Problema 1 – Cálculo do Máximo Divisor Comum, utilizando dispositivos lógicos programáveis

Objectivo

Pretende-se analisar o funcionamento de um sistema capaz de calcular o máximo divisor comum entre dois números A e B (a implementar através de dispositivos de lógica programável do tipo FPGA). Os dois números devem ser apresentados em binário, através dos interruptores instalados nos kits de experimentação disponíveis no laboratório. A e B são dois números de 4 bits cada (os grupos podem considerar diferentes valores, mas do ponto de vista da implementação recomenda-se 4 bits por condicionalismos de equipamento de experimentação).

O procedimento de cálculo proposto baseia-se no algoritmo de Euclides, brevemente descrito mais à frente.

Após actuação nos interruptores de entrada para fornecer os números A e B, o utilizador dá a ordem para cálculo do máximo divisor comum, findo o qual o sistema fornece o resultado e activa uma saída específica sinalizando que o cálculo está concluído.

A ordem para início de cálculo é realizada através da actuação de uma variável de entrada (GO). Adicionalmente, está disponível uma outra entrada para inicialização do sistema (RESET).

O objectivo da primeira parte do problema é o de apresentar e iniciar a utilização do ambiente de desenvolvimento ISE da Xilinx para CPLDs e FPGAs. Para isso utilizar-se-á uma representação baseada em esquemáticos e os recursos de simulação disponíveis.

O objectivo da segunda parte do problema é o de introduzir a utilização da linguagem VHDL na especificação de módulos a partir do ambiente de desenvolvimento ISE da Xilinx para CPLDs e FPGAs.

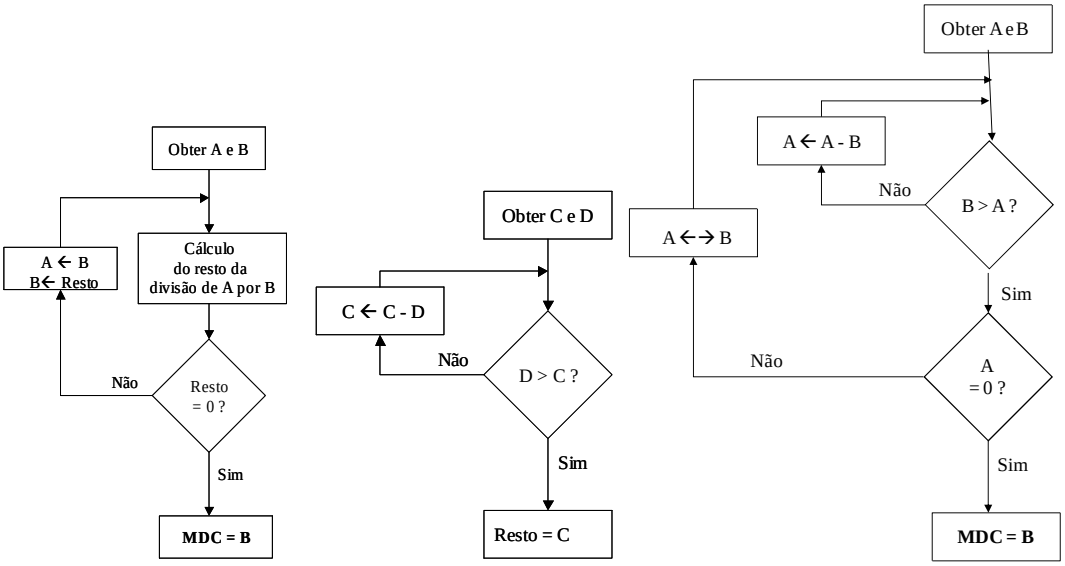
Finalmente, na terceira parte do problema, o objectivo é de utilizar VHDL para realizar a especificação da simulação do sistema.

Pretende-se realizar a implementação e teste físico do sistema nas várias fases do trabalho, através dos kits disponíveis no laboratório. Os kits podem ser requisitados pelos grupos de trabalho, para utilização dentro e fora das aulas.

Ao longo deste enunciado apresentam-se esboços de soluções possíveis, que cada grupo deverá concretizar. O presente enunciado pode ser "melhorado" sempre que o grupo o considere justificado; sempre que considere que o enunciado não é explícito ou é ambíguo, poderá (o grupo) definir a melhor forma de superar essa limitação.

