Problem 1 – Greatest Common Divider computation using programmable logic devices

Goal

It is intended to analyze the operation of a system capable of calculating the greatest common divider between two numbers A and B (to be implemented through programmable logic devices of the FPGA type). The two numbers must be presented in binary, using the switches installed in the kits available in the laboratory. A and B are two numbers of 4 bits each (the groups can consider different values, but from the point of view of the implementation it is recommended 4 bits due to constraints of experimentation equipment/libraries).

The proposed computation procedure is based on the Euclidean algorithm, briefly described later.

After actuating the input switches to provide the numbers A and B, the user gives the order for computation of the greatest common divider, at which point the system supplies the result and activates a specific output signaling that the calculation is complete.

The order to start calculation is performed through the actuation of an input variable (GO). In addition, another input for system initialization (RESET) is available.

The purpose of the first part of the problem is to introduce and start using the Xilinx ISE development environment for CPLDs and FPGAs. For this, a schematic-based representation will be used, as well as the available simulation resources.

The purpose of the second part of the problem is to introduce the use of the VHDL language in the specification of modules from the Xilinx ISE development environment for CPLDs and FPGAs.

Finally, in the third part of the problem, the objective is to use VHDL to carry out the system simulation specification.

It is intended to perform the physical implementation and testing of the system along the various phases of the work, using the experimentation kits available in the laboratory. Kits can be ordered by the students' groups for use in and out of class.

Throughout this document, possible solutions that each group should concretize are sketched. The present document can be "improved" whenever the group considers it justified; whenever it is considered that the document is not explicit or is ambiguous, the group can define the best way to overcome the limitation.

Description

The Euclidean algorithm can be described generically through the flowchart of Figure 1 (a), that is, assuming that A is greater than B, the determination of the greatest common divider is based on the calculation of the remainder of the division of A and B; where the remainder is non-zero, A is replaced by B and B by the remainder, and the rest of the division is re-calculated; when the rest is null, then the greatest common divider is equal to B.

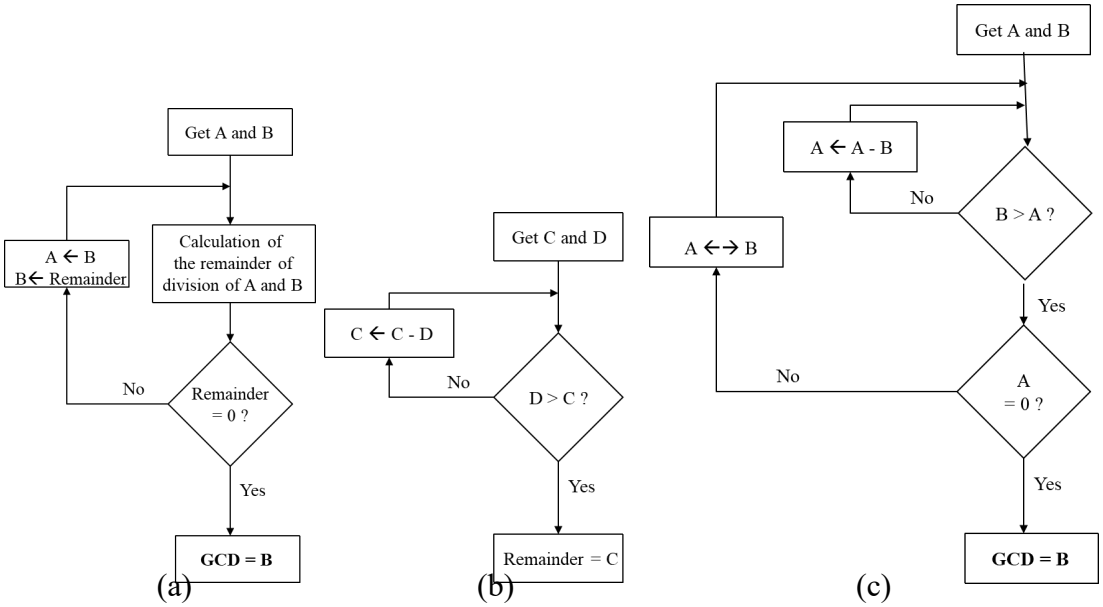


Figure 1 – (a) Flowcharts of Euclid's algorithm; (b) Flowchart for calculating the remainder of the division by successive subtraction.

