Universidade de São Paulo
Escola de Engenharia de São Carlos
Depto. de Engenharia Elétrica e de Computação

Introdução a VHDL

Aula 6

Professora Luiza Maria Romeiro Codá

Aula 6: Introdução a VHDL
sala

Conteúdo:

- Declaração de VARIABLE
- Esquemas de Iteração em Região de códigos sequenciais: comando LOOP:

1. LOOP FOR : Comandos NEXT e EXIT
2. LOOP WHILE: Comandos NEXT e EXIT

- Prática nº9: `registrador usando os comandos LOOP FOR e LOOP WHILE`

Declaração de VARIABLE (Variável):

é usada 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

Variáveis

- são utilizadas dentro de PROCESS;
- São declaradas entre a palavra PROCESS e BEGIN;
- Suas atribuições têm efeito imediato, como em programação , na linha que é modificada;
- só são válidas dentro do PROCESS que são declaradas e não podem ser utilizadas em outros Process;

Sinais:

- são utilizados para comunicação entre componentes e módulos;
- são declarados na architecture (podendo também não usualmente serem declarados na entity ou package);
- Suas atribuições são projeções para o futuro, atualizadas ao sair do process;
- São válidos onde são declarados, ex: em toda a arquitetura que foi declarada.

Comparação entre VARIABLE e SIGNAL

Tanto sinais como variáveis podem ser especificados intervalo e valor inicial:

```
VARIABLE v1 : INTEGER RANGE 1 to 60:=1;

SIGNAL s1 : INTEGER RANGE 1 to 60:=1;

VARIABLE v2: STD_LOGIC:='0';

SIGNAL s1 : STD_LOGIC:='0';
```

Comparação entre VARIABLE e SIGNAL

	SIGNAL	VARIABLE
Declaração	Na architecture ou como PORT na entity	Dentro de PROCESS
Utilização	utilizados para comunicação entre componentes.	usadas apenas em processos e subprogramas (funções e procedimentos)
Interpretação	podem ser interpretados como fios físicos, reais.	usualmente não estão disponíveis para múltiplos componentes e processos
Atribuição	Recebe valor atribuído na suspensão do Process. Sómente a última atribuição é válida	Todas as atribuições de variáveis tem efeito imediato. Recebe valor atribuído imediatamente. Toda atribuição é válida
Atraso	Inercial e de transporte	Não há

Operações com Sinais e Variáveis

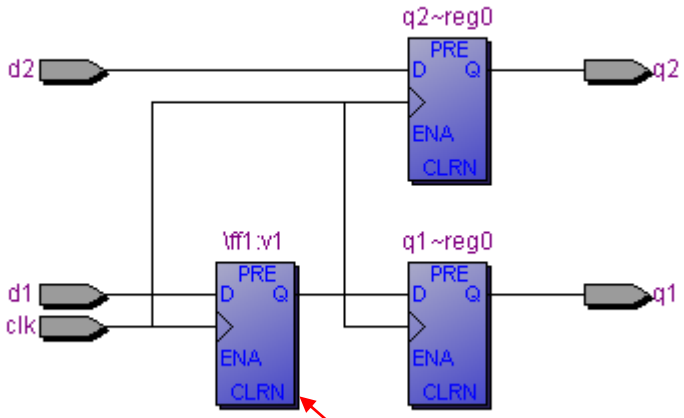
Retirado de: http://paginapessoal.utfpr.edu.br/erig/logica-reconfiguravel/VHDL/sigval.vhd/view

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sigval IS
PORT (        a,b,c : IN STD_LOGIC; --entradas
              sigout1, sigout2, sigout3, varout1,varout2, varout3: OUT STD_LOGIC); -- saidas
END sigval;

