Relatório Prática 05 – 22/09/2023
Arthur Souza/João Paulo – PN1

Iniciou a prática criando uma entidade nomeado FlipFlopD no software QUARTUS II 13.0, especificando qual chip está sendo utilizado, no caso o EP2C35F672C6.

A partir disso, criou o primeiro arquivo em VHDL de mesmo nome que a entidade no qual foi colado uma parte do código disponibilizado:

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;

entity FlipFlopD is
        port( clock: in std_logic;
                D: in std_logic;
            Q: out std_logic
           );
end FlipFlopD;

architecture RTL of FlipFlopD is
begin
        process(clock)
        begin
                if (clock='1' and clock'event) then
                        Q <= D;
                end if;
        end process;
end RTL;
```

Utilizou-se a estrutura logica WHEN-ELSE em architecture para alterar a descrição de comportamental para fluxo de dados, como visto abaixo:

```
architecture RTL of FlipFlopD is begin
        Q <= '0' when(reset='0') else
        D when(clock='1' and clock'event);
end RTL;
```

Simulado e atestado ausência de erros, verificou-se a esquematização do código em circuito pela opção RTL viewer e Technology Map Viewer. Segue abaixo as imagens:

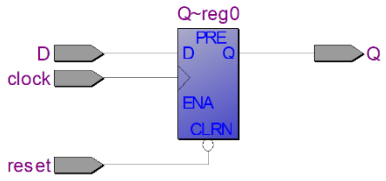

Figura 1: Circuito FlipFlopDr

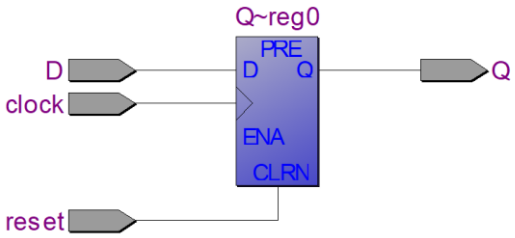

Figura 2: Diagrama do circuito FlipFlopD

Seguiu-se para a compilação do testbench criado, este chamado de tb_FlipFlopD, e que irá definir os testes do projeto. Ele está descrito abaixo:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tb_FlipFlopD is
end tb_FlipFlopD;

architecture teste of tb_FlipFlopD is

component FlipFlopD is
        port(
                        clock: in std_logic;
                        D: in std_logic;
                        reset: in std_logic;
              Q: out std_logic
             );
end component;

signal fio_Cl, fio_D, fio_Re, fio_Q: std_logic;

begin

-- Note que o componente é instanciado com apenas 4 bits nas entradas para facilitar a simulação:
instancia_FlipFlopD: FlipFlopD port map(clock=>fio_Cl, D=>fio_D, reset=>fio_Re, Q=>fio_Q);

-- Dados de entrada de 4 bits são expressos em "hexadecimal" usando "x":
fio_Cl<='0', '1' after 50ns,'0' after 100ns, '1' after 150ns,'0' after 200ns,'1' after 250ns,'0' after 300ns;
fio_D<='0','1' after 25ns,'0' after 75ns, '1' after 175ns;
fio_Re<='0','1' after 2ns, '0' after 200ns;
end teste;
```

Com ele, podemos simular o funcionamento do circuito através do software MULTISIM. Basta apenas indica o arquivo como testbench em simulation e começar a simulação em RTL. Irá abrir um gráfico de sinais com valores determinados pelo testbench e irá auxiliar na verificação da lógica.

