XOR, uma para cada bit, onde quatro entradas correspondem aos bits de dados (X ou Y) e uma outra é um sinal de comando (K_X ou K_Y), que comandam a função de complemento a um. A Figura 4 apresenta o circuito desses módulos.

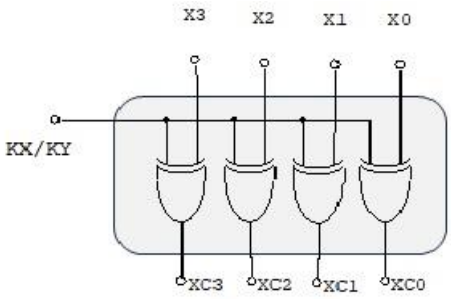


Figura 04 – Circuito de Complemento a Um (Fonte: Autor)

Onde:

$XC_0 = X_0 \oplus K_X(K_Y)$
$XC_1 = X_1 \oplus K_X(K_Y)$
$XC_2 = X_2 \oplus K_X(K_Y)$
$XC_3 = X_3 \oplus K_X(K_Y)$

2.2.1. CÓDIGO DOS BLOCOS COMP1 E COMP2

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

entity comp is
-- declaração das portas de entrada
port (      X : in std_logic_vector (3 downto 0);
            sel: in std_logic;
-- declaração das portas de saída
            Y: out std_logic_vector (3 downto 0)) ;
end comp;

architecture clcomp of comp is
begin
            -- funcionamento da arquitetura do módulo comp (complemento a um)
            -- as saídas são equivalentes ao 'xor' com o sinal de controle KX ou KY
             Y(0) <= X(0) xor sel;
             Y(1) <= X(1) xor sel;
             Y(2) <= X(2) xor sel;
             Y(3) <= X(3) xor sel;
end clcomp;
```

2.3. MÓDULO CL AND

O circuito para realização de AND bit a bit é composto por quatro portas AND, uma para cada par de bits, correspondentes as entradas de dados (X e Y). A Figura 5 apresenta o circuito desse módulo.

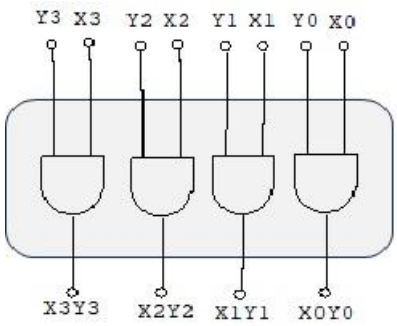


Figura 05 – Circuito de AND bit a bit (Fonte: Autor)

2.3.1. CÓDIGO DO BLOCO CL AND

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

entity cland is
-- declaração das portas de entrada
   port (      A,B: in std_logic_vector (3 downto 0);
-- declaração das portas de saída
            Y: out std_logic_vector (3 downto 0) ) ;
```

```
end cland;

architecture andbit of cland is
begin
   -- funcionamento da arquitetura do módulo cl and
   -- as saídas são equivalentes ao 'and' bit a bit com os sinais de entrada
    Y(0) <= A(0) and B(0);
    Y(1) <= A(1) and B(1);
    Y(2) <= A(2) and B(2);
    Y(3) <= A(3) and B(3);
end andbit;
```

2.4. ;MÓDULO CL OR

O circuito para realização de OR bit a bit é composto por quatro portas OR, uma para cada par de bits, correspondentes as entradas de dados (X e Y). A Figura 6 apresenta o circuito desse módulo.

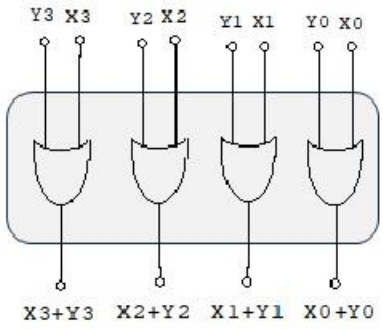


Figura 6 – Circuito de OR bit a bit (Fonte: Autor)

2.4.1. CÓDIGO DO BLOCO CL OR

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

entity clor is
-- declaração das portas de entrada
   port (      A,B: in std_logic_vector (3 downto 0);
-- declaração das portas de saída
                Y: out std_logic_vector (3 downto 0) ) ;
end clor;

architecture orbit of clor is
   begin
   -- funcionamento da arquitetura do módulo cl or
   -- as saídas são equivalentes ao 'or' bit a bit com os sinais de entrada
    Y(0) <= A(0) or B(0);
    Y(1) <= A(1) or B(1);
    Y(2) <= A(2) or B(2);
    Y(3) <= A(3) or B(3);
end orbit;
```

2.5. MÓDULO CL MAG

O circuito para realização da comparação de magnitude de um número de 4 bits é apresentado na Figura 7, onde:

- O sinal de saída X > Y é descrito por:

$(X > Y) = (X_3Y_3') + (X_3 \oplus Y_3)' (X_2Y_2') + (X_3 \oplus Y_3)' (X_2 \oplus Y_2)' (X_1Y_1') + (X_3 \oplus Y_3)' (X_2 \oplus Y_2)' (X_1 \oplus Y_1)' (X_0Y_0')$

- O sinal de saída X = Y é descrito por:

$(X = Y) = (X_3 \oplus Y_3)' (X_2 \oplus Y_2)' (X_1 \oplus Y_1)' (X_0 \oplus Y_0)'$

- O sinal de saída X < Y é descrito por:

$(X < Y) = (X > Y)' (X = Y)'$

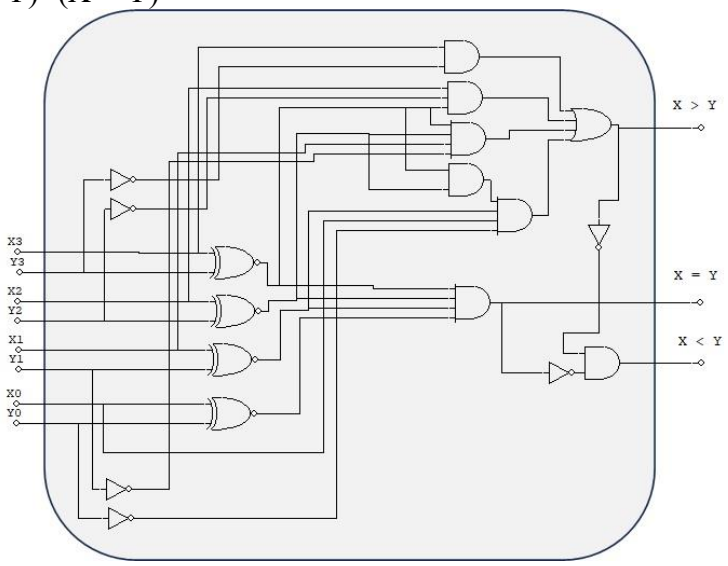


Figura 7 – Circuito do Comparador de Magnitude de número de 4 bits (Fonte: Autor)

O resultado desse módulo é apresentado na saída com a seguinte codificação:

0001 – A maior que B

0010 – A igual a B

0100 – A menor que B

2.5.1. CÓDIGO DO BLOCO CL MAG

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

entity clmag is
   -- declaração das portas de entrada
   port (      A,B: in std_logic_vector (3 downto 0);
   -- declaração das portas de saída
               magnitude: out std_logic_vector (3 downto 0) ) ;
end clmag;

architecture compmag of clmag is
signal maior, igual:  std_logic;
begin
```

```
-- funcionamento da arquitetura do módulo cl mag

        maior <= (A(3) and not(B(3))) or ((not(A(3) xor B(3))) and (A(2) and not(B(2)))) or ((
not(A(3) xor B(3))) and ( not(A(2) xor B(2))) and (A(1) and not(B(1)))) or (( not(A(3) xor B(3))) and
( not(A(2) xor B(2))) and ( not(A(1) xor B(1))) and (A(0) and not(B(0))));
        igual <= (not(A(3) xor B(3))) and (not(A(2) xor B(2))) and (not(A(1) xor B(1))) and
(not(A(0) xor B(0)));
        magnitude(0) <= maior;
        magnitude(1) <= igual;
        magnitude(2) <= not(maior) and not(igual);
        magnitude(3) <= '0';
end compmag;
```

2.6. MÓDULO CHAVE 1

O circuito desse módulo possui 4 multiplexadores 2:1 com uma única chave seletora, KM_0. A entrada zero está ligada diretamente a saída do módulo COMP 2 e a entrada 1 do mux está ligada ao nível lógico zero, para que as operações que envolvem apenas a entrada X possam ser efetuadas. A Figura 8 apresenta o circuito desse módulo.

