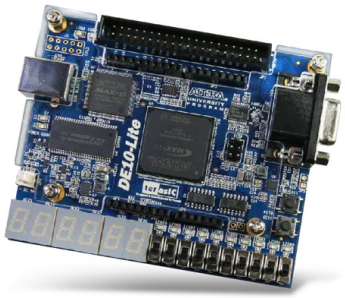
Departamento de Engenharia
Eletrónica e Telecomunicações
e de Computadores

Licenciatura em Engenharia Informática e de Computadores
Licenciatura em Engenharia Informática, Redes e Telecomunicações



RELATÓRIO DO 1^0 TRABALHO

Circuitos Combinatórios

Trabalho realizado por:

(52718) Alexandre Silva

(52599) Duarte Rodrigues

Turma: 12D
Docente: Pedro Miguens, Sérgio André e Diogo Bastos

Lógica e Sistemas Digitais
2024 / 2025 inverno

Conteúdo

Índice **i**

1 Objetivo **1**

2 Descrição do Circuito a Projetar **1**

3 Desenvolvimento do Projeto **2**

3.1 Funções lógicas dos circuitos 2

3.2 Simplificação lógica 3

3.3 Diagrama de blocos 5

4 Resultados e Discussão **8**

4.1 Simulação 9

4.2 Teste 10

5 Conclusão **11**

Referências **12**

A VHDL **13**

A.1 alu.vhd 13

A.2 decoder.vhd 20

A.3 mux_2inputs.vhd 22

A.4 flags/flags_main.vhd 23

A.5 flags/odd_parity_checker.vhd 25

A.6 arithmetic_unit/arithmeticunit.vhd 27

A.7 arithmetic_unit/adder_subtractor_4bits.vhd 31

A.8 arithmetic_unit/flags.vhd 35

A.9 arithmetic_unit/full_adder.vhd 37

A.10 arithmetic_unit/half_adder.vhd 40

A.11 arithmetic_unit/inner_arithmetic.vhd 41

A.12 logic_module/logic_module.vhd 44

A.13 logic_module/logical_shift_right.vhd 50

A.14 logic_module/arithmetic_shift_right.vhd 51
A.15 logic_module/logical_shift_left.vhd 52
A.16 logic_module/mux_3inputs.vhd 53
A.17 logic_module/mux_4inputs.vhd 55
A.18 logic_module/nand_gate.vhd 57

B Atribuição de Pinos **58**

1 Objetivo

O objetivo do trabalho é descrever e simular um circuito aritmético e lógico (*ALU - Arithmetic and Logical Unit*) baseado em VHDL estrutural, simulá-lo e implementá-lo na placa de desenvolvimento *DE-10 Lite* da *Intel*.

2 Descrição do Circuito a Projetar

O trabalho tem como objetivo projetar um circuito capaz de realizar as operações aritméticas de adição (X + Y + CBi), subtração (X - Y - CBi), incrementação (X - CBi), decrementação (X - CBi) e as operações deslocamento (X»>1, X»1 X«<1) e a operação lógica de NAND($\overline{X \cdot Y}$)

As entradas são X e Y, operandos de 4 *bits*, CBi um operando de 1 *bit* e a OP, de 3 *bits*, que seleciona a operação a realizar, operação esta que é descodificada a partir do módulo *decoder*.

A saída R é o resultado da operação, estando este, no mesmo domínio dos operandos. Assim, o resultado das operações, (R) gera os indicadores e saídas da *ALU* (*flags*) *Carry/Borrow* (CBo), *Overflow* (OV), Zero(Z), *Greater or Equal* (GE), *Below or Equal* (BE) e *Parity* (P). Na figura 1 encontra-se as operações realizadas pela *ALU*.

OP	Operação	CBo	OV	Z	P	GE	BE
0 0 0	$X + CBi$	X	X	X	X	–	–
0 0 1	$X - CBi$	X	X	X	X	–	–
0 1 0	$X + Y + CBi$	X	X	X	X	–	–
0 1 1	$X - Y - CBi$	X	X	X	X	X	X
1 0 0	$X >>> 1$	X	–	X	X	–	–
1 0 1	$X >> 1$	X	–	X	X	–	–
1 1 0	$X <<< 1$	X	–	X	X	–	–
1 1 1	$\overline{X.Y}$	–	–	X	X	–	–

Figura 1: Tabela das funções do circuito *ALU*

3 Desenvolvimento do Projeto

3.1 Funções lógicas dos circuitos

As funções do *decoder* foram implementadas através da sua simplificação lógica conforme os mapas de *Karnaugh* utilizados no 3.2, obtendo assim uma tabela de sinais internos correspondestes aos *bits* da entrada OP, Tabela 1.

Na equação (1) podemos obter os mintermos de Z.

$$f({R_3}^+, R_2, R_1, {R_0}^-) = m(0) \tag{1}$$

Na equação (2) podemos obter os mintermos de P.

$$f({R_3}^+, R_2, R_1, {R_0}^-) = \sum m(1, 2, 4, 7, 8, 11, 13, 14) \tag{2}$$

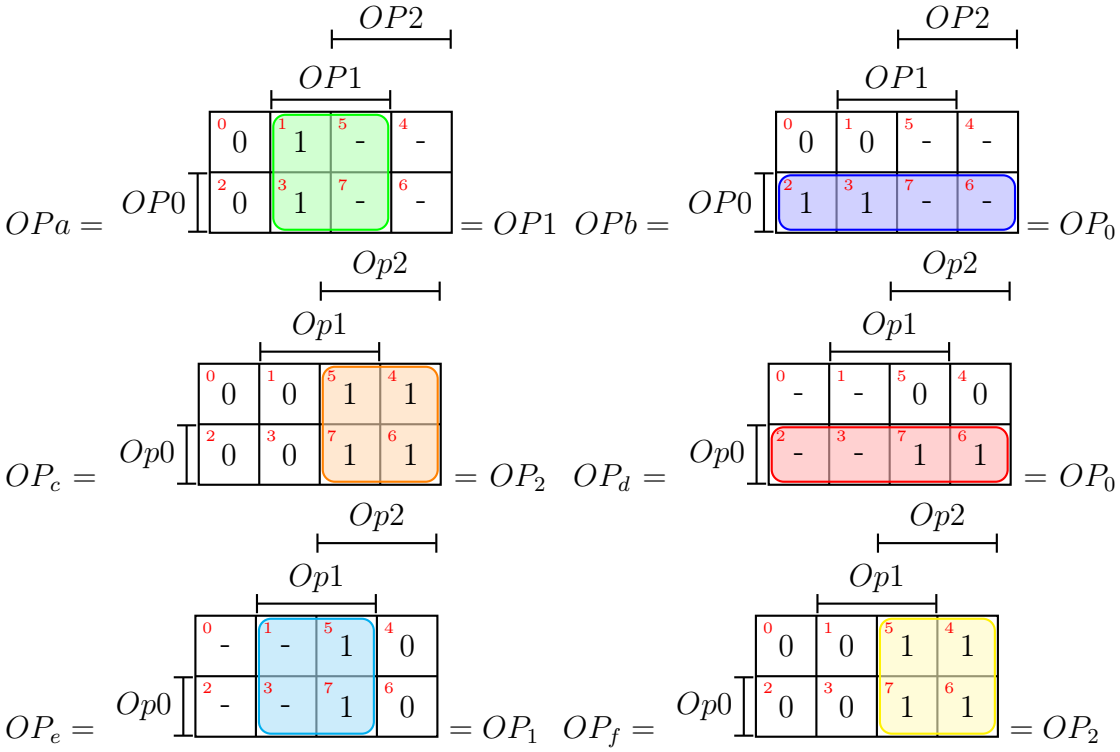
Os *outputs* do bloco de *flags* são obtidos através de um R, resultado da avaliação lógica do *MUX* entre o módulo lógico e o módulo aritmético, um *carry* do LABc3 *module* e um *carry* do *Logic module*, um *overflow* vindo também do *LABc module* e o sinal interno de OPf. Assim, temos que:

- $Ov = (A_3 \cdot B_3 \cdot \overline{R_3}) or (\overline{A_3} \cdot \overline{B_3} \cdot R_3)$
- $CBo = (iCB \cdot \overline{OPf}) or (Cy \cdot OPf)$
- $BE = (iCB) or (Z)$
- $GE = (\overline{R_3} \cdot \overline{Ov}) or (R_3 \cdot Ov))$

Nestes casos, as expressões vêm das entradas ou saídas de diversos componentes da ALU (figura 2), sendo assim necessária fazer uma correspondência através dos diversos blocos. Desta forma, conseguimos compreender que por exemplo $A_3 = X_3$, o $B_3 = (Y_3 \cdot OPa) or (0 \cdot \overline{OPa})$ e o CBo do *LABc module*(figura 3) é o iCB usado nas funções devido ao diagrama de blocos do ALU.

3.2 Simplificação lógica

Nos seguintes mapas de *Karnaugh*, podemos constatar que conseguimos obter uma simplificação lógica de cada sinal interno de OP, através de um dos *bits* de $OP_{3..0}$.



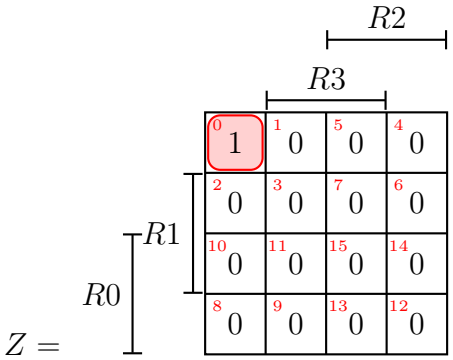
Assim, através destes mapas de *Karnaugh*, temos que:

Sinal Interno	**$OP_{3..0}$**
OPa	OP1
OPb	OP0
OPc	OP2
OPd	OP0
OPe	OP1
OPf	OP2

Tabela 1: Simplificações lógicas para o *decoder*.

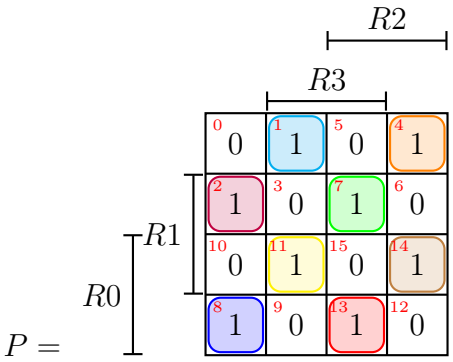
Desta forma, é possível fazer a correspondência entre os sinais internos de OPa a OPf e os 3 *bits* de entrada de OP.

No seguinte mapa de *Karnaugh*, podemos constatar que a função Z (zeros), não tem simplificação lógica pelo que a função foi descrita através do mintermo(0), como anteriormente mencionado.



$Z = m(0)$

No seguinte mapa de *Karnaugh*, podemos constatar que a função P (paridade ímpar), não tem simplificação lógica pelo que a função foi descrita através do somatório dos seus mintermo associados, como anteriormente mencionado.



$P = \sum m(1, 2, 4, 7, 8, 11, 13, 14)$

3.3 Diagrama de blocos

Na Figura 2 encontra-se o Diagrama de blocos ALU.

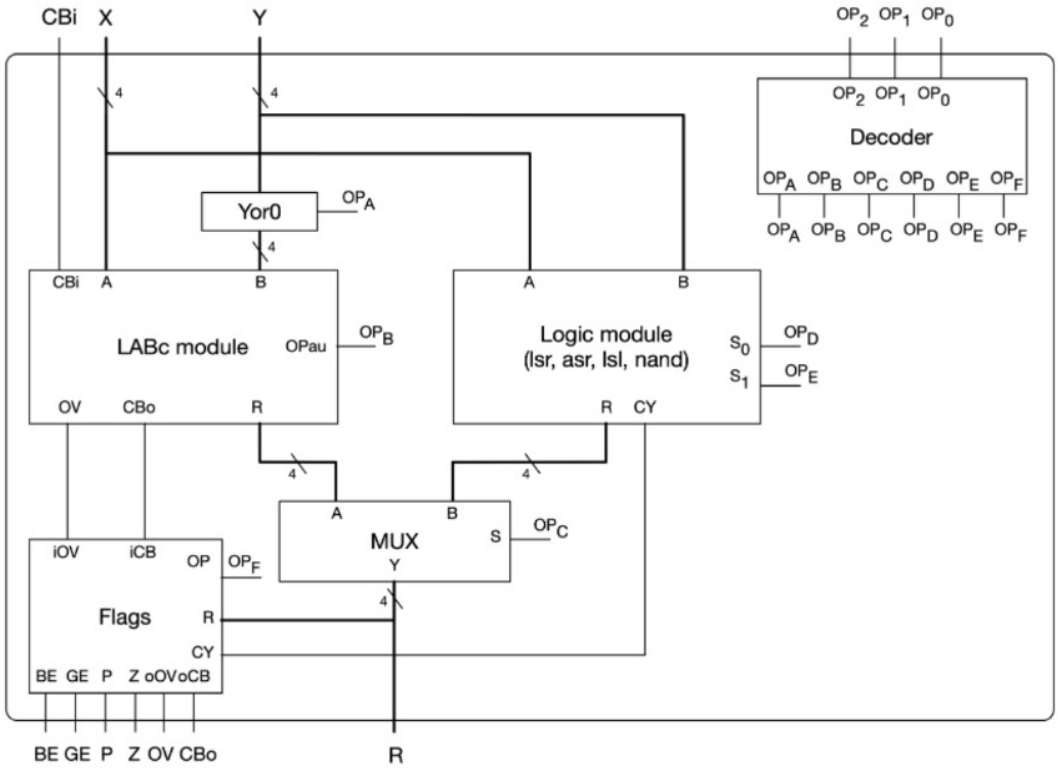


Figura 2: Diagrama de Blocos *ALU*.