Descrição

O algoritmo de Euclides pode ser descrito genericamente através do fluxograma da Figura 1(a), ou seja, admitindo que A é maior que B, a determinação do máximo divisor comum baseia-se no cálculo do resto da divisão de A por B; sempre que o resto é diferente de zero, procede-se à substituição de A por B e de B pelo resto obtido, voltando a executar-se o cálculo do resto da divisão; quando o resto for nulo, então o máximo divisor comum é igual a B.



(a) (b) (c)

Figura 1 – (a) Fluxogramas do algoritmo de Euclides; (b) Fluxograma para cálculo do resto da divisão através de subtracções sucessivas.

No algoritmo descrito, a parte claramente mais complexa é o cálculo do resto da divisão de A por B. Um procedimento possível para a sua implementação pode ser obtido através de subtrações sucessivas, como descrito no fluxograma da Figura 1(b).
Desta forma, a descrição do procedimento para o cálculo do máximo divisor comum utiliza uma descrição organizada hierarquicamente, contendo dois níveis.
A integração dos dois níveis conduz ao algoritmo apresentado na Figura 1(c) onde se prevê a troca entre os valores de A e B (como operação atómica).

Análise e preparação da implementação

A análise do sistema que se segue deve ser encarada como uma sugestão; os grupos de trabalho dispõem de toda a liberdade para proporem e implementarem caracterizações alternativas (ou resultantes de optimizações às sugestões apresentadas).
Para permitir a implementação do sistema, várias atitudes podem ser tomadas pelo projectista, por exemplo:

- tomar a caracterização algorítmica hierárquica com dois níveis apresentada atrás e implementar os dois níveis como componentes separados, mas interdependentes, em que se garanta a dependência funcional hierárquica entre os dois componentes;
- obter uma caracterização global para o sistema, contendo apenas um nível na descrição a utilizar.

Considere-se, como ponto de partida, uma caracterização genérica do sistema baseada numa decomposição do sistema numa parte de controlo e numa parte de dados. Para o caso específico do sistema a implementar, uma decomposição possível (a justificar mais à frente) é apresentada na Figura 2, em que:

- a parte de controlo, a implementar através de um circuito sequencial síncrono, sob controlo de um sinal de relógio (representado por CLK na Figura 2), dispõe de dois sinais de entrada externos (RESET e GO), que permitirão ao utilizador inicializar o sistema e indicar o início da operação, de duas saídas

para sinalizar “início de utilização” e “fim de operação” e de um conjunto de entradas e saídas que permitirão a interligação à parte de dados;

- a parte de dados dispõe de entradas de dados (os dois números A e B), de uma saída de dados (o máximo divisor comum de A e B), e de um conjunto de entradas e saídas permitindo a interligação à parte de controlo. Os elementos de memória necessários à sua implementação recebem o mesmo sinal de relógio utilizado pela parte de controlo (solução síncrona).

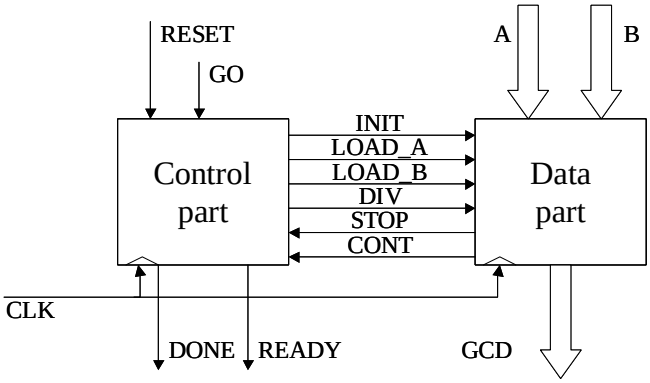


Figura 2 – Decomposição em parte de dados e controlo.

Caso pretenda considerar uma implementação com base nos dois componentes hierarquicamente organizados, será necessário considerar para cada um dos componentes uma decomposição semelhante, em partes de controlo e de dados. Deixa-se a sugestão como exercício adicional.