In the described algorithm, the clearly more complex part is the calculation of the remainder of the division of A through B. A possible procedure for its implementation can be obtained by successive subtractions, as described in the flowchart of Figure 1 (b).
In this way, the description of the procedure for calculating the common maximum divider uses a hierarchically organized description, containing two levels.
The integration of the two levels leads to the algorithm shown in Figure 1 (c) where it is expected to exchange the values of A and B (as atomic operation).

Analysis and preparation of implementation

The analysis of the following system should be regarded as a suggestion; the students groups are free to propose and implement alternative characterizations (or resulting from optimizations to the suggestions presented).
To allow the implementation of the system, various attitudes can be taken by the designer, for example:

- take the hierarchical algorithmic characterization with two levels presented above and implement the two levels as separate but interdependent components, guaranteeing the hierarchical functional dependence between the two components;
- obtain a global characterization for the system, containing only one level in the description to use.

Consider as a starting point a generic characterization of the system based on a decomposition of the system into a control part and a data part. For the specific case of the system to be implemented, a possible decomposition (to be explained later) is presented in Figure 2, where:

- the control part, implemented through a synchronous sequential circuit, under the control of a clock signal (represented by CLK in Figure 2), has two external input signals (RESET and GO), which will enable the user to

initialize the system and indicate the start of the operation, two outputs to signal "start of use" and "end of operation" and a set of inputs and outputs that will allow interconnection to the data part;

- the data part has data inputs (both numbers A and B), a data output (the greatest common divider of A and B), and a set of inputs and outputs allowing interconnection to the control part . The memory elements required for its implementation receive the same clock signal used by the control part (synchronous solution).

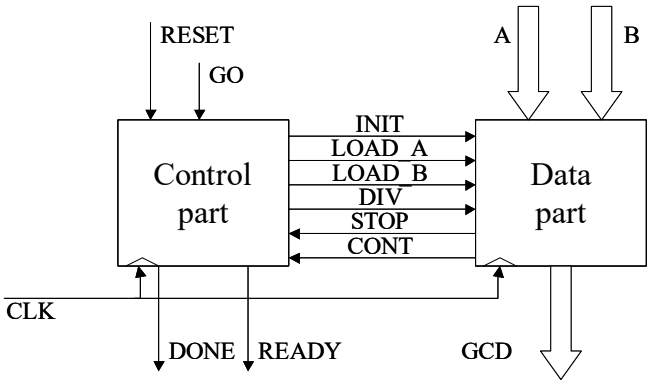


Figure 2 – Decomposition into control and data parts.

If you want to consider an implementation based on the two hierarchically organized components, you need to consider for each of the components a similar decomposition, in control and data parts. The suggestion is left as an additional exercise.

In the following paragraphs, the functions associated with each of the signals mentioned in Figure 2 are briefly described:

- External inputs for the control part::
  - ✓ RESET to initialize the system (affects only the control part, which will be responsible for initializing the data part);
  - ✓ GO to indicate that it is intended to perform a calculation of the greatest common divisor;

- Outputs from the control part:
  - ✓ READY, intended to signal "start-up", ie waiting for calculation order;;
  - ✓ DONE, intended to signal "end of operation" and that the result is available;

- External inputs for the data part:
  - ✓ A and B, the two numbers for which the common maximum divider is to be calculated; number of bits: 4 each(typical);

- Data outputs:
  - ✓ GCD (greatest common divisor), is the highest common divisor;

- Inputs for the data part from the control part:

- ✓ INIT, LOAD_A, LOAD_B and DIV, specifying control signals (see time diagrams and associated description);

➢ Inputs to the control part from the data part:
- ✓ STOP, indicating that the common maximum divider has been found,
- ✓ CONT, indicating that the division is not yet complete.

