Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Sistemas Eletrônicos - PSI

PSI-2553- Projeto de Sistemas Integrados

Exp. 1A: Implementação RTL do Processador Fibonacci (Teoria) (17)

Objetivos

Nesta experiência serão criados os modelos VHDL RTL (estruturais, ou seja suas descrições esquemáticas) do *datapath,* da unidade de controle e do módulo de Fibonacci. Estes modelos referem-se aos resultados da síntese RTL desenvolvido na aula de anterior e servirão de base para a validação (por simulação, a ser realizada na exp1B) e para a síntese do sistema baseado em processador (a ser realizada na exp4) dedicado ao cálculo da sequência de Fibonacci.

Parte Teórica

2.1. Modelo da arquitetura RTL do processador

A metodologia de projeto de um processador de aplicação específica resulta numa arquitetura RTL composta de dois componentes: a **parte de controle** (uma máquina de estados finitos, FSM) e uma **parte operativa** (***datapath***).

Desta forma a descrição VHDL do processador é hierárquica: primeiro são descritos os modelos dos 2 componentes separadamente e estes são depois juntados (no nível superior) para compor o modelo do processador.

A fim de padronizar os resultados de cada dupla, a parte experimental desta experiência fornece um método estruturado para gerar o código VHDL de cada componente a partir de modelos (VHDL RTL) genéricos que chamaremos de padrões (*templates*).

2.1.1. Parte de controle - a máquina de estados finitos (FSM)

A Figura 1 ilustra a estrutura do padrão (*template*) do controlador fornecida no seu arquivo VHDL.

O código da arquitetura é dividido em três (3) processos concorrentes:

- **SEQ:** a parte sequencial que executa a mudança (ou não) de estados a cada ciclo de relógio. A cada ativação (borda de subida) do *clock*, ou a cada ativação do *reset*, o valor do **next_state** é atualizado para o **state,** ou estado inicial S0, no caso de *reset*. O caso em que **next_state = state** equivale a não haver mudança de estado.

- **COMB**: a parte lógica combinacional para a computação do valor de **next_state**. Pela definição da máquina de estados finitos, o estado futuro depende de entradas e do estado presente. Portanto, qualquer mudança de estado ou de algum valor de entrada (ver a lista de sensibilidade) determina um novo cálculo de **next_state**.

- **SAI:** a parte lógica combinacional para o cálculo dos valores dos sinais de saída. Por se tratar de uma Máquina de Moore, a saída depende apenas do estado presente. Portanto, qualquer mudança de estado determinará uma nova saída (ver a lista de sensibilidade).

Cada dupla de alunos deverá personalizar o código VHDL padrão de acordo com o resultado obtido em sala de aula (dia 17/03).

