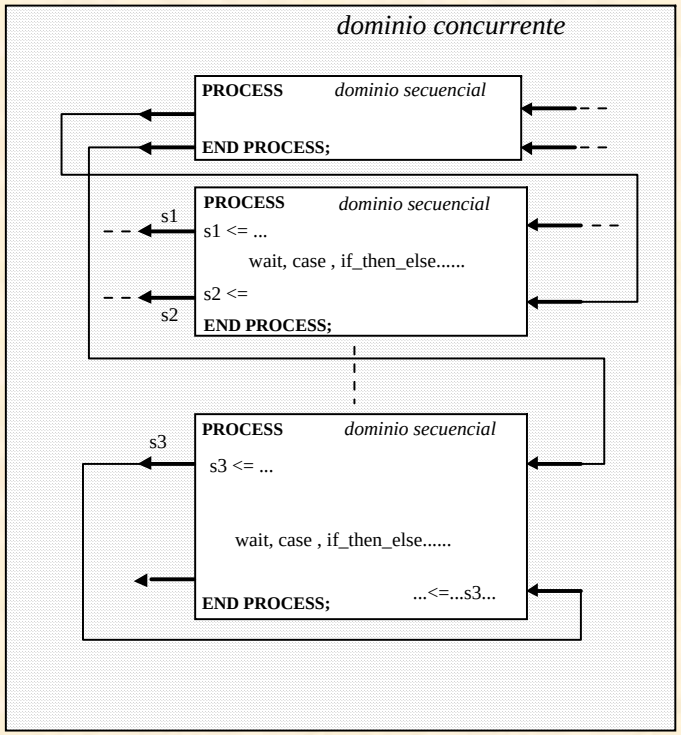
# Sesión 3: Modelo temporal: VHDL concurrente

## Dominios secuencial y concurrente en VHDL

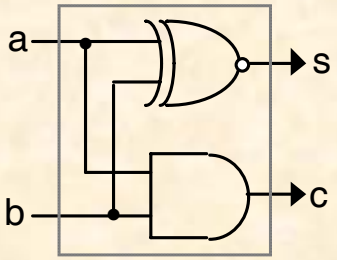
```
ENTITY dispositivo IS
    PORT(pe1,...:IN tipoe; ps1,...:OUT tipos; pes1,...:INOUT tipoes
END  dispositivo

ARCHITECTURE ejemplo OF dispositivo IS
    SIGNAL  s1, s2,... : tipo;
BEGIN
```

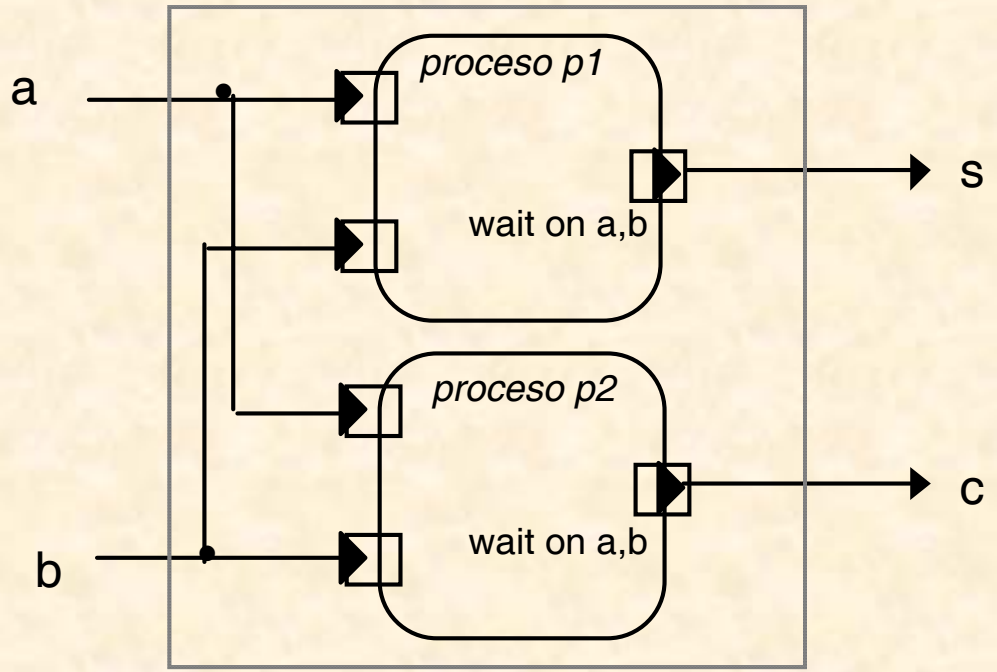



```
END ejemplo;
```

## Ejemplo de Programa Concurrente

```
ENTITY medio_sumador IS
  PORT(a,b : IN bit; s,c : OUT bit);
END medio_sumador;

ARCHITECTURE concurrente OF medio_sumador IS
BEGIN

  p1 : PROCESS
  BEGIN
    s <= a XOR b;
    WAIT ON a,b;
  END PROCESS;

  p2 : PROCESS
  BEGIN
    c <= a AND b;
    WAIT ON a,b;
  END PROCESS;

END concurrente;
```