Nos parágrafos seguintes descrevem-se, sumariamente, as funções associadas a cada um dos sinais referidos na Figura 2:

- Entradas externas para a parte de controlo:
  - ✓ RESET destinado a inicializar o sistema (afecta só a parte de controlo, que será responsável pela inicialização da parte de dados);
  - ✓ GO destinado a indicar que se pretende realizar um cálculo do máximo divisor comum;

- Saídas da parte de controlo:
  - ✓ READY, destinado a sinalizar “início de utilização”, isto é que está à espera de ordem de cálculo;
  - ✓ DONE, destinado a sinalizar “fim de operação” e que o resultado se encontra disponível;

- Entradas externas para a parte de dados:
  - ✓ A e B, os dois números em relação aos quais se pretende calcular o máximo divisor comum; número de bits típico de 4 cada;

- Saídas da parte de dados:
  - ✓ GCD (greatest common divisor), é o máximo divisor comum;

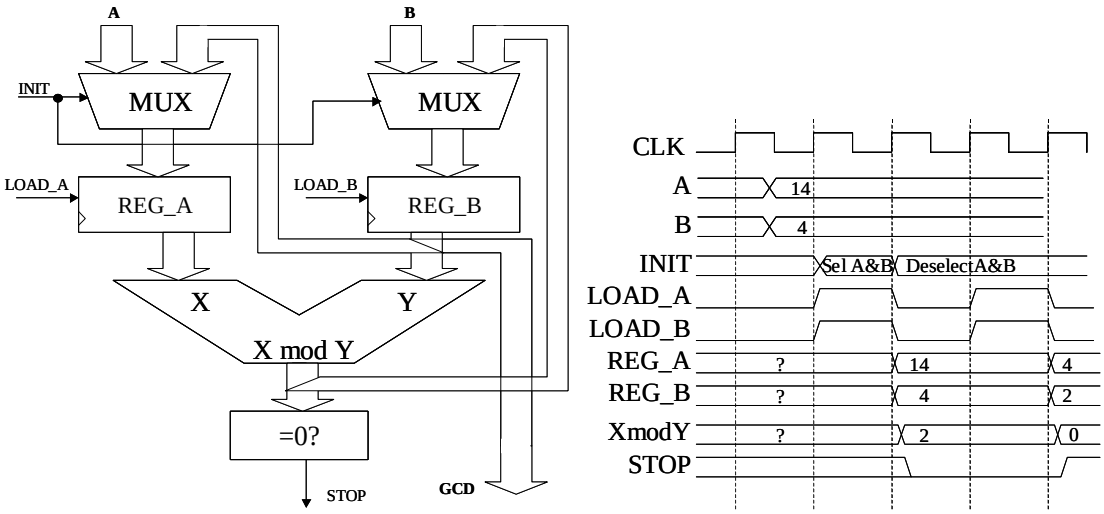
- Entradas para a parte de dados vindas da parte de controlo:
  - ✓ INIT, LOAD_A, LOAD_B e DIV, sinais específicos de controlo (ver diagramas temporais e descrição associada);

- Entradas para a parte de controlo vindas da parte de dados:
  - ✓ STOP, indicando que foi encontrado o máximo divisor comum,
  - ✓ CONT, indicando que a divisão ainda não está concluída.

Especificação de implementação da parte de dados

Quanto à parte de dados, recomenda-se (normalmente) a utilização de uma arquitectura de transferência entre registos. É por ela que necessitamos de começar, de modo a definir posteriormente a parte de controlo.

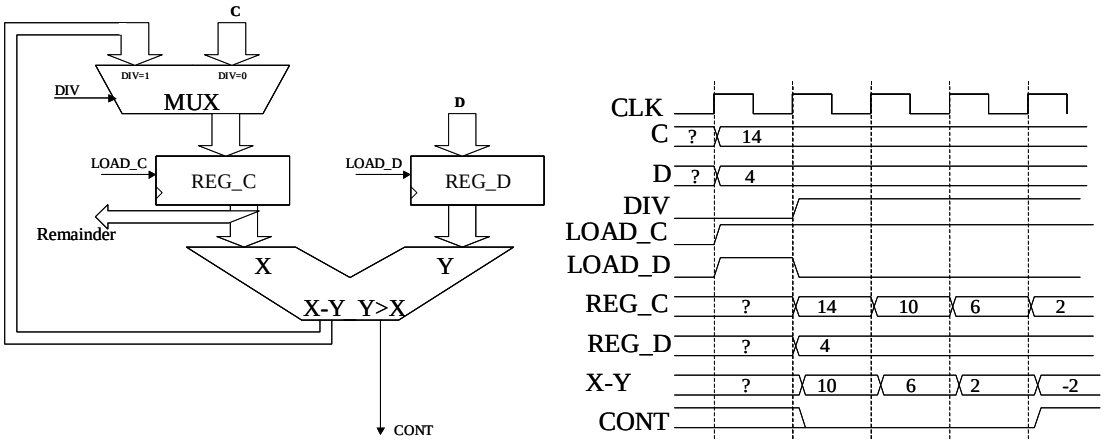
Na Figura 3(a) apresenta-se uma arquitetura que permitirá computar o máximo divisor comum, tendo a Figura 1(a) como referência, isto é, tomando apenas a representação algorítmica de nível topo. Na Figura 3(b) apresenta-se um hipotético diagrama temporal de evolução dos sinais presentes, considerando que se fornecem os valores de 14 e 4 para A e B e que os sinais LOAD_A, LOAD_B e INIT evoluem da forma indicada. No diagrama temporal representam-se as zonas de variação dos vários sinais, bem como as zonas em que o sinal é determinado (0 ou 1) e aquelas em que não se garante um valor (no presente exemplo).



