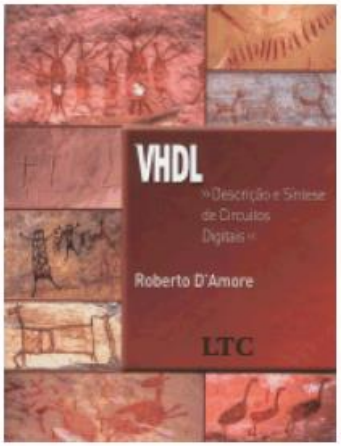
SEL5752/SEL0632 – Linguagens de Descrição de Hardware
Aula 15 – Arquivos e Testes

Prof. Dr. Maximiliam Luppe

Livro adotado:

VHDL - Descrição e Síntese de Circuitos Digitais
Roberto d'Amore

ISBN 85-216-1452-7
Editora LTC www.ltceditora.com.br



Para informações adicionais consulte: **www.ele.ita.br/~damore/vhdl**

Operações com arquivos

Tópicos

- Definição de tipos arquivo
- Pacote TEXTIO

Tópicos avançados

Tópicos

- Declaração ASSERT
- Declaração REPORT

Teste

Tópicos

- Testes
- Teste de uma descrição

Emprego de arquivos

- **LRM (Language Reference Manual):**

  " O tipo arquivo é usado para definir objetos representando arquivos no ambiente do sistema hospedeiro"

  "O valor de um objeto tipo arquivo é uma seqüência de valores armazenados em um arquivo "

- **A manipulação de informações armazenadas em arquivos necessita:**

  1- declaração de um tipo arquivo

  2- objetos declarados segundo um tipo arquivo definido

Declaração de um tipo arquivo

- **Declaração de um tipo arquivo:**
  - identificador do tipo: arquivos_de_tipo_x
  - tipo que um objeto FILE arquivos_de_tipo_x pode conter: tipo_contido

```
TYPE arquivos_de_tipo_x  IS FILE OF tipo_contido;

TYPE arqv_inteiro    IS FILE OF INTEGER;    -- tipo FILE p/ objetos armaz. INTEGER
TYPE arqv_bit_vector IS FILE OF BIT_VECTOR; -- tipo FILE p/ objetos armaz. BIT_VECTOR
TYPE arqv_caracter   IS FILE OF CHARACTER;  -- tipo FILE p/ objetos armaz. CHARACTER
```

- **Outros exemplos**:

```
-- tipo FILE para objetos armazenando vetores de 8 bits
TYPE arqv_byte IS FILE OF BIT_VECTOR(0 TO 7);

-- tipo FILE para objetos armazenando um sub-tipo
SUBTYPE sub_inteiro IS INTEGER RANGE 0 TO 99;
TYPE arqv_inteiro_0_99   IS FILE OF sub_inteiro;
```

Versão VHDL-1993

- **Versão VHDL-1987**:
  - abertura do arquivo ocorre na declaração de um objeto do tipo arquivo

- **Opção é possível na versão VHDL-1993**

  sintaxe → pequenas alterações com relação a versão VHDL-1987:

```
--FILE nome_objeto : arqv_de_tipo_x IS   modo    "nome_externo";           -- 1987
--FILE nome_objeto : arqv_de_tipo_x OPEN modo IS "nome_externo";           -- 1993


-- Exemplos:
--FILE leitura     : arqv_inteiro    IS    IN         "/vhdl/dados_r.dat";  -- 1987
  FILE leitura     : arqv_int       OPEN Read_Mode IS "/vhdl/dados_r.dat";  -- 1993

--FILE escrita     : arqv_caracter   IS   OUT          "/vhdl/dados_w.dat"; -- 1987
  FILE escrita     : arqv_int       OPEN Write_Mode IS "/vhdl/dados_w.dat"; -- 1993
```

Versão VHDL-1993

- **Nova opção para abertura de arquivo:**

 - omitir informações da abertura do arquivo na declaração de um objeto do tipo arquivo:

```
FILE nome_objeto : arqv_de_tipo_x;
```

- Opção anterior com a informação de abertura (vista na imagem anterior):

```
--FILE nome_objeto : arqv_de_tipo_x OPEN  modo       IS "nome_externo";

  FILE leitura     : arqv_int       OPEN Read_Mode  IS "/vhdl/dados_r.dat";

  FILE escrita     : arqv_int       OPEN Write_Mode IS "/vhdl/dados_w.dat";
```

Versão VHDL-1993

- **Subprogramas implicitamente definidos** (após a declaração do tipo FILE) :

```
READ   (arquivo_leitura, dado);        -- procedimento para leitura
WRITE  (arquivo_escrita, dado);        -- procedimento para escrita
ENDFILE(arquivo_c)                     -- funcao fim de arquivo, retorna tipo boleano
FILE_OPEN(arquivo_a, "nome_externo", Read_Mode);      -- abre um arquivo leitura
FILE_OPEN(arquivo_a, "nome_externo", Write_Mode);     -- abre um arquivo escrita
FILE_OPEN(arquivo_a, "nome_externo", Append_Mode);    -- abre um arquivo escrita
FILE_CLOSE(arquivo_a);                                -- fecha um arquivo
```

- **Operação dos subprogramas READ WRITE ENDFILE**:
  - similar a operação dos subprogramas com mesmo nome da versão anterior
- **File_Open Read_Mode**: abre arquivo para leitura
- **File_Open Write_Mode**: abre arquivo para escrita (limpa arquivo se existente)
- **File_Open Append_Mode**: abre arquivo para escrita (insere novos dados no final)
- **File_Close**: fecha arquivo

Versão VHDL-1993 (exemplo operação de escrita - similar a descrição anterior)

- **Tipo arqv_int**: tipo FILE para elementos do tipo INTEGER
- **Objeto arquivo_wr**: objeto do tipo arqv_int
- **Arquivo fechado no término da simulação**

```
 1 -- versao VHDL-1993
 2 -- o arquivo e' fechado apos o termino da simulacao
 3 ENTITY arqv_b IS
 4 END arqv_b;
 5
 6 ARCHITECTURE teste OF arqv_b IS
 7 BEGIN
 8   escreve: PROCESS
 9     TYPE arqv_int IS FILE OF INTEGER;
10     FILE arquivo_wr : arqv_int OPEN Write_Mode IS "/vhdl/dado_b.dat";
11    BEGIN
12    FOR dado IN 48 TO 57 LOOP  -- valor 48 ASCII corresponde a caracter "0"
13      WRITE(arquivo_wr, dado); -- esqueve valores 48 a 57 no aquivo
14    END LOOP;                  -- correspondem aos caracteres "0" a "9" ASCII
15   WAIT;
16   END PROCESS escreve;
17 END teste;
```

Versão VHDL-1993 (exemplo operação de escrita - abre e fecha arquivo)