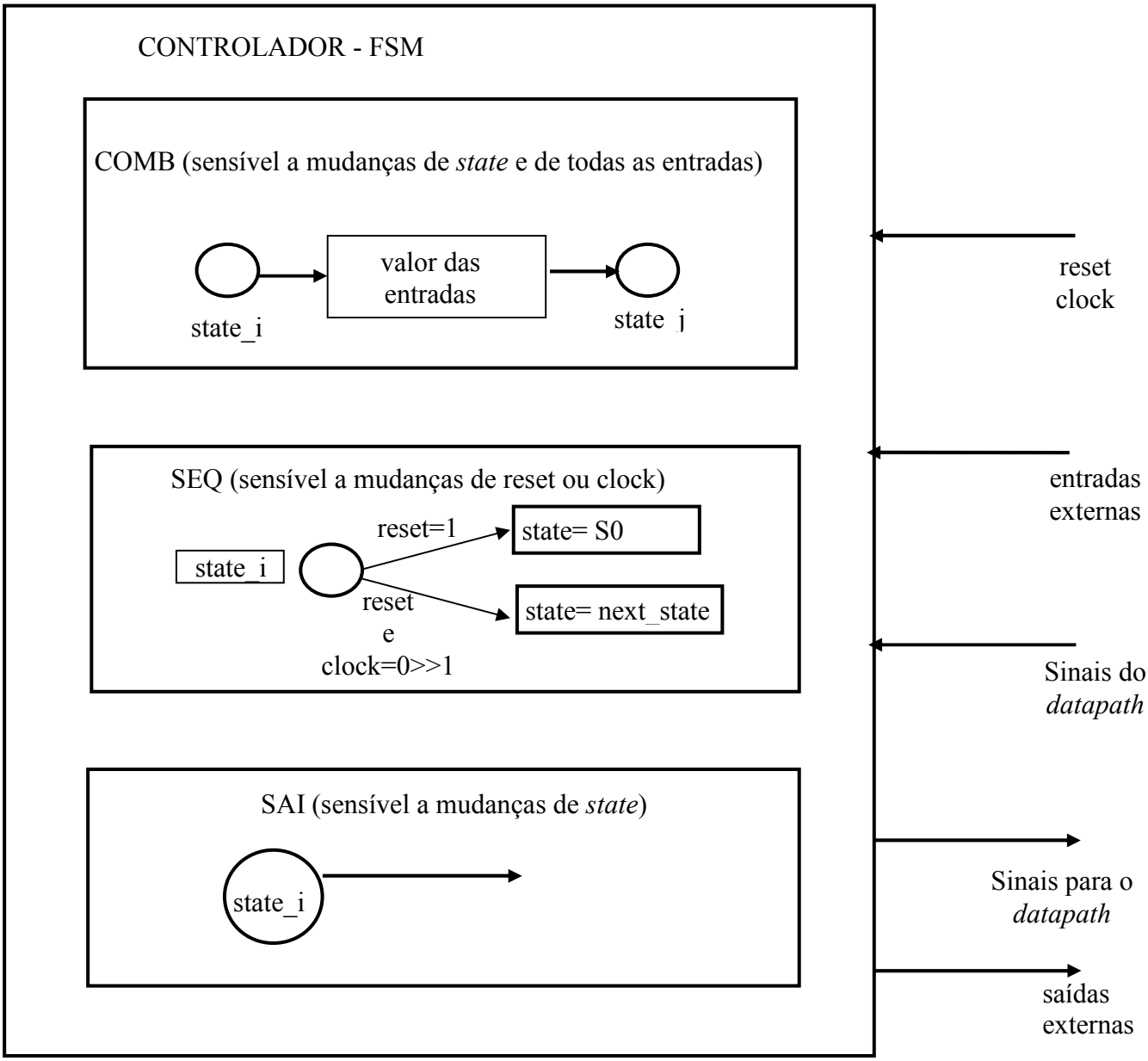

Figura 1. Estrutura do t*emplate* do controlador

2.1.2. Parte Operativa (datapath)

A Figura 2 ilustra a estrutura da entidade do *datapath* fornecida no seu arquivo VHDL.

O modelo VHDL do *datapath* também segue um padrão (*template*) como ilustrado no final deste texto. Primeiramente, são definidos os componentes RTL que compõem a arquitetura (esquema) do *datapath*. Todos os dados de entrada ou de saída são do tipo *standard_logic_vector* e seus tamanhos são definidos pelo valor de um parâmetro definido pela estrutura GENERIC. Os componentes (de biblioteca) fornecidos para compor o *datapath* são os seguintes:

- multiplexador 2x1 de n bits, com um sinal de seleção.
- registrador controlado por load (com sinal de *reset*)

- somador de n bits nas entradas e saída (sem indicador de overflow).
- subtrator de n bits nas entradas e saída.
- comparador (= ou /=) de n bits (se A=B a saída é 1).

Cada componente é definido pela sua entidade cuja arquitetura é comportamental.
Em seguida são instanciados os componentes que compõem o *datapath* e são indicadas as suas conexões.

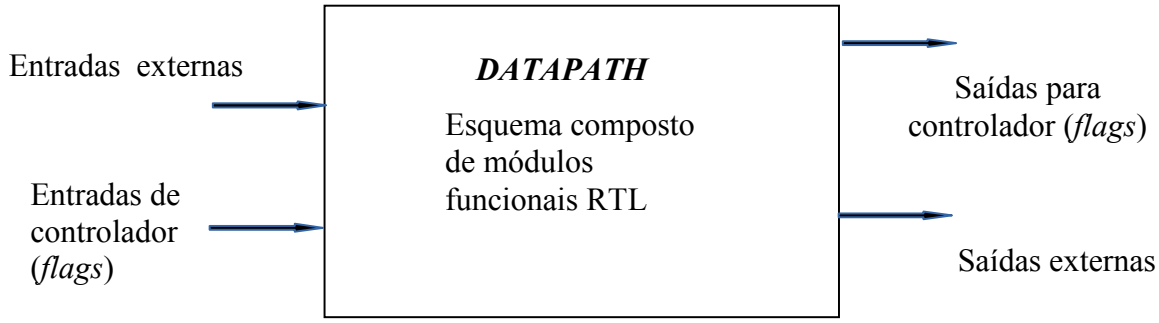

Figura 2 – Entidade do *template* do *datapath*

2.1.2. Processador M_F completo

A Figura 3 ilustra a estrutura do padrão do processador M_F fornecida no seu arquivo VHDL.

