Relatório Prática 03 – 01/09/2023
Arthur Souza/João Paulo – PN1

Iniciou a prática criando uma entidade nomeado Comparador 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 do testbench disponibilizado e adicionado uma seção para realizar comparação das palavras considerando-as com complemento de 2:

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


entity Comparador is

	generic
	(
		DATA_WIDTH : natural := 16
	);

	port
	(
		a	: in std_logic_vector  ((DATA_WIDTH-1) downto 0);
		b	: in std_logic_vector  ((DATA_WIDTH-1) downto 0);
		maior  : out std_logic;
		menor : out std_logic;
		igual   : out std_logic
	);
end Comparador;

architecture dataflow of Comparador is
begin
  igual<= '1' when signed(a)=signed(b) else '0';
	maior<= '1' when signed(a)>signed(b) else '0';
	menor<= '1' when signed(a)<signed(b) else '0';
end dataflow;
```

Utilizou-se a estrutura logica WHEN-ELSE para comparação de cada palavra.

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:

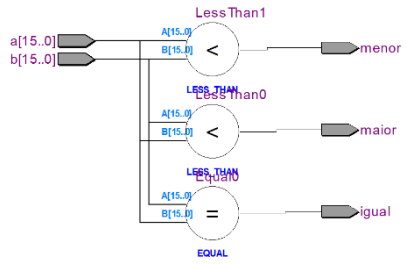

Figura 1: Circuito Comparadorr

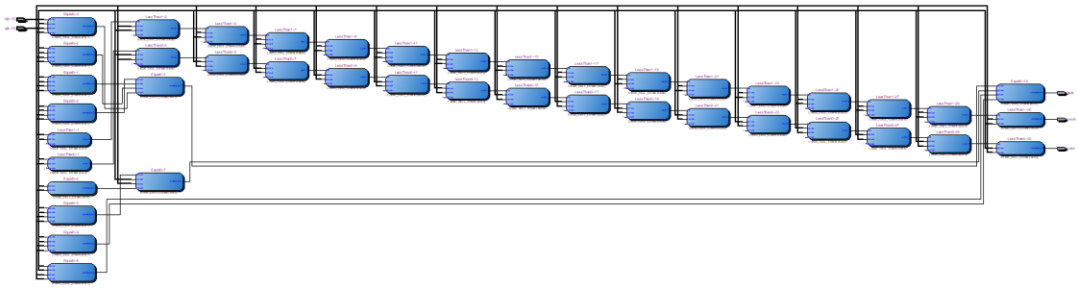

Figura 2: Diagrama do circuito Comparador

No map viewer foi possível observar a organização interna de cada bloco comparador, como visto no exemplo abaixo, inclusive vendo o nível de complexidade para um circuito de 16bits.

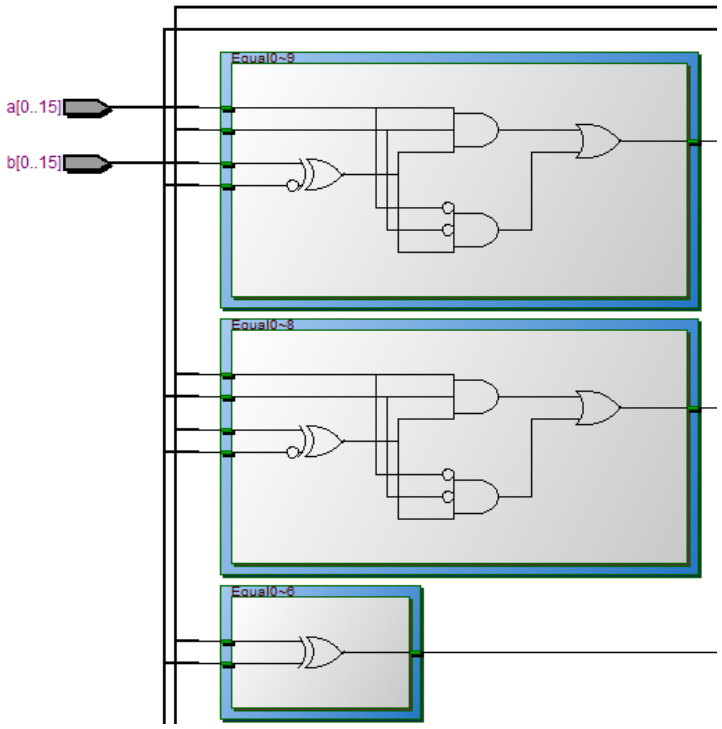

Figura 3: Circuito interno do Bloco

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

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

entity tb_comparador is
end tb_comparador;

architecture teste of tb_comparador is

component Comparador is

	generic
	(
		DATA_WIDTH : natural := 16
	);

	port
	(
		a	: in std_logic_vector  ((DATA_WIDTH-1) downto 0);
