TECNOLOGÍA DE COMPUTADORES

Tema 2

"Descripción de VHDL"

(1/2)

Agustín Álvarez Marquina

Elementos básicos de VHDL

Elementos básicos

- Palabras reservadas.
- Identificadores.
- Tipos de objetos y datos.
- Literales.
- Operadores.
- Atributos.

Palabras reservadas

- **Tienen un significado específico.**
- **Sirven para definir las sentencias del lenguaje.**

abs	else	map	register #	variable
access #	elsif	mod	reject Δ #	wait
after #	end		rem	when
alias #	entity	nand	report #	while
all #	exit	new #	return	with #
and		next	rol Δ	
architecture	file #	nor	ror Δ	xnor Δ
array	for	not		xor
assert #	function	null	select #	
attribute			severity #	
	generate	of	shared Δ #	
begin	generic	on #	signal	
block	group Δ	open #	sla Δ	
body	guarded #	or	sll Δ	
buffer		others	sra Δ	
bus #	if	out	srl Δ	
	impure Δ #	package	subtype	
case	in	port		
component	inertial Δ #	postponed Δ #	then	
configuration	inout	procedure	to	
constant	is	process	transport #	
		pure Δ	type	
disconnect #	label #			
downto	library	range	unaffected Δ #	
	linkage #	record	units #	
	literal Δ		until	
	loop		use	

no soportada por las herramientas de síntesis.
Δ soportada por el estandar del 93 no por el del 87.

Identificadores

- **Identificadores: Sirven para denominar a los elementos.**
  - No pueden utilizarse palabras reservadas.
  - Deben empezar con una letra, no pueden terminar en un subrayado, ni tener dos subrayados intermedios seguidos.
  - No hay diferencias entre mayúsculas y minúsculas.
    - No deben emplearse acentos.
  - No tienen una longitud limitada.
  - Recomendación: deben ser significativos de lo que representan, así se facilita la tarea de documentación.

Tipos de objetos y datos (I)

- **Todos los objetos tienen que declararse antes de su utilización.**

- Tipos de objetos.
  - Constantes.
  - Variables.
  - Señales.
  - Ficheros.

- Tipos de datos.
  - Escalares.
    - Enumerados.
    - Enteros.
    - Físicos.
    - Coma flotante.
  - Compuestos.
    - Vector/matriz.
    - Registro.
  - Acceso.

    Punteros.

Tipos de objetos (I)

Constantes.

- Mantienen su valor durante toda la ejecución.
- Sintaxis:

  CONSTANT nombreConstante**:** tipoDatos[**:=** valorInicial]**;**

- Ejemplos:

```
CONSTANT retardo: time:= 1 ns;
CONSTANT direccion: bit_vector:= “10001110”;
CONSTANT tabla: tipoTabla(0 TO 2):= (1, 7, 8);
```

Tipos de objetos (II)

Variables.

- Almacenan datos intermedios en casos donde las instrucciones son ejecutadas en serie:
  - En procesos (cuando se usa sentencia *PROCESS*) y
  - Subprogramas.
- No tienen significado físico directo.
  - Las asignaciones ocurren inmediatamente, no tienen tiempo asociado.

Tipos de objetos (III)

Variables.

- Sintaxis:

  VARIABLE nombreVariable**:** tipo[**:=** valorInicialOpcional]**;**

- Ejemplos:

```
VARIABLE v1, v2: BIT:= ´1´;
VARIABLE semaforo: tresColores:= rojo;
```

Tipos de objetos (IV)

Señales.

- Una señal mantiene una lista de valores.
  - La lista incluye su valor actual y un conjunto de posibles valores futuros.
- Las asignaciones no son instantáneas, se actualizan en el siguiente paso de simulación (sentencia *WAIT* o si ha terminado una sentencia concurrente).
- Sirven para comunicar procesos e interconectar componentes.
  - Si no se especifica un retardo (mediante sentencia *AFTER*), se aplica automáticamente un retardo de tipo delta.

Tipos de objetos (V)

Señales.

