ETSIST-UPM
Dpto. de Ing. Telemática y Electrónica

Diseño Digital 2
Bloque temático 1

BT1_A1_P4

Diseño jerárquico

Técnicas de simulación y verificación de circuitos complejos

Técnicas de simulación y verificación de circuitos complejos

- En la realización de *Test-Benchs* VHDL suelen utilizarse construcciones del lenguaje que rara vez se emplean para construir modelos sintetizables de circuitos
  - Ficheros
  - Funciones y Procedimientos definidos por el usuario
  - Sentencias ASSERT

- Sirven para:
  - La definición de estímulos:
    - Ficheros para la lectura de estímulos generados automáticamente.
    - Procedimientos y funciones para el modelado de alto nivel del entorno de funcionamiento real del circuito y para la modularización de la asignación de estímulos en procesos
  - La verificación automática de los resultados de las simulaciones
    - Escritura de resultados sobre ficheros para, posteriormente, procesarlos.
    - Autoverificación en test-bench mediante sentencias ASSERT

Objetos VHDL

- Para asignar valores a variables se usa el símbolo ‘:=’
- Para asignar valores a señales se usa el símbolo ‘<=’
- Las sentencias de asignación de valor a una variable se ejecutan secuencialmente (como en otros lenguajes de programación) y no de forma paralela.
- Es decir, si en el siguiente extracto de código s_a = “0000” y s_b = “1111” antes de la ejecución del proceso. Una vez finalizada la ejecución s_a tomará el valor “1010”, mientras que s_b tomará el valor previo de s_a, es decir, “0000”.

```
signal s_a,s_b : std_logic_vector(3 downto 0);

process (clk)
   variable var : std_logic_vector(3 downto 0);

begin
   if clk’event and clk = ‘1’ then
         var := “1010”;
         s_a <= var;
         s_b <= s_a;
   end if;
end process;
```

Objetos VHDL

- A un objeto **sólo** pueden asignársele valores del tipo sobre el que está definido

```
variable flag1, flag2 : bit;
variable flag3        : std_ulogic;

flag1 := '0';  -- Asignación válida
flag1 := 'U';  -- Inválida: el caracter 'U' no pertenece
               -- al tipo BIT
flag1 := 0;    -- Invalida: se le asigna un valor entero
flag1 := flag2; -- Válida: variables del mismo tipo
flag1 := flag3; -- Inválida: variables de distinto tipo
```

Objetos VHDL

- Para poder asignar a un objeto valores de un tipo distinto, pueden usarse funciones de conversión

```
variable flag : bit;
signal nodo   : std_ulogic;

nodo <= To_StdULogic(flag);
flag := To_bit(nodo, '0');
```

Subprogramas: Funciones y Procedimientos

- Las **funciones** VHDL:
  - ✓ **no son** sentencias
  - ✓ devuelven un valor
  - ✓ dentro de ellas, no puede asignarse valor a señales o incluirse sentencias **WAIT**.
- Los **procedimientos** VHDL:
  - ✓ **son** sentencias
  - ✓ pueden ejecutarse secuencial o concurrentemente
  - ✓ pueden incluir sentencias **WAIT** y asignar valores a señales
- **Declaración** de funciones y procedimientos
  - ✓ Puede realizarse en cualquier zona declarativa: Declaración de Paquete, Zona de declaración de un Cuerpo de Arquitectura o de un proceso
- **Definición** de funciones y procedimientos
  - ✓ La definición del procedimiento puede realizarse por separado o en la propia declaración
  - ✓ Cuando la función se declara en una Declaración de Paquete, la definición se realiza en el Cuerpo del Paquete

Sintaxis de la Declaración de funciones y procedimientos

```
FUNCTION    NOMBRE  (LISTA_DE_ARGUMENTOS) RETURN TIPO_DE_DATOS;
PROCEDURE   NOMBRE  (LISTA_DE_ARGUMENTOS);
```

La lista de argumentos es el conjunto de parámetros que devuelve o se pasan al subprograma:

- ✓ En las funciones todos los argumentos son de entrada
- ✓ En los procedimientos, los parámetros pueden ser: de entrada (IN), de salida (OUT) y bidireccionales (INOUT)
- ✓ Los argumentos pueden ser cualquier tipo de objeto
- ✓ En la declaración de un argumento puede omitirse la dirección y el tipo de objeto; en tal caso:
  - Se considerará por defecto que es un argumento de entrada
  - Si se omite el tipo de objeto se considerará que es una CONSTANTE, si es un argumento de entrada, y una VARIABLE, si es de salida o bidireccional
- ✓ Las funciones devuelven un valor del TIPO DE DATOS indicado en su declaración

Ejemplos de declaración de funciones y procedimientos

```
function retardo_en_tclk (constant retardo, tclk: in time) return natural;
```

```
function dentro_de_rango (signal a: in std_logic_vector;
                          constant sup, inf: integer) return boolean;
```

```
procedure tecleo ( signal      ena_cmd:      out std_logic;
                   signal      cmd_tecla:    out std_logic_vector(3 downto 0);
                   signal      clk:          in  std_logic;
                   constant    tecla:        in  std_logic_vector(3 downto 0));
```

Sintaxis de la Definición de funciones y procedimientos

```
FUNCTION/PROCEDURE NOMBRE(LISTA_DE_ARGUMENTOS) [RETURN…] IS
    ZONA DE DECLARACIÓN
BEGIN
    ALGORITMO DE PROCESAMIENTO SECUENCIAL
END FUNCTION/PROCEDURE;
```

- ✓ En la zona de declaración del subprograma pueden declararse constantes, variables, tipos de datos e, incluso, otros subprogramas
- ✓ Todas las declaraciones son locales
- ✓ El algoritmo puede construirse con sentencias de ejecución secuencial
- ✓ En los procedimientos pueden realizarse asignaciones de valor a señal; en las funciones no se puede
- ✓ Las funciones deben devolver un valor, del tipo de datos indicado en su declaración, utilizando una sentencia **RETURN**

Ejemplo de definición de funciones

```
function hora_to_natural (hora: std_logic_vector(23 downto 0)) return natural is
    variable resultado: natural := 0;

  begin
    resultado := 10*conv_integer(hora(23 downto 20));
    resultado := resultado + conv_integer(hora(19 downto 16));
    resultado := resultado * 3600;
    resultado := resultado + 600*conv_integer(hora(15 downto 12));
    resultado := resultado + 60*conv_integer(hora(11 downto 8));
    resultado := resultado + 10*conv_integer(hora(7 downto 4));
    resultado := resultado + conv_integer(hora(3 downto 0));
    return resultado;

  end function;
```

Ejemplo de definición de procedimientos

```
procedure tecleo(signal   ena_cmd:   out std_logic;
        signal     cmd_tecla: out std_logic_vector(3 downto 0);
        signal      clk:           in    std_logic;
        constant tecla:         in    std_logic_vector(3 downto 0)) is

begin
        wait until clk'event and clk = '1';
        ena_cmd <= '1';
        cmd_tecla <= tecla;

        wait until clk'event and clk = '1';
        ena_cmd <= '0';

        wait until clk'event and clk = '1';

 end procedure;
```

Sintaxis de las sentencias de funciones y procedimientos

NOMBRE (PROCEDURE) (LISTA_DE_ASOCIACIÓN);
OBJETO <= NOMBRE (FUNCIÓN) (LISTA_DE_ASOCIACIÓN);

- ✓ La lista de asociación puede construirse mediante conexión explícita o por posición
- ✓ Si se omite algún parámetro, es obligatorio que en la declaración del subprograma tenga asignado un valor por defecto
- ✓ Los procedimientos pueden ejecutarse secuencial o concurrentemente –si alguno de sus parámetros de entrada es una señal
- ✓ Una función puede estar en la parte derecha –asignación- de una sentencia concurrente si alguno de sus parámetros de entrada es una señal

EJEMPLO: **tecleo (ena_cmd, cmd_tecla, clk, X"F");**

Paquetes VHDL: conceptos (I)