Na Figura 3 encontra-se o Diagrama de blocos *LABc module.*

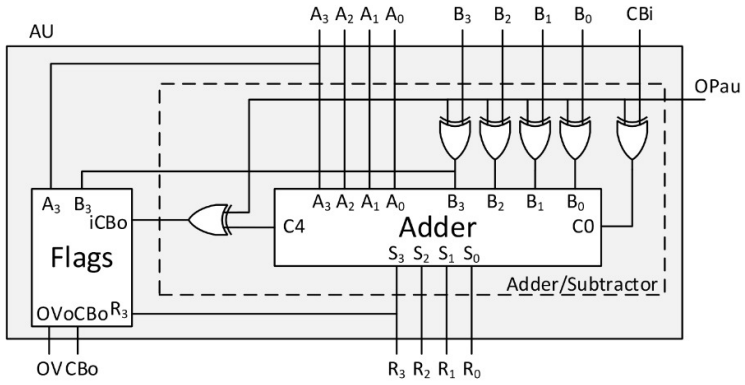


Figura 3: Diagrama de Blocos LABc module.

Na Figura 4 encontra-se o Diagrama de blocos. *Logic module*

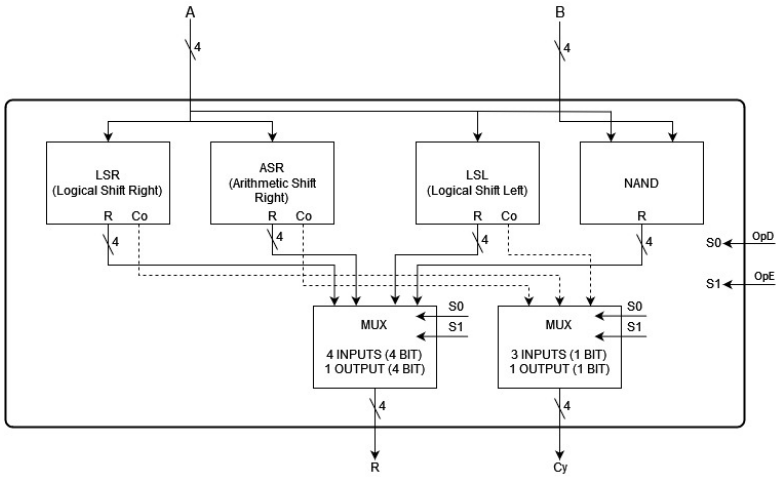


Figura 4: Diagrama de Blocos Logic module.

Na Figura 5 encontra-se o Diagrama de blocos *Flags*.

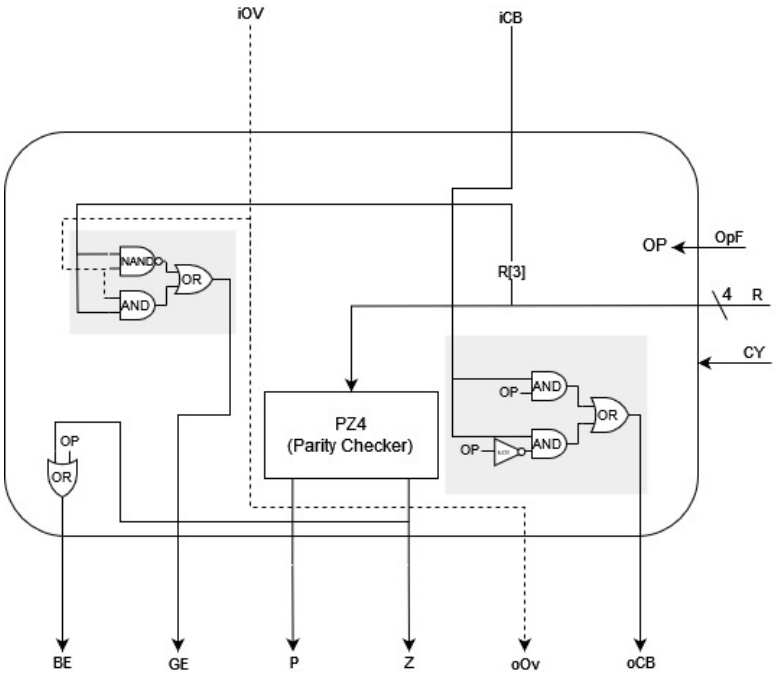


Figura 5: Diagrama de Blocos Flags.

4 Resultados e Discussão

Após a realização de todos os componentes e ligações necessária, o nosso objetivo era verificar se os valores que obtiamos nas simulações estavam de acordo com o esperado.

Para isso, fizemos uso do recurso ao *ModelSim*, onde podemos verificar o comportamento do nosso circuito em relação às nossas entradas (X, Y, CBi e OP) e saídas (R, CBo, OV, Z, P, GE e BE).

O circuito ALU foi implementado visando realizar as operações aritméticas (soma, subtração, incremento e decremento), operações de deslocamento (LSL, LSR, ASR) e a operação lógica (NAND), além de gerar os *outputs* apropriados das *flags* e o resultado R.

Durante o processo de simulação pudemos constatar que com diferentes combinações de entradas aplicadas, todos os modulos apresentaram os valores esperados e os sinais intermédios corresponderam sempre aquilo que se pretendia, demonstrando o pleno funcionamento do circuito.

No entanto, nesta fase do trabalho, caso os valores de algum dos modulos demonstrasse valores fora do normal, ou aparecesse como indefinido, através do *ModelSim*, somos capazes de fazer *debug* de verificar os valores intermédios 1 a 1 até encontrar o problema de ligação ou de expressão que estaria a originar o problema.

4.1 Simulação

Realizou-se a simulação do circuito projetado com o objetivo de verificar a funcionalidade deste.

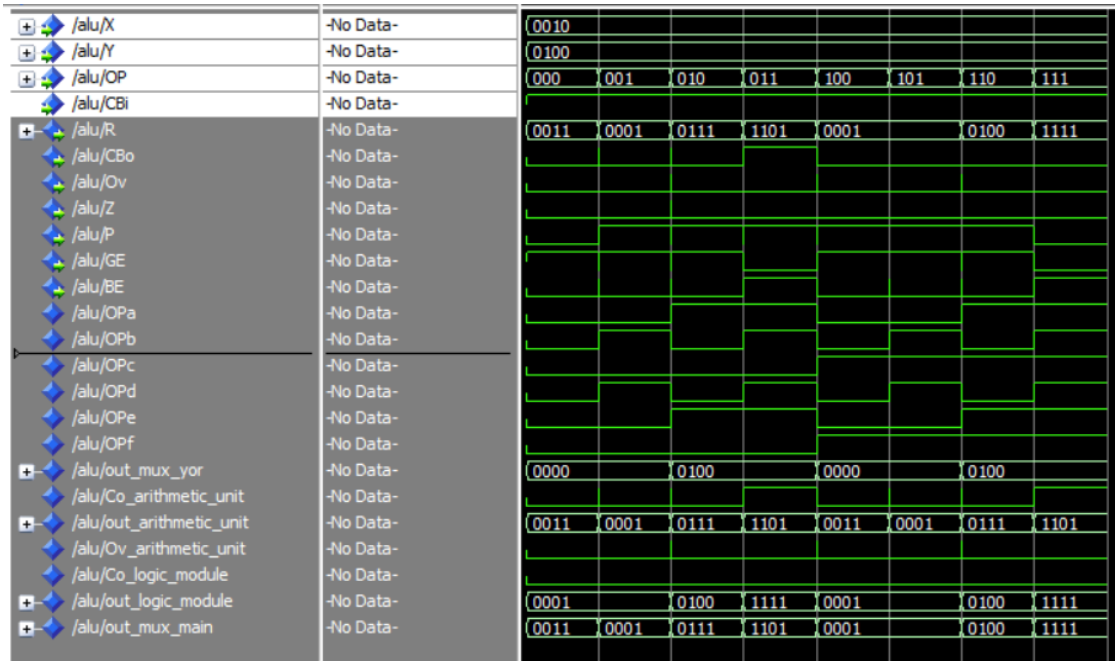


Figura 6: Waves com inputs X=0010, Y=0100, CBi=1.