- **Declaração do objeto arquivo_wr: não abre o arquivo**
- **Arquivo aberto na declaração da linha 11**
- **Arquivo fechado durante a simulação - linha 15**

```
 1 -- versao VHDL-1993
 2 ENTITY arqv_c IS
 3 END arqv_c;
 4
 5 ARCHITECTURE teste OF arqv_c IS
 6 BEGIN
 7   escreve: PROCESS
 8     TYPE arqv_int IS FILE OF INTEGER;
 9     FILE arquivo_wr  : arqv_int;
10   BEGIN
11     FILE_OPEN(arquivo_wr, "/vhdl/dado_c.dat",Write_Mode);
12     FOR dado IN 48 TO 57 LOOP   -- valor 48 ASCII corresponde a caracter "0"
13       WRITE(arquivo_wr, dado);  -- esqueve valores 48 a 57 no aquivo
14     END LOOP;                   -- correspondem aos caracteres "0" a "9" ASCII
15     FILE_CLOSE(arquivo_wr);     -- fecha o arquivo
16   WAIT;
17   END PROCESS escreve;
18 END teste;
```

Pacote TEXTIO

- **Fornece subprogramas para operação com arquivos no formato texto**
  - permitem uma leitura direta das informações
  - podem ser manipulados por um editor de texto
  - compatíveis com diferentes plataformas

- **Disponível em qualquer implementação da linguagem**

- **Desvantagem** (comparando com arquivos no formato binário)
  - interação com a ferramenta de simulação mais lenta
    - executada através de subprogramas

- **Formato dos arquivos:**
  - compostos por linhas
  - cada linha constituída por uma série de caracteres
  - final de cada linha: caracter de retorno de carro

- **Para tornar o pacote visível:**
  - incluir a cláusula USE STD.TEXTIO.ALL

Pacote TEXTIO

- **Tipos e sub-tipos definidos:**

```
TYPE    LINE IS ACCESS STRING;    -- LINE e' um ponteiro para um tipo STRING
TYPE    TEXT IS FILE OF STRING;   -- Tipo FILE composto de elementos do tipo STRING
TYPE    SIDE IS (RIGHT, LEFT);    -- Empregados para o posicionamento do dado
SUBTYPE WIDTH IS NATURAL;         -- Definir o tamanho de uma informacao
```

- **Tipo LINE**: ponteiro para um valor do tipo STRING
  - operações de escrita e leitura de dados em arquivos:
    - executadas com objetos do tipo LINE

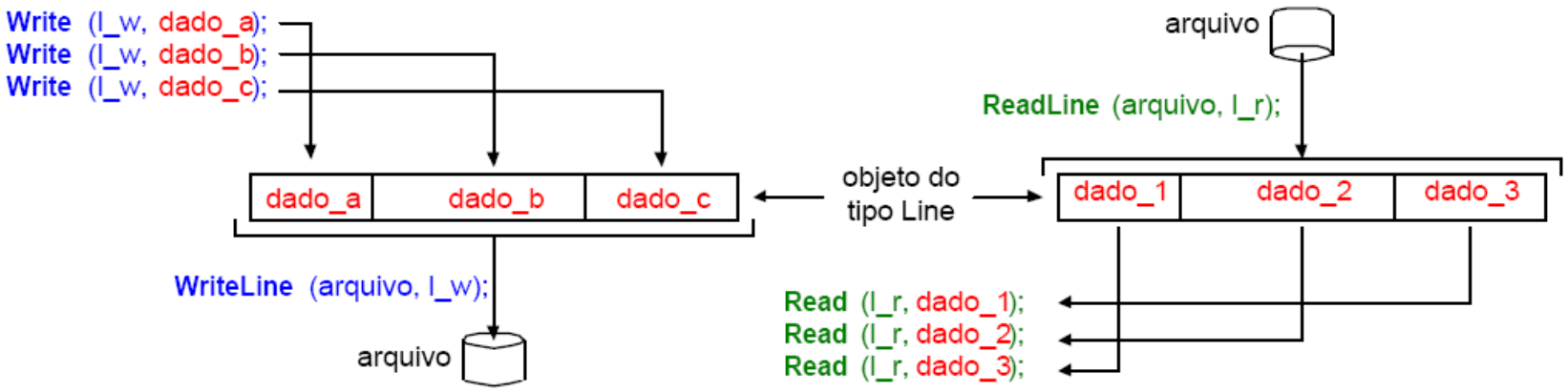
- **Tipo TEXT**: tipo arquivo composto de elementos do tipo STRING

- **Tipo SIDE**:
  - empregado para definir a justificação dos dados no campo de escrita:
    - RIGHT alinha no lado direito do campo
    - LEFT para o lado esquerdo.

- **Sub-tipo WIDTH**:
  - para especificar o tamanho do campo empregado na escrita de um dado

Pacote TEXTIO

- **Operações com arquivos no pacote TEXTIO**:



- **Escrever uma informação**:
  - transferir um ou mais dados para um objeto do tipo LINE
  - escrever o conteúdo do objeto no arquivo

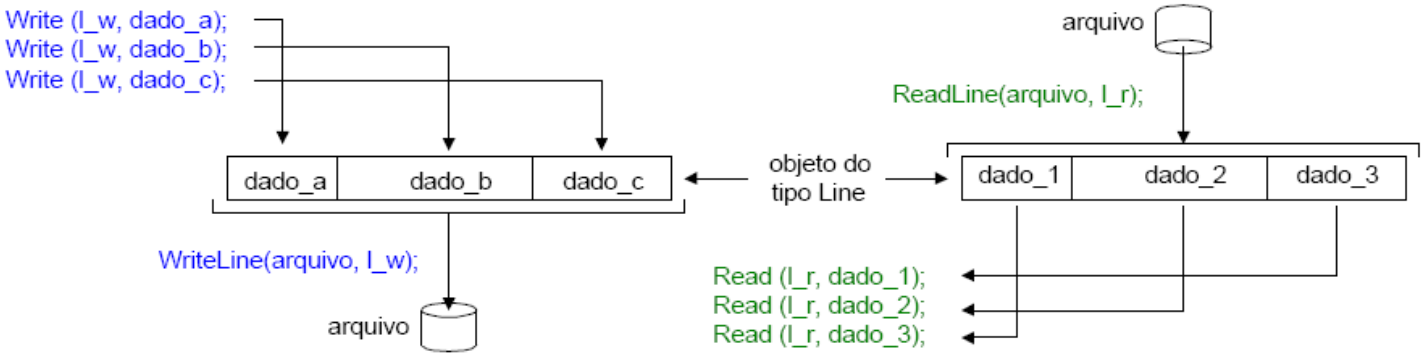
- **Ler uma informação**:
  - transferir informação para um objeto do tipo LINE
  - extrair os dados do objeto

Pacote TEXTIO

- **Procedimentos definidos no pacote TEXTIO**:

```
WRITE     (linha_wr, dado);          -- WRITE(tipo_LINE, tipo_definido_pacote_padrao)
WRITELINE (arquivo_escrita, linha_wr);
READLINE  (arquivo_leitura, linha_rd);
READ      (linha_rd, dado);          -- WRITE(tipo_LINE, tipo_definido_pacote_padrao)
```