Implementation specification of the data part

For the data part, the use of an inter-register transfer architecture is (usually) recommended. We need to start specifying this part, and the control part later.
Figure 3 (a) presents an architecture that will allow computing the greatest common divider, with Figure 1 (a) as a reference, that is, taking only the top-level algorithmic representation. Figure 3 (b) shows a hypothetical time evolution diagram of the present signals, considering that the values of 14 and 4 are given for A and B, and that the LOAD_A, LOAD_B and INIT signals are as shown. In the time diagram the zones of variation of the various signals are represented, as well as the zones where the signal is known (0 or 1) and those in which a value is not guaranteed (in the present example).

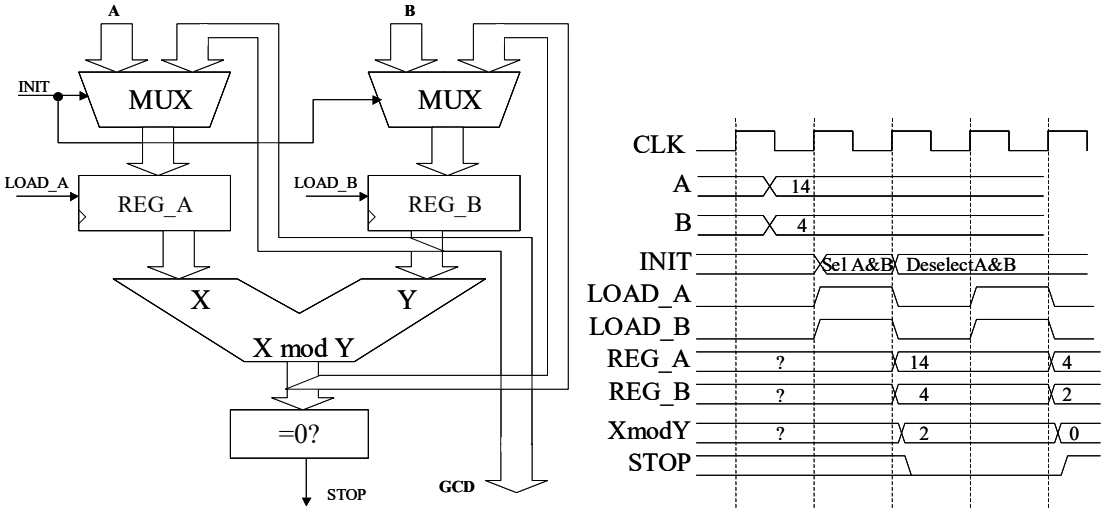
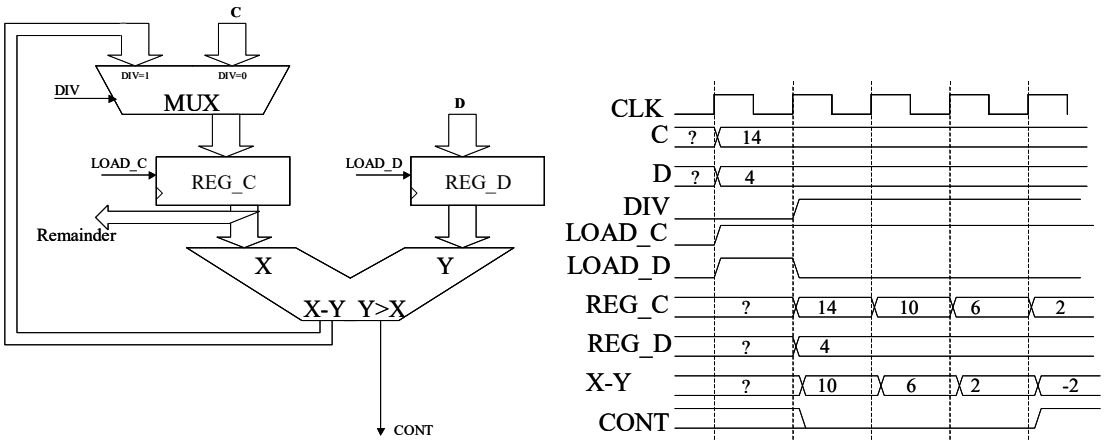


Figure 3 –Data part I (a) Architecture associated with the high-level description of Euclid's algorithm (b) Exploratory timing diagram.