# Sentencia *Process*

***Sintaxis:***

```
[rótulo_proceso : ]
        PROCESS [(lista_sensibilización)]
                parte_declarativa_proceso
        BEGIN
                sentencias_secuenciales
        END PROCESS [rótulo_proceso];
```

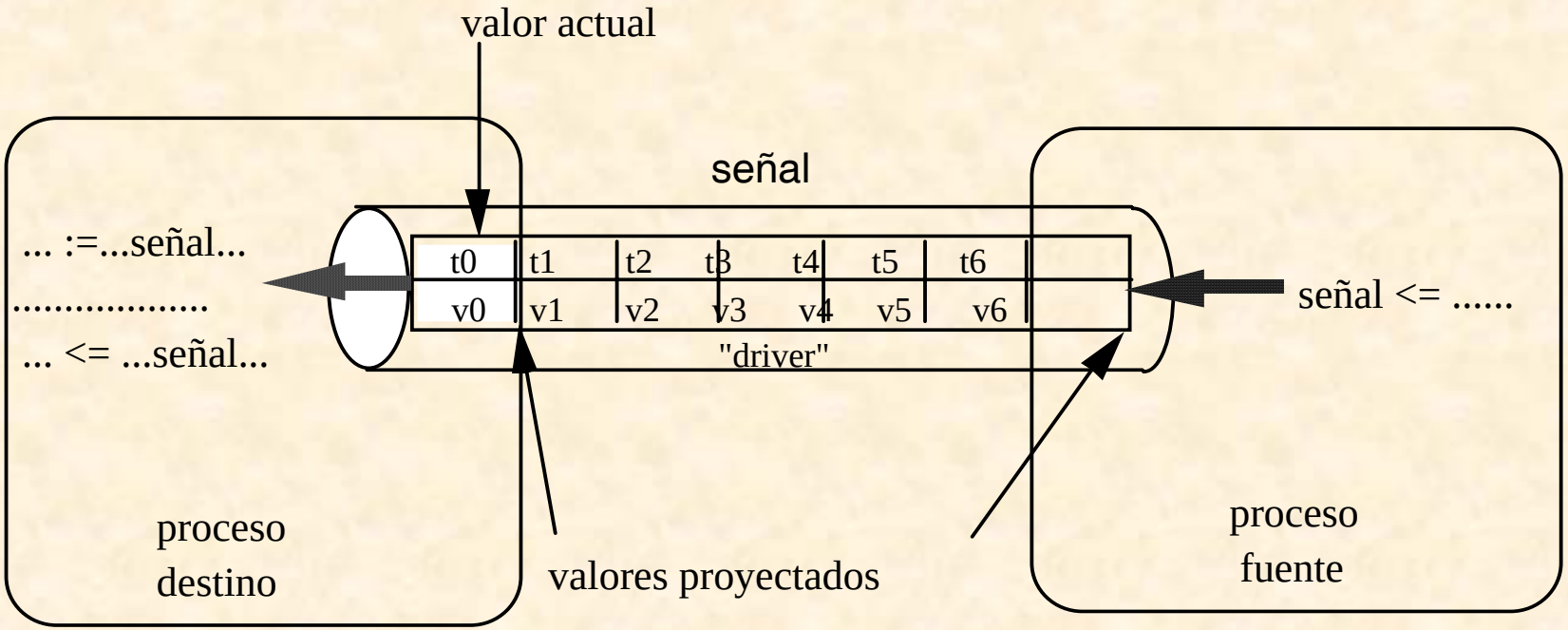
- Un proceso puede estar *activo* o *suspendido*..
- Cuando comienza la simulación todos los procesos están activos.
- En un proceso activo las sentencias se ejecutan continuamente, comenzando de nuevo por la primera cuando se alcanza la última.
- Un proceso suspende cuando ejecuta una sentencia WAIT, y permanece en este estado hasta que se cumplen los requisitos de la reactivación: *sensibilización, condición y temporización* .