Na figura 6 está representada a simulação de todas as operações possíveis, feitas com o X=0010, Y=0100 e o CBi=1, sendo assim possível verificar que o circuito está funcional e com os valores corretos.

Além disso, podemos constatar que a simulação continua a dar-nos valores como Ge e Be, mesmo quando a operação a ser realizada não é a de X-Y-CBi, pelo que devemos ter atenção e decidir se os valores fazem sentido de acordo com as operações efetuadas.

4.2 Teste

									Resultado Teórico									Resultado Experimental								
#	OP	CBi	X(2)	X(N)	X(Z)	Y(2)	Y(N)	Y(Z)	R(2)	R(N)	R(Z)	CB0	OV	Z	P	GE	BE	R(2)	R(N)	R(Z)	CB0	OV	Z	P	GE	BE
1	000	0	0000	0	0	1111	15	-1	0000	0	0	0	0	1	0	-	-	0000	0	0	0	0	1	0	-	-
2	000	1	0000	0	0	1111	15	-1	0001	1	+1	0	0	0	1	-	-	0001	1	+1	1	0	0	1	-	-
3	000	1	0111	7	+7	0101	5	+5	1000	8	-8	0	1	0	1	-	-	1000	8	-8	0	1	0	1	-	-
4	000	1	1111	15	-1	1010	10	-6	0000	0	0	1	0	1	0	-	-	0000	0	0	1	0	1	0	-	-
5	001	1	0000	0	+0	1111	15	-1	1111	15	-1	0	0	0	0	-	-	1111	15	-1	0	0	0	0	-	-
6	001	1	0001	1	+1	0101	5	+5	0000	0	0	0	0	1	0	-	-	0000	0	0	0	0	1	0	-	-
7	001	1	1000	8	-8	1010	10	-6	0111	7	+7	1	1	0	1	-	-	0111	7	+7	1	1	0	1	-	-
8	010	0	0001	1	+1	1110	14	-2	1111	15	-1	0	0	0	0	-	-	1111	15	-1	0	0	0	0	-	-
9	010	1	0001	1	+1	1110	14	-2	0000	0	0	1	0	1	0	-	-	0000	0	0	1	0	1	0	-	-
10	010	1	1000	8	-8	1111	15	-1	1000	8	-8	1	1	0	1	-	-	1000	8	-8	1	1	0	1	-	-
11	010	1	0010	2	+2	0101	5	+5	1000	8	-8	0	0	0	1	-	-	1000	8	-8	0	0	0	1	-	-
12	011	0	0011	3	+3	0101	5	+5	1110	14	-2	0	0	0	1	0	1	1110	14	-2	0	0	0	1	0	1
13	011	0	1110	6	+6	1100	12	-4	0010	2	+2	1	0	0	1	1	0	0010	2	+2	1	0	0	1	1	0
14	011	1	0110	6	+6	0110	6	+6	1111	15	-1	1	0	0	0	0	1	1111	15	-1	1	0	0	0	0	1
15	011	1	1000	8	-8	0111	7	+7	0000	0	0	0	1	1	0	0	1	0000	0	0	0	1	1	0	0	1
16	100	1	0001	1	+1	1100	12	-4	0000	0	0	1	-	1	0	-	-	0000	0	0	1	-	1	0	-	-
17	100	0	1101	13	-3	1001	9	-7	0110	6	+6	1	-	0	0	-	-	0110	6	+6	1	-	0	0	-	-
18	101	0	1010	10	-6	1111	15	-1	1101	13	-3	0	-	0	1	-	-	1101	13	-3	0	-	0	1	-	-
19	110	1	1001	9	-7	0110	6	+6	0010	2	+2	1	-	0	1	-	-	0010	2	+2	1	-	0	1	-	-
20	111	1	1111	15	-1	0100	4	+4	1101	11	-5	-	-	0	1	-	-	1101	11	-5	-	-	0	1	-	-
21	001	0	0110	6	+6	0101	5	+5	0110	6	+6	0	0	0	0	-	-	0110	6	+6	0	0	0	0	-	-
22	010	0	0110	6	+6	0101	5	+5	1011	11	-5	0	1	0	1	-	-	1011	11	-5	0	1	0	1	-	-
23	100	1	0110	6	+6	0101	5	+5	0011	3	+3	0	-	0	0	-	-	0011	3	+3	0	-	0	0	-	-
24	111	1	0110	6	+6	0101	5	+5	1011	11	-5	-	-	0	1	-	-	1011	11	-5	-	-	0	1	-	-

Figura 7: Tabela dos testes na Placa DE10-Lite

Nesta fase do trabalho, completamos a tabela com os resultados teóricos, de acordo com os *inputs* ou resultados fornecidos e fizemos a comparação entre os resultados teóricos e os resultados obtidos na placa de testes *DE10-Lite*.

Ao fazermos a comparação, chegamos à conclusão que a placa *DE10-Lite* estava a obter valores em conformidade com o esperado e pudemos concluir que o circuito estava a funcionar em plenas condições. 7

5 Conclusão

O trabalho consistiu em descrever um circuito aritmético e lógico capaz de realizar as operações aritméticas de adição (X + Y + CBi), subtração (X - Y - CBi), incrementação (X - CBi), decrementação (X - CBi) e as operações deslocamento (X»>1, X»1 X«<1) e a operação lógica de NAND($X \cdot Y$).

Este projeto foi desenvolvido no decorrer da disciplina de Lógica e Sistemas Digitais com o acesso à aplicação *Quartus Prime 20.1 Lite Edition*.

Para tal, foi usado como recurso os mapas de *Karnaugh*, apresentado no ponto 3.2 do relatório, para determinar as funções lógica que melhor descreviam as expressões necessárias para resolver o problema proposto.

Após a utilização dos mapas de *Karnaugh*, concluímos que não existe simplificação para a expressão lógica da função de paridade ímpar, pelo que a implementámos através do somatório dos mintermos dessa dada função.

A função Z foi implementada através do mintermo (0) como é demonstrado no ponto 3.1 do relatório.

Os *outputs* das flags foram obtidos através do resultado do *MUX* entre os módulos lógico e aritmético e as entradas e saídas destes mesmos módulos, como foi descrito no ponto 3.1.

Além disso foi realizada uma simplificação lógica de forma a obtermos as expressões dos OP usados no *decoder*, no ponto 3.2.