(a) (b)

Figura 3 – Parte de dados I (a) Arquitectura associada a descrição de alto-nível do algoritmo de Euclides (b) Diagrama temporal exploratório.

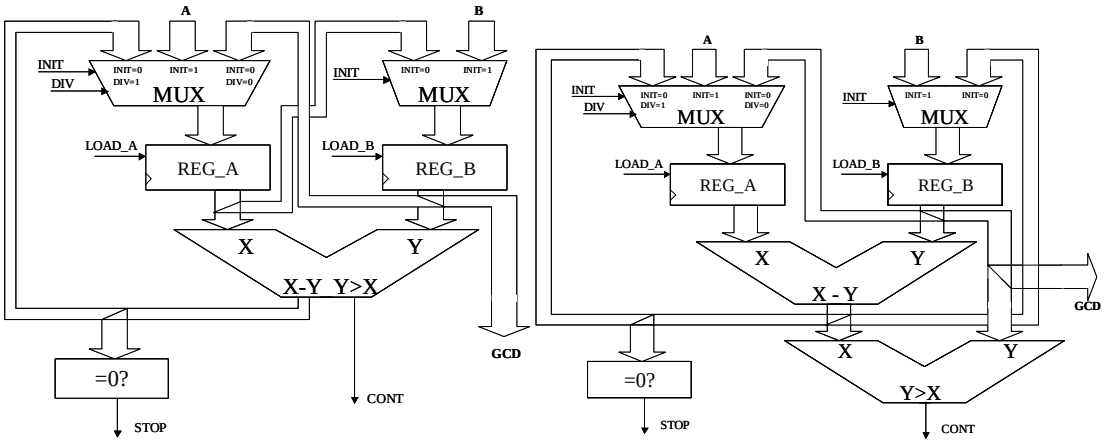
A Figura 4(a) apresenta uma arquitetura adequada para suportar a implementação do fluxograma apresentado na Figura 1(b) responsável pelo cálculo do resto da divisão inteira através de subtrações sucessivas; de modo semelhante, a Figura 4(b) ilustra um possível diagrama temporal associado.



(a) (b)

Figura 4 – Parte de dados II (a) Arquitectura associada à computação do resto
(b) Diagrama temporal exploratório.

Tendo como objectivo a realização da parte de dados de acordo com a Figura 1(c), considerando uma descrição não hierárquica, o arquitecto do sistema pode encontrar várias soluções para a parte de dados. A Figura 5(a) e a Figura 5(b) apresentam duas soluções possíveis para arquitecturas de partes de dados globais, integrando as apresentadas nas figuras anteriores.



(a) (b)

Figura 5 – Parte de dados III (a) Arquitectura global
(b) Arquitectura global alternativa.

Sugere-se a análise das arquitecturas apresentadas. Siga (justificadamente) uma delas na realização do seu trabalho prático.
Nas soluções analisadas até aqui, considerou-se que o operando A era maior que o operando B. Sugere-se que analise as alterações necessárias a realizar nas arquitecturas apresentadas para garantir um bom funcionamento independentemente dos valores de A e B.

Especificação de implementação da parte de controlo

Quanto à parte de controlo, deverá ser uma máquina de estados, síncrona, arquitectura de Moore recomendável... Na Figura 6 esboça-se uma possível solução que deverá utilizar como ponto de partida da sua análise.