- Deben declararse en la parte declarativa de la arquitectura y son visibles a todos los procesos y bloques de la arquitectura.
  - Puede utilizarse en entornos **secuenciales** y **concurrentes**.
- Sintaxis:

  SIGNAL nombreSeñal**:** tipo**:=** [valorInicialOpcional]**;**
- Ejemplos:

```
SIGNAL a: BIT:= '0';
SIGNAL busDatos: std_logic_vector(7 DOWNTO 0);
```

Tipos de objeto (VI)

Ficheros.

- Permiten comunicar un diseño VHDL con un entorno externo.
  - Útil para introducir estímulos de simulación y salvar resultados de simulación.
- Sintaxis:

  TYPE identificadorTipoFichero **IS FILE OF** tipoDatos;

  FILE identificador**:** tipo **OPEN read_mode IS** “nombre”**;**

  FILE identificador**:** tipo **OPEN write_mode IS** “nombre”**;**

- Ejemplo:

```
TYPE vectores IS FILE OF integer;
FILE datos: vectores OPEN write_mode IS “datos.out”;
```

Tipos de datos (I)

- Cada objeto debe ser de un tipo de datos concreto.
  - El tipo de datos determinará el conjunto de valores que puede tomar y las operaciones que se podrán realizar con ese objeto.
- El conjunto de valores que los objetos pueden tomar no puede cambiar durante la simulación.
- No hay tipos implícitos aunque sí predefinidos.
  - Por ejemplo en los paquetes *STD* y *STD_LOGIC_1164*.
- No hay conversiones implícitas, hay que realizarlas explícitamente.
  - Ejemplo erróneo: *4.0 + 3= ¿?*

Tipos de datos (II)

❍ Tipos de datos escalares.

❑ Enumerados.

- Conjuntos ordenados de identificadores o caracteres definidos por el usuario, útiles para definir los estados de una máquina de estados finitos.

TYPE identificadorTipo **IS** (valor _ 1, valor _ 2, ..., valor _ N);

```
TYPE variosLogicos IS ('X', '0', '1', 'Z');
TYPE semaforo IS(rojo, verde, ambar, apagado);
```

- Ejemplos de declaración de objetos a partir de la declaración de tipos.

```
SIGNAL estadoActual: semaforo:= apagado;
VARIABLE logicaMitad: variosLogicos;
```

Tipos de datos (III)

Tipos de datos escalares.

Enteros.

- Definidos en el paquete estándar

```
TYPE integer IS RANGE -2147483648 TO 2147483647;
```

- Tienen asociados las operaciones matemáticas de suma (+), resta (-), multiplicación (*) y división (/).
- El rango puede ser especificado en orden:
  - Ascendente: (límiteInferior ***TO*** límiteSuperior).
  - Descendente: (límiteSuperior ***DOWNTO*** límiteInferior).

Tipos de datos (IV)

Tipos de datos escalares.

Enteros.

TYPE nombre **IS RANGE** intervalo del rango de enteros;

```
TYPE diasMes IS RANGE 31 DOWNTO 1;
TYPE diasSemama IS RANGE 1 TO 7;
TYPE diasAnno IS RANGE 1 TO 365;
```

- Ejemplos de declaración de objetos a partir de la declaración de tipos.

```
SIGNAL aniversario: diaMes:= 22;
ENTITY fiestasPuentes IS
   PORT(mes: IN diasMes;
        dia: IN diasSemana;
        fiestas: OUT diasAnno);
END fiestasPuentes;
```

Tipos de datos (V)

Tipos de datos escalares.

Físico.

- Es un tipo enumerado que se utiliza para representar magnitudes físicas (tiempo, distancia, capacidad).
  - Tienen un valor y unas unidades.
  - Internamente son considerados como enteros.

```
TYPE nombreMagnitud IS RANGE restricción de rango
UNITS
    Identificador= valores unidades;
END UNITS;
```

Tipos de datos (VI)

Tipos de datos escalares.

Físico.