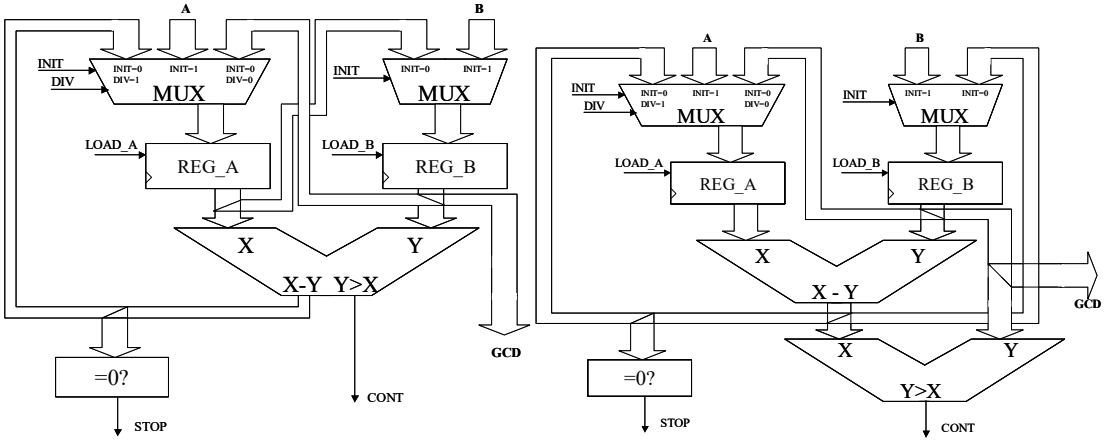
Figure 4 (a) presents a suitable architecture to support the implementation of the flowchart shown in Figure 1 (b) responsible for calculating the rest of the entire division through successive subtractions; Similarly, Figure 4 (b) illustrates a possible associated timing diagram.



(a) (b)

Figure 4 – Data part II (a) Architecture associated to the remainder computation. (b) Exploratory timing diagram.

In order to implement the data part according to Figure 1 (c), considering a non-hierarchical description, the system's designer can propose several solutions for the data part. Figure 5 (a) and Figure 5 (b) present two possible solutions for architectures of global data parts, integrating those presented in the previous figures.



(a) (b)

Figure 5 – Data part III (a) Global architecture (b) Alternative global architecture.

It is suggested the analysis of the presented architectures. Follow (and justify) one of them in the accomplishment of your practical work.
In the solutions analyzed so far, it was considered that the operand A was larger than the operand B. It is suggested to analyze the necessary changes to be made in the presented architectures to guarantee a good operation independently of the values of A and B.

Implementation specification of the control part

For the control part, it should be a state machine, synchronous, Moore's architecture... Figure 6 outlines a possible solution that should use as a starting point for its analysis.

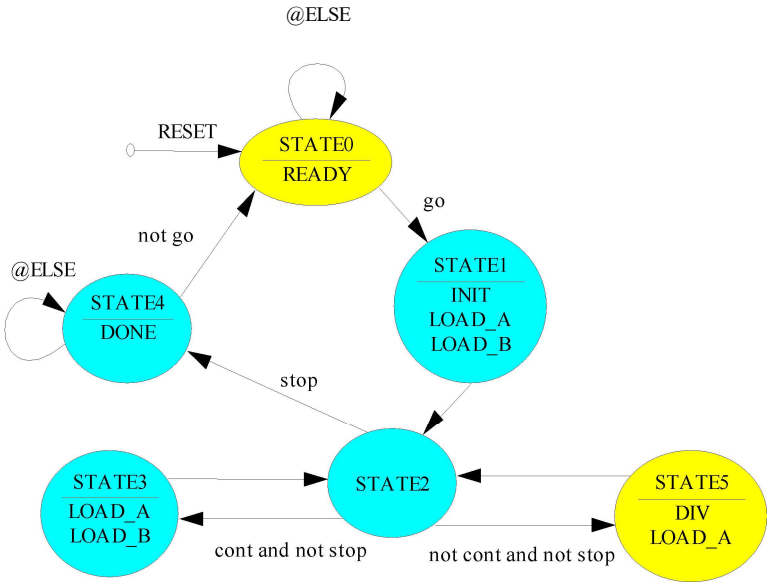


Figure 6 – State diagram responsible for controlling the data part shown in Figure 5a).

It is suggested, as an exercise, the construction of an associated timing diagram.