**Ejemplo:**

| | |
|---|---|
| PROCESS(s1, s2, s3, ...)<br>BEGIN<br>.<br>.<br>.<br>END PROCESS; | PROCESS<br>BEGIN<br>.<br>.<br>.<br>WAIT ON s1, s2, s3, ...);<br>END PROCESS; |

## Señales

- Una señal consta de un *valor actual* y un conjunto de *valores proyectados* (posibles valores futuros).
- El valor actual es el leido por los procesos destino de la señal.
- Los valores proyectados y el actual constituyen el *driver* de la señal, esto es, una secuencia de pares <*valor,tiempo*> denominados *transacciones*..
- Las transacciones las crea el proceso fuente cuando asigna *valores* a la señal, disponiendose en el driver en orden ascendente de la componente de tiempo.
- Cada vez que una señal cambia de valor se dice que ha ocurrido un *evento* sobre ella.



## Tipos de retardo en las asignaciones de señal

### Retardo de Transporte

- Si el tiempo de retardo implica la creación de una nueva transacción con la componente de tiempo mayor que la de las demás transacciones del *driver*, la nueva transacción se añade al final del *driver*.
- Si por el contrario existen transacciones con tiempos mayores que la nueva transacción, estas transacciones se eliminan del *driver*.

**Ejemplo:**

```
SIGNAL s : integer := 0;
  p1 : PROCESS
  BEGIN
    s <= TRANPORT 1 AFTER 1 NS, 3 AFTER 3 ns, 5 AFTER 5 NS;
    s <= TRANSPORT 4 AFTER 4 NS;
    WAIT;
END PROCESS;
```

***drivers***

después de ejecutar:
s <= TRANPORT 1 AFTER 1 NS, 3 AFTER 3 ns, 5 AFTER 5 NS;

| 1 ns | 3 ns | 5 ns |
|---|---|---|
| 1 | 3 | 5 |

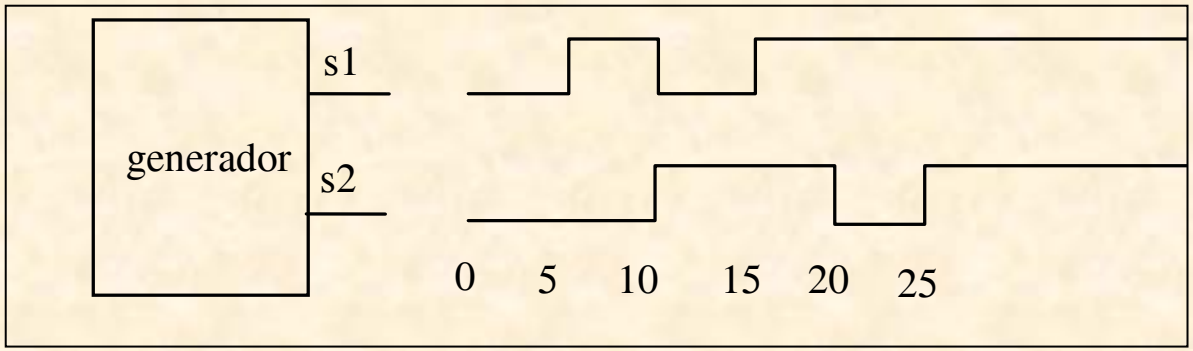
después de ejecutar:
s <= TRANSPORT 4 AFTER 4 NS;

| 1 ns | 3 ns | 4 ns |
|---|---|---|
| 1 | 3 | 4 |

# Generador de Formas de Onda

• Con retardos de transporte podemos diseñar generadores de cualquier tipo de forma de onda utilizando asignaciones de conjuntos ordenados de transacciones.

```
ENTITY generador IS
END generador;
ARCHITECTURE transporte OF generador IS
        SIGNAL s1,s2 : BIT;
BEGIN
                p1 : PROCESS
                BEGIN
s1 <= TRANSPORT '1' AFTER 5 ns, '0' AFTER 10 ns, '1' AFTER 15 ns; WAIT;
                END PROCESS;
                p2 : PROCESS
                BEGIN
s2 <= TRANSPORT '1' AFTER 10 ns, '0' AFTER 20 ns, '1' AFTER 25 ns; WAIT;
                END PROCESS;
END transporte;
```



## Retardo Inercial

- El efecto de una nueva transacción sobre todas las posteriores en el *driver* es el mismo que en el caso de transporte, esto es, se eliminan.
- La diferencia está en el efecto sobre las transacciones anteriores a la nueva transacción:
  - Si la componente valor de una transacción del *driver* es diferente al valor de la nueva transacción, se elimina del *driver*.