Para cada multiplexador 2:1 tem-se como relação entrada-saída.

$Y_i = (KM_0' A) + (KM_0 B)$

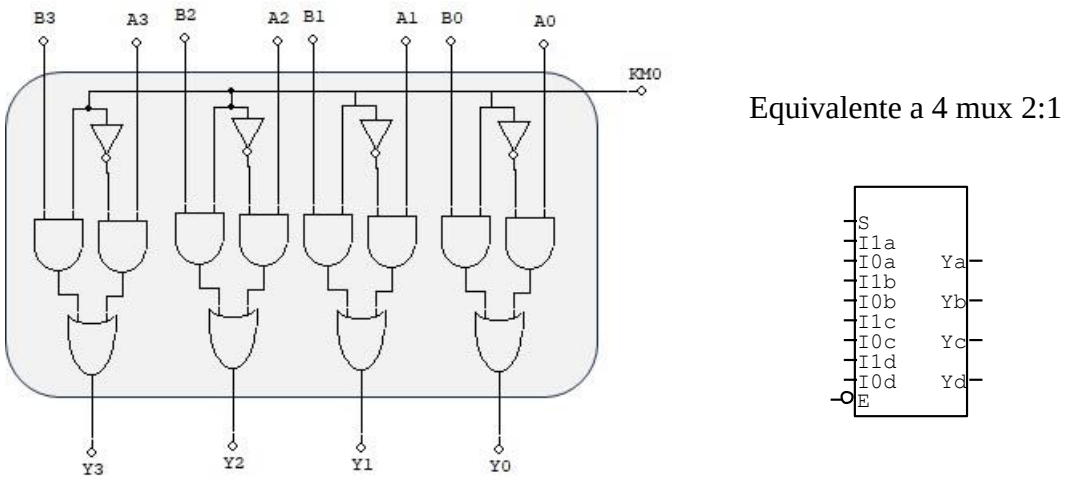


Figura 8 – Circuito do Módulo CHAVE 1 (Fonte: Autor)

2.6.1. CÓDIGO DO BLOCO CHAVE 1

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

entity mux2_1 is
        Port ( SEL : in  STD_LOGIC;
               A   : in  STD_LOGIC_VECTOR (3 downto 0);
               B   : in  STD_LOGIC_VECTOR (3 downto 0);
               Y   : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end mux2_1;

architecture mux421 of mux2_1 is
begin
        Y <= A when (SEL = '0') else B;
end mux421;
```

2.7. MÓDULO CHAVE 2

O circuito desse módulo possui 4 multiplexadores 4:1 com duas chaves seletoras, KM_1 e KM_2. Cada multiplexador tem suas entradas ligadas ao bit *i* das saídas dos módulos SOMADOR, CL AND, CL OR e CL MAG, dessa forma as saídas dos multiplexadores constroem a resposta da ULA, de acordo com a seleção da operação. A Figura 9 apresenta o circuito desse módulo. Por exemplo, se ($KM_2 = KM_1 = 0$, as saídas dos multiplexadores fornecerão os bits oriundos do somador.

Para cada multiplexador 4:1 tem-se como relação entrada-saída.

$D_i = (KM_2' \; KM_1' \; I_0) + (KM_2' \; KM_1 \; I_1) + (KM_2 \; KM_1' \; I_2) + (KM_2 \; KM_1 \; I_3)$

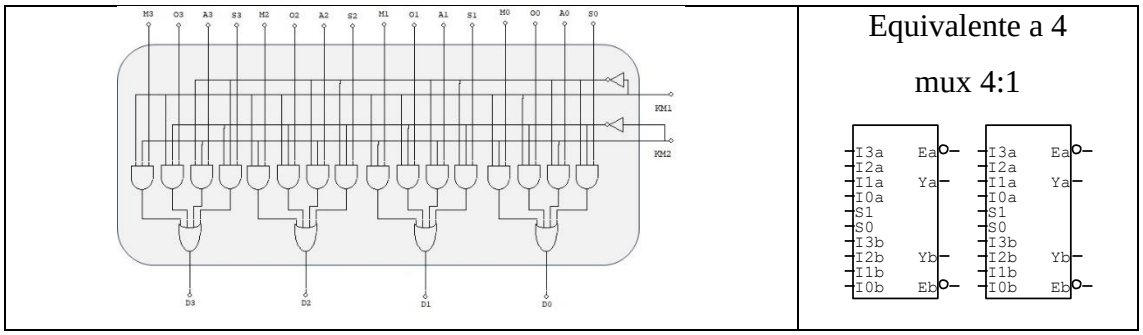


Figura 9 – Circuito do Módulo CHAVE 2 (Fonte: Autor)

2.7.1. CÓDIGO DO BLOCO CHAVE 2

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

entity mux4_1 is
   Port ( S1,S0 : in  STD_LOGIC;
               A   : in  STD_LOGIC_VECTOR (3 downto 0);
               B   : in  STD_LOGIC_VECTOR (3 downto 0);
               C: in  STD_LOGIC_VECTOR (3 downto 0);
               D: in  STD_LOGIC_VECTOR (3 downto 0);
               S   : out STD_LOGIC_VECTOR (3 downto 0));
end mux4_1;

architecture mux441 of mux4_1 is
   begin
   process (A,B,C,D,S0,S1) is
   begin
                if (S0 ='0' and S1 = '0') then
                   S <= A;
                elsif (S0 ='1' and S1 = '0') then
                   S <= B;
```

```
        elsif (S0 ='0' and S1 = '1') then
            S <= C;
        else
            S <= D;
        end if;

    end process;
end mux441;
```

2.8. MÓDULO SOMADOR

O módulo SOMADOR é composto por quatro somadores completos, possui uma entrada de carry (C_0) e as entradas de dados. Em sua saída encontram-se os bits de carry dos último e penúltimo bits, para posterior construção de flag de overflow. A Figura 10 apresenta o circuito desse módulo.

Para cada somador completo tem-se como relação entrada-saída.

$S_i = X_i \oplus Y_i \oplus C_{in}$

$C_{out} = (X_i Y_i) + (X_i C_{in}) + (Y_i C_{in})$

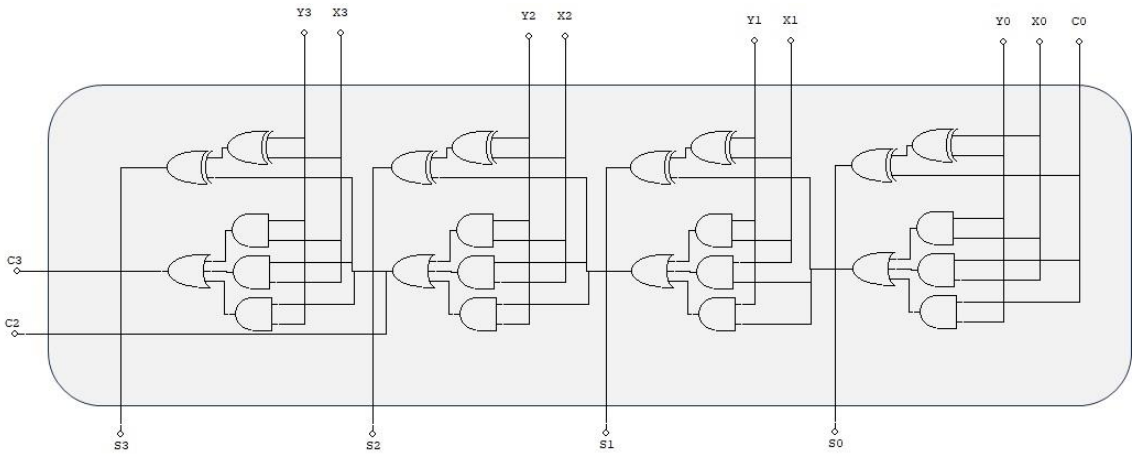


Figura 10 – Circuito do Módulo SOMADOR (Fonte: Autor)