ARCHITECTURE regras OF sigval IS
    SIGNAL sig1,sig2 : STD_LOGIC; -- sinais auxiliares
        BEGIN
            PROCESS (a, b, c)
            VARIABLE var1,var2 : STD_LOGIC; -- variaveis auxiliares
            BEGIN
             -- atribuições iniciais para um sinal e para uma variável
                    sig1 <= '1';
                    var1 := '1';
             -- a definição é enviada para as saidas
                    sigout1 <= sig1; -- um sinal pode ser enviado para uma saída
                    varout1 <= var1; -- ima variável também pode ser enviada para saida.
             -- operação com sinais e variáveis
                    sig1 <= a AND b; --  a atribuição anterior de sig1 vai ser ignorada.
                    var1 := a AND b; --  a atribuição anterior de var1 NÂO vai ser ignorada.
             -- resultado é enviado para saída.
                    sigout2 <= sig1;
                    varout2 <= var1;
                    sig2 <= sig1 OR c; -- operações com sinais
                    var2 := var1 OR c; -- operações com variaveis
                    sigout3 <= sig2;
                    varout3 <= var2;
                    END PROCESS;
END regras;
```

Emprego de VARIABLE

https://www.youtube.com/watch?v=TeceQQaVuS0

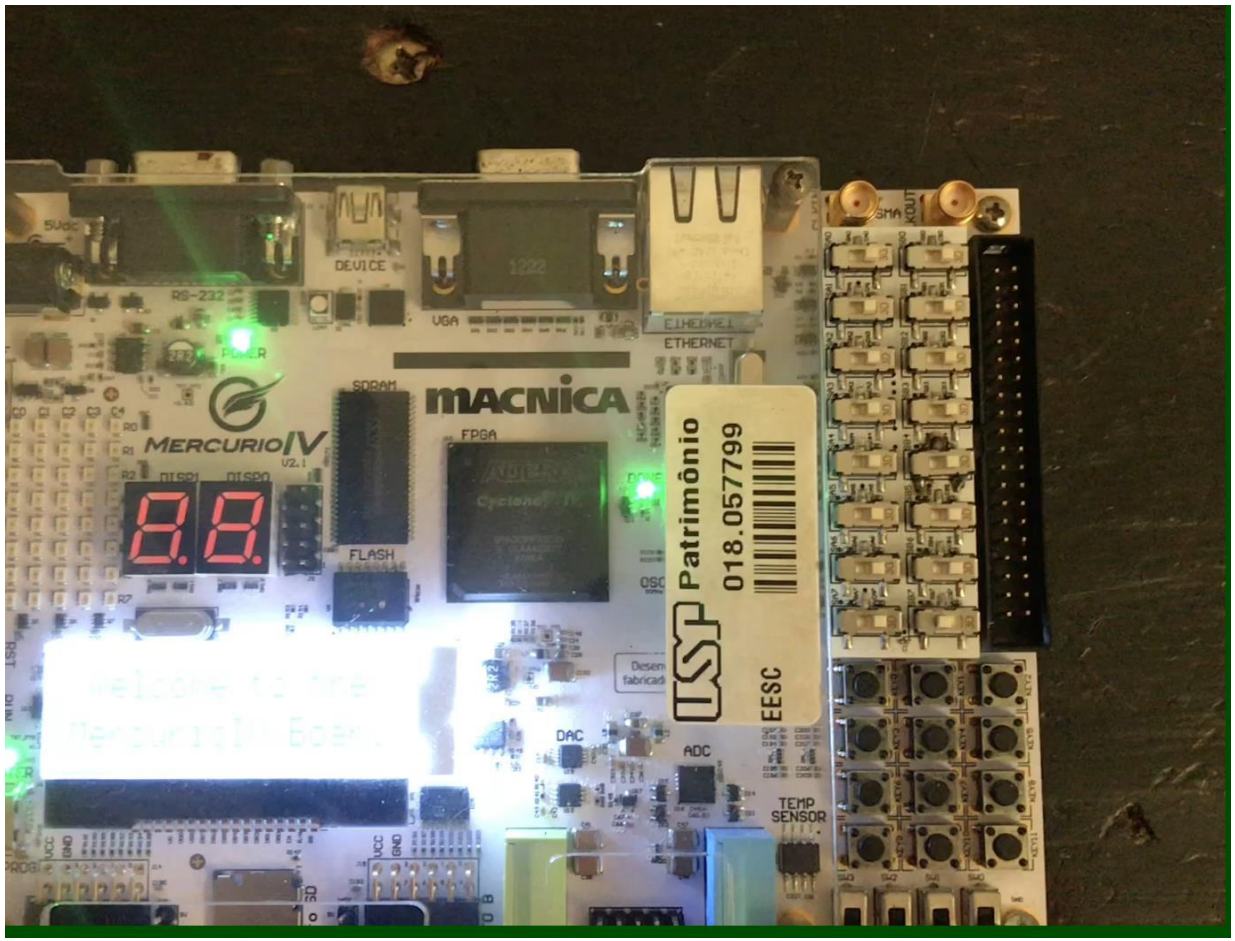
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY FF_VAR IS
    PORT( clk ,d1, d2 : IN STD_LOGIC;
                  q1,q2 : OUT STD_LOGIC);
END FF_VAR;
ARCHITECTURE variavel OF FF_VAR IS
BEGIN
 ff1: PROCESS (clk)
            VARIABLE v1 : STD_LOGIC;
            BEGIN
          IF   RISING_EDGE (clk)THEN
            q1 <= v1; -- variável v1 utilizada sem receber novo valor
             v1 := d1; -- variável v1 recebe novo valor
            END IF;
  END PROCESS ff1;
ff2: PROCESS (clk)
            VARIABLE v2 : STD_LOGIC;
            BEGIN
          IF  RISING_EDGE (clk) THEN
             v2 := d2; -- variável v2 recebe  valor inicial
            q2 <= v2;
            END IF;
  END PROCESS ff2;
END variavel;
```