- **WRITE**: insere valor num objeto do tipo LINE
- **WRITELINE**: escreve a informação do objeto tipo LINE no arquivo
- **READLINE**: lê arquivo, informação retorna no objeto tipo LINE
- **READ**: extrai um dos valores do objeto tipo LINE
- **WRITE e READ sobrecarregam procedimentos de mesmo nome**



Pacote TEXTIO

- **Relembrando a definição do tipo LINE**:

```
TYPE    LINE IS ACCESS STRING;     -- LINE e' um ponteiro para um tipo STRING
```

- **Um objeto tipo LINE pode conter vários valores**

- **É um ponteiro para um valor STRING**

- **Cada chamada do procedimento WRITE:**
  - um novo dado é inserido no objeto tipo LINE
  - ponteiro movido para uma nova posição

- **Procedimento WRITELINE:**
  - retorna o ponteiro à posição inicial
  - valores contidos no objeto tipo LINE são perdidos

- **Cada chamada do procedimento READ**
  - dado removido do objeto tipo LINE
  - o ponteiro é movido

Pacote TEXTIO (detalhe dos parâmetros na próxima imagem)

- **Procedimentos para escrita - pacote TEXTIO - tipos do pacote STD**

```
WRITELINE (file F: TEXT; L: INOUT LINE);
WRITE (L: INOUT LINE; V: IN BIT;        J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN BIT_VECTOR; J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN BOOLEAN;    J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN CHARACTER;  J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN INTEGER;    J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN STRING;     J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN REAL;       J: IN SIDE:= RIGHT;  F: IN WIDTH := 0;
                                                                D: IN NATURAL:= 0);
WRITE (L: INOUT LINE; V: IN TIME;       J: IN SIDE:= RIGHT;  F: IN WIDTH := 0;
                                                                U: IN TIME:= ns);
```

- **Note**: os procedimentos WRITE são definidos com mais de dois parâmetros
  - eles têm um valor inicial estabelecido: podem ser omitidos

Pacote TEXTIO

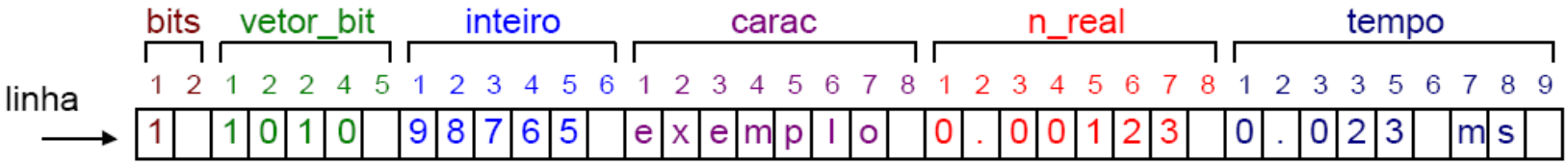
- **Parâmetro J**: dado é alinhado a esquerda ou a direta do campo
  - valores válidos: RIGHT LEFT
- **Parâmetro F**: tamanho do campo para o dado
  - 0: campo do tamanho exato para escrita sem nenhuma margem
- **Parâmetro D**: determina o número de dígitos a direta do ponto decimal
  - 0: valor expresso com mantissa normalizada e expoente (exemplo: 3.1416E00)
- **Parâmetro U**: unidade empregada na representação do valor tipo TIME

```
WRITE (L: INOUT LINE; V: IN BIT;         J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN BIT_VECTOR; J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN BOOLEAN;     J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN CHARACTER;  J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN INTEGER;     J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN STRING;      J: IN SIDE:= RIGHT;  F: IN WIDTH := 0);
WRITE (L: INOUT LINE; V: IN REAL;        J: IN SIDE:= RIGHT;  F: IN WIDTH := 0;
                                                                 D: IN NATURAL:= 0);
WRITE (L: INOUT LINE; V: IN TIME;        J: IN SIDE:= RIGHT; F: IN WIDTH := 0;
                                                                 U: IN TIME:= ns);
```

Pacote TEXTIO

- **Exemplos**: procedimento empregando mais de 2 parâmetros

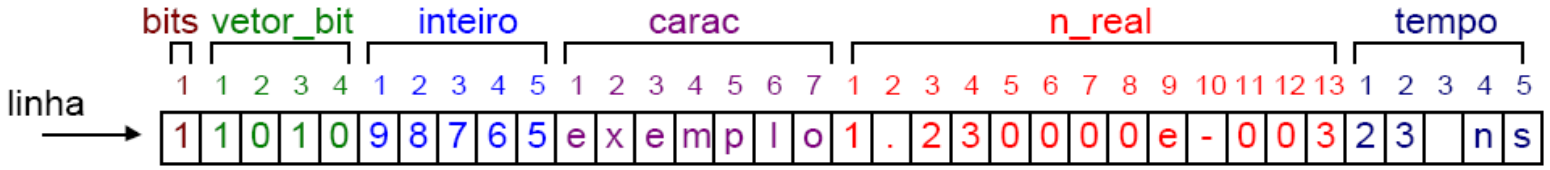
```
Write (linha, bits,        Left, 2);        -- bits := 1              tipo Bit
Write (linha, vetor_bit, Left, 5);          -- vetor_bit :=1010       tipo Bit_Vector
Write (linha, inteiro,     Left, 6);        -- inteiro := 98765       tipo Integer
Write (linha, carac,     Left, 8);          -- carac := exemplo       tipo String
Write (linha, n_real,    Left, 8, 5);       -- n_real := 12.3E-4      tipo Real
Write (linha, tempo,     Left, 9, ms);      -- tempo := 23 us         tipo Time
```



Pacote TEXTIO

- **Exemplos**: procedimento empregando 2 parâmetros

```
Write (linha, bits);          -- bits := 1
Write (linha, vetor_bit);     -- vetor_bit :=1010
Write (linha, inteiro);       -- inteiro := 98765
Write (linha, carac);         -- carac := exemplo
Write (linha, n_real);        -- n_real := 12.3E-4
Write (linha, tempo);         -- tempo := 23 us
```



Pacote TEXTIO - Versão VHDL-1993

- **Possível empregar as opções disponíveis versão VHDL-1993**
  - abertura do arquivo na declaração de um objeto tipo arquivo texto (linhas em verde)
  - apenas declarar um objeto tipo arquivo texto (linhas em azul)

```
FILE leitura     : TEXT    OPEN Read_Mode IS  "nome_externo";
FILE escrita     : TEXT    OPEN Write_Mode IS "nome_externo";
FILE nome_objeto : TEXT;
```

Pacote TEXTIO - Versão VHDL-1993 (exemplo escrita e leitura de dados)