- Ejemplo:

```
TYPE intensidad IS RANGE 0 TO 10000000;
UNITS
   na;            -- nanoamperio
   ua= 1000 na;   -- microamperio
   ma= 1000 ua;   -- miliamperio
   a = 1000 ma;   -- amperio
END UNITS;
```

Tipos de datos (VII)

Tipos de datos escalares.

Coma flotante.

- Números reales definidos en el paquete estándar

```
TYPE real IS RANGE -1.0E38 TO 1.0E38;
```

- Tienen asociados las operaciones matemáticas de suma (+), resta (-), multiplicación (*) y división (/).
- Declarados de igual forma que los enteros.

TYPE nombre **IS RANGE** intervalo del rango de números reales;

- Ejemplos.

```
TYPE notas IS RANGE 10.0 DOWNTO 0.0;
TYPE probabilidad IS RANGE 0.0 TO 1.0;
```

Tipos de datos (VIII)

Tipos de datos compuestos.

Vectores y matrices.

- Es una colección indexada de elementos que son del mismo tipo.
- Cada elemento de un vector o una matriz puede ser accedido por uno o más índices.
- Son especialmente útiles para modelar memorias (RAM-ROM).

TYPE identificador **IS ARRAY** rango **OF** tipoDatos;

Tipos de datos (IX)

Tipos de datos compuestos.

Vectores y matrices

- **Restringidos:**
  - Los límites del índice se establecen en la declaración del tipo.

```
TYPE vector1 IS
ARRAY(natural RANGE(31 DOWNTO 1)) OF real;

TYPE vector2 IS
ARRAY(1 TO 6,  1 TO 10) OF integer;
```

- **No restringidos.**

```
TYPE vector3 IS
ARRAY(integer RANGE<>) OF bit;
```

Tipos de datos (X)

Tipos de datos compuestos.

Vectores y matrices

- A partir de la declaración del tipo, se pueden declarar objetos de ese tipo:

```
VARIABLE diasMes: vector1;
```

- Se pueden dar valores individualmente a cada elemento e incluso todos a la vez.

```
diaMes(25):= 25;
diaMes:= 31, 30, 29, 28, 27, 26, 25, 24,
23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13,
12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1;
```

Tipos de datos (XI)

Tipos de datos compuestos.

Registros

- Pueden agrupar objetos de un mismo o diferentes tipos.
  - Son útiles para modelar paquetes de datos.
- Son referenciados mediante un nombre.
- No tienen nada que ver con la idea de registro en hardware.

```
TYPE nombreRegistro IS
RECORD
    identificador_1: tipo_1;
    ...
    identificador_n: tipo_n;
END RECORD nombreRegistro;
```

Tipos de datos (XII)

Tipos de datos compuestos.

Registros

```
TYPE codOperacion IS (sum, res, halt, sta);
TYPE instrucción IS
RECORD
   codigo: codOperacion;
   fuente: integer;
   destino: integer;
END RECORD instrucción;
```

- A partir de la declaración del registro, se pueden declarar objetos de este tipo:

```
VARIABLE ultimaInst: instruccion;
```

Tipos de datos (XIII)

Tipos de datos compuestos.

Punteros

- No se usan mucho.
- Solamente las variables pueden declararse de esta manera.

TYPE nombre **IS ACCESS** tipoDatosApuntado

```
TYPE apuntaEnteros IS ACCESS integer;
VARIABLE enteroMemoria: apuntaEnteros;
```

Tipos de datos (XIV)

Tipos de datos compuestos.

Punteros

- Ejemplos:
  - Para crear un nuevo objeto:
    ```
    enteroMemoria:= NEW INTEGER;
    ```
  - Para acceder al dato apuntado por el objeto:
    ```
    enteroMemoria.ALL:= 20;
    ```
  - Para liberar la memoria ocupada cuando ya no se requiere el objeto:
    ```
    Deallocate(enteroMemoria);
    ```

Tipos de datos (XV)

Subtipos

- Se utilizan para definir subconjuntos ya declarados en un tipo de datos.
  - Son útiles para detectar si un objeto toma valores dentro del rango esperado.
- Un tipo y un subtipo se consideran el mismo tipo de datos y pueden mezclarse en una operación.