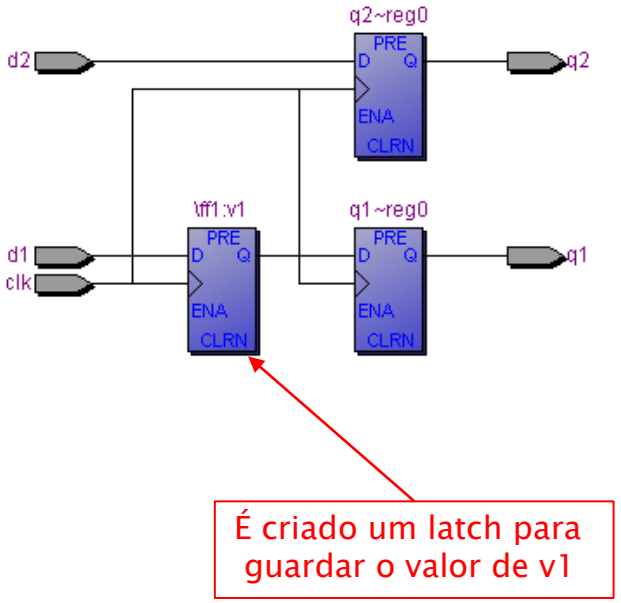


É criado um latch para guardar o valor de v1

Emprego de VARIABLE

https://www.youtube.com/watch?v=TeceQQaVuS0





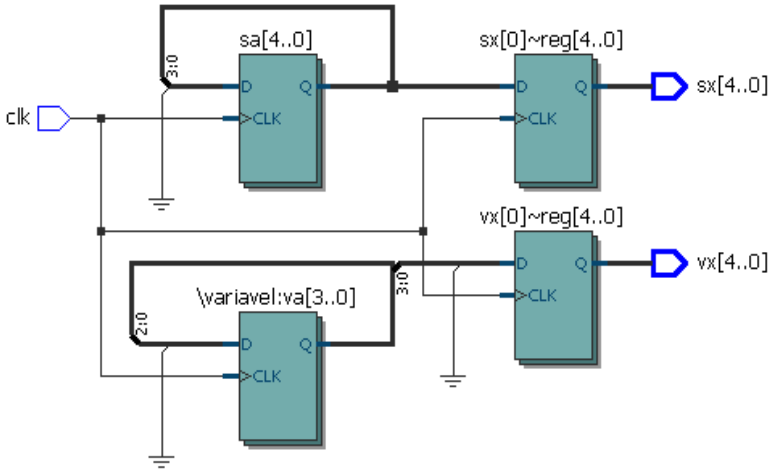
VARIABLE X SIGNAL

A diferença fundamental entre variáveis e sinais é o atraso da atribuição

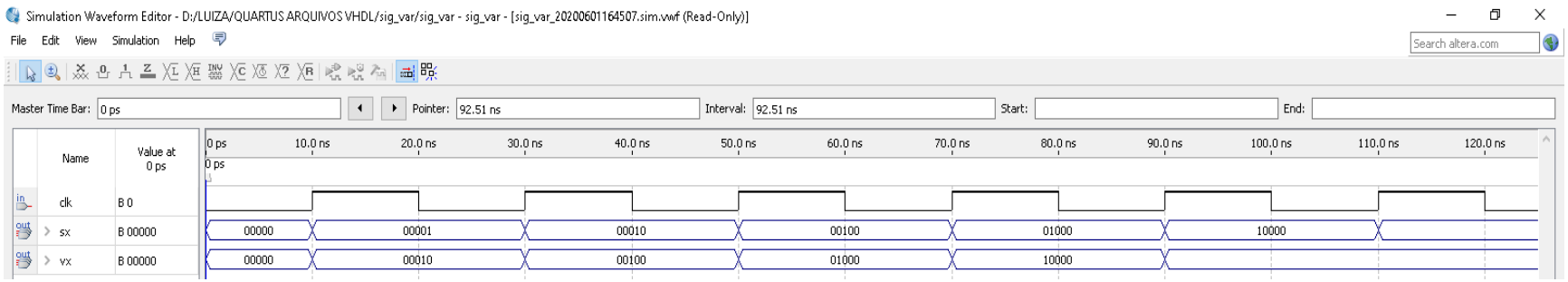
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY sig_var IS
GENERIC (n: NATURAL :=16);
PORT( clk: IN STD_LOGIC;
        sx : OUT INTEGER RANGE 0 TO n; -- saída de teste do sinal
        vx : OUT INTEGER RANGE 0 to n); -- saída de teste da variável
END sig_var;