2.8.1. CÓDIGO DO MÓDULO SOMADOR

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

entity sum4bits is
port (
-- declaração das variáveis de entrada
    X , Y: in std_logic_vector (3 downto 0);
    C0: in std_logic;
    -- declaração das variáveis de saída
    S : out std_logic_vector (3 downto 0);
    Cn,Cn_1: out std_logic);
end sum4bits;

architecture somador of sum4bits is
```

```
-- sinais de carry out intermediários
    signal C1,C2,C3: std_logic;

begin
    -- primeiro somador
    S(0)<=X(0) XOR Y(0) XOR C0;
    C1<= (X(0) AND Y(0)) OR (X(0) AND C0) OR (Y(0) AND C0);
    --segundo somador
    S(1)<=X(1) XOR Y(1) XOR C1;
    C2<= (X(1) AND Y(1)) OR (X(1) AND C1) OR (Y(1) AND C1);
    --terceiro somador
    S(2)<=X(2) XOR Y(2) XOR C2;
    C3<= (X(2) AND Y(2)) OR (X(2) AND C2) OR (Y(2) AND C2);
    --quarto somador
    S(3)<=X(3) XOR Y(3) XOR C3;
    Cn<= (X(3) AND Y(3)) OR (X(3) AND C3) OR (Y(3) AND C3);
    --envio do terceiro carry para o pino de saída
    Cn_1<=C3;

end somador;
```

2.9. MÓDULO FLAGS

A ULA proposta possui flags de sinalização, que indicam os casos de overflow, resultado igual a zero, número negativo e existência de carry out. A Figura 11 apresenta o circuito desse módulo.

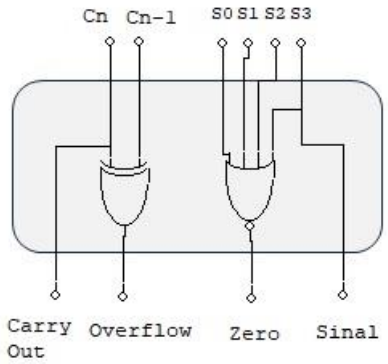


Figura 11 – Circuito do Módulo FLAGS (Fonte: Autor)

2.9.1. CÓDIGO MÓDULO FLAGS

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

entity flags is
-- declaração das portas de entrada
            port (      S : in std_logic_vector (3 downto 0);
                        Cn_1,Cn: in std_logic;
-- declaração das portas de saída
                        zeros,overflow,sinal,carryout: out std_logic ) ;
end flags;
```

```
architecture sinalizador of flags is
begin
        -- funcionamento da arquitetura do módulo flags

         zeros <= not(S(0) or S(1) or S(2) or S(3));
         overflow <= Cn_1 xor Cn;
         sinal <= S(3);
         carryout <= Cn;
end sinalizador;
```

2.10. MÓDULO CONTADOR

O módulo contador foi implementado diretamente na linguagem VHDL.

A contagem de 0 a 15 é acionada pela detecção da borda de descida do sinal da entrada *clk*.

O sinal de entrada *rst* zera o contador independentemente do valor do contador, e é acionado pela borda de subida deste sinal.

2.10.1. CÓDIGO MÓDULO CONTADOR

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

entity contador is

   port(
        clk, rst : in std_logic;
        X                       : out std_logic_vector(3 downto 0));
end contador;

architecture rtl of contador is
   signal q_temp : unsigned (3 downto 0);
begin
   contagem : process(clk, rst)
   begin
   -- reset assincrono
        if rst = '1' then
                q_temp <= (others => '0');

        elsif falling_edge(clk) then
                q_temp <= q_temp + 1;
        end if;
   end process;
   X <= std_logic_vector(q_temp);
end rtl;
```

2.11. MÓDULO DECODIFICADOR

O módulo decodificador foi implementado diretamente na linguagem VHDL. Ele tem como objetivo disponibilizar os valores dos operandos X, Y e os resultados das operações

no formato de 7 segmentos, ou seja, sua entrada é um código binário de 4 bits e sua saída é um código compatível com um display de 7 segmentos. Adota-se a seguinte convenção.

2.11.1. CÓDIGO MÓDULO DECODIFICADOR

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

entity decoder is
        port(
        A                       : in std_logic_vector(3 downto 0);
        A_dec7 : out std_logic_vector(6 downto 0));
end decoder;

architecture dec7 of decoder is
begin
  process(A)
  begin
  case A is
        when "0000" => A_dec7 <= not "1111110";
        when "0001" => A_dec7 <= not "0110000";
        when "0010" => A_dec7 <= not "1101101";
        when "0011" => A_dec7 <= not "1111001";
        when "0100" => A_dec7 <= not "0110011";
        when "0101" => A_dec7 <= not "1011011";
        when "0110" => A_dec7 <= not "1011111";
        when "0111" => A_dec7 <= not "1110000";
        when "1000" => A_dec7 <= not "1111111";
        when "1001" => A_dec7 <= not "1111011";
        when "1010" => A_dec7 <= not "1110111";
        when "1011" => A_dec7 <= not "0011111";
        when "1100" => A_dec7 <= not "1001110";
        when "1101" => A_dec7 <= not "0111101";
        when "1110" => A_dec7 <= not "1001111";
        when "1111" => A_dec7 <= not "1000111";


        when others => A_dec7 <= "0000000";
  end case;
  end process;

end dec7;
```

2.12. ULA

A Figura 11 apresenta o diagrama esquemático da unidade logica aritmética.

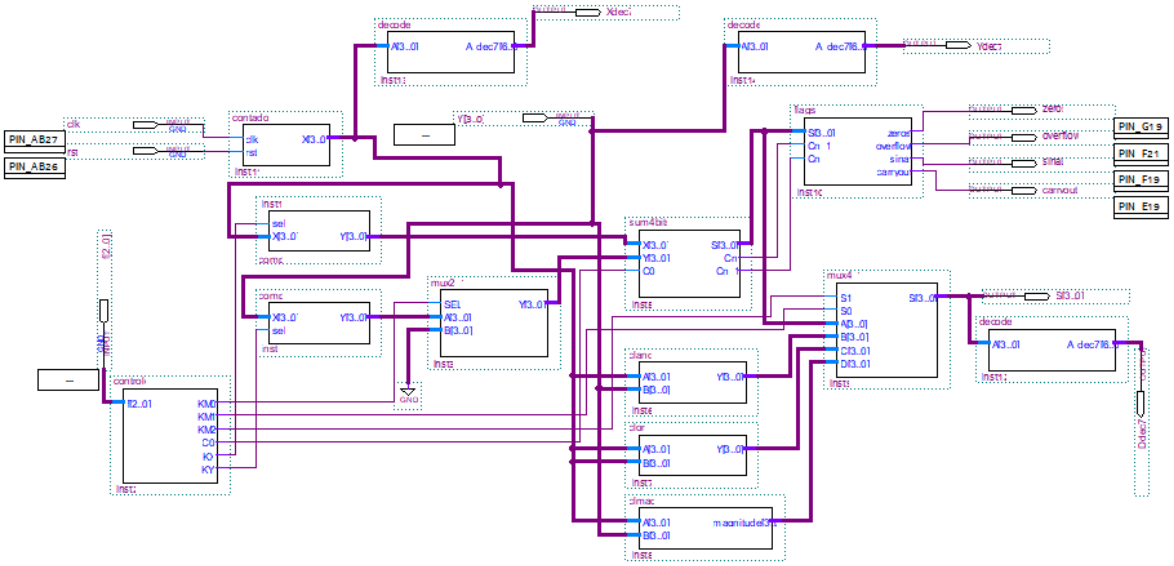


Figura 11 – Circuito da ULA (Fonte: Autor)

2.12.1. CÓDIGO DA UNIDADE LOGICA ARITMÉTICA

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

entity ULAV2 is
-- declaração das portas de entrada
        port (      clk, rst: in std_logic:='0'; Y : in std_logic_vector (3 downto 0);
                    f: in std_logic_vector (2 downto 0);
        -- declaração das portas de saída
        Res : out std_logic_vector (3 downto 0); -- resultado
        zeros, overflow, carryout, sinal: out std_logic ;
        X_dec7: out std_logic_vector (6 downto 0);-- saída X codificada para 7 segmentos
        Y_dec7: out std_logic_vector (6 downto 0);-- saída Y codificada para 7 segmentos
        D_dec7: out std_logic_vector (6 downto 0) -- saída D codificada para 7 segmentos
        ) ;
        end ULAV2;


architecture ULA_SD of ULAV2 is
-------------------------------------------------------------
-- DECLARAÇÃO DOS COMPONENTES DA ULA
-------------------------------------------------------------
-----------------------------------------
        component decoder is
                        port(
                        A           : in std_logic_vector(3 downto 0);
                        A_dec7      : out std_logic_vector(6 downto 0));
        end component decoder;

