Modelos de Representação de um Sistema top-level em VHDL: Comportamental,Fluxo de dados e Estrutural.

Disciplina:
Laboratório de Sistemas Digitais

Professor:
Jhonattan Córdoba Ramírez

Alunos:
João Pedro Copelli - 2021014414
João Victor Gomes - 2020072690
Marcelle Christine Aquino Silva - 2021014546

22 de outubro de 2022

Sumário

1 Introdução

A declaração da arquitetura (“architecture”) descreve o comportamento da entidade, define o seu funcionamento interno, isto é, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos. Para tal, utiliza-se uma série de comandos de operação. A declaração de uma arquitetura pode conter comandos concorrentes ou seqüenciais. Sua organização pode conter declaração de sinais, constante, componentes, operadores lógicos, etc, assim como comandos(ex: BEGIN, END). VHDL permite ter mais de uma architecture para a mesma entidade.

- A arquitetura de uma entidade pode ser descrita de três formas distintas de abstração, mas que, em geral, conduzem a uma mesma implementação. este relatório abordará Modelos de Representação de um Sistema top-level em VHDL:
- Descrição estrutural
- Descrição por Fluxo de dados (“data-flow”)
- Descrição comportamental

2 Parte Teórica

2.1 Como descrever uma arquitetura em VHDL?

A declaração da arquitetura mostra como deve ser o funcionamento interno da entidade, ou seja, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos. E para isso, utiliza-se uma série de comandos de operação. A declaração de uma arquitetura pode conter comandos concorrentes ou sequenciais. Sua organização pode conter declaração de sinais, constantes, componentes, operadores lógicos, etc, assim como comandos(ex: BEGIN, END).

2.2 2) Quais são as diferenças entre os modelos de representação de sistemas: Comportamental, Fluxo de dados, Estrutural?

Descrição Comportamental: consiste na descrição de sistemas sequenciais cujo comando fundamental é o process o qual pode ser, opcionalmente prece-

dido de um label e seguido de uma lista de sensibilidade que indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída. Descrição Fluxo de Dados: Descreve o que sistema deve fazer utilizando expressões lógicas e comandos concorrentes, Neste tipo de descrição, os valores de saída são atribuídos diretamente, através de expressões lógicas. Todas as expressões são concorrentes no tempo, ou seja, as atribuições ocorrem simultaneamente. Geralmente descrevem o fluxo de dados no sistema. Descrição Estrutural: Descreve como é o hardware em termos de interconexões entre componentes.as atribuições de sinais são feitas através do mapeamento de entradas e saídas de componentes. Ou seja, é como se fosse uma lista de ligações entre componentes básicos pré-definidos, onde:

Component: é exatamente a descrição de um componente; Port map: é um mapeamento deste componente em um sistema maior.

2.3 Quais as vantagens/desvantagens dos modelos para representação de sistemas top-level?

As vantagens dos modelos para representação de sistemas top-level incluem a facilidade de identificação dos componentes principais (que seriam desenvolvidos para a dita finalidade e suas interfaces), a forma sintetizada pela qual se descreve todos os sistemas e processos utilizados, a maior facilidade de se entender o comportamento desempenhado pelo sistema desenvolvido e também uma maior facilidade de substituir por outros processos, uma vez que se sabe sua lógica, em decorrência justamente dos atributos previamente mencionados. As desvantagens desses modelos de representação é que eles não expõem de forma muito detalhada o design lógico do sistema e também não detalham cada uma das lógicas dos componentes. Isso contribui para que eventuais falhas internas sejam mais difíceis de se identificar e reparar.

2.4 Quais são as declarações concorrentes que envolvem cada modelo?

Os modelos estrutural e de fluxo de dados possuem declarações concorrentes, enquanto o comportamental não possui. Possuir declarações concorrentes significa que todos os comandos são executados paralelamente. Em uma descrição de arquitetura por fluxo de dados ou comportamental, podem ser utilizados alguns comandos com o objetivo de facilitar a descrição de circuitos mais complexos. Esses comandos podem ser concorrentes, e alguns deles se contidos em regiões específicas de códigos, como dentro de processos, são avaliados na sequência em que são apresentados.

2.5 .Qual é o modelo de representação de sistemas mais usado?