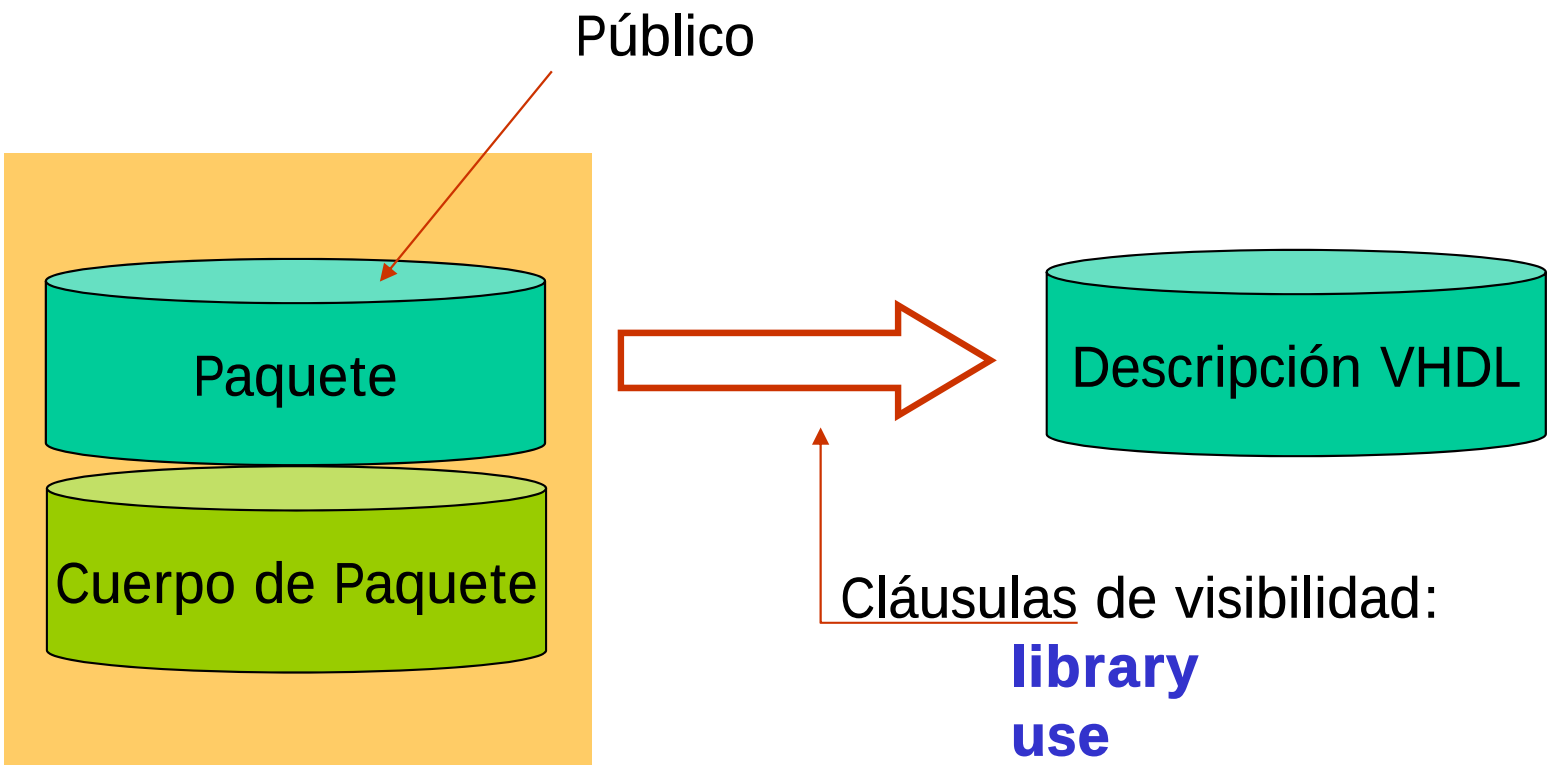
- Cumplen funciones equivalentes a las de las librerías en los lenguajes de programación de alto nivel

- Contienen, fundamentalmente, tipos de datos y operadores definidos por el usuario, subprogramas y componentes

- Se construyen con dos unidades del lenguaje:
  - La **Declaración de Paquete**, que es la vista pública del Paquete
  - El **Cuerpo de paquete**, que contiene la definición de los operadores y subprogramas que aparecen en la Declaración del Paquete

Paquetes VHDL: conceptos (II)

- Los elementos declarados en un Paquete pueden ser utilizados en otras unidades VHDL haciendo uso de cláusulas de visibilidad