**Ejemplo:**

```
SIGNAL s : integer := 0;
 p1 : PROCESS
  BEGIN
    s <= INERTIAL 1 AFTER 1 NS, 3 AFTER 3 ns, 5 AFTER 5 NS;
    s <= INERTIAL 3 AFTER 4 NS, 4 AFTER 5 NS;
    WAIT;
END PROCESS;
```

después de ejecutar:
s <= INERTIAL 1 AFTER 1 NS, 3 AFTER 3 ns, 5 AFTER 5 NS;

***drivers***

| 1 ns | 3 ns | 5 ns |
|---|---|---|
| 1 | 3 | 5 |

después de ejecutar:
s <= INERTIAL 3 AFTER 4 NS, 4 AFTER 5 NS;

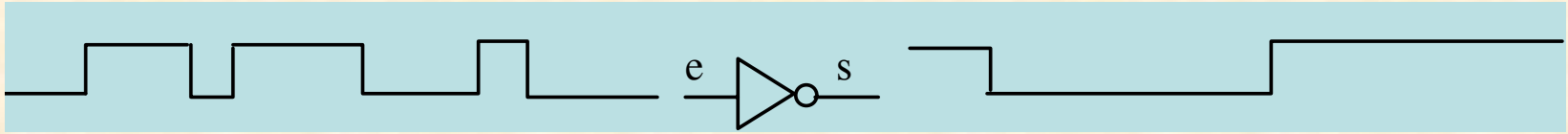
| 3 ns | 4 ns | 5 ns |
|---|---|---|
| 3 | 3 | 4 |

# Efecto del Retardo Inercial

- El transporte inercial produce en un componente la eliminación de los pulsos de entrada con una anchura inferior al valor del retardo especificado.
- Por defecto, si no se especifica nada en la sentencia de asignación de señal, se supone un retardo de tipo inercial.

- **Ejemplo**: el siguiente inversor con retardo inercisal de 10 ns, no responderá a los pulsos inferiores a este valor:

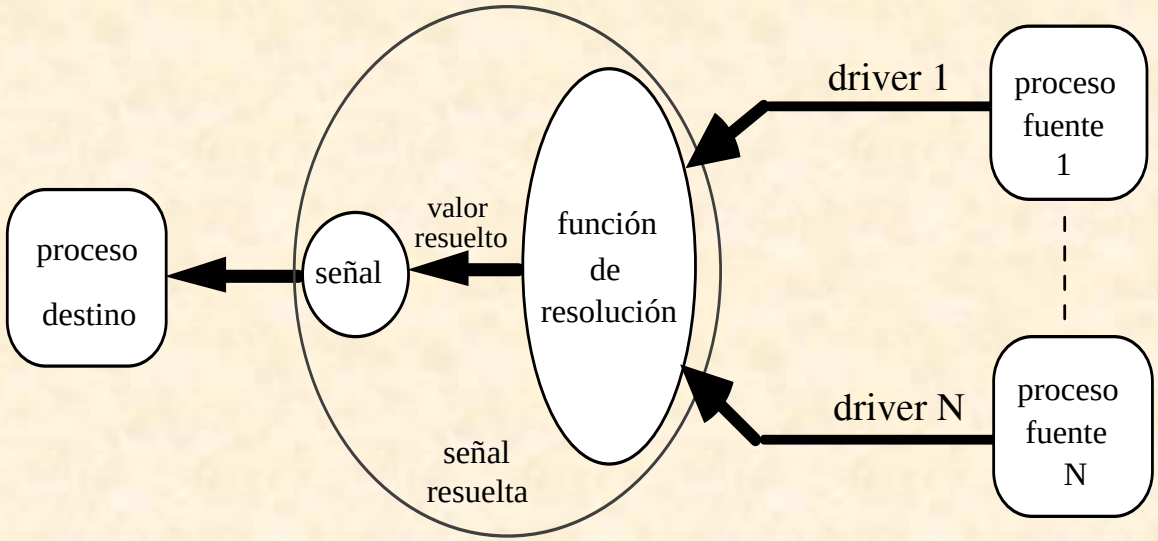
```
ENTITY inv IS
        PORT(e : IN bit; s : OUT bit);
END inv;
ARCHITECTURE inercial OF inv IS
BEGIN
        PROCESS(s3)
        BEGIN
  d <= NOT s3 AFTER 10 ns;
        END PROCESS;
END inercial;
```



## Señales resueltas

- Para que una señal pueda tener más de un proceso fuente y por consiguiente más de un *driver* asociado es obligatorio que sea una *señal resuelta*..
- Se dice que una señal es (o está) resuelta cuando tiene asociada una función (*función de resolución* ) que determina cual de los diferentes valores actuales de los correspondientes drivers se toma como valor de la señal.
- En general la función de resolución se asocia al tipo al que pertenece la señal resuelta.