Implementation of the data part (first objective): use of schematics

Starting from the previously characterized decomposition of the system into data and control parts, it is intended that by using the hierarchical schematics editing facilities of ISE, select the appropriate modules for the implementation of our data part.

Taking the solution presented in Figure 5a) for the data part should get something close to the following figures, where the user-defined modules and those of the library are represented.

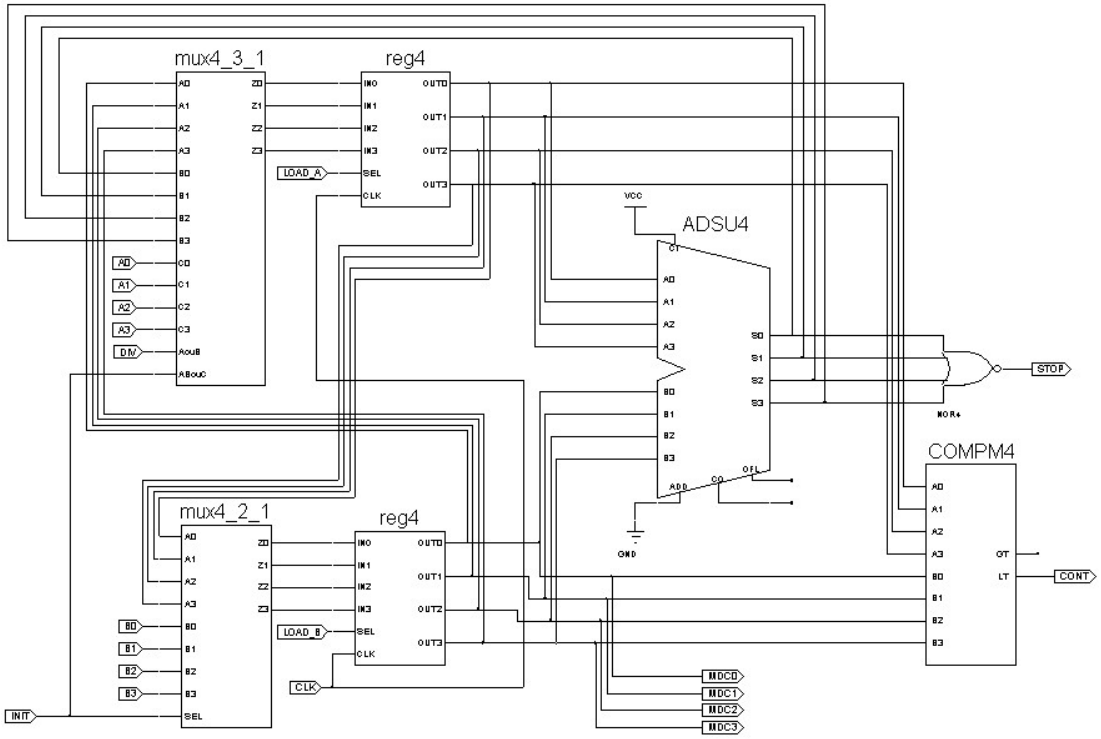


Figure 7 - Top hierarchy schematic of the data part.