Este circuito foi testado na placa *DE10-Lite* da *Intel* durante a aula laboratorial da disciplina de forma a ter o circuito validado. Os resultados experimentais confirmam o bom funcionamento do circuito, de acordo com os resultados da simulação feita através da *Testbench* utilizada para validar o circuito.

Referências

Referências

[1] Pedro Miguens Matutino, Sérgio André *Guia de Instalação do Quartus Prime Lite Edition 20.1.1.720*, 2022, ISEL.

[2] Terasic *DE10-lite User Manual*, 2020, Terasic.

A VHDL

A.1 alu.vhd

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

-- This is the main entity of the project, responsible for
   starting every
-- operation, allowing
entity alu is

  port (
    X     : in std_logic_vector(3 downto 0);
    Y     : in std_logic_vector(3 downto 0);

    OP    : in std_logic_vector(2 downto 0);
    CBi   : in std_logic;

    R     : out std_logic_vector(3 downto 0);
    CBo   : out std_logic;
    Ov    : out std_logic;
    Z     : out std_logic;
    P     : out std_logic;
    GE    : out std_logic;
    BE    : out std_logic
  );

end alu;


architecture behavioral of alu is

  -- Import the arithmetic unit
  component arithmeticunit
```

```
  port (
    A    : in std_logic_vector(3 downto 0);
    B    : in std_logic_vector(3 downto 0);

    Ci   : in std_logic;
    Co   : out std_logic;

    OPbit : in std_logic;
    R     : out std_logic_vector(3 downto 0);
    Ov    : out std_logic
  );

end component;

-- Import the logic module
component logic_module

  port (
    A  : in std_logic_vector(3 downto 0);
    B  : in std_logic_vector(3 downto 0);

    S0 : in std_logic;
    S1 : in std_logic;

    Co : out std_logic;
    R  : out std_logic_vector(3 downto 0)
  );

end component;

-- Import the decoder
component decoder

  port (
```

```
    OP2 : in std_logic;
    OP1 : in std_logic;
    OP0 : in std_logic;

    OPa : out std_logic;
    OPb : out std_logic;
    OPc : out std_logic;
    OPd : out std_logic;
    OPe : out std_logic;
    OPf : out std_logic
  );

end component;

-- Import the two inputs MUX
component mux_2inputs

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    S : in std_logic;

    R : out std_logic_vector (3 downto 0)
  );

end component;

-- Import the two inputs MUX
component flags_main

  port (
    Ov     : in std_logic;
    CBi_au : in std_logic;
    CBi_lm : in std_logic;
```

```
        R_mux   : in std_logic_vector(3 downto 0);
        OPf     : in std_logic;

        CBo     : out std_logic;
        Ov_out  : out std_logic;
        BE      : out std_logic;
        GE      : out std_logic;
        P       : out std_logic;
        Zs      : out std_logic
      );

    end component;

    -- Defines all the operation signals to be used
    signal OPa : std_logic;
    signal OPb : std_logic;
    signal OPc : std_logic;
    signal OPd : std_logic;
    signal OPe : std_logic;
    signal OPf : std_logic;

    -- Defines all the output signals
    signal out_mux_yor          : std_logic_vector(3 downto 0);
    signal Co_arithmetic_unit   : std_logic;
    signal out_arithmetic_unit  : std_logic_vector(3 downto 0);
    signal Ov_arithmetic_unit   : std_logic;
    signal Co_logic_module      : std_logic;
    signal out_logic_module     : std_logic_vector(3 downto 0);
    signal out_mux_main         : std_logic_vector(3 downto 0);

begin

    -- Instantiate the main decoder, to translate the operation
        bits to the A-F signals
    instance_decoder : decoder
```

```
  port map (
    OP2 => OP(2),
    OP1 => OP(1),
    OP0 => OP(0),

    OPa => OPa,
    OPb => OPb,
    OPc => OPc,
    OPd => OPd,
    OPe => OPe,
    OPf => OPf
  );


-- Instantiates the 2 inputs MUX that will serve as a YOR
instance_yor : mux_2inputs

  port map (
    A => "0000",
    B => Y,
    S => OPa,

    R => out_mux_yor
  );


-- Instantiates the arithmetic unit to perform math
   operations
instance_arithmetic_unit : arithmeticunit

  port map (
    A => X,
    B => out_mux_yor,
```

```
    Ci => CBi,
    Co => Co_arithmetic_unit,

    OPbit => OPb,
    R     => out_arithmetic_unit,
    Ov    => Ov_arithmetic_unit
  );


-- Instantiates the logic module for any relational or
   logical operations
instance_logic_module : logic_module

  port map (
    A => X,
    B => Y,

    S0 => OPd,
    S1 => OPe,

    Co => Co_logic_module,
    R  => out_logic_module
  );


-- Instantiates the MUX used to connect the logic module
   and arithmetic unit
instance_mux : mux_2inputs

  port map (
    A => out_arithmetic_unit,
    B => out_logic_module,
    S => OPc,

    R => out_mux_main
```

```
    );


  -- Instantiates the flags module
  instance_flags : flags_main

    port map (
      Ov      => Ov_arithmetic_unit,
      CBi_au  => Co_arithmetic_unit,
      CBi_lm  => Co_logic_module,
      R_mux   => out_mux_main,
      OPf       => OPf,

      CBo     => CBo,
      Ov_out  => Ov,
      BE      => BE,
      GE      => GE,
      P       => P,
      Z_out   => Z
    );

  R <= out_mux_main;

end behavioral;
```

A.2 decoder.vhd

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

-- This entity is responsible for acting as a decoder,
   allowing us to select
-- different operations on the ALU
entity decoder is

  port (
    OP2 : in std_logic;
    OP1 : in std_logic;
    OP0 : in std_logic;

    OPa : out std_logic;
    OPb : out std_logic;
    OPc : out std_logic;
    OPd : out std_logic;
    OPe : out std_logic;
    OPf : out std_logic
  );

end decoder;


architecture structural of decoder is
begin

  OPa <= OP1;
  OPb <= OP0;
  OPc <= OP2;
  OPd <= OP0;
  OPe <= OP1;
  OPf <= OP2;
```

```
end structural;
```

A.3 mux_2inputs.vhd

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

-- This entity is responsible for acting like a MUX, and
   giving the output based
-- on the selected inputs
entity mux_2inputs is

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    S : in std_logic;

    R : out std_logic_vector(3 downto 0)
  );

end mux_2inputs;