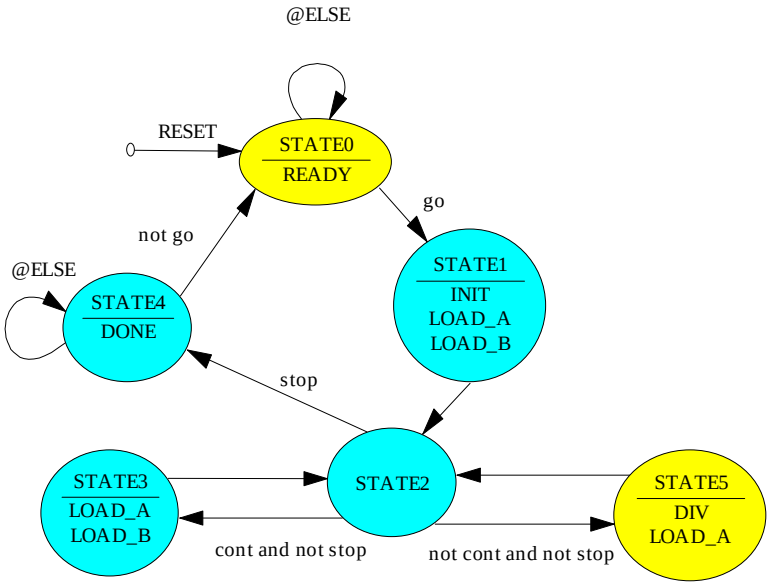


Figura 6 – Diagrama de estados responsável pelo controlo da parte de dados apresentada na Figura 5a).

Sugere-se, como exercício, a construção de um diagrama temporal associado.

Implementação da parte de dados (primeiro objetivo): utilização de esquemáticos

Partindo da decomposição do sistema em parte de dados e de controlo, caracterizada previamente, pretende-se que recorrendo às facilidades de edição hierárquica de esquemáticos do ISE, seleccionar os módulos adequados para a implementação da nossa parte de dados.

Tomando a solução apresentada na Figura 5a) para a parte de dados deverá obter algo próximo das seguintes figuras, onde se representam os módulos definidos pelo utilizador e os que são de biblioteca.

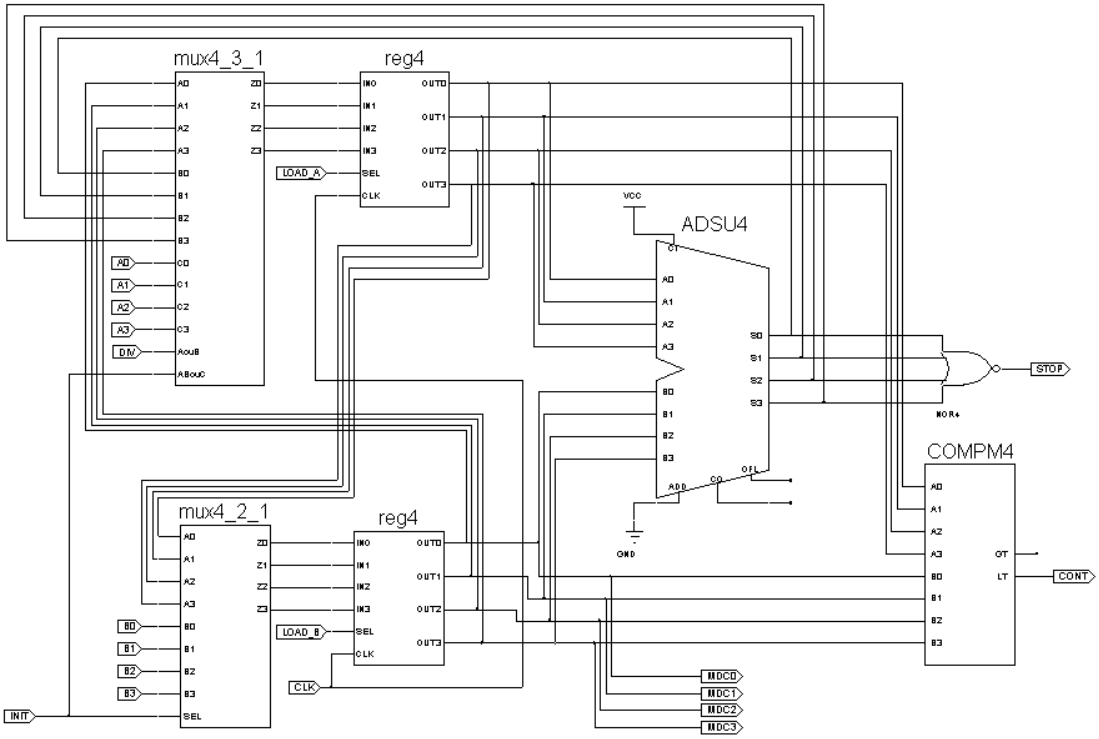


Figura 7 - Esquemático de topo de hierarquia de parte de dados.