O modelo de representação estrutural é a mais utilizada em VHDL. Isso ocorre devido a esse tipo de modelagem suportar conceitos de design hierárquico. Além disso, suporta a reutilização de unidades de design, bem como a capacidade de usar bibliotecas predefinidas de módulos.

3 Parte Prática

Criamos um novo projeto no Quartus II Utilizando como base o arquivo $FlipFlopD.vhd$ disponível,em seguida foi descrito em VHDL um Flip-FlopD usando um modelo de representação de sistemas em fluxo de dados.

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FlipFlopD is
Port ( d,clk : in  STD_LOGIC;
q,qb : inout STD_LOGIC);
end FlipFlopD;
architecture dataflow of FlipFlopD is
signal d1,s1,r1:STD_LOGIC;
begin
s1 <= d nand clk;
d1 <= d nand d;
r1 <= d1 nand clk;
q <= s1 nand qb;
qb <= r1 nand q;
end dataflow
```

Figura 1: código flipflop data flow

e escrito um arquivo testbench para que fosse possivel testar a descrição vhdl sintetizada.

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

entity tb_FlipFlopD is
end tb_FlipFlopD;

architecture teste of tb_Aula8 is

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

signal D,Q: std_logic;
signal clock : std_logic := '0';
constant clk_period : time := 2 ns;

begin

instancia_FlipFlopD: FlipFlopD port map(clock=>clock,D=>D,Q=>Q);
clk_process :process
    begin
         clock <= '0';
         wait for clk_period/2;  --for 0.5 ns signal is '0'.
         clock <= '1';
         wait for clk_period/2;  --for next 0.5 ns signal is '1'.
    end process;

D <= '0', '1' after 3 ns, '0' after 5 ns, '1' after 7 ns;
end FlipFlopD;
```

Figura 2: código testbench flipflop data flow

seguindo as atividades propostas voltou-se ao quartus para gerar o rtl e tambem gerar o post-map verificando sua implementação interna o resultado pode ser visto na figura abaixo

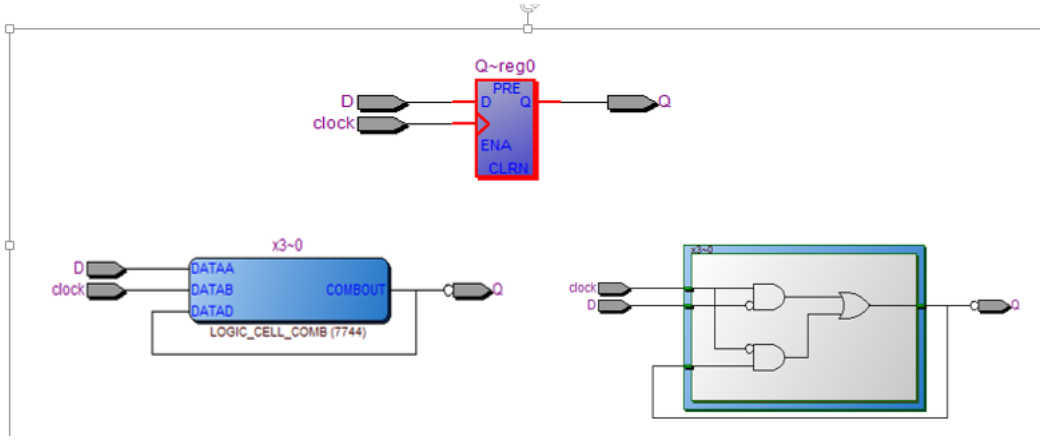

Figura 3: FIGURA 1: Rtl e post maping do flipflop D usando um modelo de representação em fluxo de dados

o diagrama representa corretamente um flip flop sensível a borda de subida apresentando um comportamento coerente ao descrito no código posteriormente Utilizando como base o arquivo “fulladder.vhd” disponível, foi feita a descrição em VHDL de um full adder usando um modelo de representação de sistemas comportamental e feito um arquivo testbench para testar.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.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
signal tmp: std_logic_vector(1 downto 0);
begin
   process(x,y,cin)
   begin
      if(x='0' and y ='1' and Cin = '1') or
        (x='1' and y ='0' and Cin = '1') or
        (x='1' and y ='1' and Cin = '0') or
        (x='1' and y ='1' and Cin = '1')
      then Cout <= '1';
else
Cout <= '0';
end if;
      if (x='0' and y ='0' and Cin = '1') or
         (x='0' and y ='1' and Cin = '0') or
         (x='1' and y ='0' and Cin = '0') or
         (x='1' and y ='1' and Cin = '1')
      then s <= '1';
else
     s <= '0';