		b	: in std_logic_vector  ((DATA_WIDTH-1) downto 0);
		maior  : out std_logic;
		menor : out std_logic;
		igual   : out std_logic
	);

end component;

signal fio_A, fio_B: std_logic_vector(3 downto 0);
signal fio_maior, fio_menor, fio_igual: std_logic;

begin

-- Note que o componente é instanciado com apenas 4 bits nas entradas para facilitar a simulação:
instancia_comparador: Comparador generic map (DATA_WIDTH => 4) port map(a=>fio_A,b=>fio_B,maior=>fio_maior, menor=>fio_menor,igual=>fio_igual);

-- Dados de entrada de 4 bits são expressos em "hexadecimal" usando "x":
fio_A <= x"0", x"8" after 20 ns, x"7" after 40 ns, x"4" after 60 ns;
fio_B <= x"0", x"7" after 10 ns, x"8" after 30 ns, x"1" after 50 ns;
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.

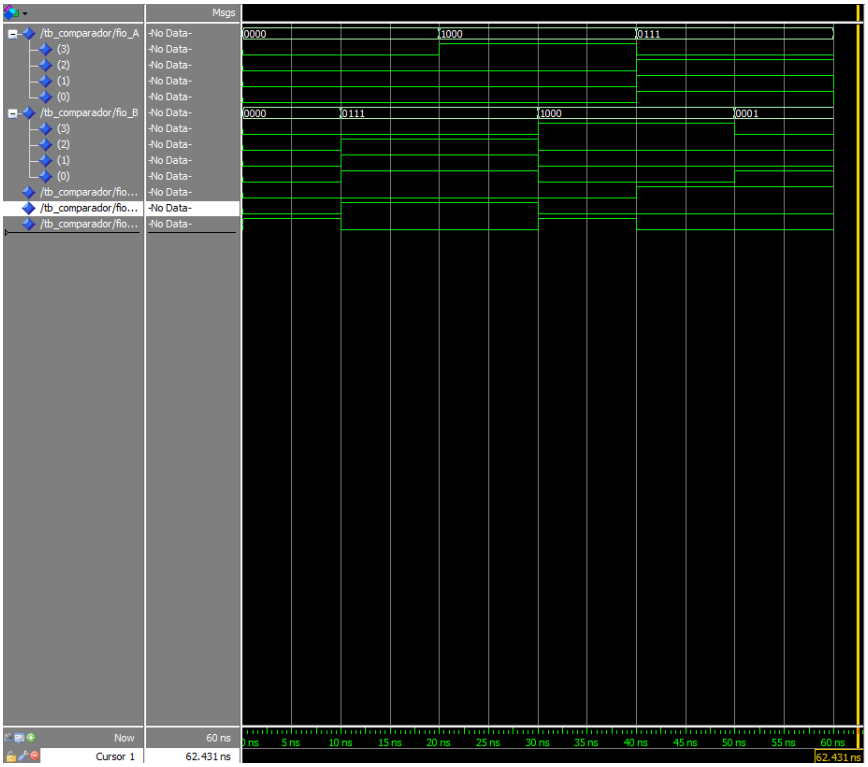

Figura 4: Simulação Comparador no Multisim (signed)