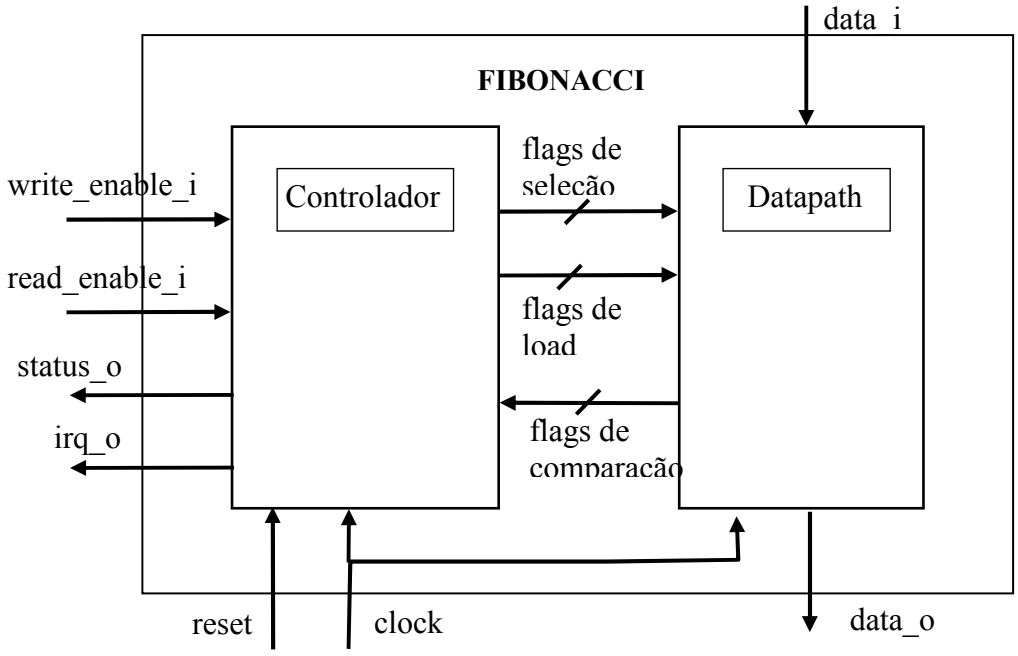

Figura 3 – Estrutura do *template do* processador M_F

O modelo VHDL do processador é o *netlist* composto pelos 2 componentes acima.