   end if;
   end process;

end RTL;
```

Figura 4: código fulladder comportamental

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity tb_fulladder is
end tb_fulladder;

ARCHITECTURE teste OF tb_fulladder IS


 COMPONENT fulladder is
 PORT(
         x   : IN std_logic;
         y   : IN std_logic;
         Cin : IN std_logic;
         s   : OUT std_logic;
         Cout: OUT std_logic
 );
 END COMPONENT;

 signal a : std_logic;
 signal b : std_logic;
 signal Ce : std_logic;

 signal ss : std_logic;
 signal Cs : std_logic;

BEGIN


 instance_fulladder: fulladder PORT MAP (
         x => a,
         y => b,
         Cin => Ce,
         s => ss,
         Cout => Cs
 );


a  <= '0', '0' after 5ns ,'1' after 10ns, '1' after 15ns, '0' after 20ns ,'0' after 25ns, '1' after 30ns, '1' after 35ns;
b   <= '0', '1' after 5ns ,'0' after 10ns, '1' after 15ns, '0' after 20ns ,'1' after 25ns, '0' after 30ns, '1' after 35ns;
Ce <= '0','0' after 5ns ,'0' after 10ns,'0' after 15ns, '1' after 20ns ,'1' after 25ns,'1' after 30ns,'1' after 35ns;



end teste;
```

Figura 5: código testbench fulladder comportamental

em seguida foi gerado o rtl e o post-mapping e seus diagramas e estruturas analizados.

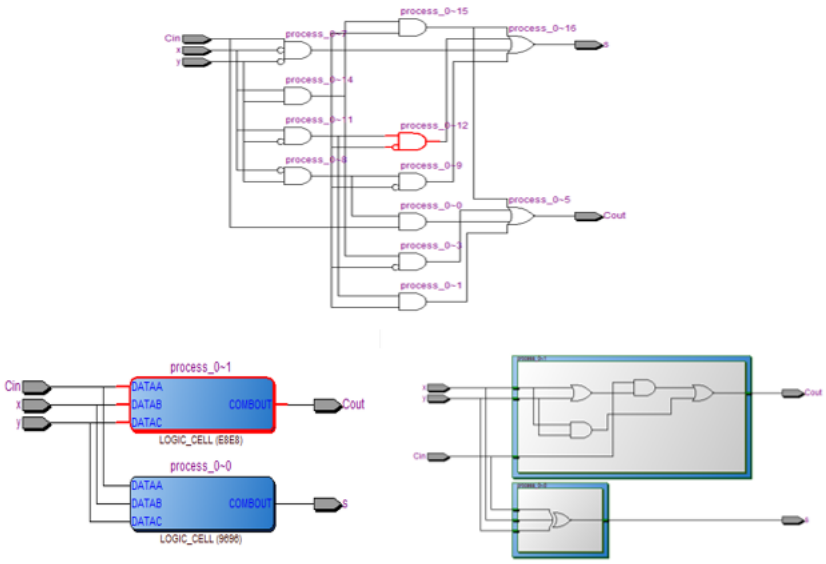

Figura 6: FIGURA 2: Rtl e post maping do Full Adder usando um modelo de representação Comportamental