-----------------------------------------
        component contador is
        -- Declaração do clk e rst de entrada e a saída X
                port(
                        clk, rst    : in std_logic;
                        X           : out std_logic_vector(3 downto 0));
                end component contador;
```

```
-------------------------------------------
        component controle is
                port (f : in std_logic_vector (2 downto 0);
                                KM0, KM1, KM2, C0, KX, KY: out std_logic ) ;
        end component controle;
-------------------------------------------
        component clor is
        -- declaração das portas de entrada
                port (A,B: in std_logic_vector (3 downto 0);
        -- declaração das portas de saída
                        Y: out std_logic_vector (3 downto 0) ) ;
        end component clor;
-------------------------------------------
        component clmag is
                -- declaração das portas de entrada
                port (A,B: in std_logic_vector (3 downto 0);
                -- declaração das portas de saída
                        magnitude: out std_logic_vector (3 downto 0) ) ;
        end component clmag;

-------------------------------------------------
        component cland is
                -- declaração das portas de entrada
                port (A,B: in std_logic_vector (3 downto 0);
                -- declaração das portas de saída
                                Y: out std_logic_vector (3 downto 0) ) ;
        end component cland;
-------------------------------------------------
        component mux2_1 is
                Port ( SEL : in  STD_LOGIC;
                                A   : in  STD_LOGIC_VECTOR (3 downto 0);
                                B   : in  STD_LOGIC_VECTOR (3 downto 0);
                                Y   : out STD_LOGIC_VECTOR (3 downto 0));
        end component mux2_1;

-------------------------------------------------
        component mux4_1 is
                Port ( S1,S0 : in  STD_LOGIC;
                                A   : in  STD_LOGIC_VECTOR (3 downto 0);
                                B   : in  STD_LOGIC_VECTOR (3 downto 0);
                                C           : in  STD_LOGIC_VECTOR (3 downto 0);
                                D           : in  STD_LOGIC_VECTOR (3 downto 0);
                                S   : out STD_LOGIC_VECTOR (3 downto 0));
        end component mux4_1;
-------------------------------------------------
        component flags is
                -- declaração das portas de entrada
                port (S : in std_logic_vector (3 downto 0);
                                Cn_1,Cn: in std_logic;
                -- declaração das portas de saída
                                zeros,overflow,sinal,carryout: out std_logic ) ;
        end component flags;
-------------------------------------------------
        component sum4bits is
                port(
                -- declaração das variáveis de entrada
                        X , Y: in std_logic_vector (3 downto 0);
                        C0: in std_logic;
                        -- declaração das variáveis de saída
                        S : out std_logic_vector (3 downto 0);
                        Cn,Cn_1: out std_logic);
```

```
        end component sum4bits;
-----------------------------------------------
        component comp is
                -- declaração das portas de entrada
                port (X : in std_logic_vector (3 downto 0);
                                sel: in std_logic;
                -- declaração das portas de saída
                                Y: out std_logic_vector (3 downto 0)) ;
        end component comp;

-------------------------------------------------------------
-- DECLARAÇÃO DOS SINAIS INTERNOS
-------------------------------------------------------------

-- sinais de saída do módulo controle
-- recebem f da ula e liberam KM0,KM1,KM2,C0,KX E KY
--port (F, KM0, KM1, KM2, C0, KX, KY) ;

        signal KM0, KM1, KM2, C0, KX, KY : std_logic;

-- sinais de saida do circuito de complemento a  um ligados a entrada X
-- recebem X(0..3) e liberam CPM1_(0..3)
--port (X, sel,Y) ;

        signal CPM1: std_logic_vector (3 downto 0);

-- sinais de saida do circuito de complemento a  um ligados a entrada Y
-- recebem as entradas Y(0..3) e liberam CPM2_(0..3)
        signal CPM2 : std_logic_vector (3 downto 0);

-- sinais da chave1
-- recebem -CPM2_(0..3) e KM0 e liberam CH1_(0..3), as entradas B(0..3) são iguais a zero
-- port(A,B,sel,Y);
        signal CH1 : std_logic_vector (3 downto 0);


-- somador
--port(X,Y,C0,S,Cn,Cn_1);
-- Recebem CPM1_0, CPM1_1 , CPM1_2, CPM1_3, CPM2_0, CPM2_1 , CPM2_2, CPM2_3, C0
-- e liberam SUM, CN, CN_1
        signal      SUM : std_logic_vector (3 downto 0);
        signal CN, CN_1 : std_logic;

-- módulo and bit a bit
-- port(A,B,Y)
-- recebem X(0..3) e Y(0..3) da entrada e liberam land0,land1,land2,land3
        signal land : std_logic_vector (3 downto 0);

-- módulo or bit a bit
-- port(A,B,Y)
-- recebem X(0..3) e Y(0..3) da entrada e liberam lor0,lor1,lor2,lor3
        signal lor : std_logic_vector (3 downto 0);


-- módulo magnitude
-- port(A,B,magnitude)
-- recebem X(0..3) e Y(0..3) da entrada e liberam magnitude
        signal mag : std_logic_vector (3 downto 0);

-- módulo dos flags
-- recebem SUM(0..3), CN e CN_1 e liberam zeros,overflow,sinal,carryout
```

```
--          port (S0,S1,S2,S3,Cn_1,Cn, zeros,overflow,sinal,carryout) ;
--          zeros,overflow,sinal,carryout já são as sáídas da ula


-- módulo chave2
-- port(A,S,M, O, KM1, KM2, D);
-- D(0..3) entra no decodificador para 7 segmentos
-- e repassa o valor para res para facilitar a simulação
        signal D : std_logic_vector (3 downto 0);

-------------------------------------------------

begin

-------------------------------------------------------------------------------------------
-- DECLARAÇÃO DOS MÓDULOS DA ULA E SEUS MAPEAMENTOS
-------------------------------------------------------------------------------------------


        control: Controle
                -- faz a connexão da entrada da ula => entrada do controle
                port map(f, KM0, KM1, KM2, C0, KX, KY);

        comp1: comp
        -- faz a connexão da entrada da ula => entrada do complemento
                port map(X, KX, CPM1);


        comp2: comp
                port map(Y,KY, CPM2);

        chave1: mux2_1
                port map(KM0, CPM2,"0000",CH1);


        somador: sum4bits
                port map(CPM1,CH1, C0,SUM, CN, CN_1);

        cand: cland
                port map(X,Y,land) ;

        cor: clor
                port map(X,Y,lor) ;

        cmag: clmag
                        port map(X,Y,mag);

        flag: flags
                        port map(SUM, CN_1, CN, zeros,overflow,sinal,carryout);

        chave2: mux4_1
                        port map (KM1, KM2, SUM, land, lor ,mag, D);

        conta: contador
                        port map(clk, rst, X);

        saidaX: decoder -- saída para o display
                        port map(X, X_dec7);

        saidaY: decoder -- saída para o display
                        port map(Y, Y_dec7);
```

```
    saidaD: decoder -- saída para o display
                    port map(D, D_dec7);

