



Aula 10 - Esquemas de geração



Tópicos da aula

- Declaração **GENERIC**
- Comando **GENERATE**
- Esquema de geração **IF**
- Comando **LOOP**
- Comando **WHILE**



Declaração GENERIC

- ❑ Permite **levar informações** externas ao componente para a ferramenta de síntese
- ❑ **Apenas** a ferramenta de síntese utiliza as informações passadas na declaração GENERIC
- ❑ São definidos na **Entidade** do componente



Exemplo - GENERIC

```
1  ENTITY flipn_3 IS
2      GENERIC (n : INTEGER := 3);           -- definicao e valor do generico
3  PORT (ck : IN BIT;                       -- relógio
4         rst : IN BIT;                     -- rst=1 leva q=000 assincrono
5         d : IN BIT_VECTOR(n-1 DOWNT0 0); -- numero de bits definidos pelo generico 'n'
6         q : OUT BIT_VECTOR(n-1 DOWNT0 0)); -- numero de bits definidos pelo generico 'n'
7  END flipn_3;
8
9  ARCHITECTURE teste OF flipn_3 IS
10 BEGIN
11
12     PROCESS (ck, rst)
13     BEGIN
14         IF (rst = '1') THEN
15             q <= (OTHERS => '0');           -- q=00...0 independente de ck
16         ELSIF (ck'EVENT AND ck = '1') THEN
17             q <= d;                         -- armazena dado
18         END IF;
19     END PROCESS;
20
21 END teste;
22
```



Exercícios

- ❏ **Crie um registrador utilizando a declaração GENERIC. O valor do tamanho de dados do registrador deve ser de 8 (oito) bits**
- ❏ **Instancie 2 (duas) vezes o componente “registrador” criado no exercício anterior (em um novo projeto). Use para este novo projeto uma declaração GENERIC na qual o valor do tamanho de dados do registrador deve ser de 16 (dezesseis) bits**



Comando GENERATE

- ❏ **Permite repetir comandos **concorrentes** diversas vezes**
- ❏ **Uso interessante na geração de circuitos que observam uma regularidade na sua estrutura (somadores, multiplicadores, ALU)**



Exemplo: GENERATE

```
17  ARCHITECTURE test OF test IS
18  |
19  COMPONENT and02
20  PORT (
21      a0 : IN std_logic;
22      a1 : IN std_logic;
23      y  : OUT std_logic);
24  END COMPONENT and02;
25  |
26  BEGIN
27  |
28  G1 : FOR n IN (length-1) DOWNTO 0 GENERATE
29  PORT MAP (
30      a0 => sig1(n),
31      a1 => sig2(n),
32      y  => z(n)
33  );
34  END GENERATE G1;
35  |
36  END test;
37
```



Exercícios

- ❏ Utilizando o comando **GENERATE** implemente um componente que faça as seguintes operações bit-a-bit:
 - ❏ Lógica AND entre os bits de 0 até 5
 - ❏ Lógica OR entre os bits de 6 até 10



Comando IF

- ❏ Interessante para uso quando se quer implementar comandos **concorrentes** de acordo com diretivas de síntese (dadas na descrição GENERIC)



Exemplo: IF

```
148 -----
149 no_fifo :
150 -----
151 -----
152 IF (FIFO_TYPE="NONE") GENERATE
153     rok <= wr;
154     wok <= rd;
155     dout <= din;
156 END GENERATE;
157 -----
158 -----
159 -----
160 fifo_shift :
161 -----
162 -----
163 IF (FIFO_TYPE="SHIFT") GENERATE
164     -----
165     U0 : fifo_controller
166     -----
167     GENERIC MAP (
168         DEPTH => DEPTH
169     )
170     PORT MAP(
171         rst => rst,
172         clk => clk,
173         wr  => wr,
174         rd  => rd,
175         state => state
176     );
177 END GENERATE;
```



Exercícios

- ❏ **Utilizando o comando IF implemente um componente que possa ser implementado de duas formas:**
 - ❏ **Lógica AND entre os dados de entrada**
 - ❏ **Lógica OR entre os dados de entrada**



Universidade Federal
de Santa Catarina

Comando LOOP

- ❏ **Permite repetir a execução de um conjunto de dados sequenciais**
- ❏ **Dois esquemas são possíveis:**
 - ❏ **FOR**
 - ❏ **WHILE**