- **Linhas 15 a 21 operação de escrita dos dados**
- **Arquivo aberto modo escrita** (linha 15)
- **Arquivo fechado após operação de escrita** (linha 21)

```
 1 -- versao VHDL 1993
 2 USE STD.TEXTIO.ALL;            -- inclui o pacote TEXTIO
 3 ENTITY arqv_a3 IS
 4   PORT(dado  : OUT INTEGER);  -- caracteres recuperados no arquivo
 5 END arqv_a3;
 6
 7 ARCHITECTURE teste OF arqv_a3 IS
 8 BEGIN
 9   escreve: PROCESS
10     FILE arq_wr_rd   : TEXT;
11     VARIABLE linha   : LINE;
12     VARIABLE espaco  : CHARACTER := ' ';
13     VARIABLE valor   : INTEGER := 48; -- dado recuperado no arquivo
14   BEGIN
15     FILE_OPEN(arq_wr_rd, "dado_a3.dat",Write_Mode);
16     FOR dado_wr IN 48 TO 57 LOOP
17       WRITE (linha, dado_wr);       -- escreve o texto 48 49 ...57
18       WRITE (linha, espaco);        -- separandos por um espaco
19     END LOOP;
20     WRITELINE (arq_wr_rd, linha);  -- armazena uma linha no arquivo
21     FILE_CLOSE(arq_wr_rd);         -- fecha o arquivo
```

Pacote TEXTIO - Versão VHDL-1993 (exemplo escrita e leitura de dados - continuação)

- **Linhas 23 a 31 operação de leitura dos dados**
- **Arquivo aberto novamente - modo leitura** (linha 23)
- **Arquivo fechado após operação de leitura** (linha 31)

```
 7 ARCHITECTURE teste OF arqv_a3 IS
 8 BEGIN
 9   escreve: PROCESS
10     FILE arq_wr_rd   : TEXT;
11     VARIABLE linha   : LINE;
12     VARIABLE espaco  : CHARACTER := ' ';
13     VARIABLE valor   : INTEGER := 48; -- dado recuperado no arquivo
14   BEGIN
 .
 .
23     FILE_OPEN(arq_wr_rd, "dado_a3.dat",Read_Mode);
24     READLINE (arq_wr_rd, linha);  -- leitura de uma linha
25     WHILE valor /= 57 LOOP        -- verifica se e' o ultimo valor
26       READ (linha, valor);        -- leitura dos dados de uma linha
27       READ (linha, espaco);       -- ignorado
28       dado <= valor;
29       WAIT FOR 10 ns;             -- temporizacao para mostra "dado" no simulador
30     END LOOP;
31     FILE_CLOSE(arq_wr_rd);        -- fecha o arquivo
32     WAIT;
33   END PROCESS escreve;
34 END teste;
```

Operações com arquivos

Tópicos

- Definição de tipos arquivo
- Pacote TEXTIO

Tópicos avançados

Tópicos

- Declaração ASSERT
- Declaração REPORT

Teste

Tópicos

- Testes
- Teste de uma descrição

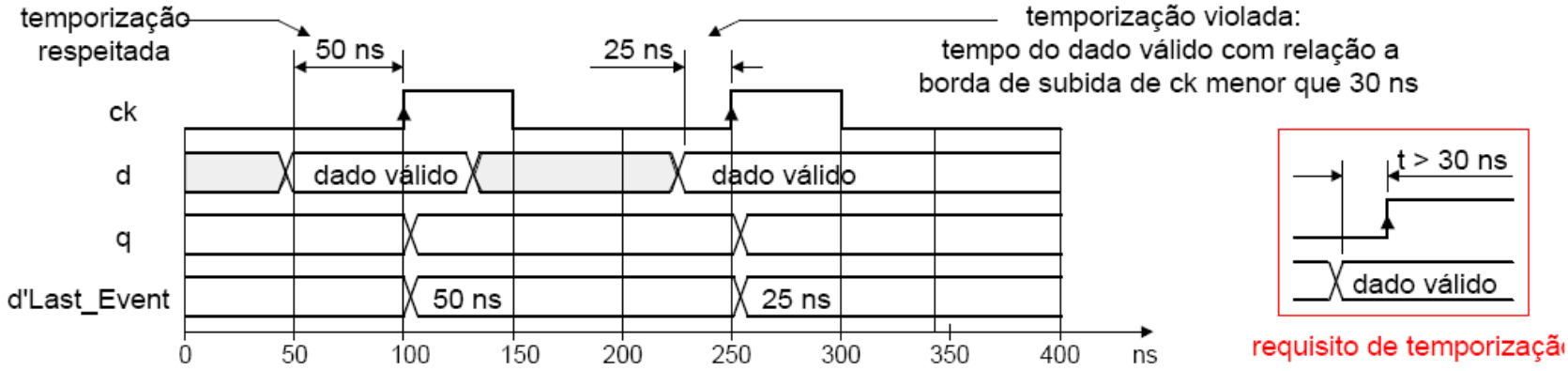
Declaração ASSERT

- **Declaração seqüencial usada para notificar condições ilegais**
- **Cláusulas opcionais**: REPORT SEVERITY
- **Condição de teste = requisito correto**
  - condição verdadeira → nenhuma ação é tomada.
- **REPORT**: especifica uma mensagem a ser apresentada pelo simulador.
  - mensagem: uma expressão do tipo STRING
  - na ausência da cláusula mensagem: Assertion violation
- **SEVERITY** : identifica o nível de gravidade
  - valores: FAILURE ERROR WARNING NOTE
  - na ausência da cláusula é assumido o nível ERROR
- **Simuladores**: opções quanto a ação a tomada (ex. abortar simulação)

```
ASSSERT condicao          -- condicao = TRUE (nada ocorre); FALSE (mensagem exibida)
 REPORT expressao_tipo_texto      -- opcional, expressao que retorna um tipo STRING
  SEVERITY expressao_de_severidade;  -- opcional: FAILURE, ERROR, WARNING, NOTE
```

Declaração ASSERT (exemplo)

- **Verificar erro de temporização em um *flip flop***
- **Requisito de temporização**:



Declaração ASSERT (exemplo - código)