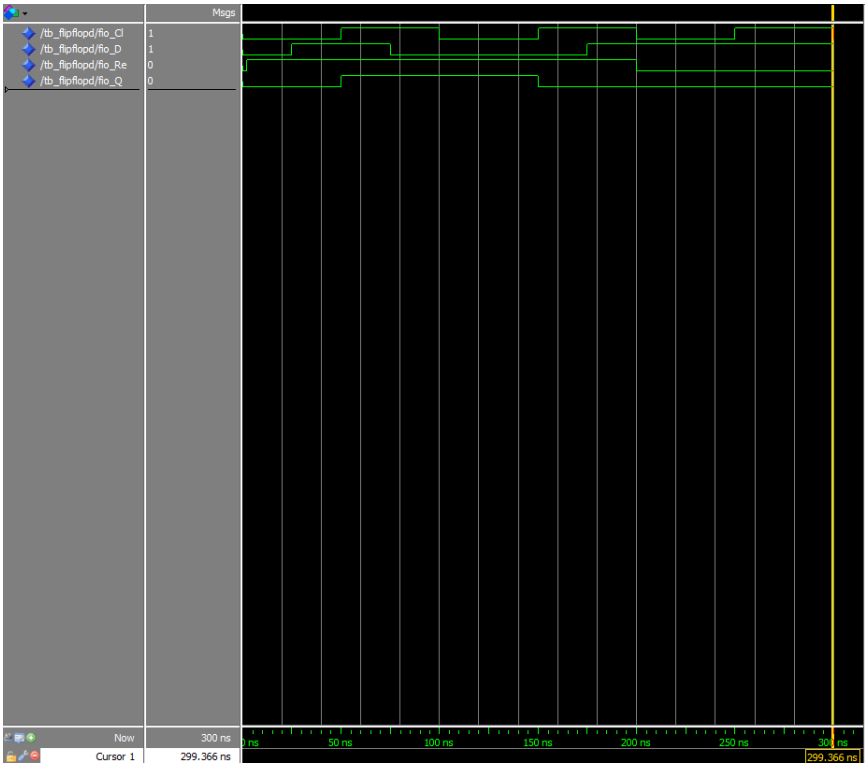

Figura 3: Simulação FlipFlopD no Multisim

Em seguida, foi feito o circuito fulladder de 4 bits, esse criado a partir de um fulladder com descrição de fluxo de dados disponibilizado no sistema, que segue abaixo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity fulladder is
   port ( Cin : in std_logic;
          x   : in std_logic;
          y   : in std_logic;
          s   : out std_logic;
          Cout : out std_logic
        );
end fulladder;
architecture RTL OF fulladder is
begin
   s <= x XOR y XOR Cin;
   Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
end RTL ;
```

O novo código, agora em 4 bits e com descrição comportamental, segue abaixo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity fulladder is
        generic
        (
                DATA_WIDTH : natural :=4
        );
        port (
                        Cin : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                        x   : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                        y   : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                        s   : out std_logic_vector ((DATA_WIDTH-1) downto 0);
                        Cout : out std_logic_vector ((DATA_WIDTH-1) downto 0)
);
end fulladder;
architecture RTL OF fulladder is
begin
        process(x, y, cin)
        begin
   s <= x XOR y XOR Cin;
   Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
        end process;
end RTL ;
```

Simulado e atestado ausência de erros, verificou-se a esquematização do código em circuito pela opção RTL viewer e Technology Map Viewer. Segue abaixo as imagens:

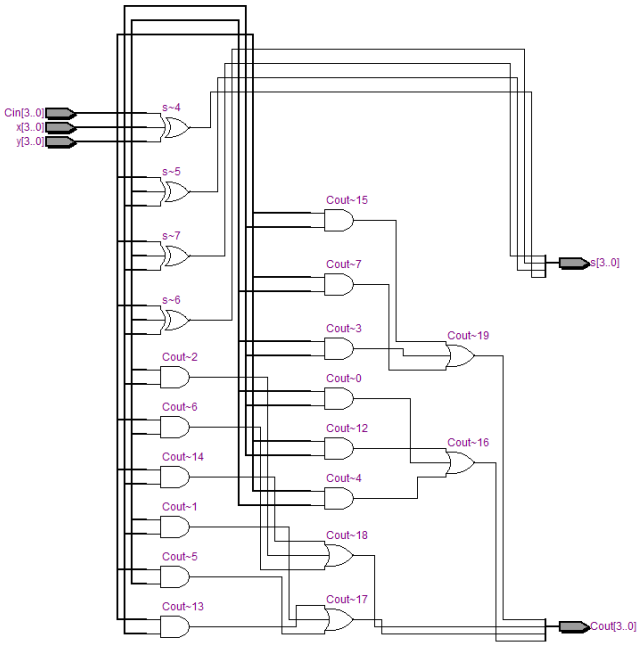

Figura 4: Circuito FullAdder 4bits

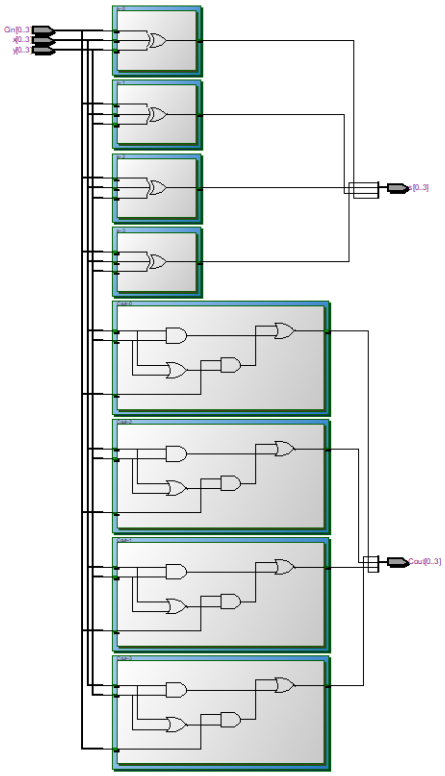

Figura 5: Diagrama circuito FullAdder 4bits

Seguiu-se para a compilação do testbench criado, este chamado de tb_fulladder, e que irá definir os testes do projeto. Ele está descrito abaixo:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity tb_fulladder is
end tb_fulladder;
architecture teste of tb_fulladder is
component fulladder is
                    generic
                    (DATA_WIDTH : natural :=4
                    );
                    port (
                            Cin : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                            x   : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                            y   : in std_logic_vector ((DATA_WIDTH-1) downto 0);