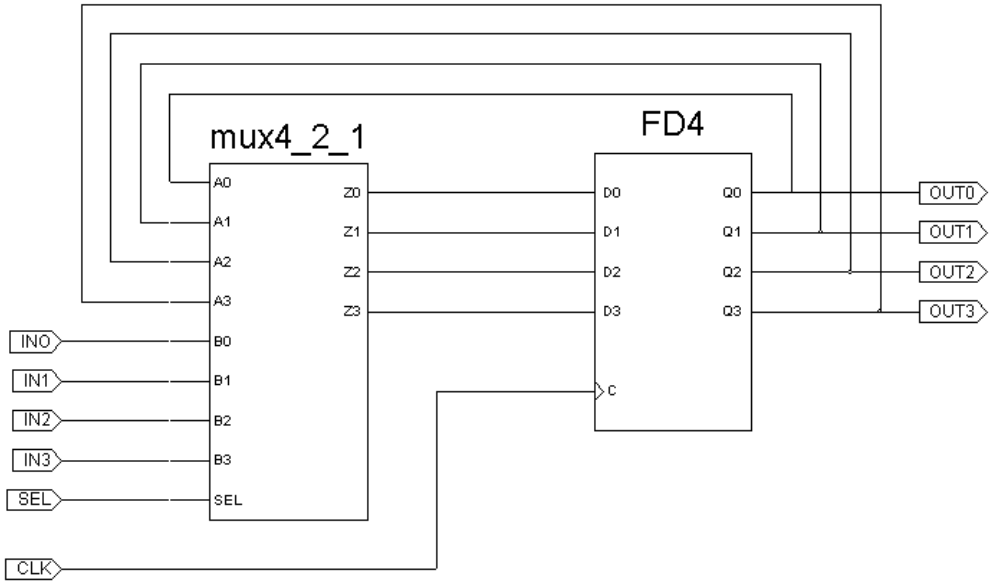


Figura 8 - Pormenor de implementação do módulo reg4.

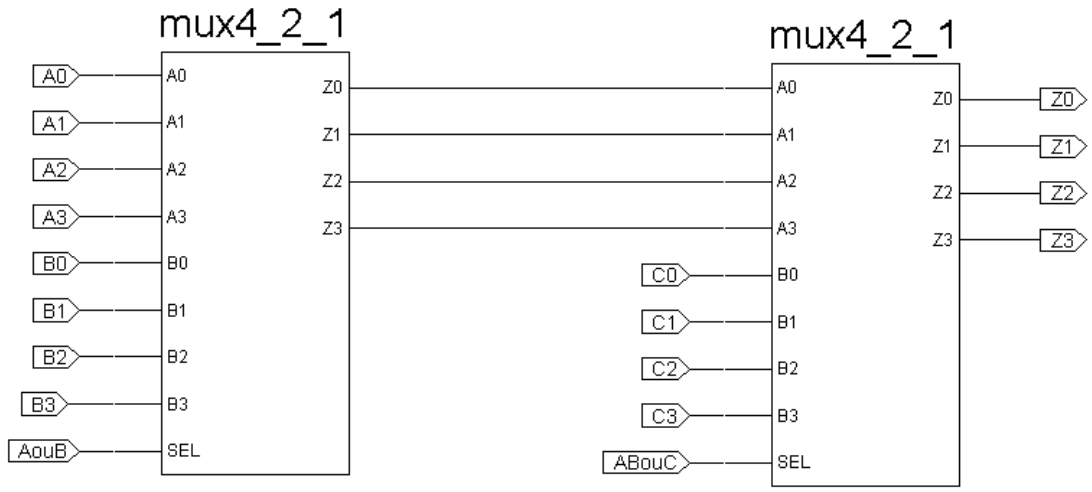


Figura 9 - Pormenor de implementação do módulo MUX4_3_1.

