Laboratório de Eletrônica Digital
SEL-EESC-USP

EESC · USP

Introdução a VHDL

Aula 6

Professora Luiza Maria Romeiro Codá

Esquemas de Iteração

Em VHDL é possível criar **esquemas iterativos** de geração. Com eles é possível repetir uma série de comandos, tanto concorrentes como sequenciais.

- ✓ Para comandos concorrentes, é utilizado o comando **concorrente** GENERATE,
- ✓ para sequenciais, o comando **sequencial** LOOP é utilizado.

Para ambos os comandos de geração, há dois esquemas. Um repete os comandos um número determinado de vezes, e o outro repete os comandos caso uma expressão de condição seja atendida.

É importante lembrar que os esquemas de geração, assim como qualquer outro comando em VHDL, não são avaliados em tempo de execução ("*run-time*"), isto é, são avaliados apenas no momento da síntese da descrição, e, portanto, geram um circuito fixo.

Obs.: O termo "tempo de execução" é aqui usado figurativamente. Os dispositivos lógicos programáveis não executam nenhum código. As descrições são interpretadas pelo *software* e transformadas em um circuito físico real.

GENERATE

O comando **concorrente** GENERATE utiliza dois esquemas de iteração para repetir comandos concorrentes:

- Esquema FOR
- Esquema IF

GENERATE – FOR

O esquema FOR repete um conjunto de comandos uma quantidade determinada de vezes. É fornecida uma variável local e os limites para esta variável. Por exemplo, o código abaixo repete os comandos concorrentes 4 vezes:

```
abc: FOR i IN 0 TO 3 GENERATE
    -- Comandos concorrentes
END GENERATE;
```

A sintaxe de declaração do esquema FOR é como a seguir:

```
<rótulo_obrigatório>: FOR <variável_local> IN <limites_da_variável> GENERATE
    -- Comandos concorrentes
END GENERATE <rótulo_opcional>;
```

GENERATE – FOR : Exemplo

Utilizando genéricos em conjunto com o comando FOR/GENERATE é possível, por exemplo, descrever facilmente uma porta AND com quantidade de entradas variável:

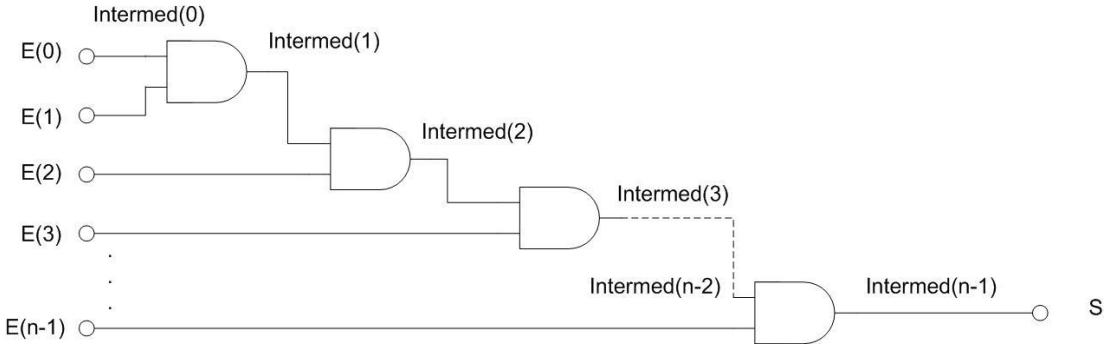
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY AND_n_entradas IS
    GENERIC(n : NATURAL := 2); -- Número de pinos de entrada
    PORT(E : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0);
         S : OUT STD_LOGIC);
END AND_n_entradas ;

ARCHITECTURE arquitetura OF AND_n_entradas IS
    -- Sinal intermediário para receber os resultados parciais
    SIGNAL Intermed : STD_LOGIC_VECTOR(n-1 DOWNTO 0);
BEGIN
    Intermed(0) <= E(0); -- Primeiro pino

    and_for: FOR i IN 1 TO n-1 GENERATE -- Cria (n-1) portas AND
        Intermed(i) <= Intermed(i-1) AND E(i);
    END GENERATE;