    Res <= D; -- uma saída em binário de 4 bits

end ULA_SD;
```

2.13. COMPONENTES UTILIZADOS

A Tabela 2 apresenta o número de portas lógicas utilizadas em cada módulo da ULA. O resultado desta Tabela foi computado considerando a composição primitiva dos multiplexadores.

Tabela 2: Número de portas lógicas da ULA

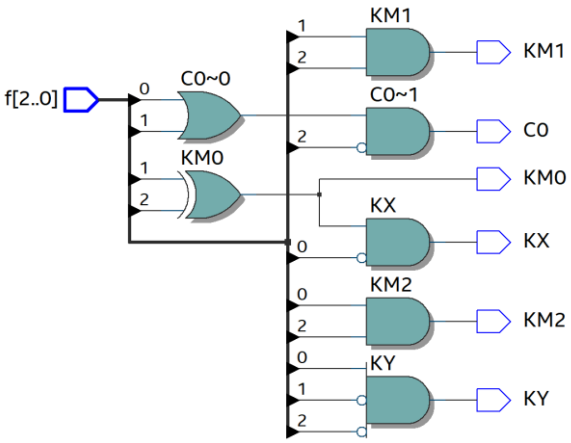
MÓDULO	PORTAS				
	AND (*)	NOT	OR (*)	XOR (*)	NOR (*)
COMP 1	-	-	-	4 (2)	-
COMP 2	-	-	-	4 (2)	-
CHAVE 1	8 (2)	4	4 (2)	-	-
CHAVE 2	16 (3)	2	4 (4)	-	-
CL OR	-	-	4 (2)	-	-
CL AND	4 (2)	-	-	-	-
FLAGS	-	-	1 (2)	-	1 (4)
SOMADOR	12 (2)	-	4 (2)	8 (2)	-
CONTROLE	4 (2) 1 (3)	3	1 (2)	1 (2)	-
CL MAG	3 (2) 1 (3) 3 (4)	6	1 (4)	4 (2)	-

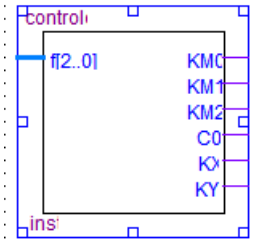
(*) Número de entradas

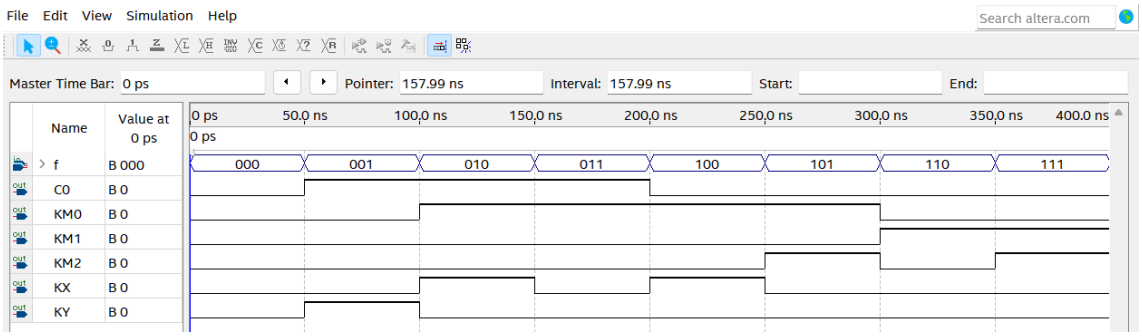
3. ARQUITETURA IMPLEMENTADA E RESULTADOS EXPERIMENTAIS

Nesta seção são apresentados os resultados da simulação de cada módulo, assim como os resultados das simulações da unidade lógica aritmética.

3.1. MÓDULO DE CONTROLE

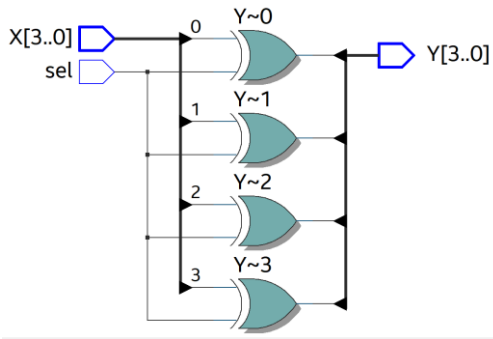


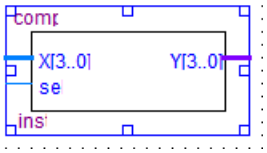


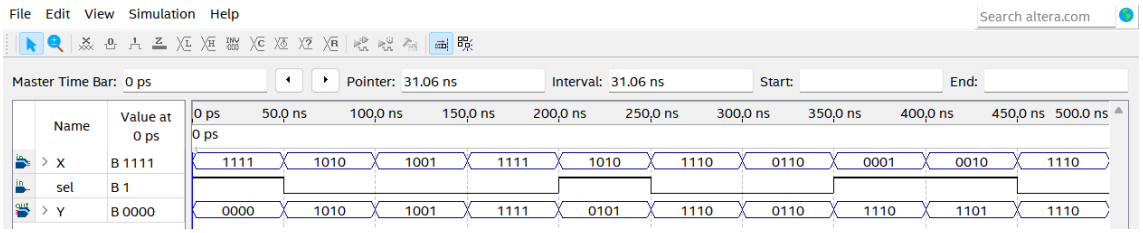


Esta simulação apresenta a relação entre as entradas de controle f_0, f_1 e f_2 e os sinais de controle da ULA. São testados os oito possíveis ajustes e verificados com as saídas, como descrito na Tabela 1. Por exemplo, quanto $f_0f_1f_2$ = 011, K_X = 0, K_Y=x, C_0=1, KM_0=1, KM_1=KM_2=0.

3.2. MÓDULOS COMP1 e COMP2

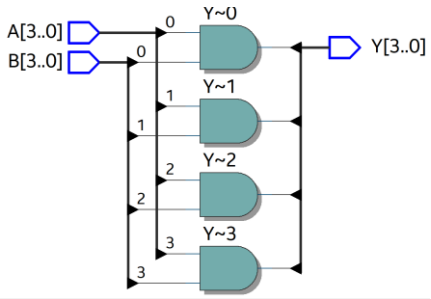


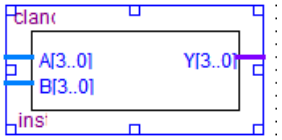


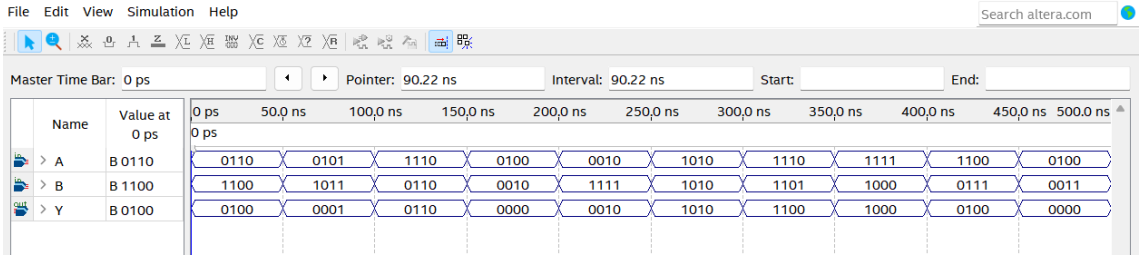