**Ejemplo:** La siguiente arquitectura contiene dos procesos fuentes para la señal *s*, por lo que se supone que pertenece a un tipo resuelto:




```
ARCHITECTURE alfa OF beta IS
  SIGNAL s : tipo_resuelto;
BEGIN
  PROCESS
  BEGIN
    s <= 1 AFTER 1 NS;
  END PROCESS;
  PROCESS;
  BEGIN
    s <= 2 AFTER 2 NS;
    WAIT;
  END PROCESS;
END alfa;
```

## Declaración de una Función de Resolución

• Una función de resolución para un tipo dado es una función que devuelve un valor de dicho tipo y acepta como único argumento un *array* no restringido de elementos de ese tipo.

**Ejemplo**, para el tipo **bit**:

FUNCTION o_cableada(arg : **bit_vector**) RETURN **bit**;

La asociación de una señal con una función de resolución puede hacerse de dos formas:

1. Utilizando un tipo resuelto (un tipo asociado a la función de resolución):

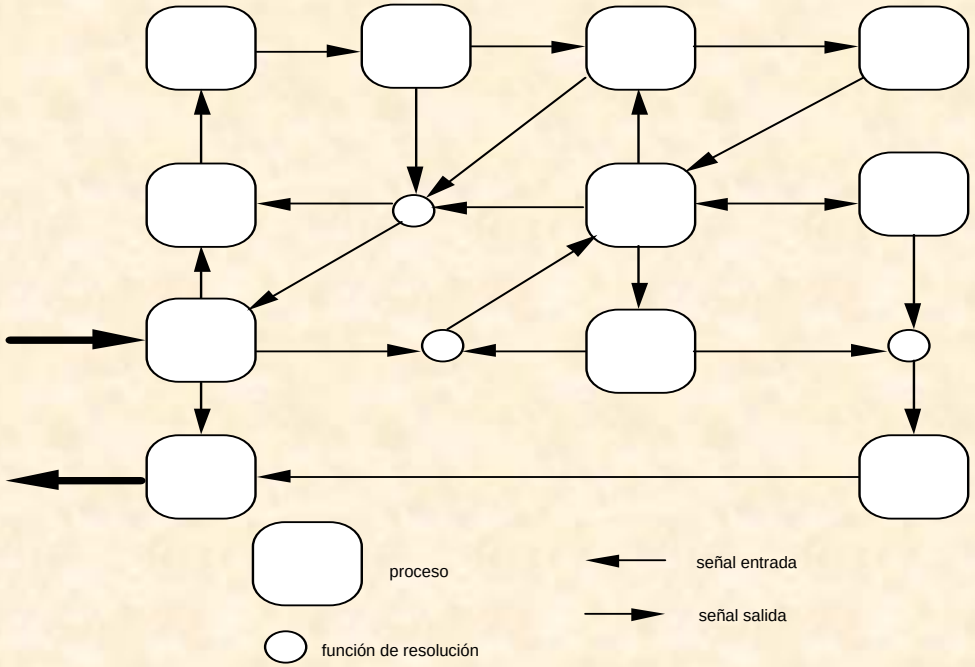
```
SUBTYPE bit_resuelto IS o_cableada bit;
SIGNAL señal_resuelta : bit_resuelto;
```

2. Directamente, asociando la señal con la función de resolución:

```
SIGNAL señal_resuelta : o_cableada bit;
```

## Simulación

- Un programa VHDL es un conjunto de procesos que se comunican mediante señales.
- Cada proceso es un algoritmo secuencial que proyecta valores para las señales de salida utilizando la sentencia de asignación de señal (<=).
- Más de un proceso puede proyectar valores para la misma señal si ésta es resuelta, es decir, tiene asociada una función de resolución para calcular el valor efectivo.
- Un proceso suspende su ejecución cuando alcanza una sentencia *wait,* y permanece en ese estado hasta que se cumplen las condiciones asociadas al *wait* .



# Proceso Kernel de Simulación Definido en el LRM

- El Manual de Referencia del Lenguaje (LRM) de VHDL utiliza un *proceso kernel* como representación conceptual del agente que coordina la actividad de los procesos de un programa durante una simulación.
- Este agente se encarga de la propagación de los valores de las señales y de detectar las condiciones que han de cumplirse para reactivar procesos.
- La ejecución de un programa VHDL consta de una *fase de inicialización* seguida de *ciclos de simulación*.