    S <= Intermed(n-1); -- Saída

END arquitetura;
```

GENERATE – FOR : Exemplo

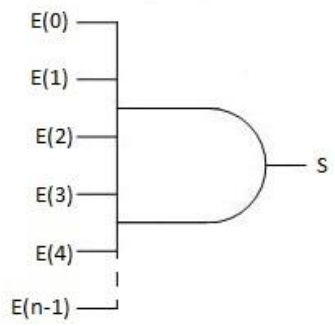
É necessário atribuir separadamente os valores de `Intermed(0)` e `S` através das linhas:

```
Intermed(0) <= E(0); -- Primeiro pino
S <= Intermed(n-1);  -- Saída
```

Isto é devido ao fato de que as operações para `i = 0` e `i = n-1` não seguem a mesma regra de geração, pois não existe **`Intermed(-1)`** e **`Intermed(n-1)`** é desnecessário. Utilizando o esquema de geração IF é possível incluir os casos citados no comando GENERATE.



Circuito Descrito



Circuito Sintetizado

GENERATE – IF

O esquema IF insere uma réplica de um conjunto de comandos caso a condição contida após a palavra reservada IF seja satisfeita. Sintaxe:

```
<rótulo_obrigatório>: IF <condição> GENERATE
    -- Comandos concorrentes
END GENERATE < rótulo_obrigatório>;
```

GENERATE – IF : Exemplo

Reescrevendo o exemplo anterior, utilizando o esquema IF, a descrição se torna:

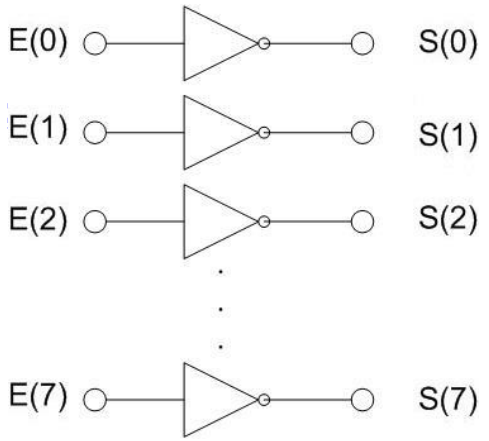
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY AND_n_entradas IS
    GENERIC(n : NATURAL := 2); -- Número de pinos de entrada
    PORT(E : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0);
         S : OUT STD_LOGIC);
END AND_n_entradas ;

ARCHITECTURE arquitetura OF AND_n_entradas IS
    -- Sinal intermediário para receber os resultados parciais
    SIGNAL Intermed : STD_LOGIC_VECTOR(n-1 DOWNTO 0);
BEGIN

    and_for: FOR i IN 0 TO n-1 GENERATE -- Cria (n-1) portas AND
        caso1: IF i = 0 GENERATE -- Bloco gerado se i = 0
            Intermed(i) <= E(i);
        END GENERATE caso1;

        caso2: IF (i > 0) AND (i < n-1) GENERATE -- Se i ∈ [1,n-2]
            Intermed(i) <= Intermed(i-1) AND E(i);
        END GENERATE caso2;

        caso3: IF i = n-1 GENERATE -- Se i = n-1
            S <= Intermed(i-1) AND E(i);
        END GENERATE caso3;

    END GENERATE and_for;

END arquitetura;
```

GENERATE – Instanciação Iterativa de Componentes

A chamada de componentes é também um comando concorrente e, portanto, pode-se utilizar o comando GENERATE para criar estruturas regulares utilizando componentes.