SUBTYPE identificadorSubtipo **IS** identificadorTipo [**RANGE** valor1 **TO/DOWNTO** valor2]

```
TYPE hexadecimal IS ('0', '1', '2', '3', '4',
'5',  '6', '7', '8', '9', 'A', 'B' , 'C', 'D',
'E', 'F');

SUBTYPE octal IS hexadecimal RANGE '0' TO '7';
```

Tipos de datos en bibliotecas (I)

Tipos de datos.

- Tipos predefinidos en el paquete estándar ***STD***.
  - No es necesario referenciarlos con la cláusula ***USE***.

```
PACKAGE STANDARD IS

    -- predefined enumeration types:
    TYPE BOOLEAN IS (FALSE,TRUE);
    TYPE BIT IS ('0', '1');

    -- predefined numeric types:
    TYPE INTEGER IS RANGE -2147483648 TO 2147483647;
    TYPE REAL IS RANGE -1.0E38 TO 1.0E38;

    TYPE SEVERITY_LEVEL IS (NOTE, WARNING, ERROR, FAILURE);
```

Continúa en la siguiente transparencia

Tipos de datos en bibliotecas (II)

- Tipos predefinidos en el paquete estándar.

```
-- predefined enumeration types:
TYPE CHARACTER IS (
        NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
        BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
        DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
        CAN, EM,  SUB, ESC, FSP, GSP, RSP, USP,
        ' ', '!', '"', '#', '$', '%', '&', ''',
        '(', ')', '*', '+', ',', '-', '.', '/',
        '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', ':', ';', '<', '=', '>', '?',
        '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
        'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
        'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
        'X', 'Y', 'Z', '[', '\', ']', '^', '_',
        '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
        'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
        'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
        'x', 'y', 'z', '{', '|', '}', '~', DEL,
        C128, C129, C130, C131, C132, C133, C134, C135,
        C136, C137, C138, C139, C140, C141, C142, C143,
        C144, C145, C146, C147, C148, C149, C150, C151,
        C152, C153, C154, C155, C156, C157, C158, C159,
        ' ', '¡', '¢', '£', '¤', '¥', '¦', '§',
        '¨', '©', 'ª', '«', '¬', '-', '®', '¯',
        '°', '±', '²', '³', '´', 'µ', '¶', '·',
        '¸', '¹', 'º', '»', '¼', '½', '¾', '¿',
        'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç',
        'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï',
        'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×',
        'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß',
        'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç',
        'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
        'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷',
        'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ');
```

Continúa en la siguiente transparencia

Tipos de datos en bibliotecas (III)

- Tipos predefinidos en el paquete estándar.

```
-- predefined type TIME:
        TYPE TIME IS RANGE - 2**62 -2**62 TO 2**62 - 1 + 2**62
                UNITS
                        FS;
                        PS = 1000 FS;
                        NS = 1000 PS;
                        US = 1000 NS;
                        MS = 1000 US;
                        SEC = 1000 MS;
                        MIN = 60 SEC;
                        HR = 60 MIN;
                END UNITS;

        SUBTYPE DELAY_LENGTH IS TIME RANGE 0 FS TO TIME'HIGH;
        -- function that returns the current simulation time:
        FUNCTION NOW RETURN DELAY_LENGTH;
```

Continúa en la siguiente transparencia

Tipos de datos en bibliotecas (IV)

- Tipos predefinidos en el paquete estándar.

```
        -- predefined numeric subtypes:
        SUBTYPE NATURAL IS INTEGER RANGE 0 TO INTEGER'HIGH;
        SUBTYPE POSITIVE IS INTEGER RANGE 1 TO INTEGER'HIGH;

        -- predefined array types:
        TYPE STRING IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;
        TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT;
        TYPE FILE_OPEN_KIND IS (READ_MODE, WRITE_MODE, APPEND_MODE);
        TYPE FILE_OPEN_STATUS IS (OPEN_OK, STATUS_ERROR, NAME_ERROR, MODE_ERROR);

        ATTRIBUTE FOREIGN: STRING;
END STANDARD;
```