Para os módulos complemento, quando *sel* = 0, X = Y e para *sel* = 1, Y (1010) é o complemento a um de X (0101).

3.3. MÓDULO CL AND

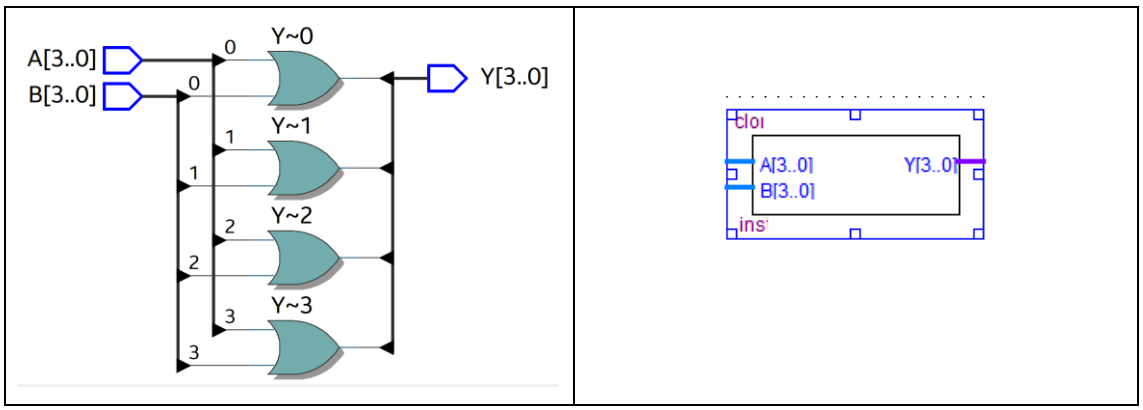


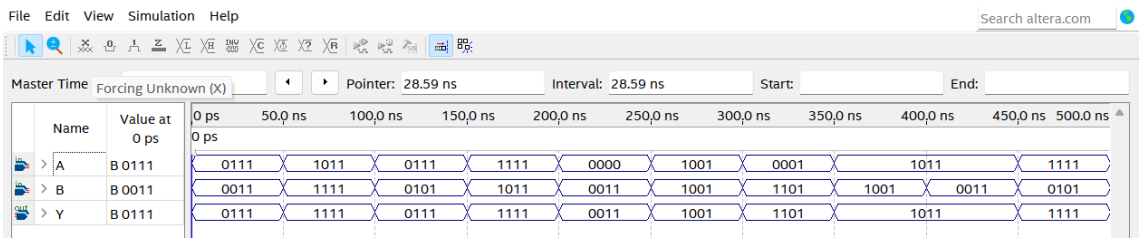




Este módulo realiza a operação *and* bit a bit. Observa-se na simulação que apenas quando os bits dos operandos possuem nível lógico 1 o resultado é o nível lógico 1 e 0 caso contrário.

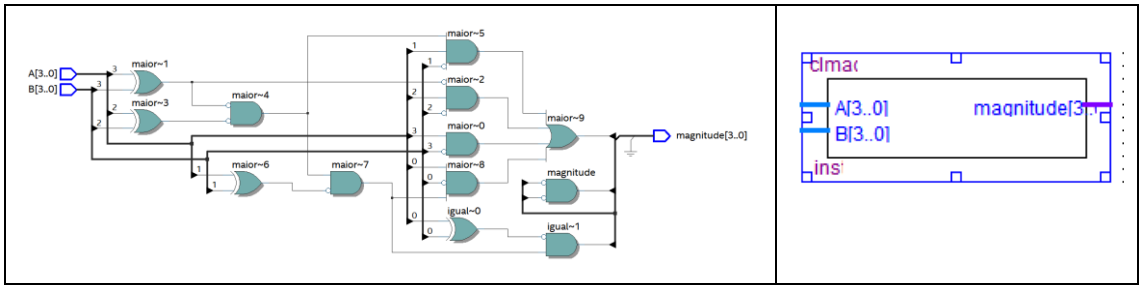
3.4. MÓDULO CL OR

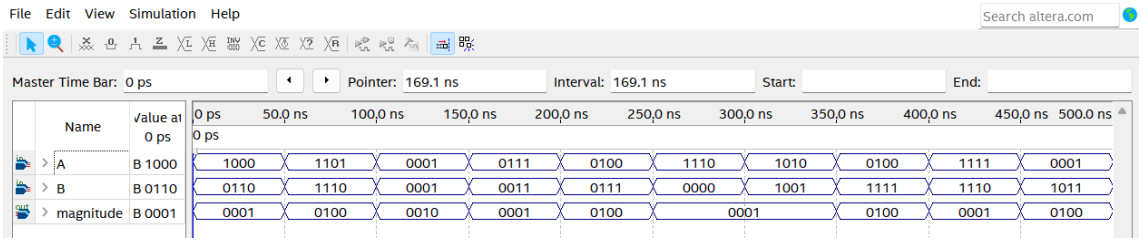




Este módulo realiza a operação *or* bit a bit. Observa-se na simulação que apenas quando o bit dos dois operandos possuem nível lógico 0 o resultado é o nível lógico 0 e 1 caso contrário.

3.5. MÓDULO CL MAG



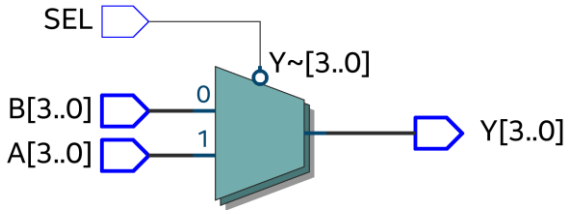


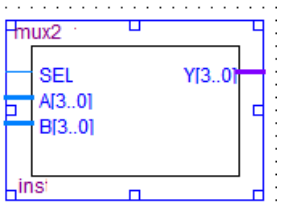
Para A > B, magnitude = 0001

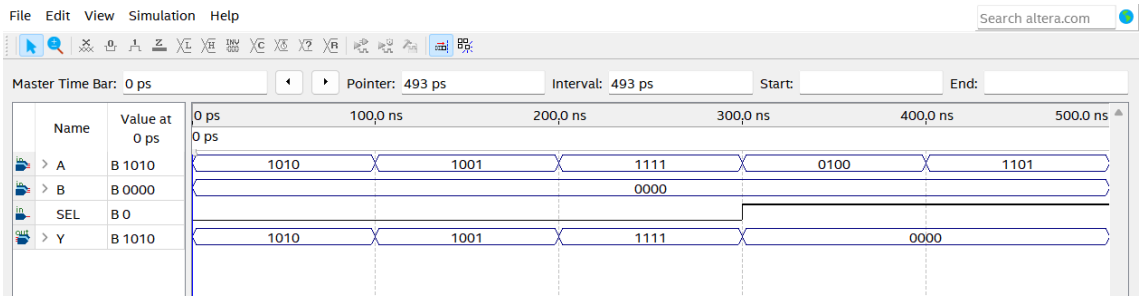
Para A = B, magnitude = 0010

Para A < B, magnitude = 0100

3.6. MÓDULO CHAVE1

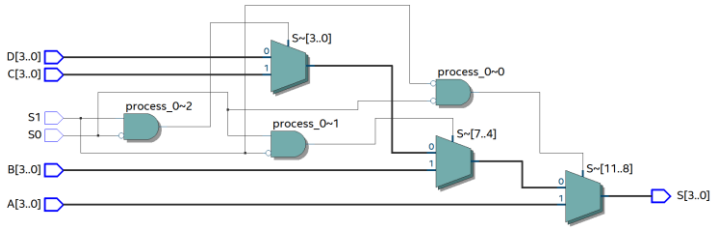


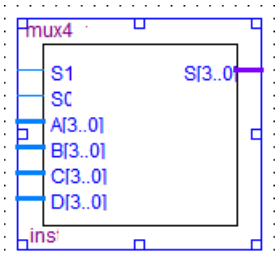


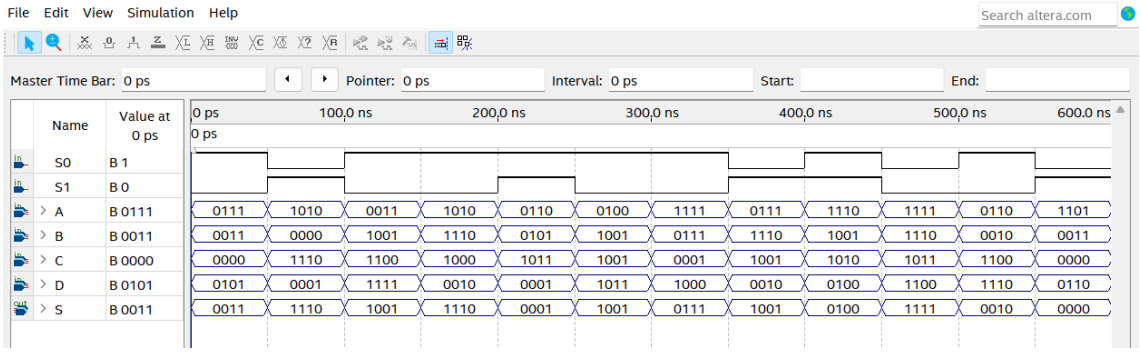


O módulo é formado por 4 multiplexadores 2:1, portanto, quando *sel* = 0 a Y = A e quando *sel* = 1, Y = B.

3.7. MÓDULO CHAVE2



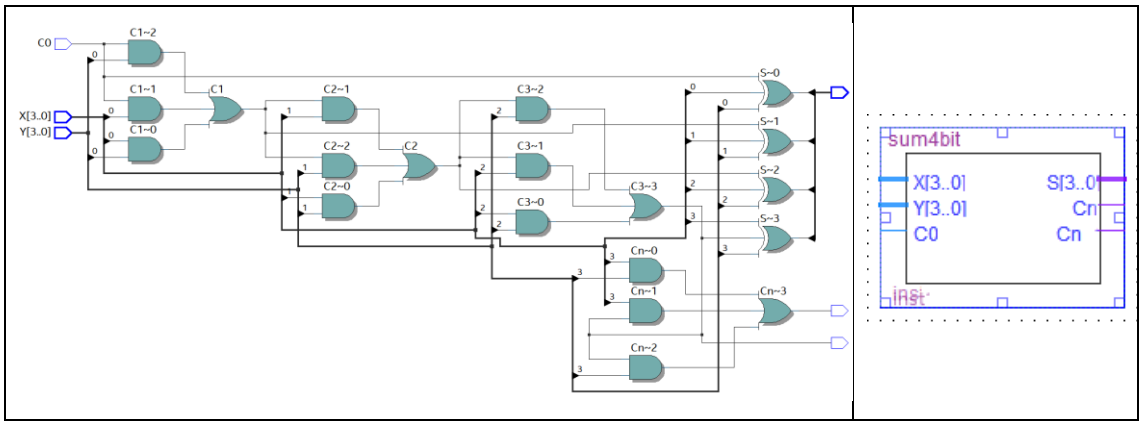


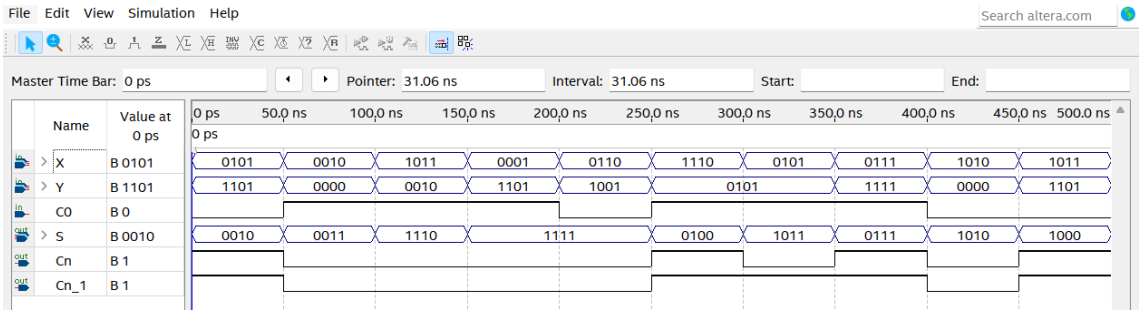


Para :

KM1 = 0, KM2 = 0, S = A (saída do somador)
KM1 = 0, KM2 = 1, S = B (saída do And)
KM1 = 1, KM2 = 0, S = C (saída do OR)
KM1 = 1, KM2 = 1, S = D (saída do comparador de magnitude)

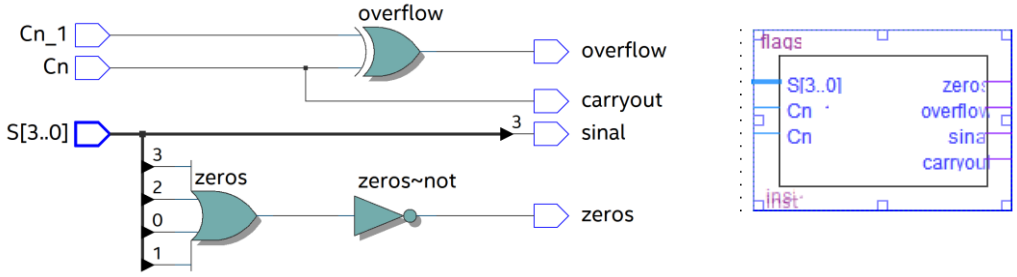
3.8. MÓDULO SOMADOR

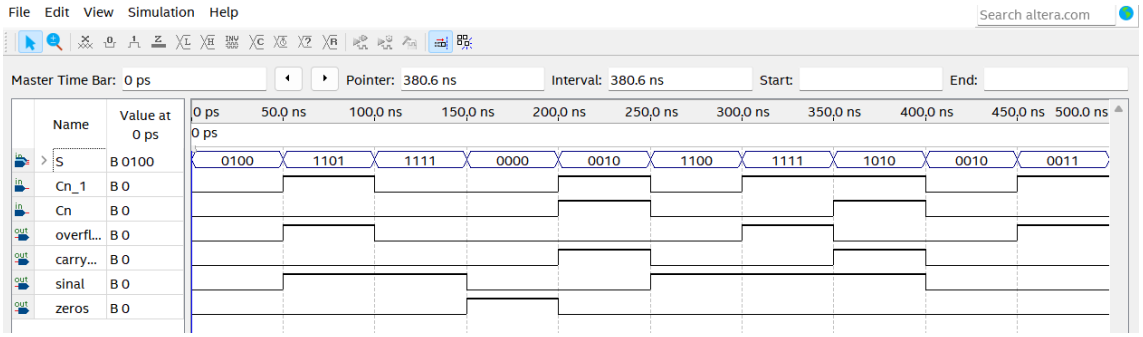