Esquema de execução FOR

```
1  ENTITY som_8a IS
2  |    GENERIC(n      : INTEGER := 3 );           -- numero de bits
3  |    PORT    (x, y  : IN  BIT_VECTOR (n-1 DOWNT0 0); -- entradas so somador
4  |              ze   : IN  BIT;                 -- vem um
5  |              s    : OUT BIT_VECTOR (n-1 DOWNT0 0); -- saida
6  |              zs   : OUT BIT);                -- vai um
7  |    END som_8a;
8
9  ARCHITECTURE teste OF som_8a IS
10 BEGIN
11
12 |    abc: PROCESS (x, y, ze)
13 |        VARIABLE v : BIT_VECTOR (n DOWNT0 0); -- vai um interno
14 |        BEGIN
15 |
16 |            v(0) := ze;
17 |
18 |            abc: FOR i IN 0 TO n-1 LOOP
19 |                s(i) <= x(i) XOR y(i) XOR v(i);
20 |                v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
21 |            END LOOP abc;
22 |
23 |            zs <= v(n);
24 |
25 |        END PROCESS;
26 |
27 END teste;
```



Esquema de iteração WHILE

```
1  ENTITY som_9a IS
2  |   GENERIC (n      : INTEGER := 3 );           -- numero de bits
3  |   PORT    (x, y   : IN  BIT_VECTOR (n-1 DOWNT0 0); -- entradas so somador
4  |           ze      : IN  BIT;                -- vem um
5  |           s       : OUT BIT_VECTOR (n-1 DOWNT0 0); -- saida
6  |           zs      : OUT BIT);                -- vai um
7  |   END som_9a;
8
9  ARCHITECTURE teste OF som_9a IS
10 BEGIN
11
12  abc: PROCESS (x, y, ze)
13  |   VARIABLE i : INTEGER;
14  |   VARIABLE v : BIT_VECTOR (n DOWNT0 0); -- vai um interno
15
16  BEGIN
17  |   i := 0;           -- deve ser atualizado a cada iteracao
18  |   v(0) := ze;
19
20  |   abc: WHILE i <= n-1 LOOP -- executado enquanto verdadeiro
21  |       s(i) <= x(i) XOR y(i) XOR v(i);
22  |       v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
23  |       i := i+1;
24  |   END LOOP abc;
25
26  |   zs <= v(n);
27  END PROCESS;
28
29 END teste;
```



Configuration (configuração)

- ❑ **Usada para fazer associações em modelos**
 - ❑ Associar uma entidade a um arquitetura
 - ❑ Associar um componente a um par entidade-arquitetura
- ❑ **Largamente usada em ambientes de simulação**
 - ❑ Facilita a exploração do espaço de projeto
- ❑ **Uso limitado em ambientes de síntese (nem sempre suportada)**

```
CONFIGURATION <identifier> OF <entity_name> IS
    FOR <architecture_name>
        END FOR;
END; -- (1076-1987 version)
END CONFIGURATION; -- (1076-1993 version)
```



Configuration (configuração)

❑ Exemplos

```
CONFIGURATION config OF mux_2x1 IS  
  FOR structural  
  END FOR;  
END;
```

```
CONFIGURATION config OF mux_2x1 IS  
  FOR behavior  
  END FOR;  
END;
```




Exemplo de *Configuration*

- ❑ Este modelo possui duas arquiteturas
 - ❑ Estrutural
 - ❑ Comportamental
- ❑ A configuração seleciona a arquitetura a ser usada

```
ENTITY mux_2x1 IS
PORT (a, b : IN BIT; -- data inputs
      sel : IN BIT; -- selector
      s : OUT BIT); -- data output
END mux_2x1;

-----

ARCHITECTURE structural OF mux_2x1 IS
BEGIN
  s <= (a AND NOT sel) OR
        (b AND sel);
END structural;

-----

ARCHITECTURE behavior OF mux_2x1 IS
BEGIN
  PROCESS (a,b,sel)
  BEGIN
    IF (sel='0') THEN
      s <= a;
    ELSE
      s <= b;
    END IF;
  END PROCESS;
END behavior;

-----

CONFIGURATION config OF mux_2x1 IS
  FOR behavior
  END FOR;
END;
```



Exemplo de *Configuration*

- ❑ Este modelo possui duas arquiteturas
 - ❑ Estrutural
 - ❑ Comportamental
- ❑ A configuração seleciona a arquitetura a ser usada

```
ENTITY mux_2x1 IS
PORT (a, b : IN BIT; -- data inputs
      sel : IN BIT; -- selector
      s : OUT BIT); -- data output
END mux_2x1;

-----

ARCHITECTURE structural OF mux_2x1 IS
BEGIN
    s <= (a AND NOT sel) OR
         (b AND sel);
END structural;

-----

ARCHITECTURE behavior OF mux_2x1 IS
BEGIN
    PROCESS(a,b,sel)
    BEGIN
        IF (sel='0') THEN
            s <= a;
        ELSE
            s <= b;
        END IF;
    END PROCESS;
END behavior;

-----

CONFIGURATION config OF mux_2x1 IS
    FOR behavior
    END FOR;
END;
```



Universidade Federal
de Santa Catarina

FIM AULA 10