***Fase de inicialización***

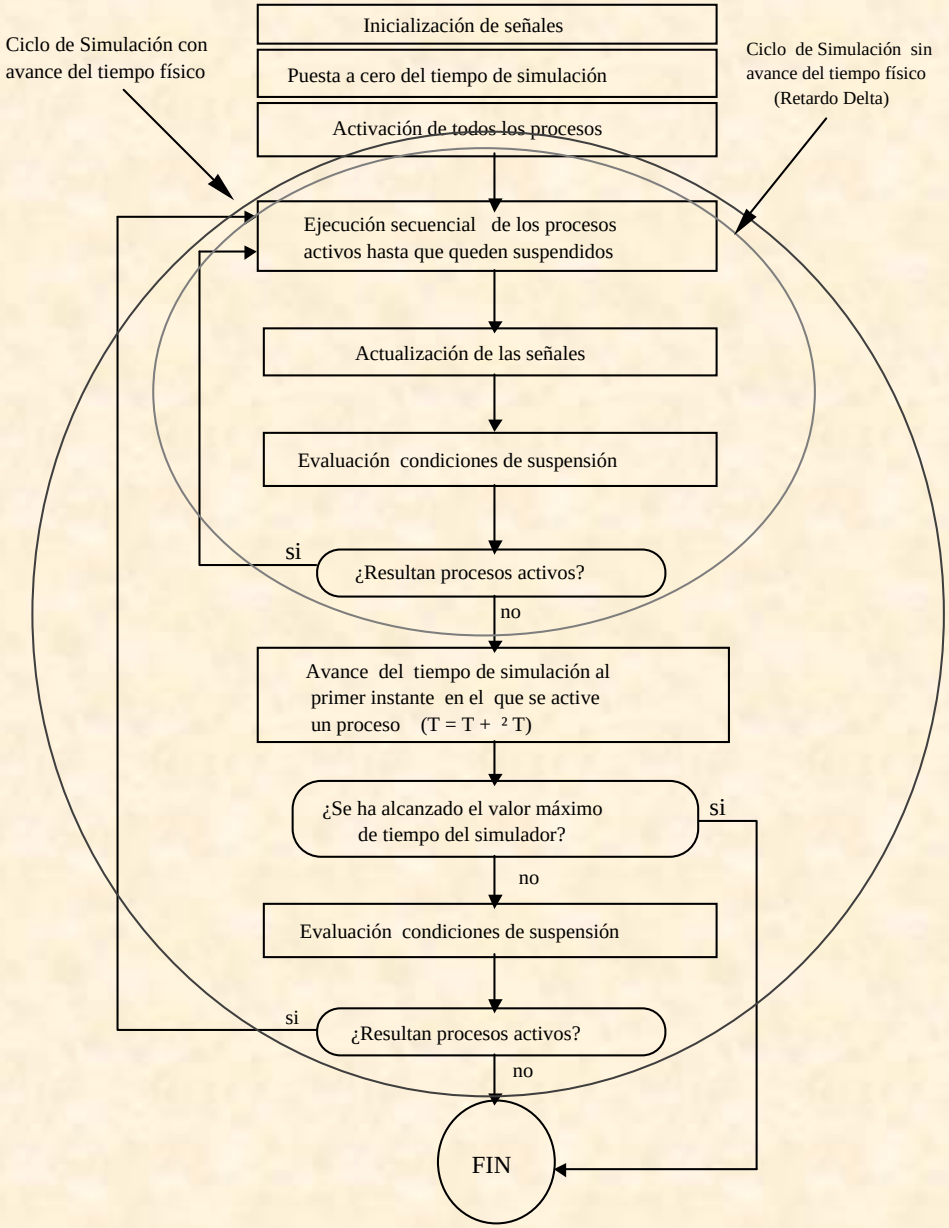
- Se inicializa el valor de todas las señales
- Se pone a cero el tiempo de simulación
- Se ejecutan todos los procesos del programa hasta que suspendan.

**El orden de ejecución de los procesos es indiferente por cuanto los nuevos valores generados para las señales no se repercuten hasta que todos los procesos quedan suspendidos.**

***Ciclos de simulación*.** Enn cada ciclo de simulación:

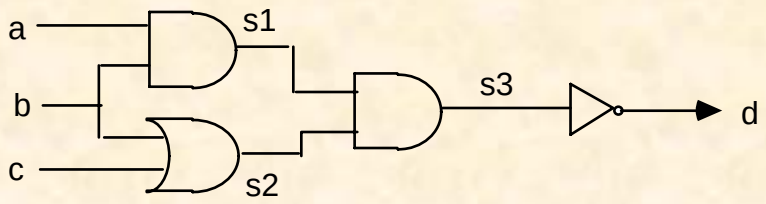
- Se actualizan los valores de las señales
- Se evalúan las condiciones de suspensión.
- Si resultan procesos activos en el tiempo de simulación actual, se ejecutan y se vuelven a actualizar las señales y a evaluar las condiciones de suspensión.
- Este ciclo se repite mientras se produzcan procesos activos en el tiempo de simulación actual.
- Una vez que todos los procesos están suspendidos en el tiempo de simulación actual, se avanza el tiempo al siguiente valor en que se produce la activación de algún proceso.
- Si no fuese posible o se alcanzase el valor máximo de tiempo admitido, finalizaría la simulación.
- En caso contrario se evalúan las condiciones de suspensión para el nuevo tiempo y se repite el ciclo para los nuevos procesos activos.