Exemplo: tomando um componente que seja uma porta NOT, pode-se criar uma estrutura que inverte todos os *bits* de um vetor:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY not_vetor IS
    PORT(E : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
         S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0))
END not_vetor ;

ARCHITECTURE arquitetura OF not_vetor IS
    COMPONENT porta_not IS
        PORT(entrada : IN  STD_LOGIC;
             saida   : OUT STD_LOGIC);
    END porta_not;

BEGIN
    for_not: FOR i IN 0 TO 7 GENERATE -- Cria 8 portas NOT
        x : porta_not PORT MAP(E(i),S(i));
    END GENERATE;

END arquitetura;
```



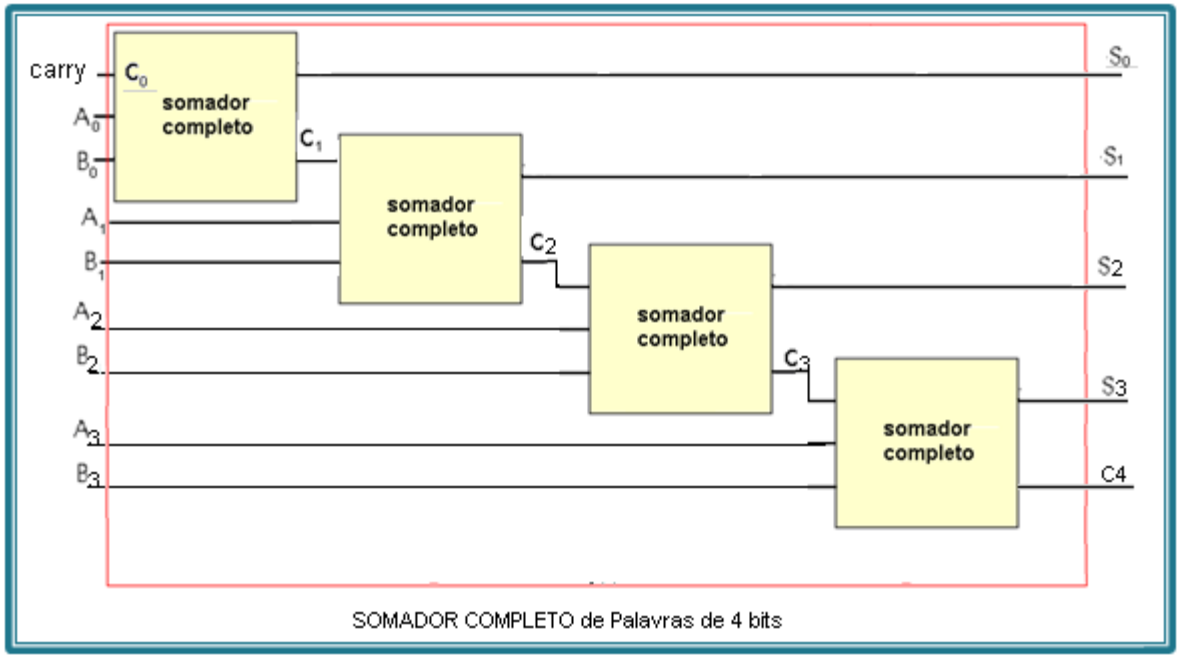
Circuito Descrito

GENERATE – Aninhamento

É permitido aninhar comandos GENERATE, a fim de se criar estruturas multidimensionais:

```
linha: FOR i IN 0 TO 7 GENERATE
    coluna: FOR j IN 0 TO 7 GENERATE
        x : componente_abc PORT MAP(Entrada_a(i),Entrada_b(j),Saída(i+1));
    END GENERATE;
END GENERATE;
```

Exercício 12 Utilizando o comando **GENERATE FOR** crie um projeto de um somador completo de palavras de 4 bits, utilizando GENERIC para determinar n bits.



LOOP

O comando **sequencial** LOOP utiliza dois esquemas de iteração para repetir comandos sequenciais:

- Esquema FOR
- Esquema WHILE

VARIABLE (Variável):

é usado para Armazenar valores imediatos em cálculos de modelos comportamental e pode ser alterada a qualquer momento.

- São utilizadas apenas em processos e devem ser declaradas entre o PROCESS e o BEGIN
- São atualizadas imediatamente e não correspondem à implementação física (como no caso dos sinais).

Sintaxe se a variável tem valor inicial:

```
variable nome_variavel : tipo [restrição] [:=valor_inicial];
```

Sintaxe se a variável não tem valor inicial:

```
variable nome_variavel : tipo [restrição];
```

Comparação entre VARIABLE e SIGNAL

	Signal	Variable
Declaração	Em *architecture* ou como *port* na *entity*.	Dentro de *process*.
Atribuição	Recebe valor atribuído na suspensão do *process*. Somente a última atribuição é válida. Ex: A<= B + C	Recebe valor atribuído imediatamente. Toda atribuição é válida. Ex: A:= B + C
Atraso	Inercial e Transporte	Não há

VARIABLE é utilizada dentro de PROCESS
Lembrando que:

- Um process, como qualquer processo em VHDL (e como qualquer pedaço de hardware), está eternamente em execução;
- Dentro de um process, a avaliação dos comandos é sequencial, ao contrário do que ocorre em VHDL fora de um process, onde tudo é avaliado em paralelo
- Cada comando pode ter efeito sobre (atribuir novos valores a) sinais e/ou variáveis
  - ✓ Atribuições a variáveis têm efeito imediato, como em programação , na linha que é modificada
  - ✓ Atribuições a sinais são projeções para o futuro, atualiza ao sair do process

LOOP

O comando **sequencial** LOOP utiliza dois esquemas de iteração para repetir comandos sequenciais:

- Esquema FOR
- Esquema WHILE (não sintetizável)

LOOP – FOR

O esquema FOR repete um conjunto de comandos sequenciais uma quantidade determinada de vezes, é fornecida uma variável local e os limites para esta variável. Por exemplo, o código abaixo repete os comandos concorrentes 4 vezes:

```
abc: FOR i IN 0 TO 3 LOOP
    -- Comandos sequenciais
END LOOP;
```

A sintaxe de declaração do esquema FOR é como a seguir:

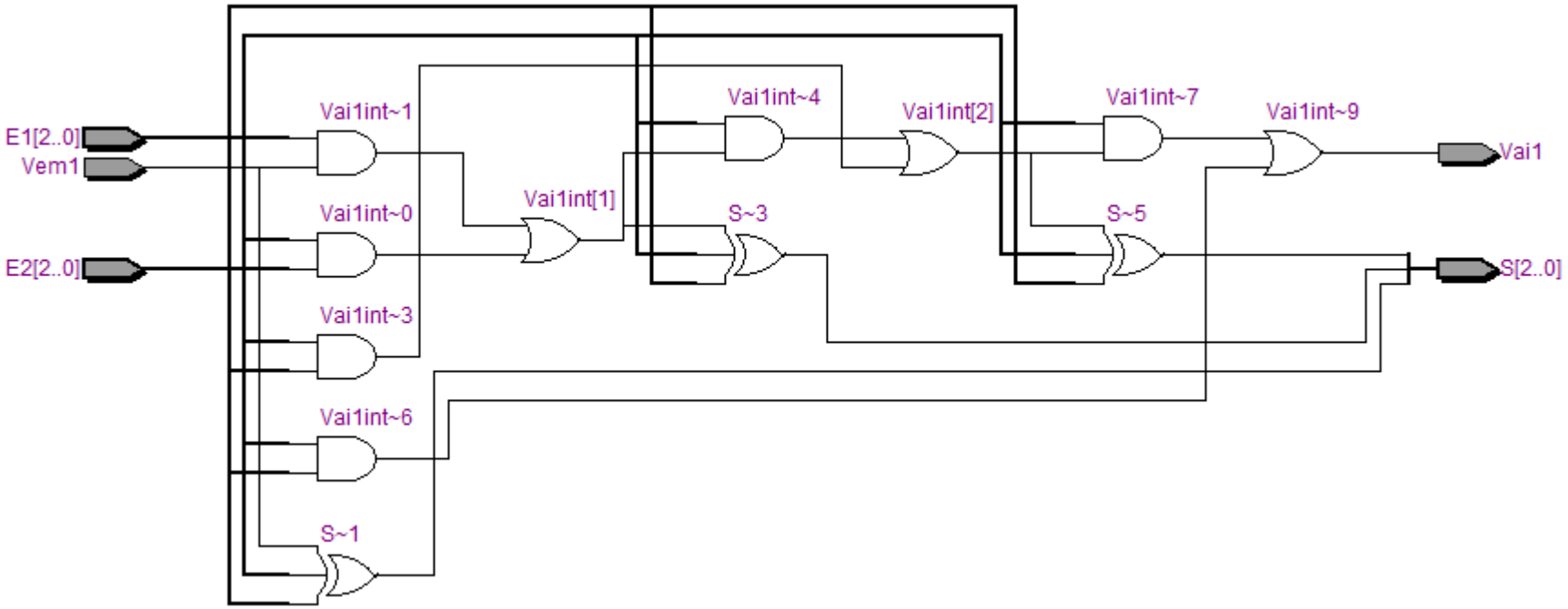
```
<rótulo_obrigatório>: FOR <variável_local> IN <limites_da_variável> LOOP
    -- Comandos sequenciais
END LOOP <rótulo_opcional>;
```

LOOP – FOR : Exemplo

Como exemplo de uso, o código a seguir descreve um somador de n bits.