Tipos de datos en bibliotecas (V)

- **Paquete *STD_LOGIC_1164*.**
  - El paquete tiene que ser llamado mediante la cláusula *USE*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

  - Incluye tipos de datos del tipo bit con valores extendidos (‘0’, ‘1’, ‘Z’, ...,)

```
std_logic
std_logic_vector(indice_1 TO/DOWNTO indice_2)
```

  - Funciones de detección de flanco en señales:

```
rising_edge(identificadorSeñal)
falling_edge(identificadorSeñal)
```

Literales (I)

Comentarios.

- Cualquier línea que contenga dos guiones consecutivos, se considera un comentario que finaliza con el final de dicha línea.
- Ejemplo:

```
-- Esta línea es un comentario
```

Cadenas de caracteres.

- Útiles para test.
- Ejemplos:

```
""
"Ejemplo en ""VHDL"" de cadena de texto"
```

Literales (II)

Numéricos (enteros, reales o físicos).

- Base decimal.
  - Enteros
    - Ejemplos:

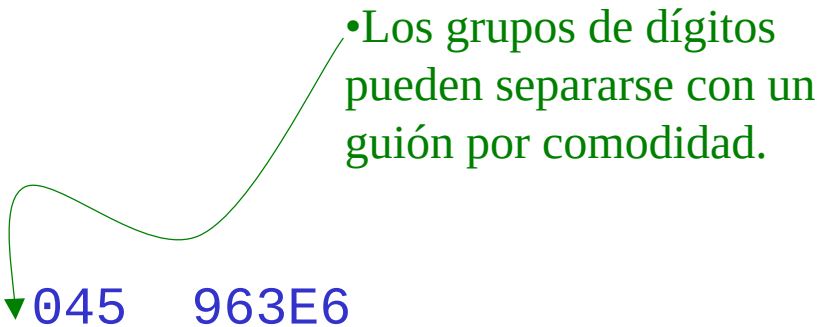
```
0   45   321_697_045   963E6
```

  - Reales.
    - Ejemplos:

```
0.0   0.9   2.356_93   12.4E-9
```

- Otra base (entre 2 y 16 siendo 10 por defecto).
  - Ejemplos:

```
2#1001_1010#    8#301#    16#FFF#
```



Literales (III)

Caracteres.

- Cualquier carácter entre comillas simples.
- Ejemplos:

```
'a', '3', 't'
```

Cadenas de bits.

- Secuencia de bits entre comillas dobles precedida por el especificador de la base
  - B: binaria, O: octal, X: hexadecimal.
- Ejemplos:

```
B"11101001"
O"126"
X"FE"
```

Literales (IV)

Enumerados.

- Son valores individuales del tipo de datos enumerado.
- Un literal enumerado puede ser un identificador, un carácter o una mezcla de ambos.
- Ejemplos para el tipo **bit**:

```
'0'  '1'
```

- Ejemplos para el tipo **std_logic**:

```
'0'  '1'  'Z'  'X'  'U'  'W'  'L'  'H'  '-'
```