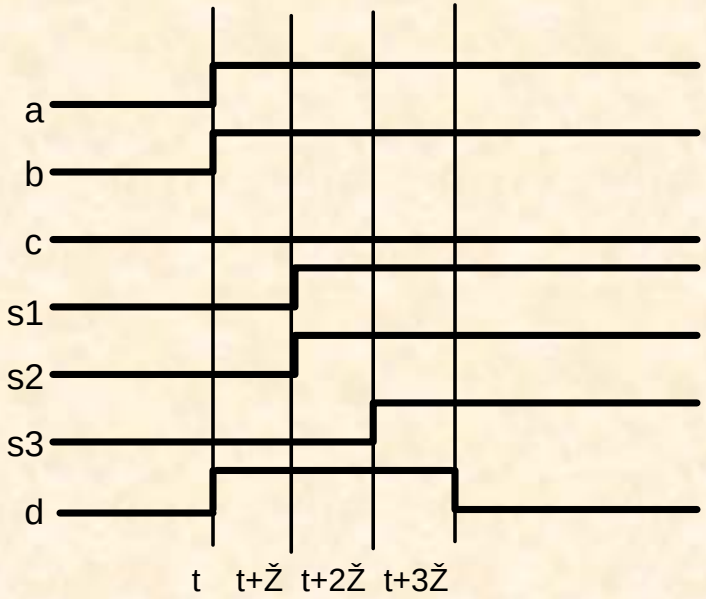
## Proceso de Simulación de un Programa VHDL



# Ejemplo de *retardos delta* (∂)

```
ENTITY prueba IS
  PORT(a,b,c : IN BIT; d : OUT BIT);
END prueba;
```



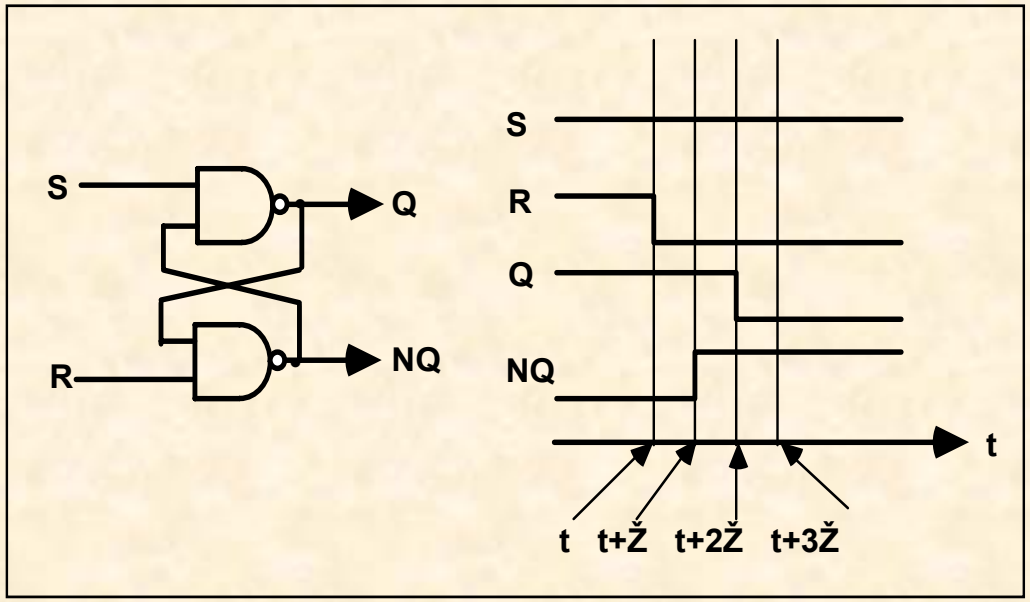

```
ARCHITECTURE delta OF prueba IS
  SIGNAL s1,s2,s3 : BIT;
BEGIN
            p1 : PROCESS (a, b)
            BEGIN
            s1 <= a AND b;
            END PROCESS;
            p2 : PROCESS(b, c)
            BEGIN
             s2 <= b OR c;
            END PROCESS;
            p3 : PROCESS(s1, s2)
            BEGIN
            s3 <= s1 AND s2;
            END PROCESS;
            p4 : PROCESS(s3)
            BEGIN
            d <= NOT s3;
            END PROCESS;
END delta;
```

| ns | delta | a | b | c | d | s1 | s2 | s3 |
|---|---|---|---|---|---|---|---|---|
| 0 | +0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | +1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | +2 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | +3 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |

S3

# Comportamiento de un Sistema Realimentado (*flip-flop rs* )

```
ENTITY ff_rs IS
        PORT( r, s : IN BIT; q, nq : OUT BIT);
END ff_rs;
```




```
ARCHITECTURE ff_rs OF ff_rs IS
BEGIN
        p1 : PROCESS(s, nq)
        BEGIN
                q <= s NAND nq;
        END PROCESS;
        p2 : PROCESS(r, q)
        BEGIN
                nq <= r NAND q;
        END PROCESS;
END ff_rs;
```

## Atributos de señal

• Existen dos grupos de atributos definidos sobre las señales:

**Atributos de señal tipo función**

• Con ellos se obtienen diferentes informaciones del comportamiento de una señal, por ejemplo, si ha cambiado, el tiempo transcurrido desde el último cambio, etc.

**Atributos de señal tipo señal**

• La información devuelta por estos atributos es similar en algunos casos a la proporcionada por los atributos tipo función.

• La diferencia estriba en que estas señales especiales se pueden utilizar en casi todos los lugares que una señal normal, incluyendo listas de sensibilización.

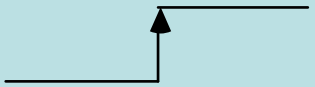
• No pueden utilizarse dentro de subprogramas.

**S'EVENT**
(tipo función)

• Devuelve *true* si ha ocurrido un evento en el *delta* actual, en caso contrario devuelve *false*.

• Es útil para determinar flancos de reloj.

**Ejemplo**: detección de un flanco positivo.



**(clk = ´1´) AND (clk'EVENT)**

## Biestable *D :*

```
ENTITY ff_d IS
        PORT(d, clk : IN BIT; q : OUT BIT);
END dff;

ARCHITECTURE ff_d OF ff_d IS
BEGIN
        PROCESS(clk)
        BEGIN
                IF (clk = ´1´) AND (clk'EVENT) THEN q <= d; END IF;
        END PROCESS;
END ff_d;
```

S**'LAST_VALUE**
(tipo función)

- Devuelve el valor de S antes del último evento.

- Es útil cuando hay que asegurarse del valor de procedencia de una señal cuando experimenta un evento.

**Ejemplo**: flanco positivo. Si *clk* está definida sobre un tipo multivaluado, por ejemplo TYPE bit3 IS (´1´,´0´,'X')

```
ENTITY ff_d IS
        PORT(d, clk : IN  bit3; q : OUT bit3);
END dff;
ARCHITECTURE ff_d OF ff_d IS
BEGIN
        PROCESS(clk)
        BEGIN
        IF (clk = ´1´) AND (clk'EVENT) (clk'LAST_VALUE = ´0´)
        THEN    q <= d; END IF;
        END PROCESS;
END ff_d;
```

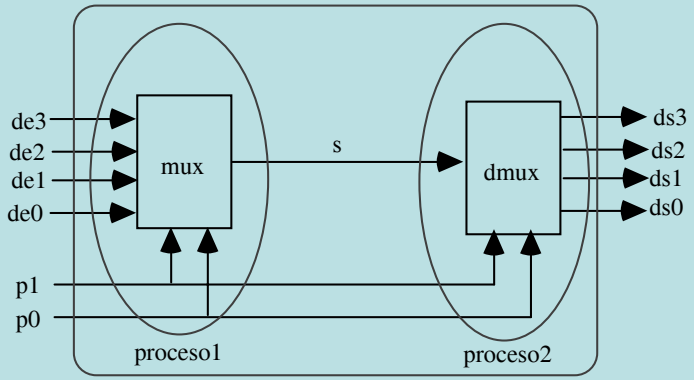
# Practica 3: VHDL concurrente

## Objetivos

1. Comprensión del modelo temporal de VHDL.
2. Diseño de pequeños sistemas concurrentes.

## Práctica a realizar

*Diseñar dos procesos correspondientes a dos módulos combinacionales, por ejemplo, un multiplexor y un demultiplexor, y conectarlos en el dominio concurrente de una arquitectura a través de señales. Se puede añadir un tercer proceso para generar los test de prueba, o bien aplicarlos desde el simulador con un archivo macro..*




## Resultados a entregar

**Documentación** del programa en la que conste:

1. *Especificación* del sistema modelado.
2. *Listado* VHDL comentado del código del programa.
3. *Relación E/S* que muestre la respuesta del sistema frente a entradas significativas.