```
ENTITY somador IS
    GENERIC(n : NATURAL := 3);                      -- Número de bits
    PORT(E1, E2 : IN  BIT_VECTOR(n-1 DOWNTO 0); -- Entradas
         Vem1   : IN  BIT;                          -- Carry in
         Vai1   : OUT BIT;                          -- Carry out
         S      : OUT BIT_VECTOR(n-1 DOWNTO 0));
END somador;

ARCHITECTURE arquitetura OF somador IS
    SIGNAL Vai1int : BIT_VECTOR(n DOWNTO 0); -- Carry out interno
BEGIN
    PROCESS(E1, E2, Vem1)
    BEGIN
        Vai1int(0) <= Vem1;
        abc: FOR i IN 0 TO n-1 LOOP
            S(i) <= E1(i) XOR E2(i) XOR Vai1int(i);
             Vai1int(i+1) <= (E1(i) AND E2(i)) OR
                             (E1(i) AND Vai1int(i)) OR
                             (E1(i) AND Vai1int(i));
        END LOOP;
        Vai1 <= Vai1int(n); -- Saída
    END PROCESS;
END arquitetura;
```

LOOP – FOR : Exemplo



Circuito Sintetizado

LOOP – WHILE

O esquema WHILE insere uma réplica de um conjunto de comandos caso a condição contida após a palavra reservada WHILE seja satisfeita. Sintaxe:

```
<rótulo_obrigatório>: WHILE <condição> LOOP
    -- Comandos sequenciais
END LOOP < rótulo_obrigatório>;
```

Obs.: Para o *software* Quartus II, se a condição de parada for um índice que é incrementado a cada laço (por exemplo, condição : i < 3, índice : i := i + 1;), então o índice deve ser do tipo VARIABLE. Caso seja usado um SIGNAL, a ferramenta não consegue sintetizar o circuito.

VARIABLE (Variável):

é usado para Armazenar valores imediatos em cálculos de modelos comportamental e pode ser alterada a qualquer momento.

- São utilizadas apenas em processos e devem ser declaradas entre o PROCESS e o BEGIN
- São atualizadas imediatamente e não correspondem à implementação física (como no caso dos sinais).

Sintaxe se a variável tem valor inicial:

```
variable nome_variavel : tipo [restrição] [:=valor_inicial];
```

Sintaxe se a variável não tem valor inicial:

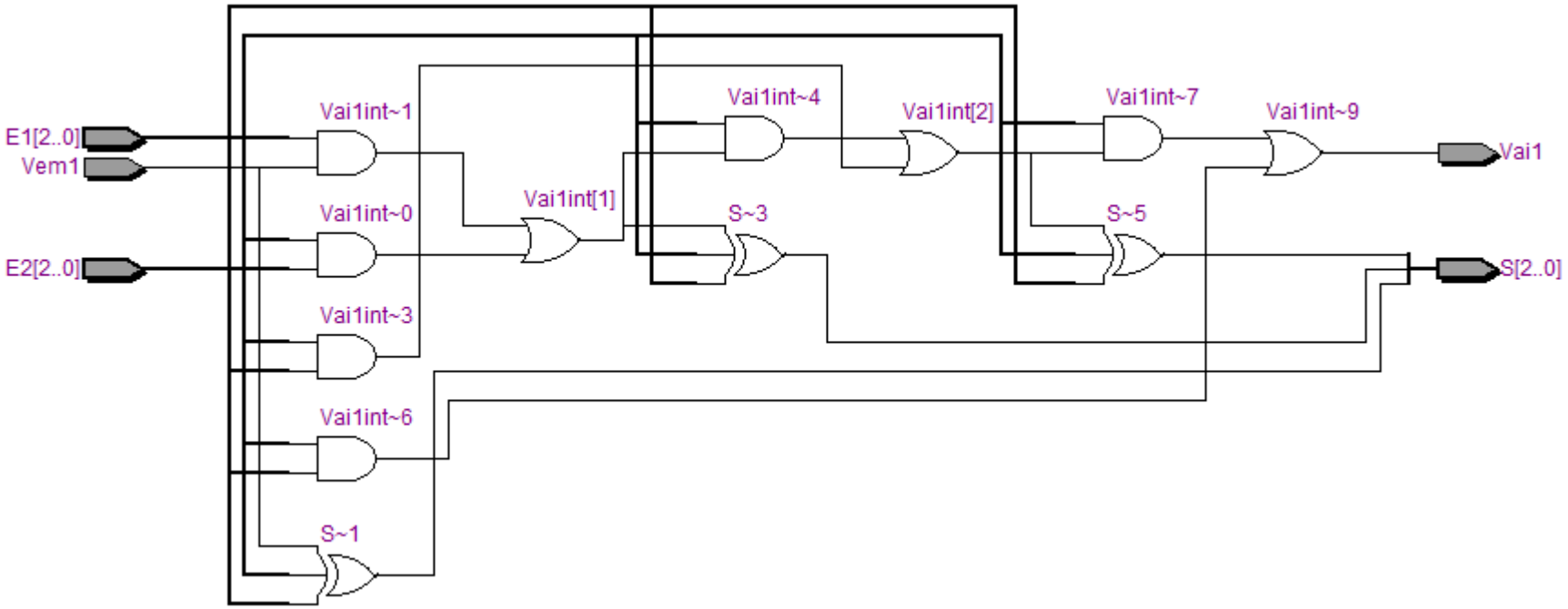
```
variable nome_variavel : tipo [restrição];
```

LOOP – WHILE : Exemplo

Usando o esquema WHILE , a descrição do somador de n bits se torna:

```
ENTITY somador IS
    GENERIC(n : NATURAL := 3);                    -- Número de bits
    PORT(E1, E2 : IN  BIT_VECTOR(n-1 DOWNTO 0); -- Entradas
         Vem1   : IN  BIT;                        -- Carry in
         Vai1   : OUT BIT;                        -- Carry out
         S      : OUT BIT_VECTOR(n-1 DOWNTO 0));
END somador;

ARCHITECTURE arquitetura OF somador IS
    SIGNAL Vai1int : BIT_VECTOR(n DOWNTO 0); -- Carry out interno
BEGIN
  PROCESS(E1, E2, Vem1)
    VARIABLE i : INTEGER RANGE 0 TO 2**n-1; -- Controle do laço
  BEGIN
    Vai1int(0) <= Vem1;
    i := 0;
    abc: WHILE i <= n-1 LOOP
        S(i) <= E1(i) XOR E2(i) XOR Vai1int(i);
        Vai1int(i+1) <= (E1(i) AND E2(i)) OR (E1(i) AND Vai1int(i)) OR
                        (E1(i) AND Vai1int(i));
        i := i + 1;
    END LOOP;
    Vai1 <= Vai1int(n); -- Saída
  END PROCESS;
END arquitetura;
```

LOOP – WHILE : Exemplo



Circuito Sintetizado