ARCHITECTURE teste OF sig_var IS
SIGNAL sa : INTEGER RANGE 0 to n :=1;
BEGIN
sinal:   PROCESS (clk)
         BEGIN
               IF RISING_EDGE (CLK) THEN
                              sa <= sa + sa;
                              sx<= sa;
               END IF;
          END PROCESS sinal;
variavel:PROCESS (clk)
       VARIABLE va : INTEGER RANGE 0 to n :=1;
       BEGIN
               IF RISING_EDGE (CLK) THEN
                              va := va + va;
                              vx <= va;
               END IF;
             END PROCESS variavel;
END teste;
```

VARIABLE X SIGNAL

Circuito Sintetizado:



O valor da variável é atribuído na mesma hora e do sinal só é atualizado após fim do PROCESS



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 região de códigos **concorrentes**, é utilizado o comando GENERATE,
- ✓ para região de códigos **sequenciais**, é utilizado o comando LOOP.

Para ambos os comandos de iteração, há dois esquemas:

- Um repete os comandos um número determinado de vezes
- o outro repete os comandos caso uma expressão de condição seja atendida.

Esquemas de Iteração – LOOP

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

- Incondicional (utilizado para simulação– não síntese)
- Esquema FOR
- Esquema WHILE (não sintetizável em algumas ferramentas)

OBS: Esses Esquemas são utilizados na região iniciada pelo comando PROCESS onde variaveis (VARIABLE) são declaradas

Esquemas de Iteração – LOOP

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

- Esquema FOR
- Esquema WHILE (não sintetizável em algumas ferramentas)

Esquemas de Iteração – LOOP FOR