- Ejemplos para el tipo **boolean**:

```
TRUE  FALSE
```

Operadores

Mayor precedencia

Operadores predefinidos

Menor precedencia

Operador	*Operación*	*Tipo del operando izquierdo*	*Tipo del operando derecho*	*Tipo del resultado*
**	exponenciación	entero o coma flotante	entero	igual que el operando izquierdo
abs	valor absoluto		numérico	igual que el operando
not	negación		***bit***, ***boolean***, vector de ***bit***, vector de ***boolean***	igual que el operando
*	multiplicación	entero o coma flotante	igual que el operando izquierdo	igual que operandos
		físico	***integer*** o ***real***	igual que el operando izquierdo
		integer o ***real***	físico	igual que el operando derecho
/	división	entero o coma flotante	igual que el operando izquierdo	igual que operandos
		físico	***integer*** o ***real***	
		físico	igual que el operando izquierdo	entero
mod		entero	igual que el operando izquierdo	igual que operandos
rem		ientero	igual que el operando izquierdo	igual que operandos
+	signo +		numérico	igual que operando
-	signo -		numérico	igual que operando
+	suma	numérico	igual que el operando izquierdo	igual que operandos
-	resta	numérico	igual que el operando izquierdo	igual que operandos
&	concatenación	vector	igual que el operando izquierdo	igual que operandos
		vector	tipo de un elemento del operando izquierdo	igual que el operando izquierdo
		tipo de un elemento del operando derecho	vector	igual que el operando derecho
		tipo de un elemento del resultado	tipo de un elemento del resultado	vector
sll	desplazamiento lógico a la izquierda	vector de ***bit*** o vector de ***boolean***	entero	igual que el operando izquierdo
srl	desplazamiento lógico a la derecha			
sla	desplazamiento aritmético a la izquierda			
sra	desplazamiento aritmético a la derecha			
rol	rotación hacia la izquierda			
ror	rotación hacia la derecha			
=	igualdad	cualquiera excepto tipo file, tipos protected o vectores de tipos discretos	igual que el operando izquierdo	***boolean***
/=	desigualdad			
<	menor que			
<=	menor o igual que			
>	mayor que			
>=	mayor o igual que			
and	and lógico	***bit***, ***boolean*** o vectores/ matrices de ***bit*** o ***boolean***	igual que el operando izquierdo	igual que operandos
or	or lógico			
nand	and lógico negado			
nor	or lógico negado			
xor	or exclusivo			
xnor	or exclusivo negado			

Atributos (I)

- **Un atributo es una característica (función, tipo, rango, señal o constante), que puede ser asociada (atribuida) a ciertos elementos del modelo VHDL.**
  - Dos posibilidades:
    - Atributos definidos por el usuario.
    - Predefinidos.
- **Atributos predefinidos para señales.**
  - Dan información sobre las señales o definen nuevas señales implícitas derivadas de las señales declaradas explícitamente.

Atributos de señal (I)

Atributo	*Tipo de resultado*	*Resultado*
S'delayed(t)	Tipo base de señal S	Retarda la señal S, t unidades de tiempo
S'stable(t)	boolean	True, cuando no ha habido eventos en S durante t
S'quiet(t)	boolean	True, cuando no ha habido transacciones en S durante t
S'transaction	bit	Señal cuyo valor cambia cuando hay transacción en S
S'event	boolean	True, cuando se ha producido un evento en la señal S
S'active	boolean	True, cuando se ha producido una transacción en S
S'last_event	time	Tiempo transcurrido desde el ultimo evento en S
S'last_active	time	Tiempo transcurrido desde la ultima transacción en S
S'last_value	Tipo base de señal S	Valor de S antes de ocurriera el último evento
S'driving	boolean	False, si el driver de S está desconectado
S'driving_value	tipo base de señal S	Valor actual de S

Atributos de señal (II)

Ejemplos para señales tipo bit	Significado
(clock='1') **AND** (clock'STABLE)	Señal clock= '1'
(clock='1') **AND** (**NOT** clock'EVENT)	Señal clock= '1'
(clock='0') **AND** (clock'STABLE)	Señal clock= '0'
(clock='0') **AND** (**NOT** clock'EVENT)	Señal clock= '0'
(clock='1') **AND** (**NOT** clock'STABLE)	Flanco de subida de señal clock
(clock='0') **AND** (**NOT** clock'STABLE)	Flanco de bajada de señal clock
(clock='1') **AND** (clock'EVENT)	Flanco de subida de señal clock
(clock='0') **AND** (clock'EVENT)	Flanco de bajada de señal clock

•Nota: para detectar flancos de señales del tipo ***std_logic_vector*** es mejor emplear las funciones **rising_edge(S)** y **falling_edge(S)**.