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

Iniciou a prática criando uma entidade nomeado Somador 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 um código padrão do software, no caso o unsigned adder e realizou-se alterações nele. No caso, o número de bits e o tipo de entradas, seguindo abaixo o arquivo utilizado:

```
-- Quartus II VHDL Template
-- Unsigned Adder

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Somador is

	generic
	(
		DATA_WIDTH : natural := 4
	);

	port
	(
		a		: in std_logic_vector ((DATA_WIDTH-1) downto 0);
		b		: in std_logic_vector ((DATA_WIDTH-1) downto 0);
		result : out std_logic_vector ((DATA_WIDTH-1) downto 0)
	);

end entity;

architecture rtl of Somador is
begin

	result <= std_logic_vector(unsigned(a) + unsigned(b));

end rtl;
```

O operador ‘+’ da biblioteca numeric_std apenas aceita operações com os tipos signed unsigned, natural e inteiro retornando apenas unsigned e signed, fazendo necessário o uso das funções conversoras para realizar a operação no final. Primariamente, converte os pinos de entrada para unsigned e, a seguir, converte o resultado da soma para vetor logico. Com isso, o funcionamento do código fica idêntico ao código anterior.

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:

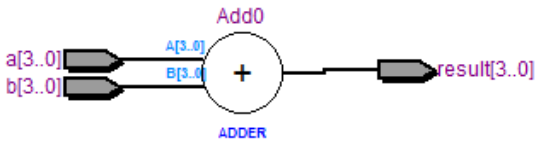


*Figura 1: Circuito Somador*

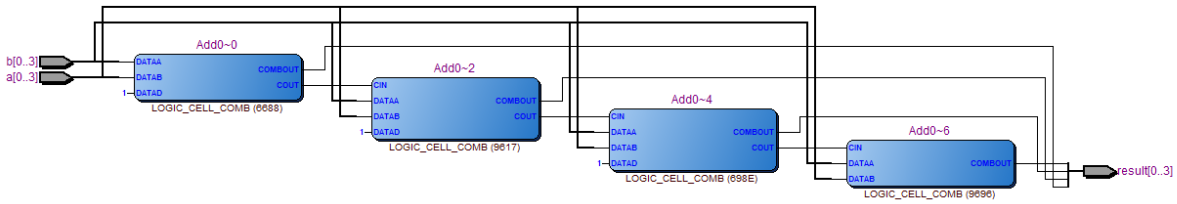


*Figura 2: Diagrama do circuito Somador*

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

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

entity tb_somador is
end tb_somador;

architecture teste of tb_somador is

component Somador is
port (   a, b  : in std_logic_vector(3 downto 0);
         result: out std_logic_vector(3 downto 0));
end component;

signal fio_a, fio_b, fio_soma: std_logic_vector(3 downto 0);
begin
instancia_somador: somador port map(a=>fio_a,b=>fio_b,result=>fio_soma);
-- x nas próximas linhas: os vetores de bits estão expressos em base hexadecimal
fio_a <= x"0", x"A" after 20 ns, x"2" after 40 ns, x"4" after 60 ns;
fio_b <= x"0", x"4" after 10 ns, x"3" 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.