-- Implements the logic of a MUX with two inputs and one
   selector
architecture structural of mux_2inputs is

  signal S4bit : std_logic_vector(3 downto 0);

begin

  S4bit <= S & S & S & S;
  R <= (A and not(S4bit)) or (B and S4bit);

end structural;
```

A.4 flags/flags_main.vhd

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

entity flags_main is

  port (
    Ov       : in std_logic;
    CBi_au   : in std_logic;
    CBi_lm   : in std_logic;
    R_mux    : in std_logic_vector(3 downto 0);
    OPf      : in std_logic;

    CBo      : out std_logic;
    Ov_out   : out std_logic;
    BE       : out std_logic;
    GE       : out std_logic;
    P        : out std_logic;
    Z_out    : out std_logic
  );

end flags_main;

architecture behavioral of flags_main is

  -- Import the odd_P_checker
  component odd_parity_checker

    port (
      I0 : in std_logic;
      I1 : in std_logic;
      I2 : in std_logic;
      I3 : in std_logic;
```

```
      P : out std_logic;
      Z : out std_logic
    );

  end component;

  signal Z_signal : std_logic;

begin

  -- Instantiation of the odd_parity_checker
  instance_odd_parity_checker : odd_parity_checker

    port map (
      I0 => R_mux(0),
      I1 => R_mux(1),
      I2 => R_mux(2),
      I3 => R_mux(3),

      P => P,
      Z => Z_signal
    );

    Z_out  <= Z_signal;
    Ov_out <= Ov;
    BE     <= CBi_au or Z_signal;
    GE     <= not (R_mux(3) or Ov) or (Ov and R_mux(3));
    CBo    <= (CBi_lm and OPf) or (CBi_au and not OPf);

end behavioral;
```

A.5 flags/odd_parity_checker.vhd

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

-- Declaration of odd_parity_checker with inputs between I0
   and I3 and outputs P and Z
entity odd_parity_checker is
  port
  (
    I0 : in std_logic;
    I1 : in std_logic;
    I2 : in std_logic;
    I3 : in std_logic;

    P : out std_logic;
    Z : out std_logic
  );
end odd_parity_checker;



architecture structural of odd_parity_checker is

  -- Declaration of the minterm signals where the output P is
      on
  signal minterm_1  : std_logic;
  signal minterm_2  : std_logic;
  signal minterm_4  : std_logic;
  signal minterm_7  : std_logic;
  signal minterm_8  : std_logic;
  signal minterm_11 : std_logic;
  signal minterm_13 : std_logic;
  signal minterm_14 : std_logic;
```

```
begin

    -- Odd input minterms
    minterm_1  <= (not (I0 or I1 or I2) and I3);
    minterm_2  <= (not (I0 or I1 or I3) and I2);
    minterm_4  <= (not (I0 or I2 or I3) and I1);
    minterm_7  <= (not I0 and (I1 and I2 and I3));
    minterm_8  <= (not (I1 or I2 or I3) and I0);
    minterm_11 <= (not I1 and (I0 and I2 and I3));
    minterm_13 <= (not I2 and (I0 and I1 and I3));
    minterm_14 <= (not I3 and (I0 and I1 and I2));

    -- P is on when either of the odd minterms is on
    -- Z is on when none of the inputs are on
    P <= (minterm_1 or minterm_2 or minterm_4 or minterm_7 or
        minterm_8 or minterm_11 or minterm_13 or minterm_14);
    Z <= (not (I0 or I1 or I2 or I3));

end structural;
```

A.6 arithmetic_unit/arithmeticunit.vhd

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

-- This entity is responsible for taking in two 4-bit
   integers and adding
-- them together, allowing for a carry in, and a carry out
   and an operation
-- variable to decide whether to subtract or add
entity arithmeticunit is

  port (
    A   : in std_logic_vector(3 downto 0);
    B   : in std_logic_vector(3 downto 0);

    Ci : in std_logic;
    Co : out std_logic;

    OPbit : in std_logic;
    R     : out std_logic_vector(3 downto 0);
    Ov    : out std_logic
  );

end arithmeticunit;


architecture behavioral of arithmeticunit is

  -- Import the flagging circuit
  component flags

    port (
      A : in std_logic;
      B : in std_logic;
```

```
    R : in std_logic;

    Ci : in std_logic;
    Co : out std_logic;

    Ov : out std_logic
  );

end component;


-- Import the inner arithmetic circuit
component inner_arithmetic

  port (
    A : in std_logic_vector;
    B : in std_logic_vector;

    Ci    : in std_logic;
    OPbit : in std_logic;

    R          : out std_logic_vector(3 downto 0);
    Co         : out std_logic;
    B3_output : out std_logic
  );

end component;


-- Declares an intermediate "out_inner_arithmetic" signal
   so that we can
-- read its bits afterwards, the carry and a xor'd b3
   signal with the
-- operation bit, use inside the flags
signal out_inner_arithmetic : std_logic_vector(3 downto 0);
```

```
  signal Ciner_arithmetic : std_logic;
  signal B3_xor : std_logic;

begin

  -- Instantiate the inner arithmetic, whose parameters are
     mostly the same
  -- as the ones we have in the AU, due to it just being a
     layer of abstraction
  instance_inner_arithmetic : inner_arithmetic

    port map (
      A => A,
      B => B,

      Ci        => Ci,
      OPbit     => OPbit,

      R          => out_inner_arithmetic,
      Co         => Ciner_arithmetic,
      B3_output => B3_xor
    );


  -- Instantiate the flags unit
  instance_flags : flags

    port map (
      A => A(3),
      B => B3_xor,
      R => out_inner_arithmetic(3),

      Ci => Ciner_arithmetic,
      Co => Co,
```

```
      Ov => Ov
    );


  R <= out_inner_arithmetic;

end behavioral;
```

A.7 arithmetic_unit/adder_subtractor_4bits.vhd

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

-- This entity is responsible for taking in two 4-bit
   integers and adding
-- them together, allowing for a carry in, and a carry out
   and an operation
-- variable to decide whether to subtract or add
entity adder_subtractor_4bits is

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    Ci : in std_logic;
    Co : out std_logic;

    R : out std_logic_vector(3 downto 0)
  );

end adder_subtractor_4bits;


architecture behavioral of adder_subtractor_4bits is

  -- Imports the full adder from the specification in the
     full_adder entity
  component full_adder

    port (
      A : in std_logic;
      B : in std_logic;
```

```
      Ci : in std_logic;
      Co : out std_logic;

      R : out std_logic
    );

  end component;


  -- Declares the output and carry variables for the 4 full
     adders
  signal out_full_adder_1 : std_logic;
  signal out_full_adder_2 : std_logic;
  signal out_full_adder_3 : std_logic;
  signal out_full_adder_4 : std_logic;

  signal carry_full_adder_1 : std_logic;
  signal carry_full_adder_2 : std_logic;
  signal carry_full_adder_3 : std_logic;
  signal carry_full_adder_4 : std_logic;

begin

  -- Instantiates a full adder with all the default values
  instance_full_adder_1 : full_adder

    port map (
      A => A(0),
      B => B(0),

      Ci => Ci,
      Co => carry_full_adder_1,

      R => out_full_adder_1
    );
```

```
-- Instantiates three more full adders with the previous
   carries as inputs
-- and with incremental bit placements for the A and B
   inputs
instance_full_adder_2 : full_adder

  port map (
    A => A(1),
    B => B(1),

    Ci => carry_full_adder_1,
    Co => carry_full_adder_2,

    R => out_full_adder_2
  );


instance_full_adder_3 : full_adder

  port map (
    A => A(2),
    B => B(2),

    Ci => carry_full_adder_2,
    Co => carry_full_adder_3,

    R => out_full_adder_3
  );


instance_full_adder_4 : full_adder

  port map (
```

```
        A => A(3),
        B => B(3),

        Ci => carry_full_adder_3,
        Co => carry_full_adder_4,

        R => out_full_adder_4
      );