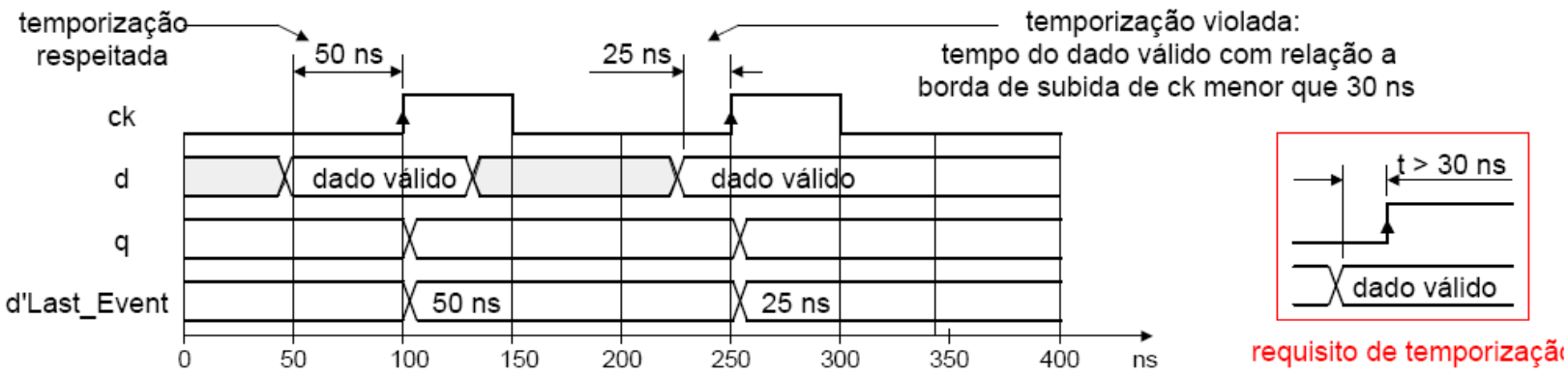
- **WAIT** (linha 17): processo verifica executado a cada borda de subida de ck
- **na execução**:
  - d'LAST_EVENT = tempo entre a borda de ck e o último evento em d
  - d'last_event > set_up = condição correta
  - d'last_event < set_up = condição incorreta (mensagem apresentada)

```
 7 ARCHITECTURE teste OF ass_3a IS
 8 BEGIN
 9  operacao: PROCESS(ck, d)
10  BEGIN
11    IF ck'EVENT AND ck='1' THEN q <= d;
12    END IF;
13  END PROCESS;
14
15  verifica: PROCESS          -- processo e' executado a cada borda subida de ck
16  BEGIN                      -- d'LAST_EVENT = intervalo de tempo ocorrido entre
17   WAIT UNTIL (ck'EVENT AND ck='1');   --    instante atual e ultimo evento em "d"
18   ASSERT d'LAST_EVENT > set_up        -- d'last_event > set_up condicao correta,
19     REPORT "Tempo set_up nao respeitado " --      se falso mensagem apressentada
20     SEVERITY ERROR;
21  END PROCESS;
22 END teste;
```

Declaração ASSERT (exemplo - código)

- **Mensagem apresentada**:

```
# ** Error:  Tempo set_up nao respeitado
#    Time: 250 ns  Iteration: 0  Instance: /ass_3a
```



Declaração REPORT

- **Declaração seqüencial usada para relatar mensagens**
- **Disponível na versão VHDL-1993**
- **Pode ser empregada em conjunto com a cláusula opcional SEVERITY**
- **Objetivo desta opção na versão VHDL-1993**:

  - evitar o uso de ASSERT FALSE p/ apresentar sempre uma mensagem

- **Formato**:

```
REPORT expressao_tipo_texto            --  expressao que retorna um tipo STRING
   SEVERITY expressao_de_severidade;    -- opcional: FAILURE, ERROR, WARNING, NOTE
```

Declaração REPORT (exemplo - código)

- **Código**: apresenta o valor da interface de entrada dado
- **Necessário converter:** dado tipo Std_logic para tipo String (função linhas 9 a 17)
- **Mensagem**:
  - expressão concatenando de texto com valor convertido
  - CR e LF pulam uma linha e retornam o cursor

```
 9 ARCHITECTURE teste OF rep_0 IS
10   FUNCTION para_string (a : STD_LOGIC) RETURN string IS
11   BEGIN
12    CASE a IS
13      WHEN '0' => RETURN " 0"; WHEN 'L' => RETURN " L";
14      WHEN '1' => RETURN " 1"; WHEN 'H' => RETURN " H";
15      WHEN 'X' => RETURN " X"; WHEN 'U' => RETURN " U"; WHEN OTHERS => RETURN " -";
16    END CASE;
17   END para_string;
18 BEGIN
19   operacao: PROCESS(dado)
20   BEGIN
21     REPORT "Valor dado= " & para_string(dado) & CR & LF
22       SEVERITY NOTE;
23   END PROCESS;
24 END teste;
```

Operações com arquivos

Tópicos

- Definição de tipos arquivo
- Pacote TEXTIO

Tópicos avançados

Tópicos

- Declaração ASSERT
- Declaração REPORT

Teste

Tópicos

- Testes
- Teste de uma descrição

Teste

- **Teste**: conjunto de ações destinadas a verificar a operação de um sistema
- **Objetivos**:
  - verificar a consistência entre:

    especificação do sistema e sistema projetado,

  - detectar problemas na implementação física do sistema
- **Execução de um teste**:
  - exercitar o sistema sob teste com uma série de estímulos e observar o resultado

Teste

- **Inconsistências com a especificação**:
  - são erros de projeto
  - devem ser detectadas na fase de simulação

- **Problemas na implementação**
  - são falhas na fabricação,
  - detectadas após a manufatura do sistema através:
    - equipamentos de teste específicos
    - pelo próprio sistema (quando dispõe de recursos para auto teste)

Teste

Estímulos para verificação de erros de projeto ≠ estímulos p/ detecção de falhas de fabricação

- **Qual o motivo da diferença**?

- **Fase de arquitetura do sistema**:
  - muitos recursos para aplicar estímulos e observar resultados.

- **Sistema implementado**:
  - difícil atingir unidades no interior do sistema
    - aplicar um estímulo e observar o resultado não é uma tarefa trivial
  - outras preocupações:
    - detectar problemas de fabricação
      (curto circuitos, abertos, componentes defeituosos)
    - estes testes procuram exercitar todos os nós do circuito

Teste

- **Estímulos de teste para verificar a consistência de uma descrição**:
  - podem ser criados com o auxílio da ferramenta de simulação empregada
    - aplicado em descrições simples
  - a criação de uma entidade em VHDL que execute as operações de teste
    - aplicado em descrições complexas

