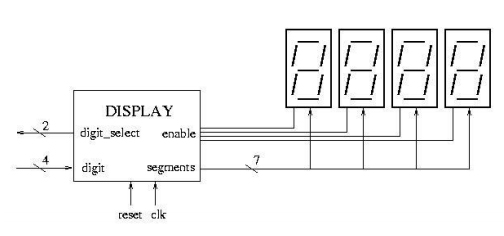
**Controller for a 7-segment Display**

***Year 2018-19***

**Riccardo Polini**

**Introduction**

The purpose of this project was to implement the description of a controller for a 7-segment display using VHDL code. Furthermore, it was required to synthetize the project by using the *Vivado* Software.



A user selects which one of the four segments to use through the **digit\_select** input and then alters the **digit** input in order to display the required number.

The controller will read the **digit\_select** input and correctly enable the required display before setting it through the **segments** output line.

The conversion between the BCD digit and the segments encoding happens according to the following truth table, found in most Electronics textbooks:

|  |  |  |
| --- | --- | --- |
| **Decimal number** | **digit** | **segments** |
| 0 | 0000 | 1111110 |
| 1 | 0001 | 0110000 |
| 2 | 0010 | 1101101 |
| 3 | 0011 | 1111001 |
| 4 | 0100 | 0110011 |
| 5 | 0101 | 1011011 |
| 6 | 0110 | 1011111 |
| 7 | 0111 | 1110000 |
| 8 | 1000 | 1111111 |
| 9 | 1001 | 1111011 |
| x | others | 0000000 |

As for the enable signal, the truth table is the following:

|  |  |
| --- | --- |
| **Digit\_select** | **enable** |
| 00 | 1000 |
| 01 | 0100 |
| 10 | 0010 |
| 11 | 0001 |

**Architecture**

The architecture for this system is very simple. The inputs are received and processed according to the truth tables in order to produce the **enable** and the **segments** signals. These are then fed to a D-FlipFlop barrier before being sent out of the network.

The DFF barrier has been placed in order to allow the network to update its outputs only when the *clock* signal is on the rising edge.

Inserire schema elettrico architettura

**Code Description**

Declaration of the various inputs/outputs:

|  |
| --- |
| entity controller is  port (  clk : in std\_logic;  reset : in std\_logic;  digit : in std\_logic\_vector(3 downto 0); --BCD input  digit\_select : in std\_logic\_vector(1 downto 0);  enable : out std\_logic\_vector(3 downto 0);  segments : out std\_logic\_vector(6 downto 0) -- 7 bit decoded output.  --segments\_out : out std\_logic\_vector(6 downto 0) -- 7 bit decoded output.  );  end controller; |

Definition of the two D-FlipFlop barriers that will hold the outputs of the controller until the rising edge of the clock arrives:

|  |
| --- |
| component dff\_N is  generic (N: integer);  port (  resetn : in std\_logic;  clk : in std\_logic;  d\_in : in std\_logic\_vector(N-1 downto 0);  q\_out : out std\_logic\_vector(N-1 downto 0)  );  end component;  component dff\_M is  generic (M: integer);  port (  resetn : in std\_logic;  clk : in std\_logic;  d\_in : in std\_logic\_vector(M-1 downto 0);  q\_out : out std\_logic\_vector(M-1 downto 0)  );  end component; |

Behaviour of the controller. An internal signal is used to assign the binary code to the output variables **segments** and **enable**.

|  |
| --- |
| begin  segments\_signal <="1111110" when digit = "0000" else -- '0'  "0110000" when digit = "0001" else  "1101101" when digit = "0010" else  "1111001" when digit = "0011" else  "0110011" when digit = "0100" else  "1011011" when digit = "0101" else  "0011111" when digit = "0110" else  "1110000" when digit = "0111" else  "1111111" when digit = "1000" else  "1110011" when digit = "1001" else "0000000";  segments <= segments\_signal;  enable\_signal <="1000" when digit\_select = "00" else  "0100" when digit\_select = "01" else  "0010" when digit\_select = "10" else  "0001" when digit\_select = "11";  enable <= enable\_signal;  end; |

**Test-bench and Output**

In the *testbench file* I map the various components with the respective testbench signals:

|  |
| --- |
| begin  controller\_map:  entity work.controller  port map (  clk =>clk\_tb,  reset =>rst\_tb,  digit => digit\_tb,  digit\_select => digit\_select\_tb,  enable => enable\_tb,  segments => segments\_tb  );  dff\_map:  entity work.dff\_N(struct)  generic map(N => 7)  port map (d\_in => segments\_tb, q\_out => output\_segments\_tb, clk => clk\_tb, resetn => rst\_tb);  dff\_map2:  entity work.dff\_N(struct)  generic map(N => 4)  port map (d\_in => enable\_tb, q\_out => output\_enable\_tb, clk => clk\_tb, resetn => rst\_tb); |

In order to test the behavior of the controller, I wrote a loop in which the input **digit** is incremented from 0 to 9 and the **digit\_select** input is also changed. This way I can verify all the configurations.

|  |
| --- |
| begin  for i in 0 to 9 loop  digit\_tb <= conv\_std\_logic\_vector(i,4);  digit\_select\_tb <= conv\_std\_logic\_vector(i mod 4,2);  wait for T\_CLK/3;  end loop; |



The above plot shows the result of a simulation of the controller. It is immediate to note that the outputs are updated only on the rising edge of the clock. Also, the correctness of the controller is easily verified by double-checking the outputs with the truth table described in paragraph 1.

**Synthesis**

**Conclusions**