    -- Defines the adder/subtractor's carry out as the last
       full adder's carry
    Co <= carry_full_adder_4;

    -- Sets the R bits in order
    R(0) <= out_full_adder_1;
    R(1) <= out_full_adder_2;
    R(2) <= out_full_adder_3;
    R(3) <= out_full_adder_4;

end behavioral;
```

A.8 arithmetic_unit/flags.vhd

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

-- This entity is simply responsible for checking whether
   there's overflow and
-- a carry out (Ov for unsigned integers and carry out for
   signed ints)
-- and outputting it for later use
entity flags is

  port (

    -- A and B are the most significant bits for both numbers
        (see diagram)
    -- If these bits are 0, it's a positive integer.
    -- If they're 1, it's a negative integer.
    A : in std_logic;
    B : in std_logic;
    R : in std_logic;  -- R is the most significant bit for
       the result

    Ci : in std_logic;
    Co : out std_logic;

    Ov : out std_logic
  );

end flags;


architecture behavioral of flags is
begin
```

```
  Co <= Ci;
  Ov <= ((A and B and not R) or (not A and not B and R));

end behavioral;
```

A.9 arithmetic_unit/full_adder.vhd

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

-- This entity is responsible for acting as a full adder,
   using two half-adders
-- in succession to account for first bit carries
entity full_adder is

  port (
    A : in std_logic;
    B : in std_logic;

    Ci : in std_logic;
    Co : out std_logic;

    R : out std_logic
  );

end full_adder;


architecture behavioral of full_adder is


  -- Imports the half adder from the specification in the
     half_adder entity
  component half_adder

    port (
      A : in std_logic;
      B : in std_logic;

      R   : out std_logic;
```

```
      Co  : out std_logic
    );

  end component;

  -- Initialises the R carriers for the half-adders
  signal out_half_adder_1 : std_logic;
  signal out_half_adder_2 : std_logic;

  signal carry_half_adder_1 : std_logic;
  signal carry_half_adder_2 : std_logic;


begin

  -- Instantiate the first half adder with the full adder's A
      and B
  instance_half_adder_1 : half_adder

    port map (
      A => A,
      B => B,

      R  => out_half_adder_1,
      Co => carry_half_adder_1
    );


  -- Instantiate the second half adder with the first half
     adder's R and
  -- carry, to account for the bit carry
  instance_half_adder_2: half_adder

    port map (
      A => out_half_adder_1,
```

```
      B => Ci,

      R  => out_half_adder_2,
      Co => carry_half_adder_2
    );


  R  <= out_half_adder_2;
  Co <= (carry_half_adder_1 or carry_half_adder_2);


end behavioral;
```

A.10 arithmetic_unit/half_adder.vhd

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

-- This entity is responsible for acting as a half adder,
   simply XOR'ing two
-- to sum them returning their value and carry.
entity half_adder is

  port (
    A : in std_logic;
    B : in std_logic;

    R  : out std_logic;
    Co : out std_logic
  );

end half_adder;


architecture structural of half_adder is
begin

  R  <= A xor B;
  Co  <= A and B;

end structural;
```

A.11 arithmetic_unit/inner_arithmetic.vhd

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

-- This entity is responsible for handling the inner
   aritmetic logic excluding
-- the circuit flags. Whether the calculation is a
   subtraction or addition is also
-- handled here
entity inner_arithmetic is

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    Ci     : in std_logic;
    OPbit : in std_logic;

    R          : out std_logic_vector(3 downto 0);
    Co         : out std_logic;
    B3_output : out std_logic
  );

end inner_arithmetic;


architecture behavioral of inner_arithmetic is

  -- Import the 4 bit adder/subtractor
  component adder_subtractor_4bits

    port (
      A : in std_logic_vector(3 downto 0);
      B : in std_logic_vector(3 downto 0);
```

```
      Ci : in std_logic;
      Co : out std_logic;

      R  : out std_logic_vector(3 downto 0)
    );

  end component;

  -- Declares a temporary variable to store the adder output
     for later
  signal carry_adder_subtractor_4bits  : std_logic;
  signal carry_adder_subtractor_4bits2 : std_logic;
  signal b_xor : std_logic_vector(3 downto 0);

begin

  b_xor(0) <= B(0) xor OPbit;
  b_xor(1) <= B(1) xor OPbit;
  b_xor(2) <= B(2) xor OPbit;
  b_xor(3) <= B(3) xor OPbit;


  -- Instantiates the adder/subtractor setting its R as the
     inner
  -- arithmetic R, and outputting the carry to XOR it with
     the operation bit
  instance_adder_subtractor_4bits : adder_subtractor_4bits

    port map (
      A => A,
      B => b_xor,

      Ci => carry_adder_subtractor_4bits2,
      Co => carry_adder_subtractor_4bits,
```

```
      R => R
    );


  carry_adder_subtractor_4bits2 <= Ci xor OPbit;
  Co          <= carry_adder_subtractor_4bits xor OPbit;
  B3_output <= b_xor(3);