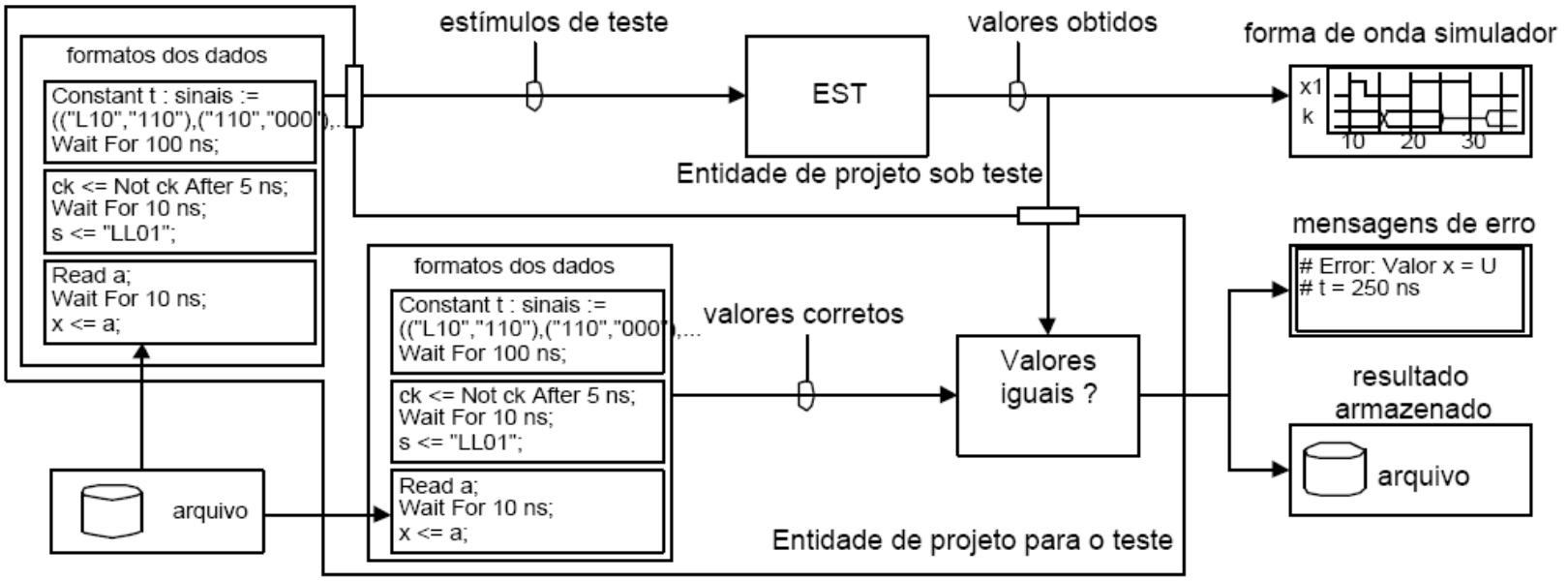
Teste de uma descrição

- **Pode empregar diferentes métodos**

- **Abordagem mais adequada deve considerar aspectos como**:
  - facilidade na elaboração
  - compatibilidade entre ferramentas de diferentes fornecedores,
  - tempo de execução
  - necessidade de armazenar resultados.

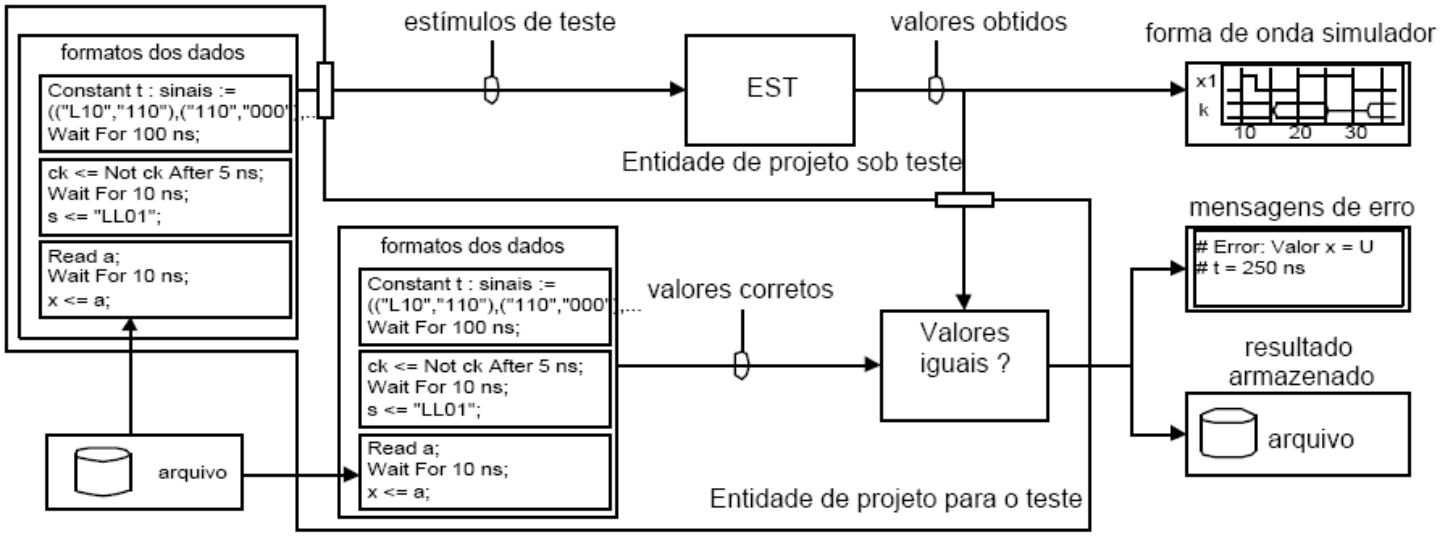
Teste de uma descrição

- **Caso geral de uma estrutura de teste, estímulos gerados por**:
  - uma constante do tipo vetor
  - um conjunto de comandos próprios da linguagem
  - um arquivo contendo os dados
  - ou uma composição destas técnicas



Teste de uma descrição

- **Observação dos resultados**:
  - através de uma janela própria da ferramenta de simulação
    - análise difícil em simulações longas
    - documentação problemática (caso incompatibilidade de ferramentas)
  - comparação entre valores obtidos e valores corretos via especificação do projeto
    - análise mais eficiente
    - documentação: independente da ferramenta (dados em arquivos texto)



Teste de uma descrição

- **Formato dos arquivos empregados**:
  - binário ou texto

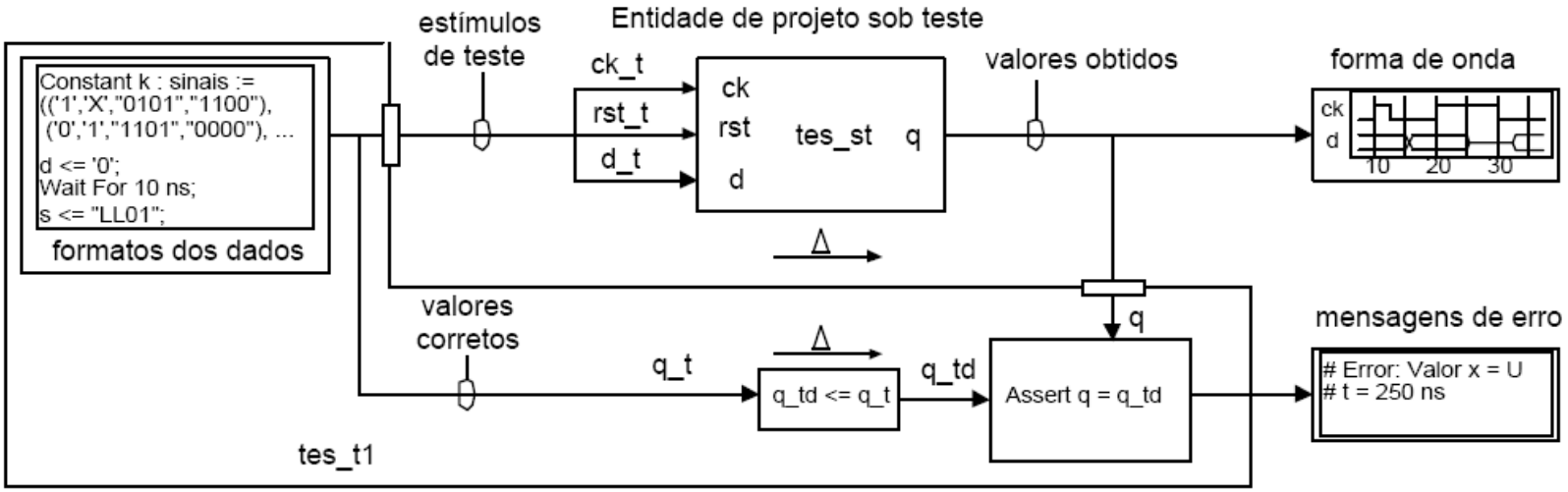
- **Binário**:
  - maior rapidez na simulação

- **Texto**:
  - compatibilidade com diferentes plataformas
  - facilidade na leitura direta da informação.