Os valores de entrada X, Y e C0 foram produzidos aleatoriamente e os resultados daas somas podem ser verificados como corretos.

3.9. MÓDULO FLAGS

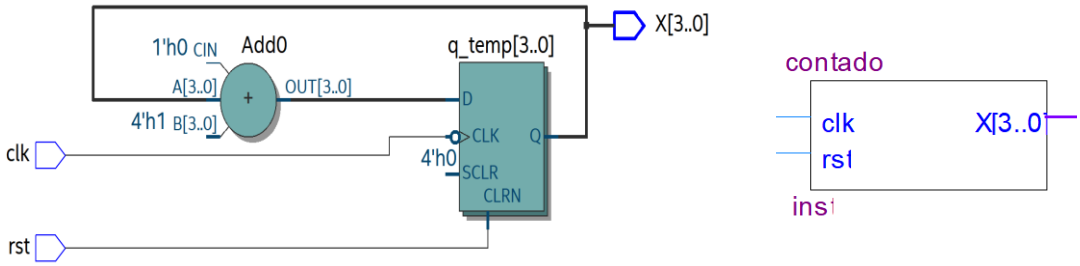


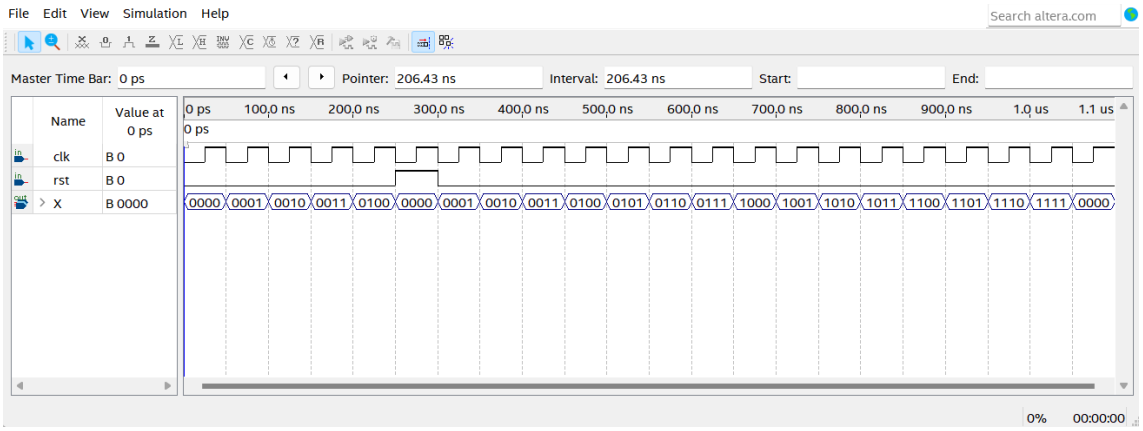


Entre 150 e 200 ns, os bits da soma são iguais a zero, fazendo com que o bit *zeros* seja igual a 1.

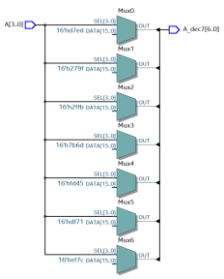
Entre 300 e 350 ns, Cn = 0 e Cn_1 = 1, apenas indicação de overflow.

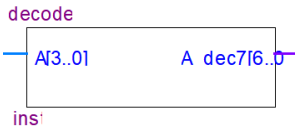
3.10. MÓDULO CONTADOR

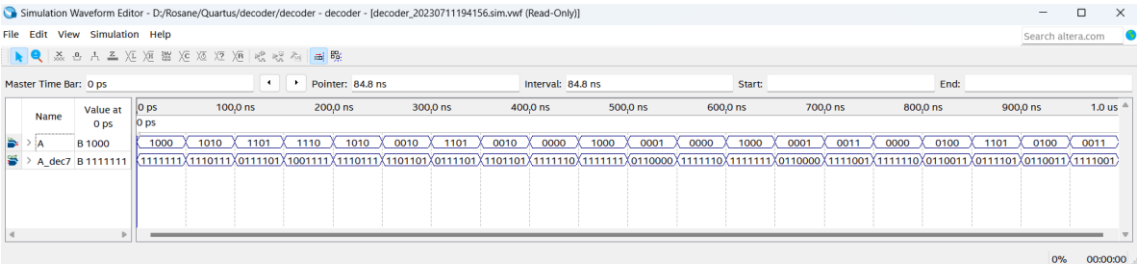




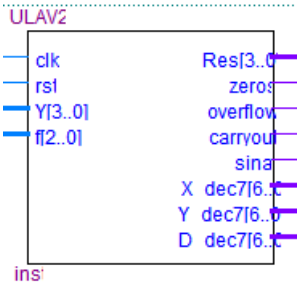
3.11. MÓDULO DECODIFICADOR

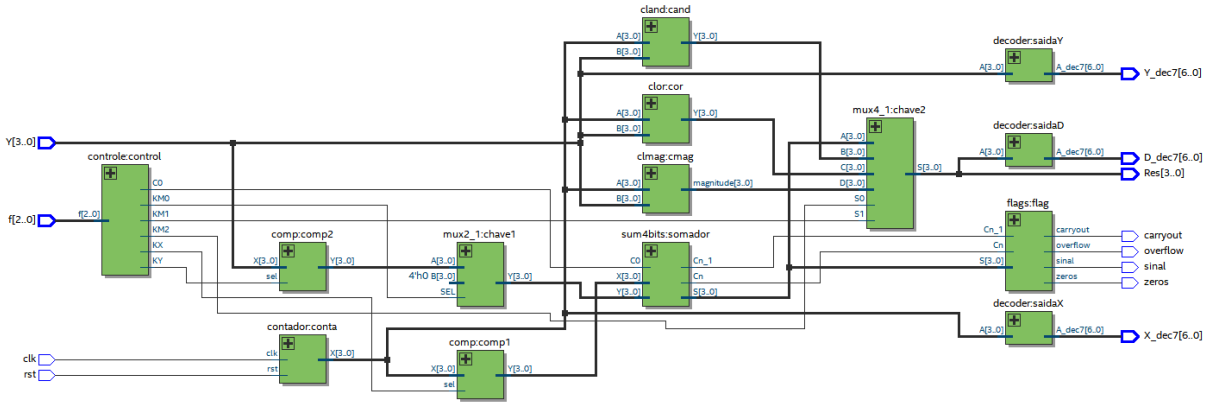


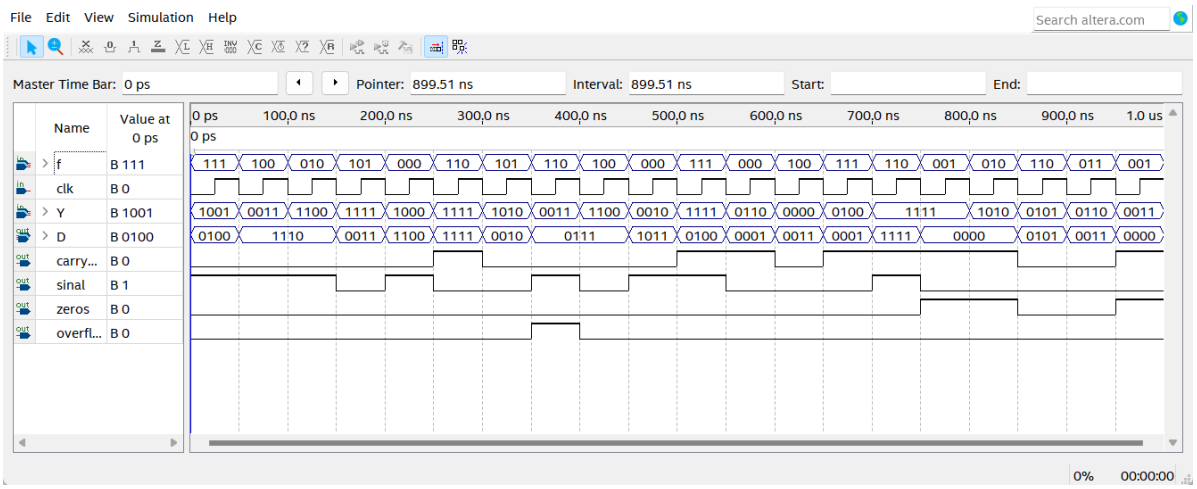




3.12. UNIDADE LÓGICA ARITMÉTICA







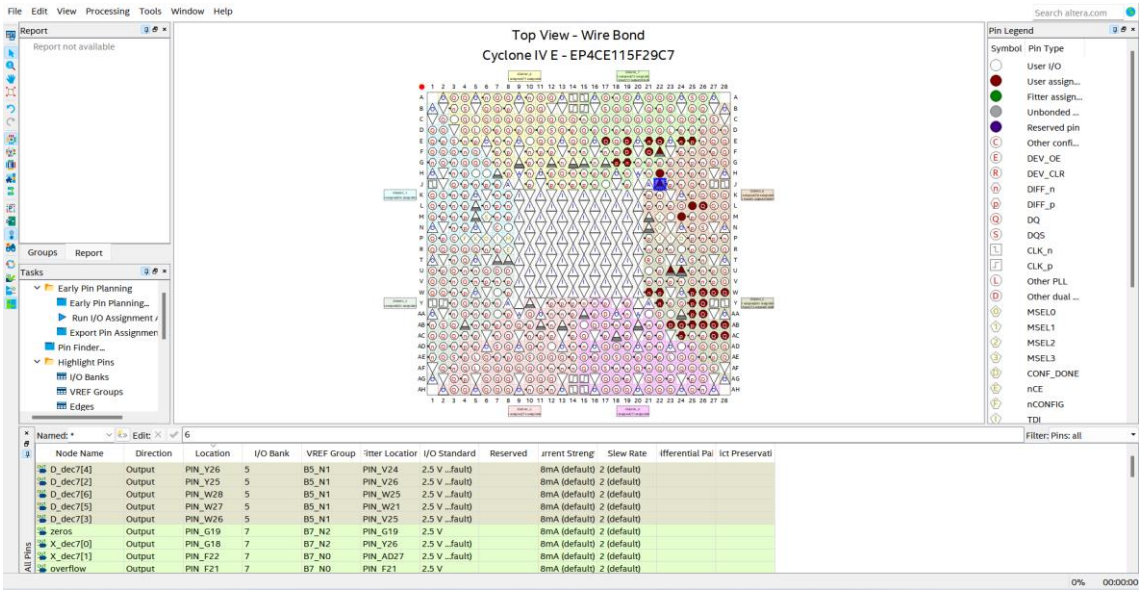
A simulação utiliza valores aleatórios de Y, um dos operandos, e de f, das opções das operações (f_0 f_1 f_2). A Tabela 2 apresenta alguns dos resultados para simples conferência da eficácia da ULA projetada.

Tabela 2: Resultados das Operações da ULA

Intervalo de Tempo (ns)	Código de Operação	Tipo de Operação	Valor de X	Valor de Y	Resultado Esperado	Resultado Obtido
0 a 50	111	MAG	0000	1001	0100	0100
50 a 100	100	COMP1	0001	-	1100	1100
100 a 150	010	TROCA ±	0010	-	1110	1110
150 a 200	101	AND	0011	1111	0011	0011
200 a 250	000	SOMA	0100	1000	1100	1100
250 a 300	110	OR	0101	1111	1111	1111
750 a 800	001	SUB	1111	1111	0000	0000
900 a 950	011	INC1	0010	-	0011	0011

1.1. PROGRAMAÇÃO DA PLACA FPGA