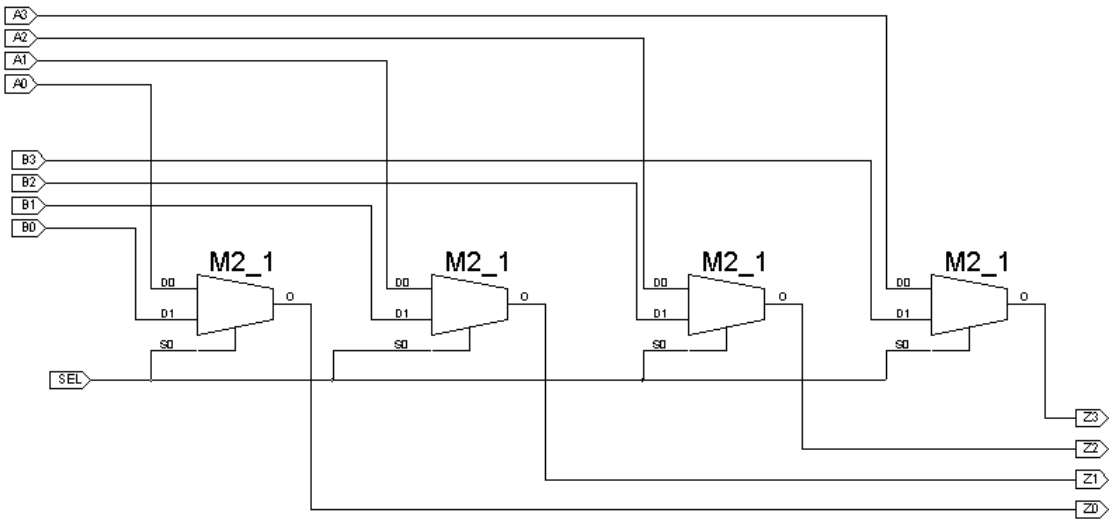


Figura 10 - Pormenor de implementação do módulo MUX4_2_1.

Recorrendo às funcionalidades de simulação disponíveis no ambiente da Xilinx, proceder à simulação da parte de dados especificada a partir dos esquemáticos.

Implementação da parte de dados (segundo objetivo): utilização de VHDL

Partindo da descrição da parte de dados apresentada previamente, proceda à substituição da parte de dados especificada com esquemáticos (objetivo anterior do trabalho) por um módulo especificado em VHDL.
Inicie o trabalho pela caracterização do interface do módulo VHDL, o que deverá ser complementado pela definição da arquitectura, identificando os diferentes módulos a especificar independentemente. O código VHDL obtido pode ser próximo do seguinte:

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

entity dados_mdc is
    Port ( clk : in std_logic;
           init : in std_logic;
           load_a : in std_logic;
           load_b : in std_logic;
           a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
           div : in std_logic;
           stop : out std_logic;
           cont : out std_logic;
           mdc : out std_logic_vector(3 downto 0));
end dados_mdc;

architecture Behavioral of dados_mdc is

signal reg_a, reg_b, resto : std_logic_vector(3 downto 0);

begin

process(clk)
begin
if clk'event and clk = '1' then
  if init = '1' then
        if load_b = '1' then
              reg_b <= b;
        end if;
  end if;
  if init = '0' then
        if load_b = '1' then
              reg_b <= reg_a;
        end if;
  end if;
end if;
end process;

process(clk)
begin
if clk'event and clk = '1' then
  if init = '1' then
        if load_a = '1' then
              reg_a <= a;
        end if;
  end if;
  if init = '0' then
        if div = '1' then
              if load_a = '1' then
                    reg_a <= resto;
              end if;
        else   if load_a = '1' then
              reg_a <= reg_b;
              end if;
        end if;
  end if;
end if;
end process;
resto <= reg_a - reg_b;
cont <= '1' when reg_b > reg_a else '0';
stop <= '1' when resto = 0 else '0';
mdc <= reg_b;