O esquema FOR LOOP repete um conjunto de comandos sequenciais um número específico de vezes (equivalente ao GENERATE, mas para código sequencial). Este comando usa um contador, e desde que o valor do contador esteja em uma certa faixa, o loop é executado. Ao término de cada execução do loop, o contador é atualizado.

Para utilizar o comando FOR LOOP deve-se fornecer uma variável local e os limites para esta variável.
Por exemplo, o código abaixo repete os comandos 4 vezes:

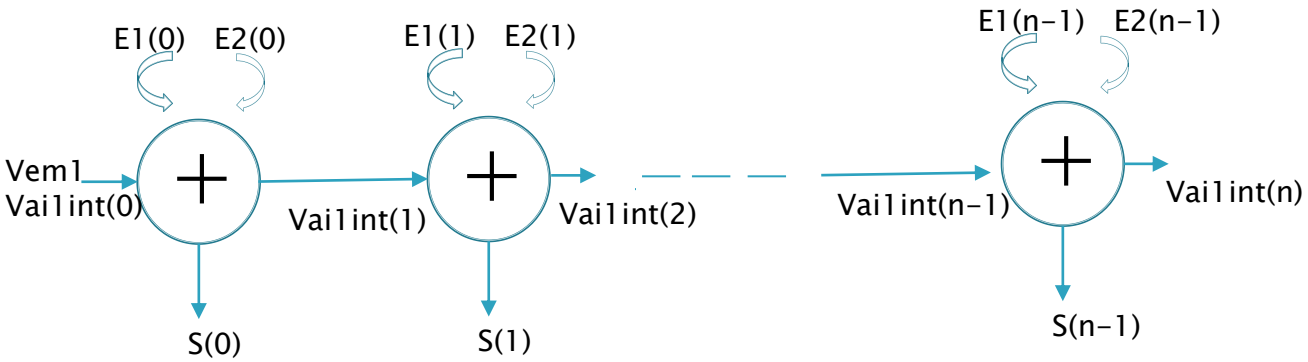
```
abc: FOR i IN 0 TO 3 LOOP
    -- Comandos sequenciais
END LOOP abc;
```

A sintaxe :

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

LOOP – FOR : Exemplo

Somador completo de 2 palavras (E1 e E2) de n bits.



$$S_k = E1_k \oplus E2_k \oplus Vai1_k$$

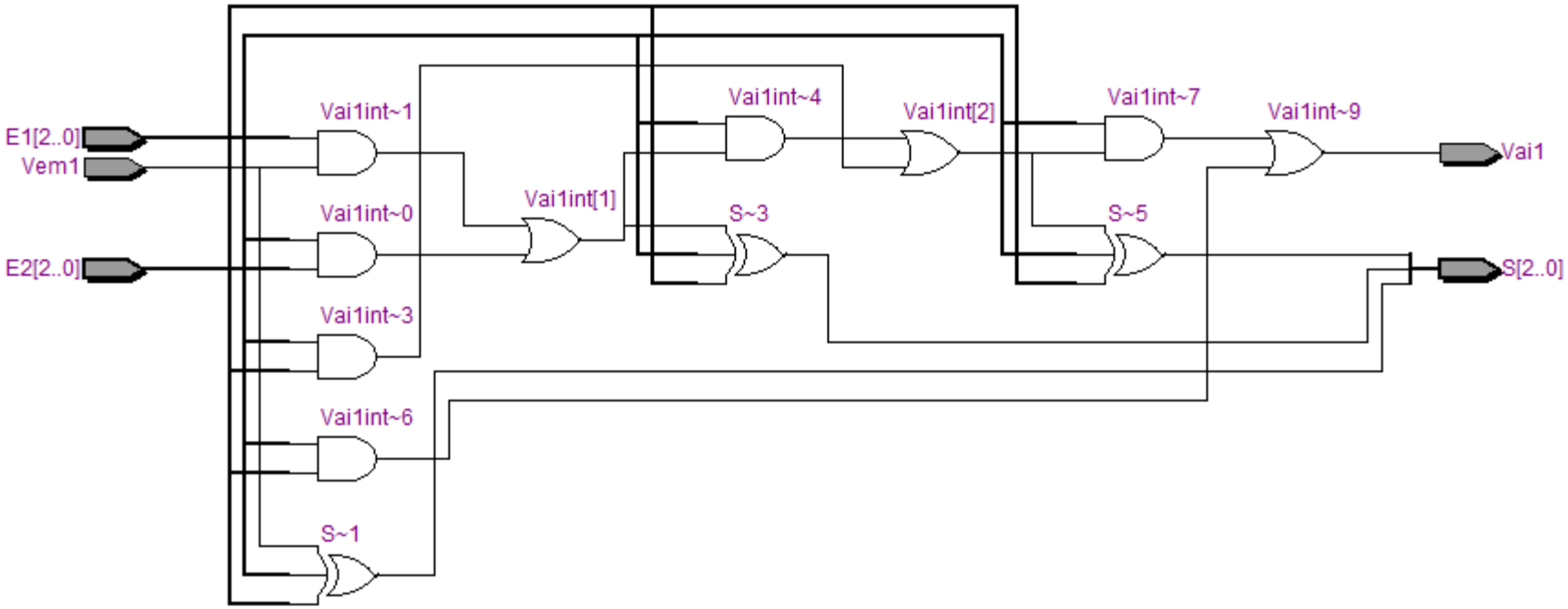
$$Vai1_{k+1} = E1_k . E2_k + E1_k . Vai1_k + E2_k . Vai1_k$$

LOOP – FOR : Exemplo

código VHDL de um somador completo de 2 palavras (E1 e E2) de n bits.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY somador IS
    GENERIC(n : NATURAL := 3);                    -- Número de bits
    PORT(E1, E2 : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0); -- Entradas
         Vem1   : IN  STD_LOGIC;                  -- Carry in
         Vai1   : OUT STD_LOGIC;                  -- Carry out
         S      : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END somador;

ARCHITECTURE arquitetura OF somador IS
    SIGNAL Vai1int : STD_LOGIC_VECTOR(n DOWNTO 0); -- Carry out interno
BEGIN
    PROCESS(E1, E2, Vem1)
    BEGIN
        Vai1int(0) <= Vem1; -- é um pino (entrada)
        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
                             (E2(i) AND Vai1int(i));
        END LOOP abc;
        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_opcional>: WHILE <condição> LOOP
    -- Comandos sequenciais
END LOOP < rótulo_opcional>;
```