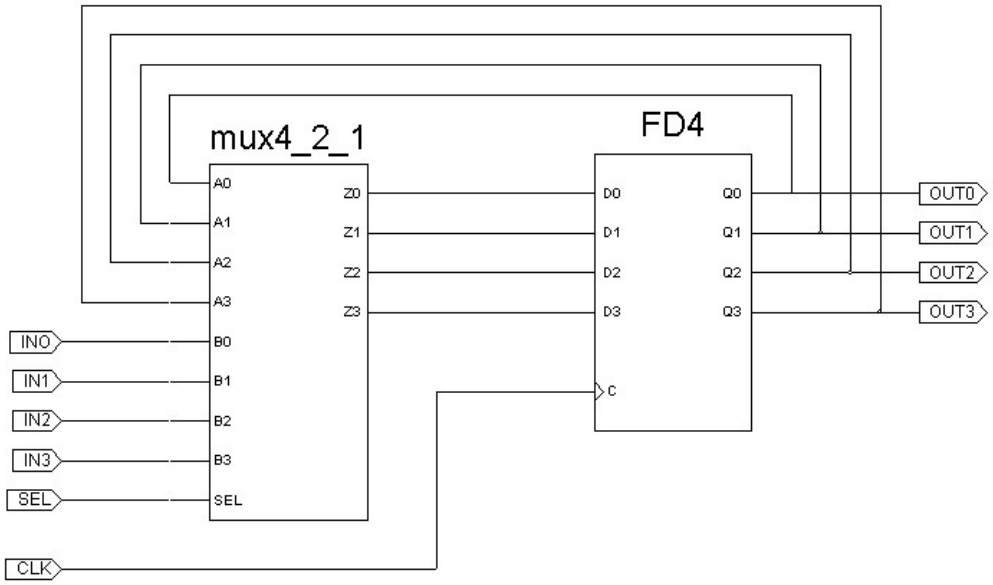


Figure 8 – Implementation of the module reg4.

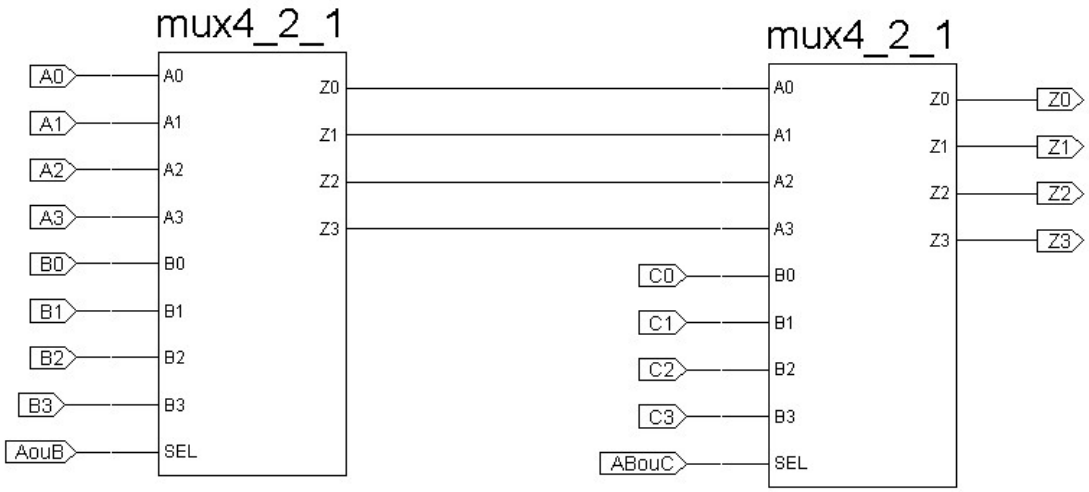


Figure 9 – Implementation of the module MUX4_3_1.

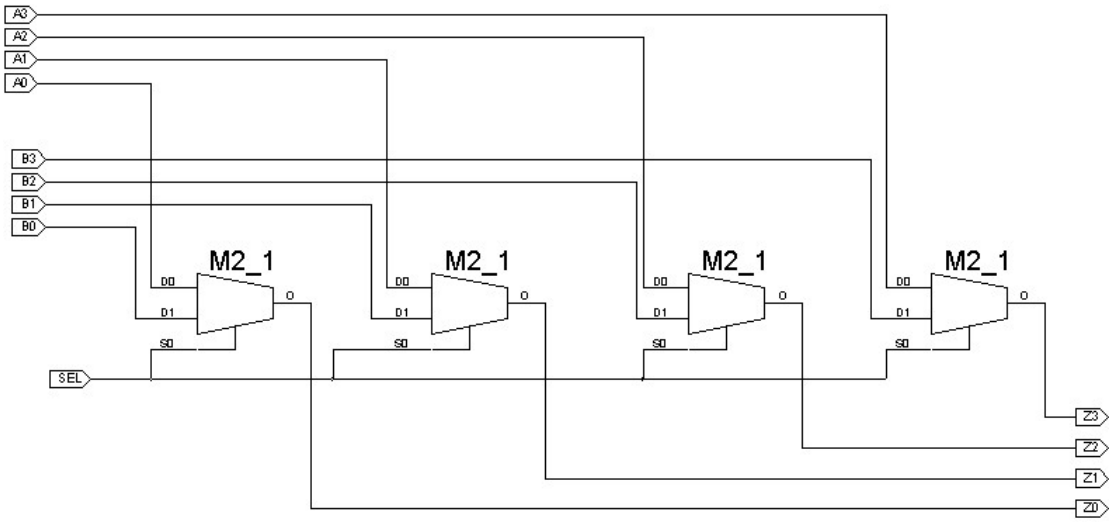


Figure 10 – Implementation of the module MUX4_2_1.