Paquetes VHDL: sintaxis

```
package {nombre del paquete} is
   {zona de declaración}
end {nombre del paquete};
```

```
package body {nombre del paquete} is
   {zona de declaración}
end {nombre del paquete};
```

Paquetes VHDL: ejemplo

```
package auxiliar is
 -- La función calc_log calcula el mínimo valor, n, que
 -- cumple que 2**n es mayor o igual que x
 function calc_log(x: in natural) return natural;

end package;

package body auxiliar is
function calc_log(x: in natural) return natural is
 begin

 for n in 1 to 16 loop
   if 2**n >= x then
     return n;
   end if;
 end loop;
 return 0; --error
end calc_log;

end package body;
```

DECLARACIÓN DE PAQUETE

PROTOTIPO (DECLARACIÓN) DE UNA FUNCIÓN (CALC_LOG) QUE REALIZA UN CÁLCULO MATEMÁTICO

SI EL PAQUETE ESTÁ ALMACENADO EN LA MISMA LIBRERÍA QUE LAS UNIDADES DE DISEÑO QUE LO USAN (EN LA LIBRERÍA WORK), PUEDE OBTENERSE VISIBILIDAD SOBRE ÉL SIN DECLARAR LA LIBRERÍA:

USE WORK.AUXILIAR.ALL;

CUERPO DE PAQUETE

DEFINICIÓN DE LA FUNCIÓN CALC_LOG

SENTENCIA RETURN

INDICA EL VALOR QUE DEVUELVE LA FUNCIÓN

Sentencia ASSERT

- Las sentencias **ASSERT** sirven para comprobar el cumplimiento de condiciones durante la ejecución de una simulación
- Las condiciones deben formularse como expresiones evaluables a un valor booleano
- Cuando la expresión es FALSE se reporta por consola un mensaje, acompañado por una indicación de la importancia del suceso –un valor del tipo predefinido **SEVERITY_LEVEL**
- Sintaxis:

```
ASSERT Condición Lógica
REPORT  String
SEVERITY Valor SEVERITY_LEVEL
```

- Los valores del tipo **SEVERITY_LEVEL** son **NOTE**, **WARNING**, **ERROR** y **FAILURE**
- La sentencia se puede ejecutar secuencial o concurrentemente –si en la expresión de la condición lógica hay señales
- Ejemplo

```
ASSERT minutos_T1(3 downto 0) = minutos(3 downto 0) - 1
REPORT “Error en el incremento de unidades de minuto”
SEVERITY ERROR;
```

Atributos

- Los atributos en VHDL proporcionan información adicional sobre el elemento del lenguaje sobre el cual son aplicados
- Pueden ser de diferentes tipos: valores, tipos, rangos, funciones o señales.
- Pueden aplicarse a distintos elementos del lenguaje: tipos de datos, objetos, etc.
- Sintaxis para hacer referencia a un atributo:

<identificador_elemento>'<identificador_atributo>

Atributos aplicables a las señales

S´EVENT
Devuelve un TRUE si ocurrió un evento en S durante el actual delta, de lo contrario devuelve FALSE.