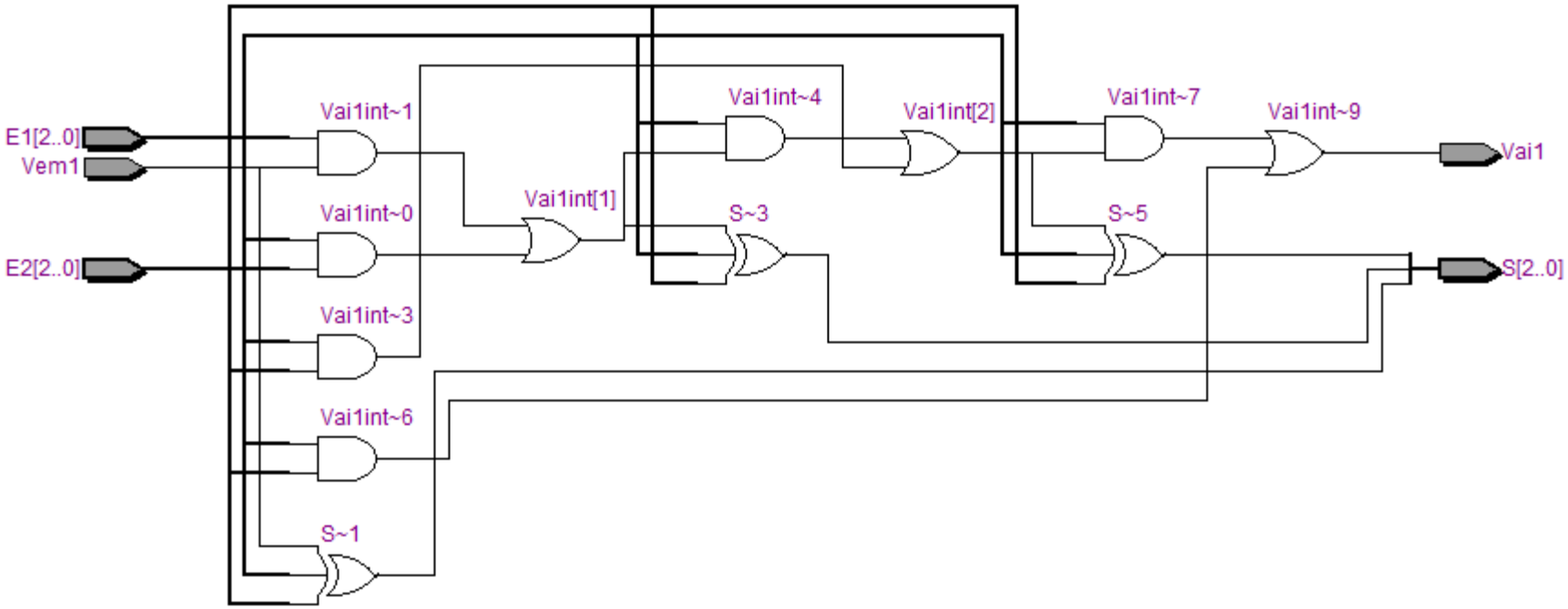
Obs.: Para o *software* Quartus II, se a condição verifica um índice que é incrementado a cada laço (por ex:, condição : i < 3, índice : i := i + 1;), então o índice deve ser do tipo VARIABLE. Caso seja usado um SIGNAL, a ferramenta não consegue sintetizar o circuito.

LOOP – WHILE : Exemplo

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

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY somador_W IS
    GENERIC(n : NATURAL := 3);                    -- Número de bits
    PORT(E1, E2 : IN  STD_LOGIC_VECTOR(n-1 DOWNTO 0); -- Entradas
         Vem1   : IN  STD_LOGIC;                  -- Carry in
         Vai1   : OUT STD_LOGIC;                  -- Carry out
         S      : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END somador_W;

ARCHITECTURE a OF somador_W IS
    SIGNAL Vai1int : STD_LOGIC_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 abc;
    Vai1 <= Vai1int(n); -- Saída
  END PROCESS;
END a;
```

LOOP – WHILE : Exemplo



Circuito Sintetizado

Comandos NEXT e EXIT

Os Comando NEXT e EXIT possibilitam alterar as sequências de operações executadas em um comando LOOP.

- **Comando** NEXT termina a execução da iteração atual e começa a subseqüente portanto, causa um salto para o final da iteração corrente, omitindo a execução dos comandos restantes no esquema de iteração e vai diretamente para a próxima interação.

- **Comando** EXIT termina a execução da iteração atual e do loop, leva a finalização do LOOP em operação.

Comando NEXT

Sintaxe:

```
<rótulo_opcional_LOOP>: FOR <variável_local> IN <limites_da_variável> LOOP
    -- Comandos sequenciais
        <rótulo_next_opcional>: NEXT <rótulo_opcional_LOOP> WHEN [condição];
    -- Comandos sequenciais

END LOOP <rótulo_opcional_LOOP>;
```

Exemplos:

NEXT; --pula para a próxima iteração no mesmo LOOP

NEXT WHEN condicao_1; --pula para a proxima iteração caso a condicao_1 seja verdadeira

Salto: NEXT WHEN condicao_1; -- idem com rótulo opcional

EXIT salto_1 WHEN condicao_3; -- pula para salto_1 onde salto_1 é um rótulo

Exemplo com Comando NEXT:

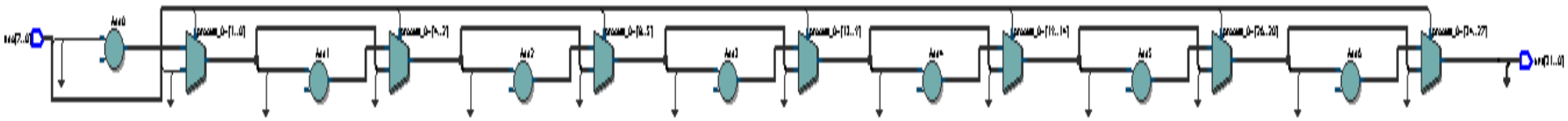
Conta número de “uns” em um vetor

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY conta_uns IS

PORT(dado : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); -- Entradas
         uns: OUT INTEGER);
END conta_uns;

ARCHITECTURE a OF conta_uns IS
BEGIN
  PROCESS(dado)
    VARIABLE nde_uns : INTEGER; -- contador de número de bits 1
  BEGIN
 nde_uns := 0;-- zera o contador de uns a cada novo dado

FOR i IN 0 TO 7  LOOP
         NEXT WHEN  dado(i)= '0'; --verifica se bit do dado é zero se verdadeiro
                          -- passa para a iteração seguinte, verificar o próximo
                               --bit do dado e não acrescenta ao valor de uns
         nde_uns := nde_uns +1;
         END LOOP;
     uns <= nde_uns;
  END PROCESS;
END a;
```

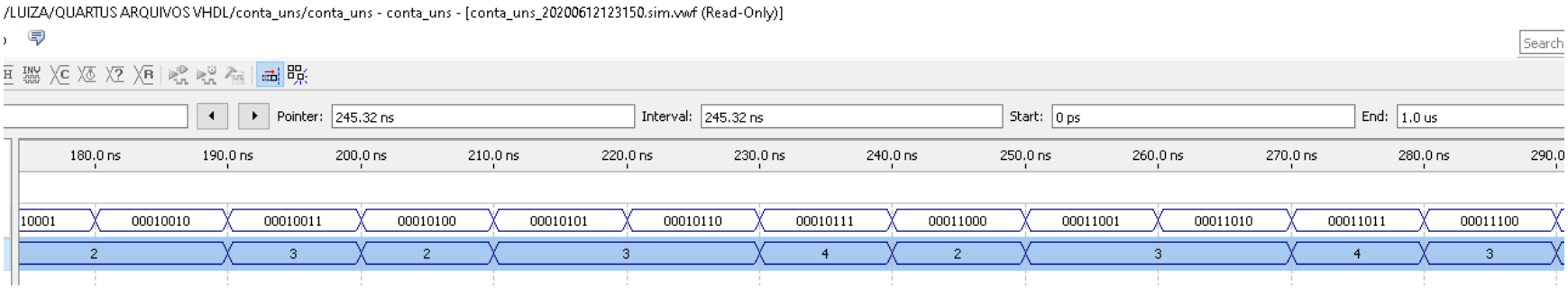
Exemplo com Comando NEXT:

Conta número de “uns” em um vetor

Circuito gerado:



Simulação:



Comando EXIT

Sintaxe:

```
<rótulo_opcional_LOOP>: FOR <variável_local> IN <limites_da_variável> LOOP
    -- Comandos sequenciais
        <rótulo_exit_opcional>: EXIT <rótulo_opcional_LOOP> WHEN [condição];
    -- Comandos sequenciais

END LOOP <rótulo_opcional_LOOP>;
```

Exemplos:

EXIT; --termina a iteração, finaliza o LOOP

EXIT WHEN condicao_1 -finaliza iteração caso a condicao_1 seja verdadeira

fim: EXIT WHEN condicao_1; -- idem com rótulo opcional

EXIT salto_1 WHEN condicao_3; --termina a iteração e pula para salto_1 que é um rótulo

Exemplo com Comando EXIT:

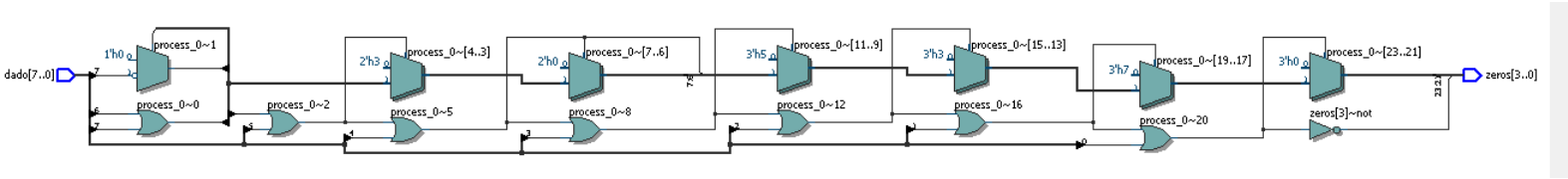
Contador de zeros à esquerda:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY conta_zeros IS

PORT(dado : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); -- Entradas com 8 bits
         zeros: OUT INTEGER RANGE 0 TO 8 ); -- Número de bits zer0
END conta_zeros;

ARCHITECTURE a OF conta_zeros IS
BEGIN
  PROCESS(dado)
    VARIABLE conta : INTEGER RANGE 0 TO 8; -- Controle do laço
  BEGIN
     conta:=0;
     FOR i IN dado'RANGE LOOP -- dado'RANGE é igual a colocar o range 7 DOWNTO 0
         CASE dado(i) IS --verifica começando do bit dado(7) e vai decrescendo
             WHEN '0' => conta := conta +1;
             WHEN OTHERS => EXIT;--se encontrar um bit diferente de 0 pára loop
         END CASE;
     END LOOP;
     zeros <= conta;
  END PROCESS;
END a;
```

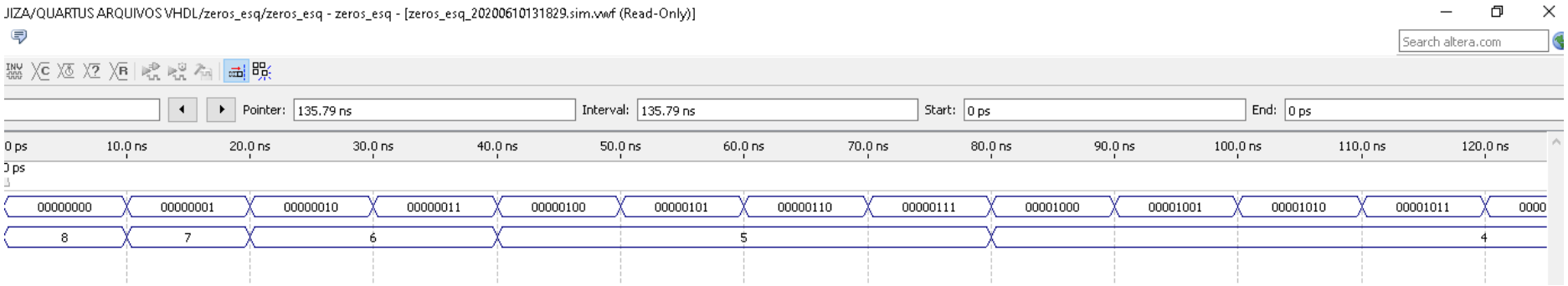
Exemplo com Comando EXIT:

Contador de zeros à esquerda:

Circuito Gerado:



Simulação



Prática nº9

Crie 2 projetos de um registrador de deslocamento da esquerda para a direita com n bits, utilizando o comando LOOP FOR e outro usando o comando LOOP WHILE. Inclua a cláusula GENERIC para determinar n bits.