Template do modelo VHDL do *datapath.vhd*

```
--Descrição do circuito feito por Mario Raffo (11)
-- Jorge Gonzalez (12)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY <nome_entidade> IS
      GENERIC (NUMBITS : NATURAL := 32);
      PORT (     rst              : IN STD_LOGIC;
                 clk        : IN STD_LOGIC;

                 entrada0   : IN <tipo de dado>;
                 ...
                 entradaN   : IN <tipo de dado>;

                 saida0           : OUT <tipo de dado>;
                 ..
                 saidaN           : OUT <tipo de dado>);
END <nome_entidade>;


ARCHITECTURE behavior OF datapath IS

      COMPONENT somador
            GENERIC (NUMBITS : NATURAL := 32);
            PORT (     SIGNAL x   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                       SIGNAL y   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                       SIGNAL XY  : OUT STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0));
      END COMPONENT;

      COMPONENT subtrator
            GENERIC (NUMBITS : NATURAL := 32);
            PORT (     SIGNAL x   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                       SIGNAL y   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                       SIGNAL XY  : OUT STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0));
      END COMPONENT;


      COMPONENT reg
            GENERIC (NUMBITS : NATURAL := 32);
            PORT( SIGNAL rst :  IN STD_LOGIC;
                    SIGNAL clk :  IN STD_LOGIC;
```

```
                  SIGNAL load  :  IN STD_LOGIC;
                  SIGNAL d  :  IN STD_LOGIC_VECTOR(NUMBITS-1
DOWNTO 0);
                  SIGNAL q   : OUT STD_LOGIC_VECTOR(NUMBITS-1
DOWNTO 0));
     END COMPONENT;

     COMPONENT multiplexor2a1
          GENERIC (NUMBITS : NATURAL := 32);
          PORT (     SIGNAL a   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                     SIGNAL b   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                     SIGNAL sel :  IN STD_LOGIC;
                     SIGNAL f   : OUT STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0));
     END COMPONENT;


     COMPONENT igual
          GENERIC (NUMBITS : NATURAL := 32);
          PORT (     SIGNAL a   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                     SIGNAL b   :  IN STD_LOGIC_VECTOR(NUMBITS-
1 DOWNTO 0);
                     SIGNAL eq  : OUT STD_LOGIC);
     END COMPONENT;



//completar com todos os sinais necessários para as conexões
     SIGNAL <nome>         : <tipo>;
     SIGNAL <nome>         : <tipo>;


BEGIN

// instanciar o número de muxes necessários

     mux1: multiplexor2a1 GENERIC MAP(
                                   NUMBITS => NUMBITS)
                           PORT MAP (
                                a     => nome real,
                                b     => nome real,
                                sel   => nome real,
                                f     => nome real);


// instanciar o número de registradores necessários

     reg1: reg GENERIC MAP(
                      NUMBITS => NUMBITS)
                 PORT MAP (
                      rst => nome real,
                      clk => nome real,
                      load  => nome real,
```

```
                        d     => nome real,
                        q   => nome real);


// instanciar o número de somadores necessários

      sum1: somador GENERIC MAP(
                              NUMBITS => NUMBITS)
                  PORT MAP (
                        x     => nome real,
                        y     => nome real,
                        XY    => nome real);

// instanciar o número de subtratores necessários

      rest1: subtrator GENERIC MAP(
                              NUMBITS => NUMBITS)
                  PORT MAP (
                        x     => nome real,
                        y     => nome real,
                        XY    => nome real);


// instanciar o número de comparadores (igual) necessários

      ig1: igual GENERIC MAP(
                                    NUMBITS => NUMBITS)
                  PORT MAP (
                        a     => n,
                        b     => uno,
                        eq    => n_eq_1);



END behavior;
```

Template **do modelo VHDL da FSM**

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

ENTITY <nome_entidade> IS
     PORT (     rst              : IN STD_LOGIC;
                clk        : IN STD_LOGIC;

                entrada0   : IN <tipo de dado>;
                ...
                entradaN   : IN <tipo de dado>;

                saida0           : OUT <tipo de dado>;
                ..
                saidaN           : OUT <tipo de dado>);
END <nome_entidade>;

ARCHITECTURE <nome_arquitetura> OF <nome_entidade> IS
     TYPE state_type IS (estado0, estado1,
estado2,.......,estadoN);  // veja quantos estados hah no seu
projeto
     SIGNAL state, next_state : state_type;

BEGIN
     ------------------------Lógica Sequencial-----------------
------
     SEQ: PROCESS (rst, clk)
     BEGIN
          IF (rst='1') THEN
               state <= estado0;
          ELSIF Rising_Edge(clk) THEN
               state <= next_state;
          END IF;
     END PROCESS SEQ;
     ----------------------Lógica Combinacional do estado
seguinte--
     COMB: PROCESS (entrada0,..,entradaN, state)  // completar
com sinais de entrada + state
     BEGIN
          CASE state IS
     // para cada estado, preencha as condições para transição
de estado
                WHEN estado0 =>
                     IF ( entrada0 = ……) THEN
                          next_state <= estado1;
                     ELSE ...
                          next_state <= estado0;
                     END IF;
                WHEN estado1 =>
                     next_state <= estado2;
                ...
                ...
                WHEN estadoN =>
```

```
                        IF ( entradaN = ……) THEN
                              next_state <= estado0;
                        ELSE ...
                              next_state <= estadoN;
                        END IF;
            END CASE;
      END PROCESS COMB;

      ------------------------Lógica Combinacional saidas--------
-------------
      SAI: PROCESS (state)
      BEGIN
            CASE state IS
// para cada estado, defina as atribuições de saída
                  WHEN estado0 =>
                        saida1 <= <valor>;
                        ...
                        saidaN <= <valor>;
                  WHEN estado1 =>
                        saida1 <= <valor>;
                        ...
                        saidaN <= <valor>;
                  ...
                  ...
                  WHEN estadoN =>
                        saida1 <= <valor>;
                        ...
                        saidaN <= <valor>;

            END CASE;
      END PROCESS SAL;

END <nome_arquitetura>;
```