end Behavioral;
```

Recorrendo às funcionalidades de simulação disponíveis no ambiente da Xilinx, proceder à simulação da parte de dados especificada utilizando VHDL.

Implementação da parte de controlo (terceiro objetivo): utilização de VHDL

Pretende-se obter a descrição VHDL da parte de controlo.
Para isso deverá ter presente a Figura 6 traduzindo as dependências expressas em VHDL.
Sugere-se a utilização dos "Language Templates" disponíveis no ambiente da Xilinx (Edit → Language Templates → VHDL → Synthesis Constructs → Coding Examples → State Machines).
Nota: Caso pretenda obter a representação gráfica da Figura 6, bem como a geração automática do código VHDL associado poderá utilizar a ferramenta StateCAD, disponível na net (30 day evaluation copy).

Implementação do sistema (quarto objetivo)

Com base na definição de módulos associados à parte de dados e à parte de controlo, poderá obter a representação de sistema como a apresentada na Figura 11. Pretende-se construir a descrição VHDL que interliga as componentes de dados e controlo.

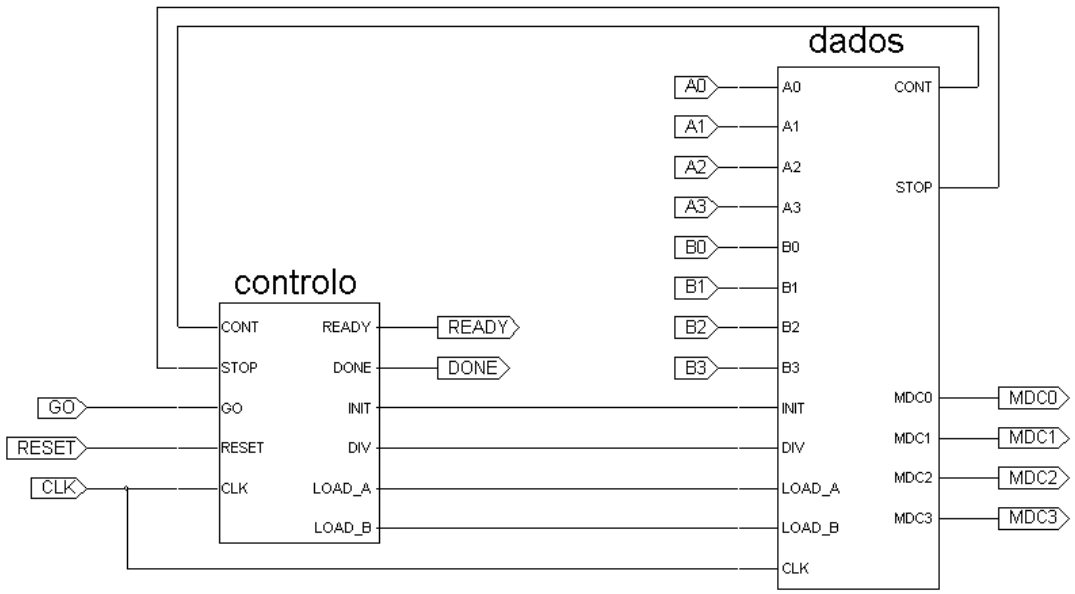


Figura 11 – Decomposição em parte de controlo e parte de dados.

Deverá obter algo próximo do seguinte código VHDL.

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

entity mdc_main is
    Port ( rst : in std_logic;
           clk : in std_logic;
```

```
           go : in std_logic;
           a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
           done : out std_logic;
           ready : out std_logic;
           mdc : out std_logic_vector(3 downto 0));
end mdc_main;

architecture Behavioral of mdc_main is

  COMPONENT controlo
  PORT(
         clk : IN std_logic;
         cont : IN std_logic;
         go : IN std_logic;
         reset : IN std_logic;
         stop : IN std_logic;
         div : OUT std_logic;
         done : OUT std_logic;
         init : OUT std_logic;
         load_a : OUT std_logic;
         load_b : OUT std_logic;
         ready : OUT std_logic
         );
  END COMPONENT;

  COMPONENT dados_mdc
  PORT(
         clk : IN std_logic;
         init : IN std_logic;
         load_a : IN std_logic;
         load_b : IN std_logic;
         a : IN std_logic_vector(3 downto 0);
         b : IN std_logic_vector(3 downto 0);
         div : IN std_logic;
         stop : OUT std_logic;
         cont : OUT std_logic;
         mdc : OUT std_logic_vector(3 downto 0)
         );
  END COMPONENT;

  signal cont, stop, div, init, l_a, l_b : std_logic;
begin

         UC: controlo PORT MAP(
         clk => clk,
         cont => cont,
         go => go,
         reset => rst,
         stop => stop,
         div => div,
         done => done,
         init => init,
         load_a => l_a,
         load_b => l_b,
         ready => ready
  );

  UD: dados_mdc PORT MAP(
         clk => clk,
         init => init,
         load_a => l_a,
         load_b => l_b,
         a => a,
         b => b,
         div => div,
         stop => stop,
         cont => cont,
```

```
            mdc => mdc
      );
    end Behavioral;
```

Recorrendo às facilidades de simulação do ISE, proceda à verificação do funcionamento do sistema.

Implementação física e teste (quinto objetivo)

Após a simulação do sistema deverá proceder à configuração de uma Xilinx Spartan3 contida no kit disponibilizado e verificar o funcionamento do circuito. Para informação complementar sobre o kit de desenvolvimento, sugere-se consulta dos materiais associados disponíveis na página www da disciplina.

DIVIRTAM-SE