Teste empregando constantes para geração dos estímulos

- **Constantes**: maneira simples realizar um teste
  - Valores são armazenados em constantes do tipo vetor
  - Um processo controlado declarações WAIT

    aplica estímulos na temporização desejada



Teste empregando constantes para geração dos estímulos - (exemplo)

- **Entidade sob teste**: registrador tipo D com inicialização assíncrona: tes_st
- **Doze estímulos de teste são propostos na constante vetores**
- **Constante vetores**: vetor composto de elementos do tipo RECORD sinais
  - cada elemento contém: os estímulos e o resultado esperado

```
 9 ARCHITECTURE teste OF tes_t1 IS
10   TYPE sinais IS
11     RECORD ck, rst  : STD_LOGIC;
12            d, q     : STD_LOGIC_VECTOR(3 DOWNTO 0);
13     END RECORD;
14  TYPE conjunto_sinais IS ARRAY (natural RANGE <>) OF sinais;
15  CONSTANT vetores : conjunto_sinais :=
16 -- ck   rst   d     q      ck  rst   d      q      ck  rst   d     q
17  (('U','1',"UUUU","UUUU"),('U','0',"UUUU","0000"),('1','1',"UUUU","0000"),--reset
18   ('0','1',"ZZZZ","0000"),('0','1',"1010","0000"),('1','1',"1010","1010"),
19   ('0','1',"ZZZZ","1010"),('0','1',"0101","1010"),('1','1',"0101","0101"),
20   ('0','1',"ZZZZ","0101"),('0','1',"1100","0101"),('1','1',"1100","0011"));
21                                                          --    \
22  COMPONENT tes_st                                        --     \__ erro
23    PORT (ck, rst : IN STD_LOGIC; d : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
24          q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));
25  END COMPONENT;
```

Teste empregando constantes para geração dos estímulos - (exemplo)

- **Aplicação dos estímulos**:
  - definida pelo laço entre as linhas 32 e 37
  - percorre todos elementos do vetor
  - cadência da aplicação é definida pela declaração WAIT no interior do laço.

```
30   operacao: PROCESS
31     BEGIN
32        FOR i IN 0 TO  vetores'LENGTH-1 LOOP
33         ck_t   <= vetores(i).ck;
34         rst_t  <= vetores(i).rst;
35         d_t    <= vetores(i).d;
36         q_t    <= vetores(i).q;
37         WAIT FOR 10 ns;
38       END LOOP;
39     WAIT;
40   END PROCESS;
```

Teste empregando constantes para geração dos estímulos - (exemplo)

- **Verificação de erros: declaração ASSERT**
- **Se q ≠ q_td gerada uma mensagem de erro**
- **11° estímulo contém um erro** (p/ verificar operação de teste da descrição)

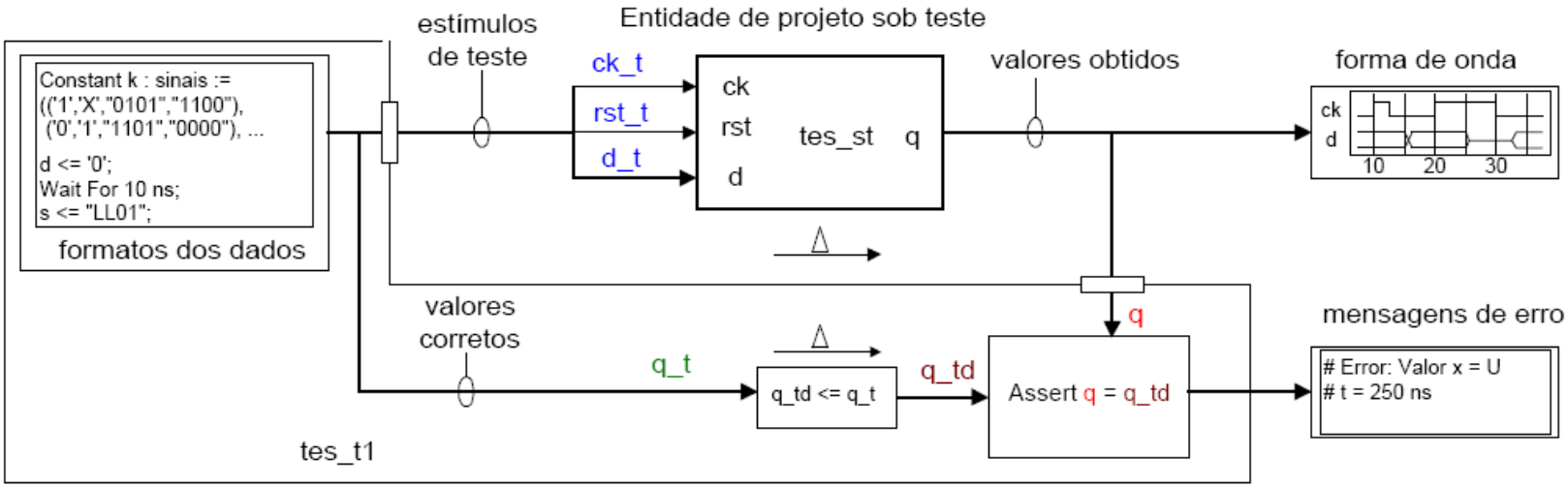
```
42  q_td <= q_t;      -- sincroniza valor "q_t" com "q", (q_t e' atrasado de 1 delta)
43  ASSERT q = q_td   -- (q=q_td) condicao correta, caso falso mensagem apresentada
44      REPORT "Valor da saida q difere do proposto" SEVERITY ERROR;
45
46  x1: tes_st PORT MAP(ck_t, rst_t, d_t, q);
```

- **Mensagem gerada pela declaração ASSERT**:

```
# ** Error: Valor da saida q difere do proposto
#    Time: 110 ns  Iteration: 2  Instance: /tes_t1
```