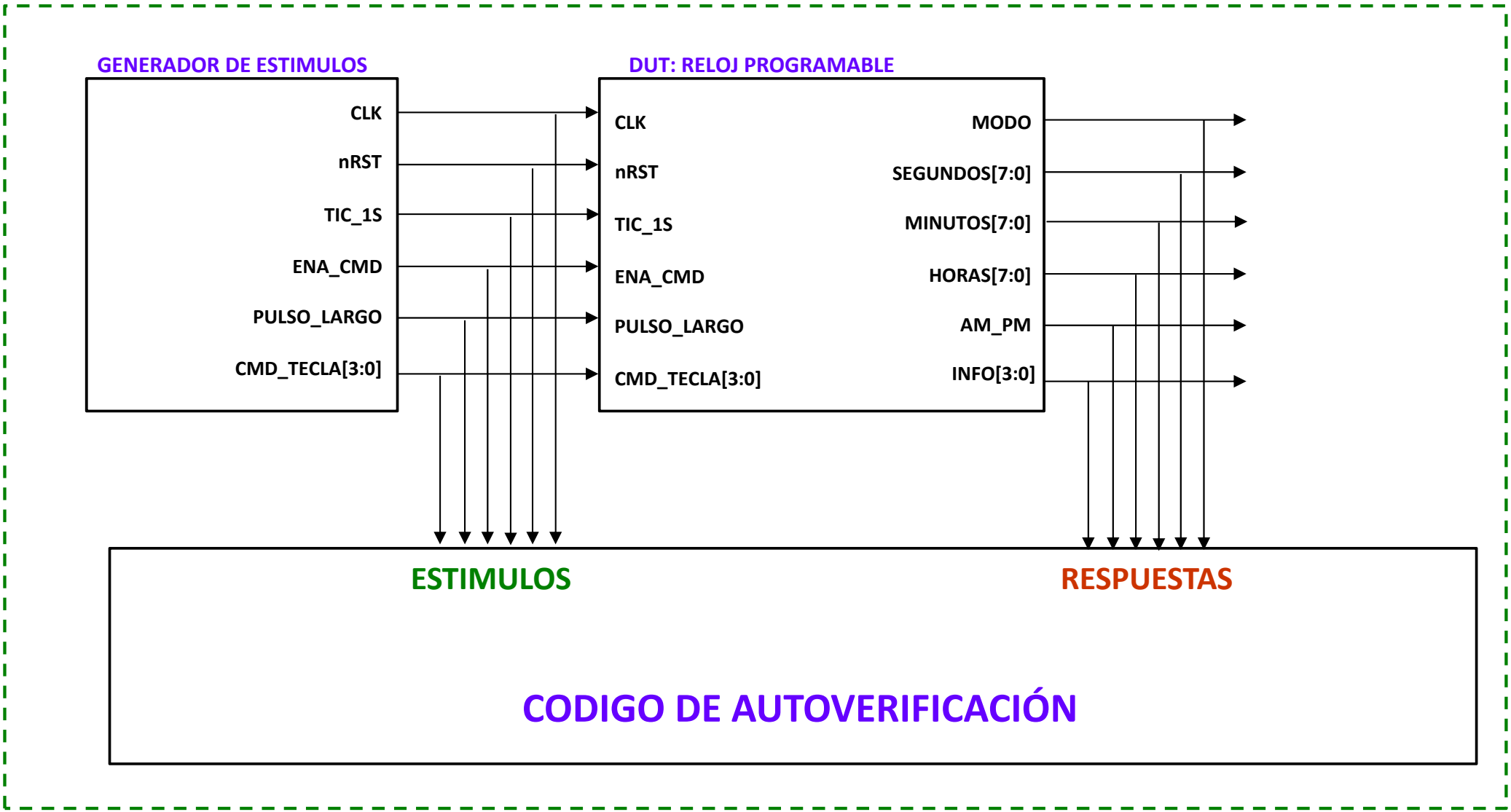
S´ACTIVE
Devuelve un TRUE si ocurrió una transacción en S durante el actual delta, de lo contrario devuelve FALSE.

S´LAST_EVENT
Retorna el tiempo transcurrido desde el último evento en la señal S.

S´LAST_VALUE
Retorna el valor previo de S antes del último evento.

Código de autoverificación: concepto

TEST-BENCH



Código de autoverificación: ejemplo

```
 -- Verificación del comando de pasar a modo de programación de reloj
process(clk, nRst)
   variable cmd_tecla_T1:   std_logic_vector(3 downto 0);
   variable ena_assert:         boolean := false;
   variable pulso_largo_T1: std_logic;
   variable info_T1:            std_logic_vector(1 downto 0);

 begin
   if nRst'event and nRst = '0' then
     ena_assert := false;

   elsif nRst'event and nRst = '1' and nRst'last_value = '0' then
     ena_assert := true;

   elsif clk'event and clk = '1' and ena_assert then
     if pulso_largo_T1 = '1' and cmd_tecla_T1 = X"A" and info_T1 = 0 then
       assert  info = 2
       report "Error detectado por el monitor :-)"
       severity error;
     end if;

     cmd_tecla_T1 := cmd_tecla;
     pulso_largo_T1 := pulso_largo;
     info_T1 := info;

   end if;
 end process;
```