Além disso, foi feito uma alteração na entidade comparador para que agora as palavras fossem consideradas como valores absolutos, sendo apenas uma troca na conversão de signed para unsigned. Abaixo segue a simulação no multisim para comparação

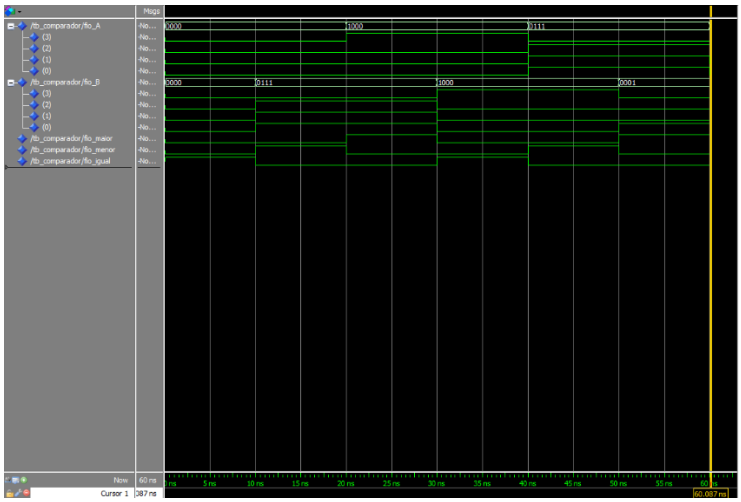

Figura 5: Simulação Comparador no Multisim (unsigned)

Para facilitar o processo e possibilitar o uso do arquivo antigo de pinagem, foi determinado que os pinos que não seriam usados fossem relacionados ao gnd para não haver interferência durante os testes na placa FPGA.

a[3] <= PIN_N25
a[2] <= PIN_N26
a[1] <= PIN_P25
a[0] <= PIN_AE14

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

maior <= PIN_AE23
igual <= PIN_AF23
menor <= PIN_AB21

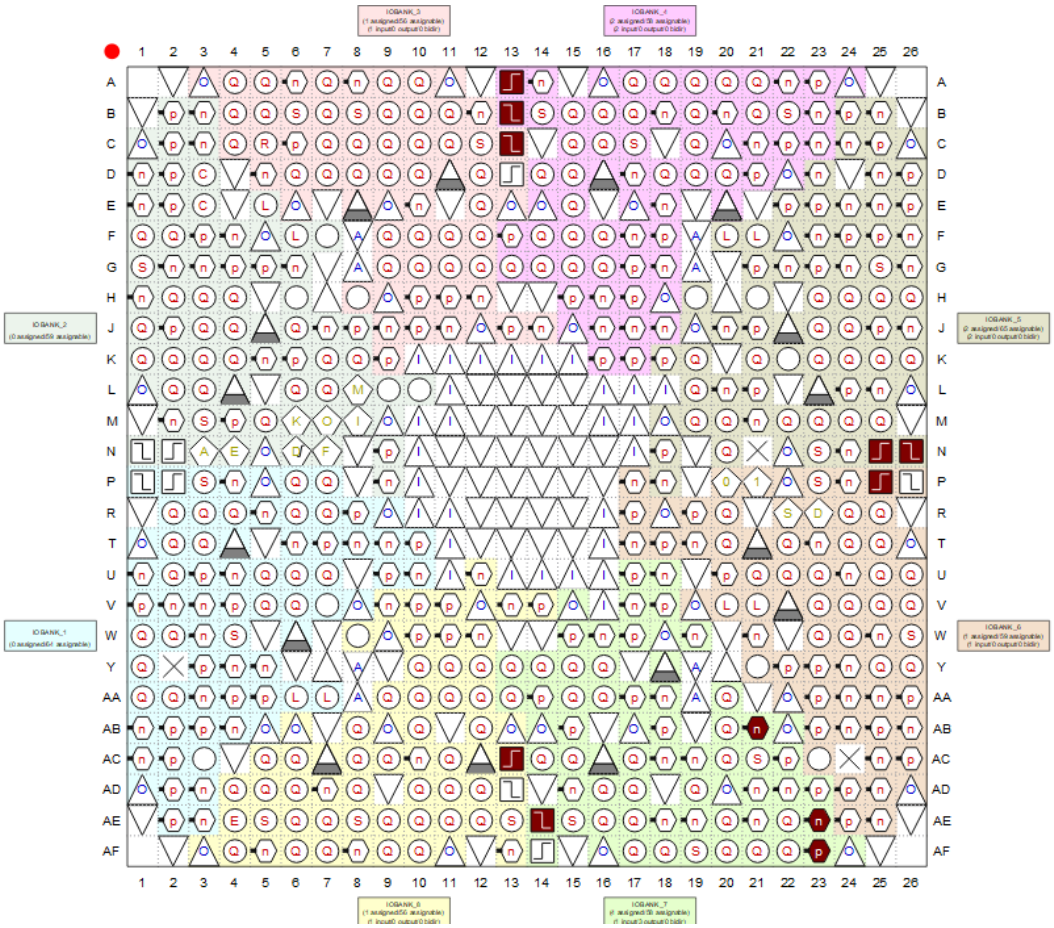

Figura 6: Pin Planner

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

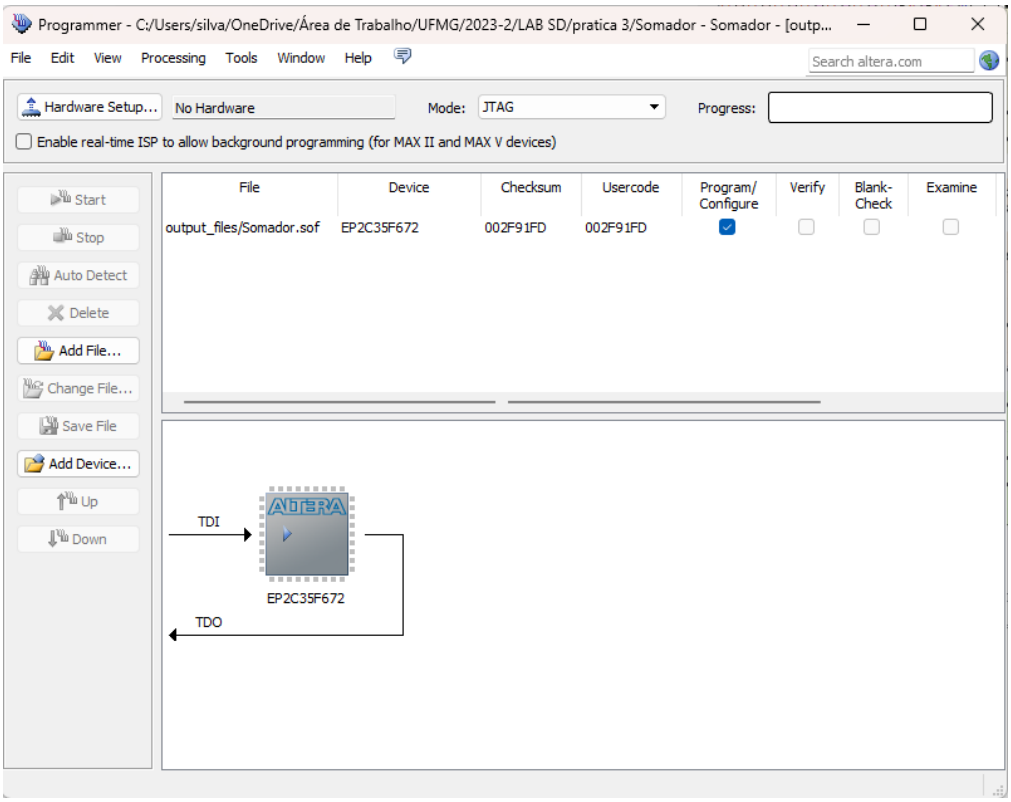

Figura 7: Programmer