Using the simulation functionalities available in the Xilinx environment, simulate the data part specified from the schematic.

Implementation of the data part (second objective): use of VHDL

From the description of the data part presented previously, substitute the specified data part with schematics (previous work objective) for a module specified in VHDL. Start the work by characterizing the interface of the VHDL module, which should be complemented by the architecture definition, identifying the different modules to be specified independently. The VHDL code obtained may be close to the following:

```
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;
```

Using the simulation functionalities available in the Xilinx environment, simulate the data part specified using VHDL..

Implementation of the control part (third objective): use of VHDL

It is intended to obtain the VHDL description of the control part.
To do this, you should keep in mind Figure 6, translating the dependencies expressed in VHDL.
It is suggested to use the "Language Templates" available within Xilinx ISE framework (Edit → Language Templates → VHDL → Synthesis Constructs → Coding Examples → State Machines).

System implementation (fourth objective)

Based on the definition of modules associated with the data part and the control part, you can obtain the system representation as shown in Figure 11. It is intended to construct the VHDL description that interconnects the data and control components.

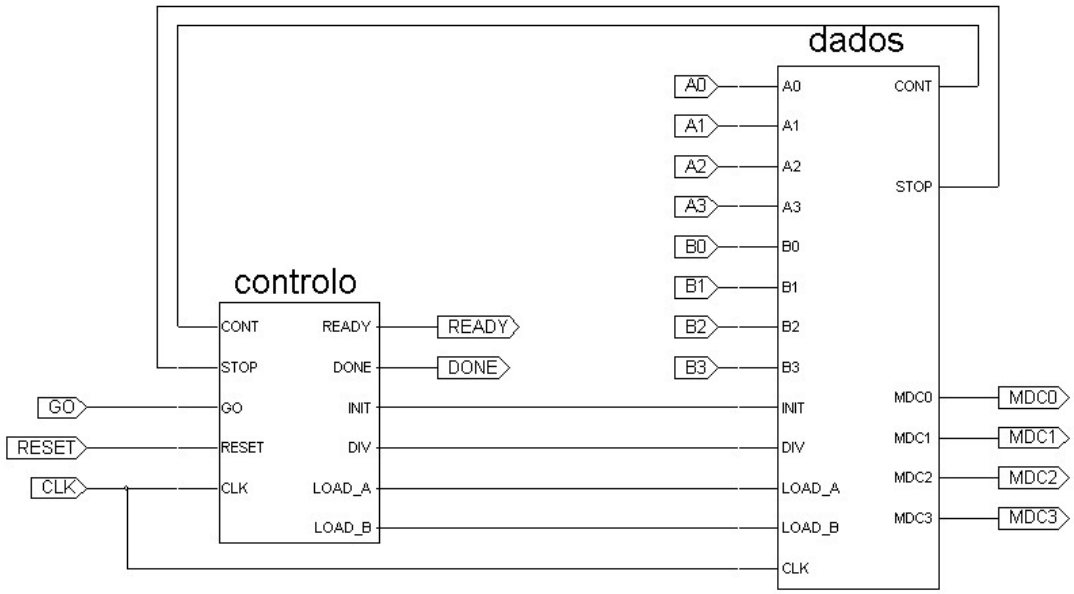


Figure 11 – Decomposition into control and data parts.

You should get something next to the following VHDL code.

```
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;
```

Using the simulation facilities of the ISE framework, check the operation of the system..

Physical implementation and testing (fifth objective)

After simulation of the system, proceed to the configuration of a Xilinx Spartan3 contained in the kit available and check the operation of the circuit. For additional information about the development kit, it is suggested to consult the associated materials available at the moodle page of the course.

HAVE FUN