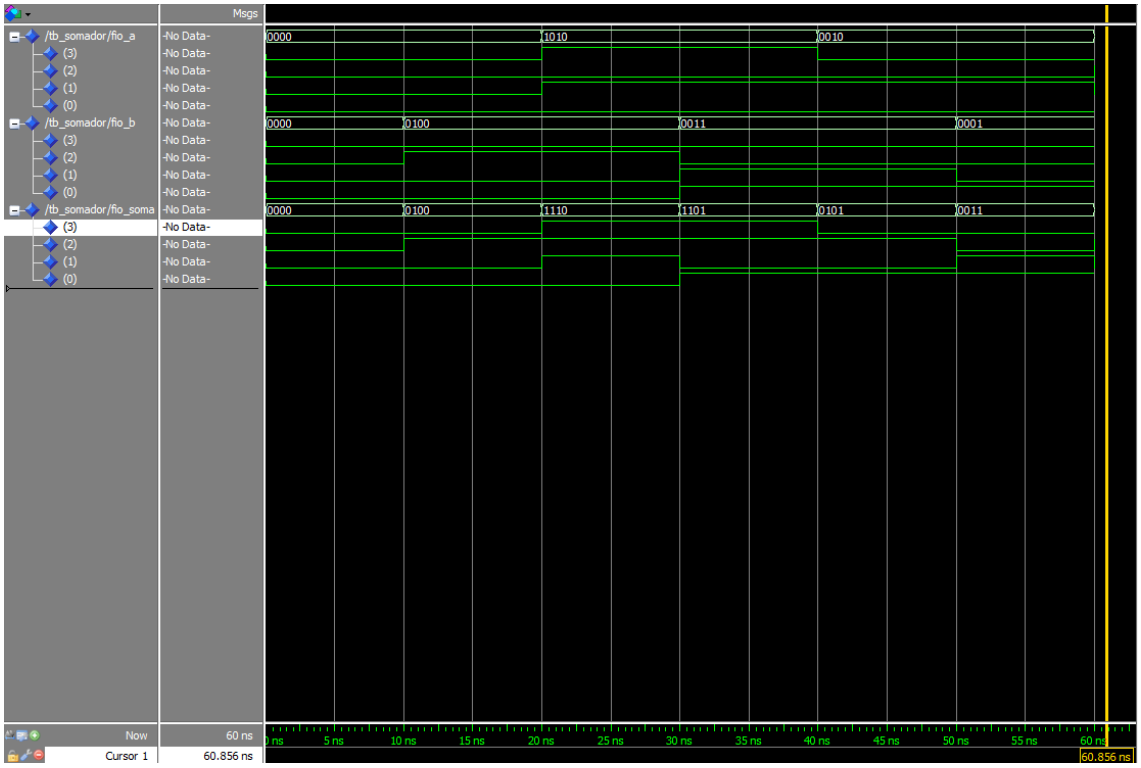


*Figura 3: Simulação Somador no Multisim*

Caso esteja tudo correto, podemos enviá-lo para o FPGA primeiro definindo a pinagem no PIN PLANNER onde, com ajuda de uma tabela, iremos associar as entradas e saídas do programa com os pinos da placa. No caso, os pinos foram configurados do seguinte modo:

***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***

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

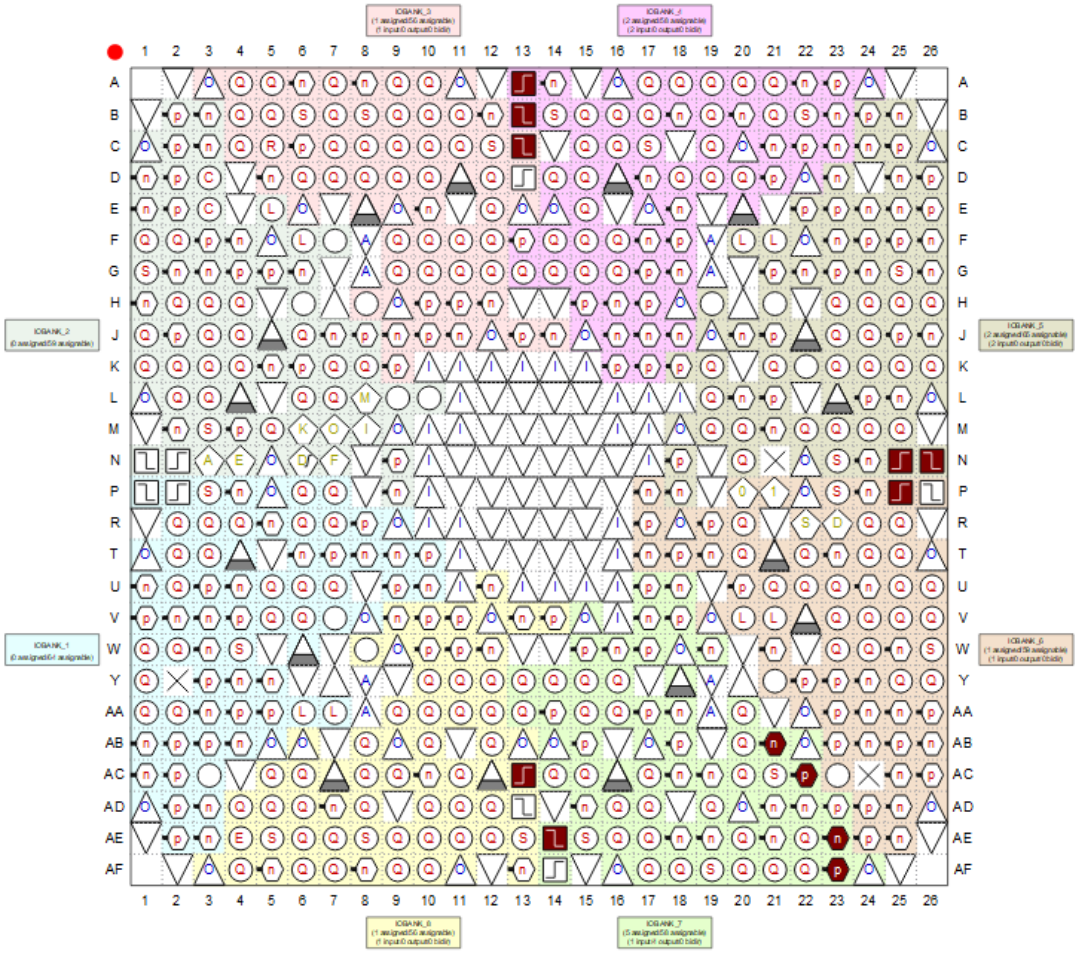

*Figura 4: Pin Planner*

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

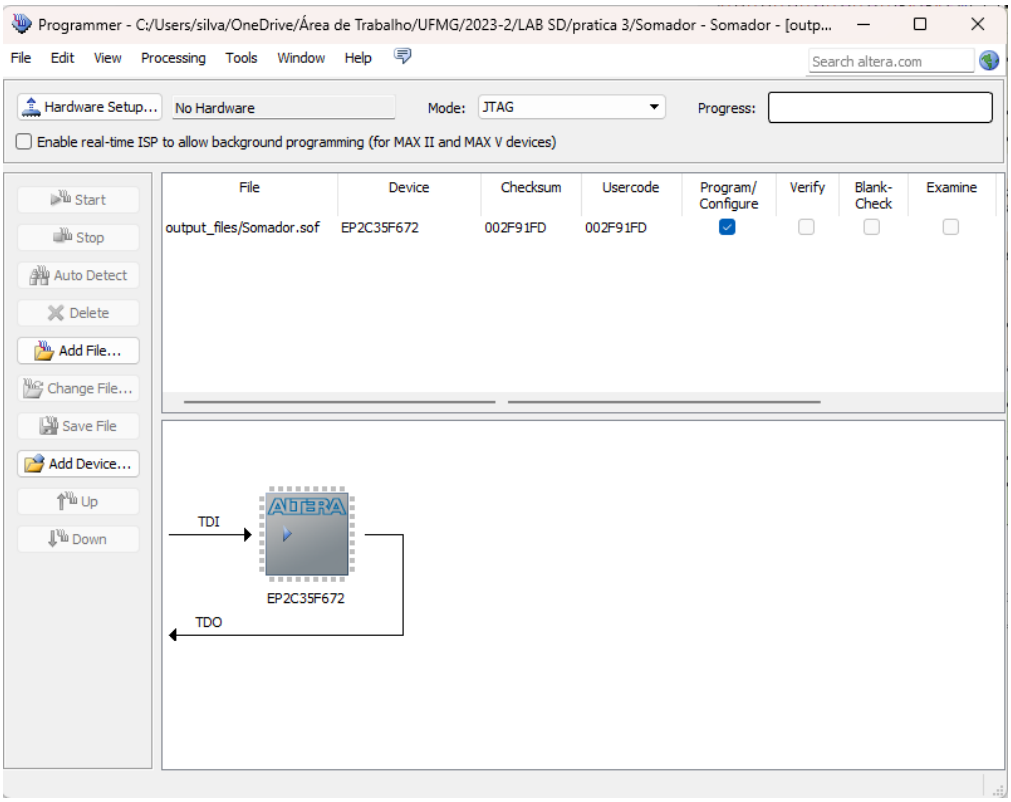


*Figura 5: Programmer*