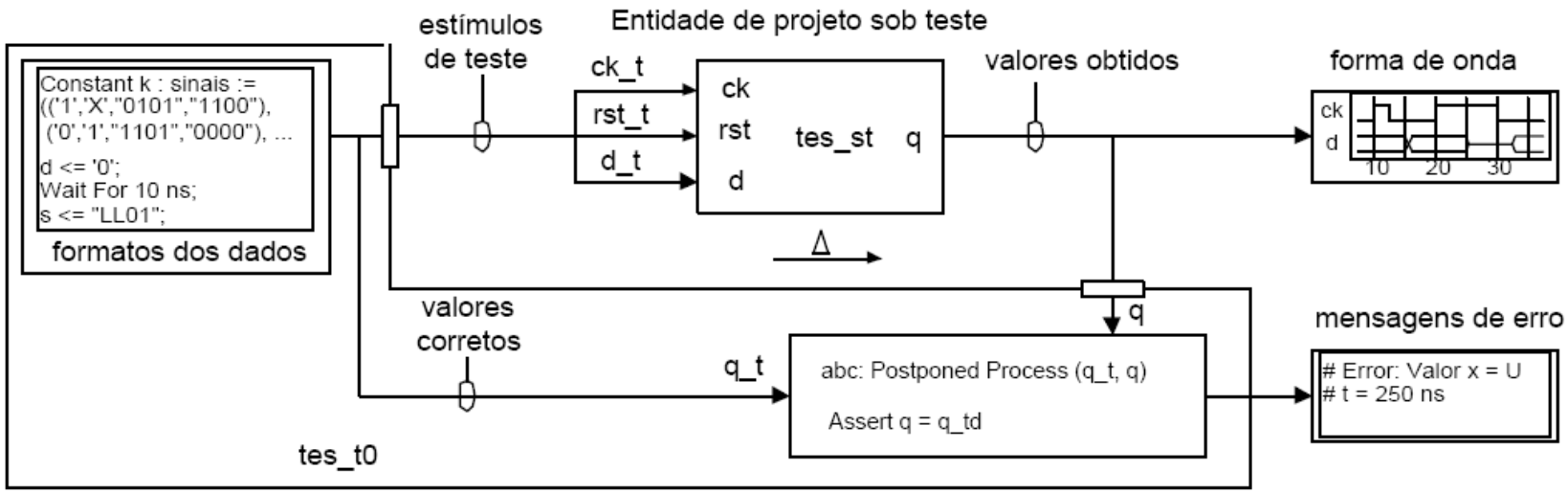
Teste empregando constantes para geração dos estímulos - (exemplo)

- **Sincronismo na comparação entre valores corretos e valores obtidos**
- **Estímulos p/ sinais ck_t , rs_t , d_t aplicados em t1**
- **Resultado q disponível após uma iteração** (t1+Δ)
- **Comparação entre q e q_t mensagem de erro** (q ≠ q_t em t1)
- **Solução:**
  - valor correto q_t foi transferido p/ o sinal q_td para sincronizar os valores



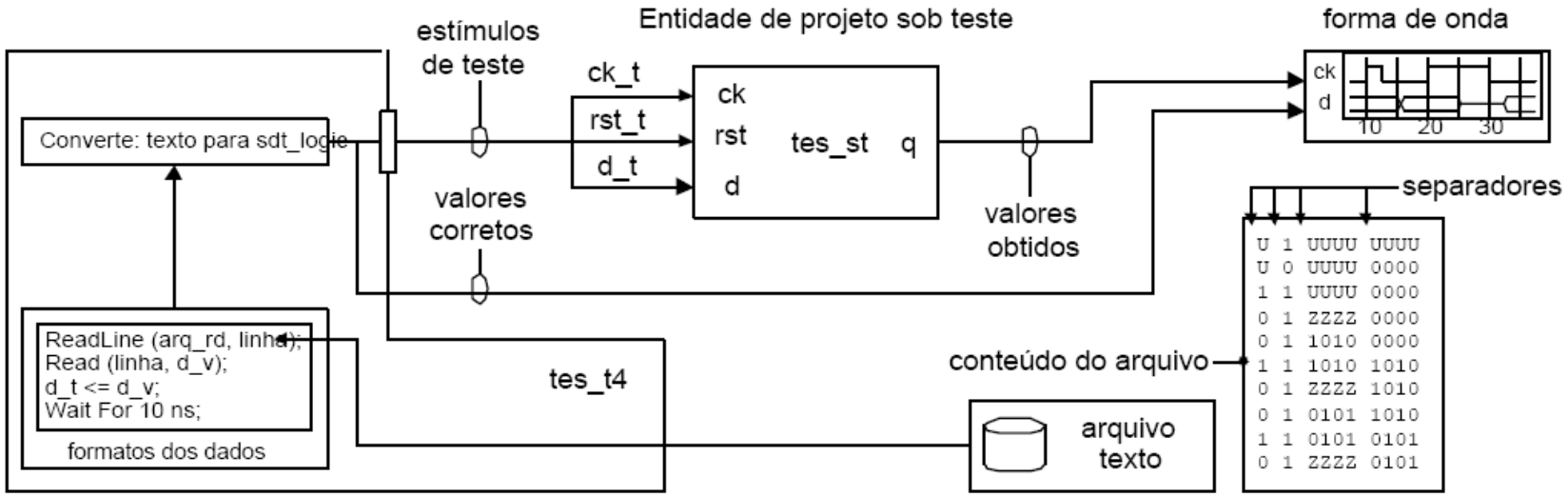
Teste empregando constantes para geração dos estímulos - (exemplo)

- **Solução melhor**:
  - pospor a execução do teste de comparação



Teste empregando dados em arquivo para geração dos estímulos

- **Dados**:
  - armazenados segundo uma formatação estabelecida
- **Aplicação dos estímulos**:
  - dados são recuperadas periodicamente até o encontro do final de arquivo



Teste empregando dados em arquivo - (exemplo)

- **Dados no formato texto**: empregados os procedimentos READLINE e READ
- **Interfaces de entrada e saída da entidade sob teste**:
  - declaradas como tipos STD_LOGIC e STD_LOGIC_VECTOR
- **Necessário a inclusão de um subprograma para conversão de tipo** (para_std)
  -(procedimentos READ do pacote TEXTIO retornam tipos definidos no pacote padrão)

```
38 le: PROCESS
39     FILE     arquivo_rd  : TEXT OPEN READ_MODE IS "estim_t3.dat"; -- arquivo formato:
40     VARIABLE linha       : LINE;                                  -- | | | | | | | | | | | | | | | |
41     VARIABLE ck_v, rst_v : STRING(2 DOWNTO 1);                    --    U   1   U U U U   U U U U
42     VARIABLE d_v,  q_v   : STRING(5 DOWNTO 1);                    --    U   0   U U U U   0 0 0 0
43   BEGIN                                                           --    1   1   U U U U   0 0 0 0
44     WHILE NOT ENDFILE(arquivo_rd) LOOP
45      READLINE (arquivo_rd, linha);                                -- leitura de uma linha
46      READ (linha, ck_v);  ck_t  <= para_std(ck_v)(0);             -- leitura dos dados
47      READ (linha, rst_v); rst_t <= para_std(rst_v)(0);            --               "
48      READ (linha, d_v);   d_t   <= para_std(d_v)(3 DOWNTO 0);     --               "
49      READ (linha, q_v);   q_t   <= para_std(q_v)(3 DOWNTO 0);     --               "
50      WAIT FOR 10 ns;                                              -- intervalo de tempo
51     END LOOP;
52     FILE_CLOSE(arquivo_rd);
53     WAIT;
54   END PROCESS le;
```