como observado na figura acima Os resultados Estão de acordo Com o esperado e por fim foi utilizado o (fulladder.vhd) para construir um somador completo de 4 bits utilizando um modelo de representação estrutural.e feito um arquivo testbench para testar a sua descrição vhdl sintetizada.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_unsigned.ALL;
4
5   entity Adder4 is
6      Port ( x    : in STD_LOGIC_VECTOR (3 downto 0);
7             y    : in STD_LOGIC_VECTOR (3 downto 0);
8             Cin  : in STD_LOGIC;
9             s    : out STD_LOGIC_VECTOR (3 downto 0);
10            Cout : out STD_LOGIC);
11  end Adder4;
12
13  architecture rtl of Adder4 is
14
15
16  component fulladder
17     Port ( x : in STD_LOGIC;
18            y : in STD_LOGIC;
19            Cin : in STD_LOGIC;
20            s : out STD_LOGIC;
21            Cout : out STD_LOGIC);
22  end component;
23
24
25  signal c1,c2,c3: STD_LOGIC;
26
27  begin
28
29  FA1: fulladder port map( x(0), y(0), Cin, s(0), c1);
30  FA2: fulladder port map( x(1), y(1), c1, s(1), c2);
31  FA3: fulladder port map( x(2), y(2), c2, s(2), c3);
32  FA4: fulladder port map( x(3), y(3), c3, s(3), Cout);
33
34  end rtl;
```

Figura 7: código somador 4bits estrutural

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_Adder4 is
end tb_Adder4;
architecture teste of tb_Adder4 is


component Adder4
   Port ( x : in STD_LOGIC_VECTOR (3 downto 0);
          y : in STD_LOGIC_VECTOR (3 downto 0);
          Cin : in STD_LOGIC;
          s : out STD_LOGIC_VECTOR (3 downto 0);
          Cout : out STD_LOGIC);
end component;

signal x,y,s:STD_LOGIC_VECTOR (3 downto 0);
signal Cin, Cout: STD_LOGIC;

begin
instancia_Adder4: Adder4 port map ( Cin=> Cin, x=>x, y=>y, s=>s,Cout=>Cout);
 x <= "0000", "0010" after 5ns ,"0110" after 10ns, "1000" after 15ns;
y  <= "0001", "0101" after 4ns ,"0111" after 9ns, "1001" after 14ns;
Cin <= '0', '1' after 5ns ,'0' after 10ns, '1' after 15ns;

end teste;
```

Figura 8: código testbench somador 4bits estrutural

Com tudo verificado e o código alterado conforme a descrição acima, fizemos a configuração do $testbench$ e executamos a simulação pelo ModelSim.

Os resultados obtidos foram esses:

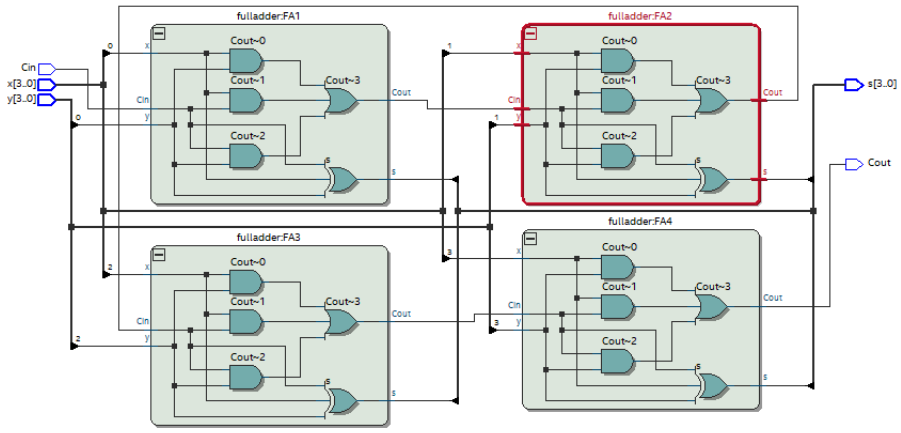

Figura 9: rtl somador 4bits

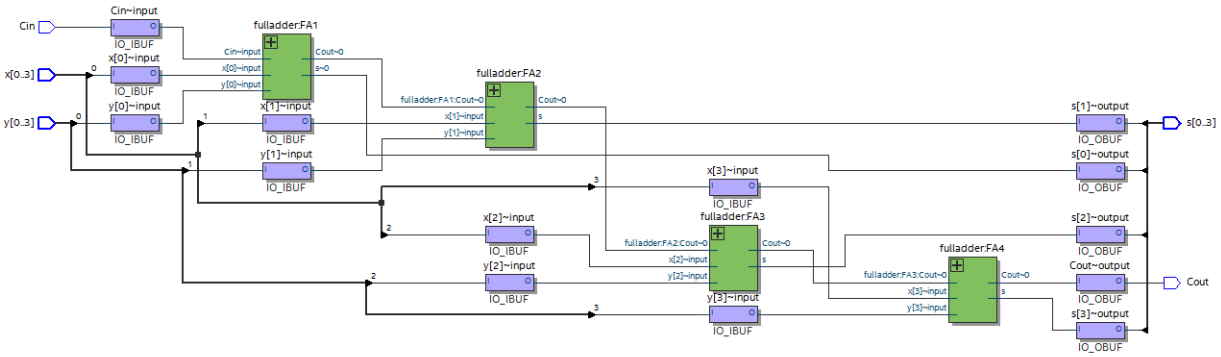

Figura 10: post mapping somador

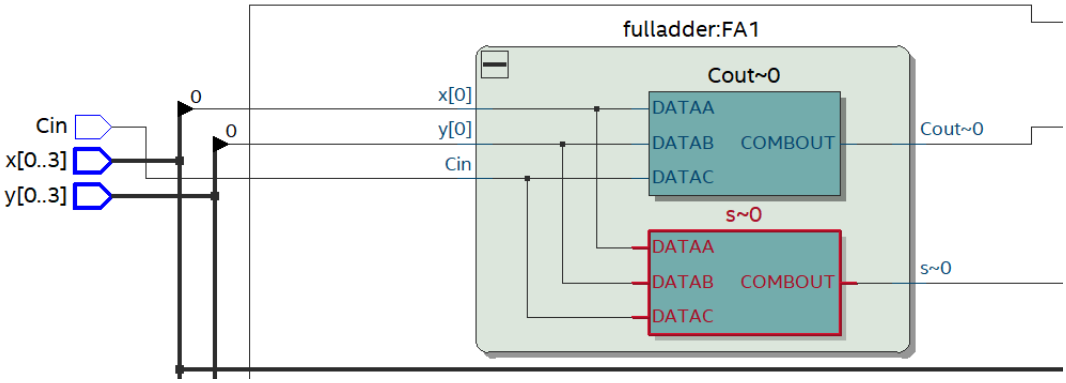

Figura 11: post maping somador

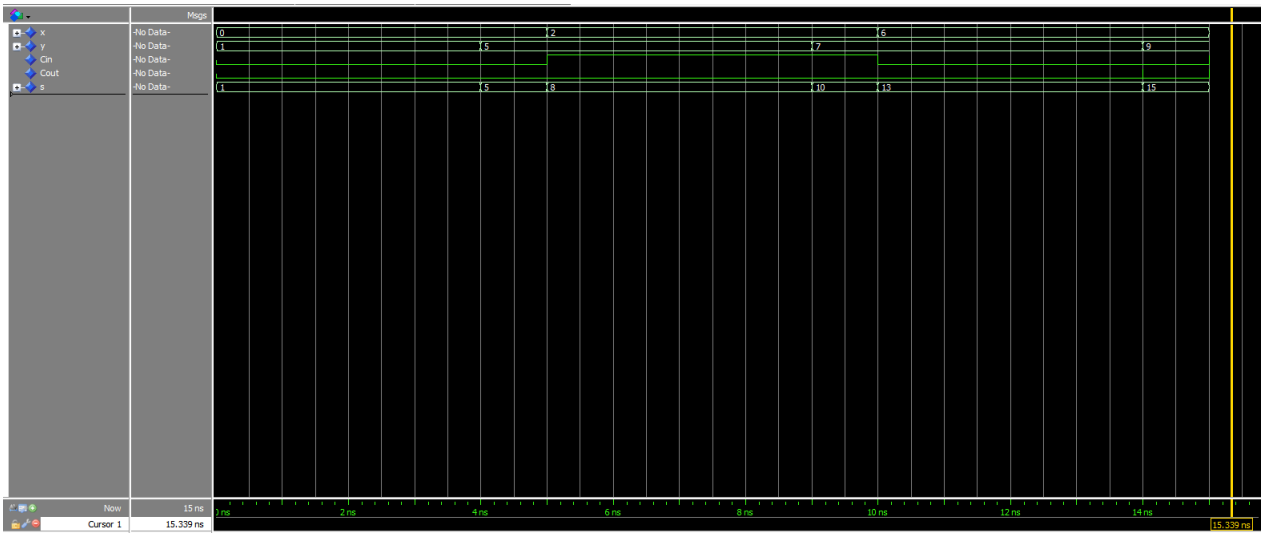

Figura 12: simulação somador

4 Conclusão

Como podemos observar pelas figuras acimas os resultados obtidos foram iguais aos resultados esperados.