end behavioral;
```

A.12 logic_module/logic_module.vhd

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

-- This entity is responsible for the operation logic
   functions
entity logic_module is

  port (
    A       : in std_logic_vector(3 downto 0);
    B       : in std_logic_vector(3 downto 0);

    S0 : in std_logic;
    S1 : in std_logic;

    Co    : out std_logic;
    R     : out std_logic_vector(3 downto 0)
  );

end logic_module;


architecture behavioral of logic_module is

  -- Import of the logical_shift_right circuit
  component logical_shift_right

    port (
      input : in std_logic_vector(3 downto 0);

      R     : out std_logic_vector(3 downto 0);
      Co    : out std_logic
    );
```

```
end component;

-- Import of the arithmetic_shift_right circuit
component arithmetic_shift_right

  port (
    input : in std_logic_vector(3 downto 0);

    R     : out std_logic_vector(3 downto 0);
    Co    : out std_logic
  );

end component;

-- Import of the logical_shift_left circuit
component logical_shift_left

  port (
    input : in std_logic_vector(3 downto 0);

    R     : out std_logic_vector(3 downto 0);
    Co    : out std_logic
  );

end component;

-- Import of the nand_gate circuit
component nand_gate

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    R : out std_logic_vector(3 downto 0)
  );
```

```
end component;

-- Import of the mux_4inputs circuit
component mux_4inputs

  port (
    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);

    S0 : in std_logic;
    S1 : in std_logic;

    R  : out std_logic_vector(3 downto 0)
  );

end component;

-- Import of the mux_3inputs circuit
component mux_3inputs

  port (
    A : in std_logic;
    B : in std_logic;
    C : in std_logic;

    S0 : in std_logic;
    S1 : in std_logic;

    R  : out std_logic
  );

end component;
```

```
  -- Signal of logical_shift_right
  signal out_logical_shift_right : std_logic_vector(3 downto
     0);
  signal carry_logical_shift_right : std_logic;

  -- Signal of arithmetic_shift_right
  signal out_arithmetic_shift_right : std_logic_vector(3
     downto 0);
  signal carry_arithmetic_shift_right : std_logic;

  -- Signal of logical_shift_left
  signal out_logical_shift_left : std_logic_vector(3 downto
     0);
  signal carry_logical_shift_left : std_logic;

  -- Signal of nand_gate
  signal out_nand_gate : std_logic_vector(3 downto 0);

  -- Signals for the mux of 4 inputs for the carries
  signal out_mux_3inputs_carries : std_logic;

begin

  -- Instantiation of the logical_shift_right
  instance_logical_shift_right : logical_shift_right

    port map (
      input => A,

      R    => out_logical_shift_right,
      Co   => carry_logical_shift_right
    );
```

```
-- Instantiation of the arithmetic_shift_right
instance_arithmetic_shift_right : arithmetic_shift_right

  port map (
    input => A,

    R    => out_arithmetic_shift_right,
    Co   => carry_arithmetic_shift_right
  );


-- Instantiation of the logical_shift_left
instance_logical_shift_left : logical_shift_left

  port map (
    input => A,

    R    => out_logical_shift_left,
    Co   => carry_logical_shift_left
  );


-- Instantiation of the nand_gate
instance_logical_nand_gate : nand_gate

  port map (
    A => A,
    B => B,

    R => out_nand_gate

  );
```

```
  -- Instantiation of the mux_4inputs
  instance_mux_4inputs : mux_4inputs

    port map (
      A => out_logical_shift_right,
      B => out_arithmetic_shift_right,
      C => out_logical_shift_left,
      D => out_nand_gate,

      S0 => S0,
      S1 => S1,

      R  => R
    );

  -- Instantiation of the mux_3inputs for the carries,
     representing
  -- the carry bits in 3bit forms
  instance_mux_3inputs : mux_3inputs

    port map (
      A => carry_logical_shift_right,
      B => carry_arithmetic_shift_right,
      C => carry_logical_shift_left,

      S0 => S0,
      S1 => S1,

      R  => out_mux_3inputs_carries
    );

  -- Extract the first bit from the R
  Co <= out_mux_3inputs_carries;

end behavioral;
```

A.13 logic_module/logical_shift_right.vhd

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

-- This entity is responsibly for shifting an input to the
   right, effectively
-- dividing the input by two

entity logical_shift_right is

  port (
    input : in std_logic_vector(3 downto 0);

    R     : out std_logic_vector(3 downto 0);
    Co    : out std_logic
  );

end logical_shift_right;


architecture structural of logical_shift_right is
begin

  R(0) <= input(1);
  R(1) <= input(2);
  R(2) <= input(3);
  R(3) <= '0';

  Co   <= input(0);

end structural;
```

A.14 logic_module/arithmetic_shift_right.vhd

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

-- This entity is responsible for shifting a number to the
   right, effectively
-- dividing the number by two, while still putting the most
   significant bit on the leftmost digit
entity arithmetic_shift_right is

  port (
    input : in std_logic_vector(3 downto 0);

    Co  : out std_logic;
    R   : out std_logic_vector(3 downto 0)
  );

end arithmetic_shift_right;

architecture structural of arithmetic_shift_right is
begin

  R(0) <= input(1);
  R(1) <= input(2);
  R(2) <= input(3);
  R(3) <= input(3);

  Co   <= input(0);

end structural;
```

A.15 logic_module/logical_shift_left.vhd

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

-- This entity is responsible for shifting a number to the
   left, effectively
-- multiplying the number by two, while adding a '0' to the
   less significant bit
entity logical_shift_left is

  port (
    input    : in std_logic_vector(3 downto 0);

    Co   : out std_logic;
    R    : out std_logic_vector(3 downto 0)
  );

end logical_shift_left;

architecture structural of logical_shift_left is
begin

  R(0) <= '0';
  R(1) <= input(0);
  R(2) <= input(1);
  R(3) <= input(2);

  Co   <= input(3);

end structural;
```

A.16 logic_module/mux_3inputs.vhd

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

-- This entity is responsible for acting like a MUX, and
   giving the output based
-- on the selected inputs
entity mux_3inputs is

  port (
    A : in std_logic;
    B : in std_logic;
    C : in std_logic;

    S0 : in std_logic;
    S1 : in std_logic;

    R  : out std_logic
  );

end mux_3inputs;


-- Implements the logic of a MUX with three inputs and two Ss
architecture structural of mux_3inputs is

  -- For readability, separate every MUX's path into
     intermediate checks
  -- based on the MUX R
  signal path_A : std_logic;
  signal path_B : std_logic;
  signal path_C : std_logic;

begin
```

```
  path_A <= A and (not S0) and (not S1);
  path_B <= B and S0 and (not S1);
  path_C <= C and (not S0) and S1;

  R <= path_A or path_B or path_C;

end structural;
```

A.17 logic_module/mux_4inputs.vhd

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

-- This entity is responsible for acting like a MUX, and
   giving the output based
-- on the selected inputs
entity mux_4inputs is

  port (
    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);

    S0 : in std_logic;
    S1 : in std_logic;

    R  : out std_logic_vector (3 downto 0)
  );

end mux_4inputs;


-- Implements the logic of a MUX with four inputs and two Ss
architecture structural of mux_4inputs is

  -- For readability, separate every MUX's path into
     intermediate checks
  -- based on the MUX R
  signal path_A : std_logic_vector(3 downto 0);
  signal path_B : std_logic_vector(3 downto 0);
  signal path_C : std_logic_vector(3 downto 0);
  signal path_D : std_logic_vector(3 downto 0);
```

```
  -- Signals for the operation bit 4bit conversion
  signal S04bit : std_logic_vector(3 downto 0);
  signal S14bit : std_logic_vector(3 downto 0);

begin

  -- Assigns every path to its logical expression (the one
     that will conduct
  -- its R signal)
  S04bit <= S0 & S0 & S0 & S0;
  S14bit <= S1 & S1 & S1 & S1;

  path_A <= A and (not S04bit) and (not S14bit);
  path_B <= B and S04bit and (not S14bit);
  path_C <= C and (not S04bit) and S14bit;
  path_D <= D and S04bit and S14bit;

  R <= path_A or path_B or path_C or path_D;

end structural;
```

A.18 logic_module/nand_gate.vhd

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

-- This entity is responsible for performing a NAND operation
    between two inputs
entity nand_gate is

  port (
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);

    R : out std_logic_vector(3 downto 0)
  );

end nand_gate;

architecture structural of nand_gate is
begin

  R <= not (A and B);

end structural;
```

B Atribuição de Pinos

Entradas	**Pino**
X(0)	C10
X(1)	C11
X(2)	D12
X(3)	C12
Y(0)	A12
Y(1)	B12
Y(2)	A13
Y(3)	A14
OP(0)	B14
OP(1)	F15
OP(2)	B8
CBi	A7
Saídas	**Pino**
BE	B11
GE	A11
OV	C13
CBo	D13
P	D14
Z	E14
R(0)	A8
R(1)	A9
R(2)	A10
R(3)	B10

Tabela 2: Atribuição de pinos para a o circuito ALU [2]