                            s   : out std_logic_vector ((DATA_WIDTH-1) downto 0);
                            Cout: out std_logic_vector ((DATA_WIDTH-1) downto 0)
    );
end component;
signal fio_Cin: std_logic_vector (3 downto 0);
signal fio_x: std_logic_vector (3 downto 0);
signal fio_y: std_logic_vector (3 downto 0);
signal  fio_s, fio_Cout: std_logic_vector (3 downto 0);
begin
instancia_fulladder: fulladder generic map (DATA_WIDTH => 4) port map(Cin=>fio_Cin,
x=>fio_x, y=>fio_y, s=>fio_s,Cout=>fio_Cout);
fio_Cin <= x"0", x"2" after 30ns, x"3" after 70ns, x"7" after 100ns;
fio_x <= x"0", x"5" after 50ns, x"3" after 90ns, x"7" after 110ns;
fio_y <= x"0", x"4" after 40ns, x"5" after 100ns, x"7" after 150ns;
end teste;
```

Com ele, podemos simular o funcionamento do circuito através do software MULTISIM. Basta apenas indica o arquivo como testbench em simulation e começar a simulação em RTL. Irá abrir um gráfico de sinais com valores determinados pelo testbench e irá auxiliar na verificação da lógica.

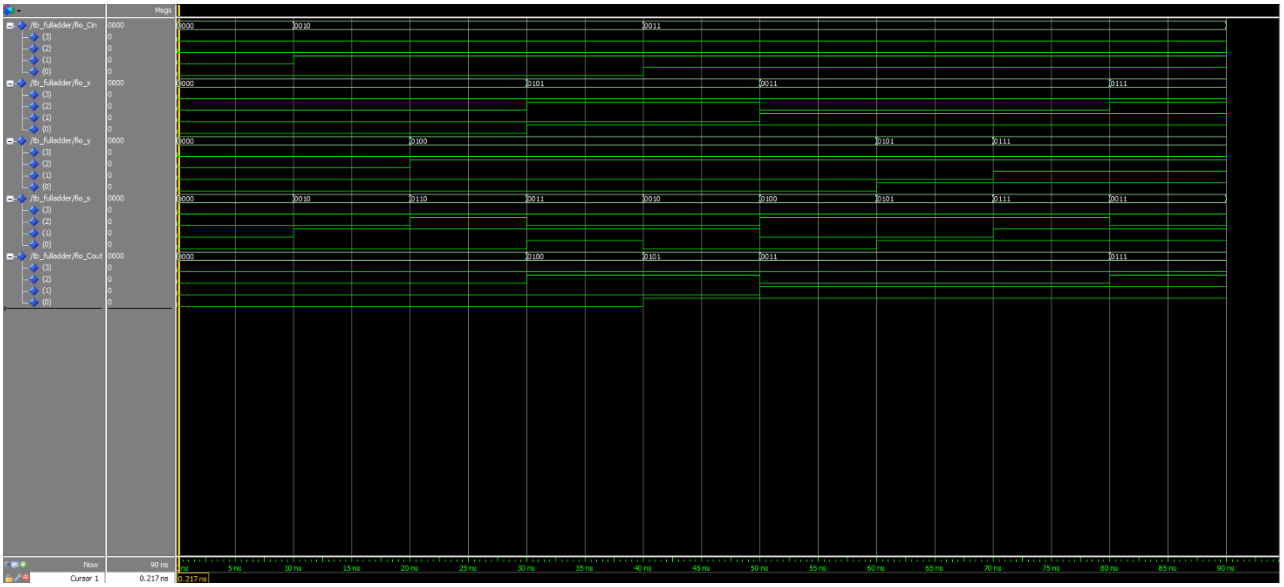

Figura 6: Simulação FullAdder 4bits no Multisim

Foi utilizado o arquivo csv do comparador para facilitar a pinagem desse projeto.
x[3] <= PIN_N25
x[2] <= PIN_N26
x[1] <= PIN_P25
x[0] <= PIN_AE14

y[3] <= PIN_AC13
y[2] <= PIN_C13
y[1] <= PIN_B13
y[0] <= PIN_A13

cin[3] <= PIN_P2
cin[2] <= PIN_T7
cin[1] <= PIN_U3
cin[0] <= PIN_U4

cout[3] <= PIN_AE23
cout[2] <= PIN_AF23
cout[1] <= PIN_AB21
cout[0] <= PIN_AC22

s[3] <= PIN_AD21
s[2] <= PIN_AC21
s[1] <= PIN_AA14
s[0] <= PIN_Y13

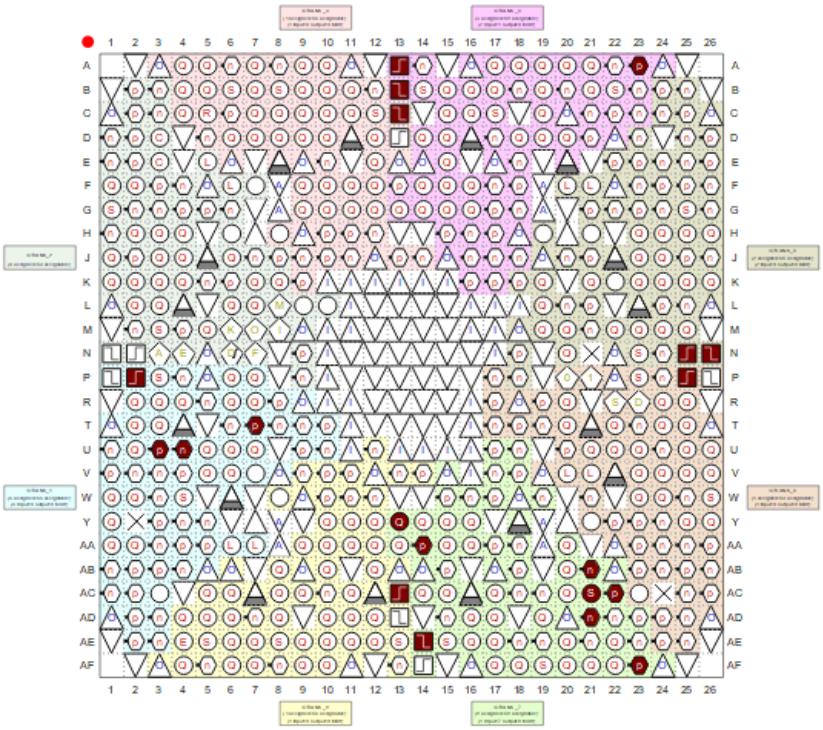

Figura 7: Pin Planner

Simulando novamente para verificação dos pinos, podemos enviar o projeto para o FPGA usando a função PROGRAMMER e o usb blaster.

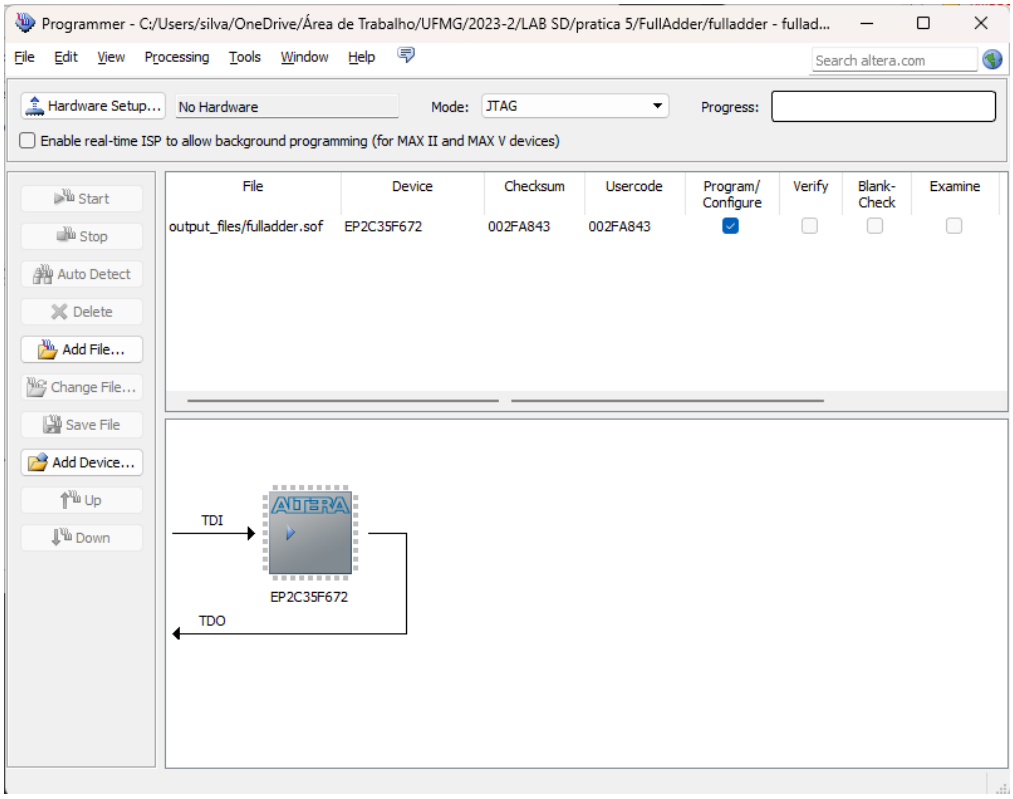

Figura 8: Programmer