As entradas dos operadores clk, rst , Y e operação f são ligados em chaves deslizantes. A saída está ligada em leds verdes e um display, enquanto os flags de sinalização de zeros, overflow, carryout e sinal são ligados em leds vermelhos. Os operandos X e Y também são apresentados em displays de sete segmentos.



2. CONCLUSÕES

A realização deste trabalho proporcionou um aprendizado significante para o desenvolvimento de projetos com circuitos lógicos. A ferramenta *Quartus Prime Lite edition*, embora tenha uma interface amigável, apresenta uma infinidade de opções que não foi possível explorá-las no curto espaço de tempo.

O projeto apresentado adotou o conceito de módulos o que permitiu sua construção passo a passo e que cada um desses módulos fosse testado separadamente para posterior integração.

Foi interessante observar que, dependendo de como se inicie o projeto, via diagrama em blocos ou programação VHDL, os resultados podem ser diferentes, em termos de circuitos lógicos.

Certamente, por se tratar de um primeiro projeto nesta linguagem muitas melhorias podem ser feitas. O código poderia ser mais minimizado e elegante, porém, como uma primeira abordagem, foi de grande aprendizado implementá-lo passo a passo para uma melhor compreensão do sistema.

Ademais, nosso projeto foi enviado para o servidor do GitHub, como solicitado. O link para acesso do código do projeto é o seguinte: https://github.com/mfdeco/trabalhovhdl

Agradecemos ao professor pelos ensinamentos dados e pelo trabalho realizado durante as aulas.

Referências:

[1] R. J. Tocci, N. S. Widmer e G. L. Moss, SISTEMAS DIGITAIS princípios e aplicações, 11ª edição, 2011.