

3. Sentencias secuenciales

Se utilizan para modelar comportamiento por procedimientos algorítmicos, por lo que su uso está permitido únicamente dentro de sentencias Process o subprogramas.

3.1 Sentencias IF

Permite seleccionar la ejecución de diferentes grupos de sentencias en función del resultado booleano de expresiones o condiciones. El formato general de la sentencia es

```
IF condición_1ª THEN      -- Puede haber varios IF anidados
    secuencia 1ª de sentencias;
{ ELSIF condición_2ª THEN  -- Cláusulas ELSIF opcionales.
    secuencia 2ª de sentencias;} -- Puede haber varias cláusulas ELSIF anidadas.
[ ELSE                     -- Cláusula ELSE opcional. Solo puede haber
    secuencia 3ª de sentencias;] -- una cláusula ELSE.
END IF ;
```

Este formato general contiene tres estructuras básicas como muestran los ejemplos :

```
IF ( semáforo = rojo ) THEN
    barrera := baja;          -- si se cumple la condición, si es cierta, se ejecutan
    intermitente := encendido; -- las sentencias que siguen
END IF;
```

```
IF ( semáforo = rojo ) THEN
    barrera := baja;          -- si se cumple la condición, si es cierta, se ejecutan
    intermitente := encendido; -- las sentencias que siguen y termina la sentencia IF

ELSE
    barrera := alta;          -- si no se cumple la condición se ejecutan las
    intermitente := apagado;  -- sentencias después de la cláusula ELSE
END IF;                      -- y se termina la sentencia IF.
```

```
IF ( semáforo = rojo ) THEN
    barrera := baja;          -- si se cumple condición 1ª, si es cierta, se ejecutan
    intermitente := encendido; -- las sentencias que siguen y termina la sentencia IF.

ELSIF ( semáforo = ambar ) THEN
    barrera := alta;          -- Si no se cumple la condición 1ª, comprobar
    intermitente := encendido; -- si se cumple la condición 2ª. Si es cierta,
                                -- se ejecutan las sentencias que siguen
                                -- y se termina la sentencia IF.

ELSE
    barrera := alta;          -- si no se cumple la condición 1ª, ni tampoco la 2ª,
    intermitente := apagado;  -- se ejecutan las sentencias después de la cláusula
END IF;                      -- ELSE y se termina la sentencia IF.
```

El uso de cláusulas ELSIF es una alternativa al uso de IFs anidados que, aún siendo legales en VHDL tienen una lectura e interpretación más complejas que aquella que resulta con cláusulas ELSIF.

Nótese que el resultado de usar cláusulas ELSIF en lugar de IFs anidados no es normalmente el mismo. No hay equivalencia directa, aunque las descripciones hechas con IFs anidados puedan sustituirse con sentencias basadas en ELSIFs. Vease un ejemplo:

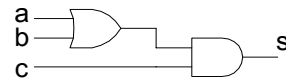
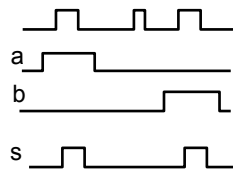
```
IF semaforo = rojo THEN
    barrera := baja;
ELSIF ticket = '1' THEN
    intermitente := pulsante;
END IF;
```

```
IF semaforo = rojo THEN
    barrera := baja;
    IF ticket = '1' THEN
        intermitente := pulsante;
    END IF;
END IF;
```

Otro aspecto importante es el uso u omisión de las cláusulas ELSE, que puede dar lugar a descripciones con comportamientos imprevistos. El siguiente ejemplo permite ver la importancia de las cláusulas ELSE al modelar circuitos:

Se supone que la descripción siguiente forma parte de una arquitectura en la que se hace una descripción secuencial de la que solo se reproduce la parte relativa al ejemplo

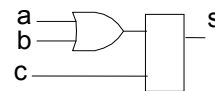
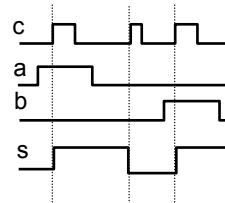
```
IF ( c = '1' ) THEN
    IF ( a = '1' OR b = '1' ) THEN
        s <= '1';
    ELSE
        s <= '0';
    END IF;
ELSE
    s <= '0';
END IF;
```



La descripción anterior corresponde al circuito de la figura y formas de onda adjuntas.

Cuando se utilizan herramientas CAD para síntesis se infieren elementos hardware y el resultado, aunque no evidente a primera vista, puede diferir notablemente por el empleo u omisión de la cláusula ELSE. En el ejemplo, si se suprime la cláusula ELSE del IF externo, el comportamiento que se describe cambia a

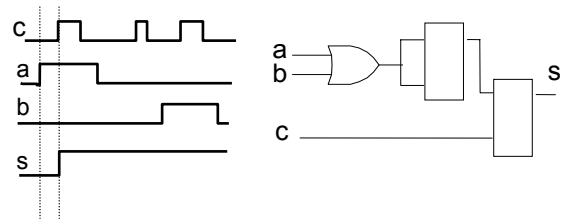
```
IF ( c = '1' ) THEN
    IF ( a = '1' OR b = '1' ) THEN
        s <= '1';
    ELSE
        s <= '0';
    END IF;
END IF;
```



Al suprimir la cláusula ELSE externa, se infiere la existencia de un elemento donde se guarda el último valor definido de la señal de salida para las condiciones de entradas en las que la señal de salida no está definida en la sentencia IF-THEN externa. Nótese que en esta descripción no se indica como responde el circuito cuando c cambia de '1' a '0'.

Si se suprime también la otra cláusula ELSE, el comportamiento cambia de nuevo a

```
IF ( c = '1' ) THEN
  IF ( a = '1' OR b = '1' ) THEN
    salida <= '1';
  END IF;
END IF;
```



Al igual que en el caso anterior, se infiere otro elemento donde se almacena la salida de la puerta OR para todos aquellos casos no especificados en la sentencia IF-THEN interna.

Los ejemplos anteriores indican que, para evitar comportamientos no previstos, es conveniente incluir siempre la cláusula ELSE en las sentencias IF, o al menos, revisar con atención los efectos de su omisión.

3.2 Sentencias CASE

Se usan cuando a partir de los valores de una expresión, se desea seleccionar alguna o algunas de las varias opciones posibles, lo que equivale en comportamiento al de la sentencia de asignación selectiva de valor a señales. Su formato básico es :

```
CASE expresión IS
  WHEN selección => secuencia de sentencias;
  { WHEN selección => secuencia de sentencias }
END CASE;
```

cuyo significado es que si el valor de una selección es alguno de los posibles que se indican de la expresión, deberán ejecutarse las sentencias que sigan a la correspondiente cláusula WHEN. Un punto importante a tener en cuenta es que deberán especificarse las acciones a tomar para todos los casos posibles o valores que pueden tenerse para la expresión que figura en la sentencia.

La *selección* puede tener varias formas :

- Un simple valor del tipo de la expresión : 5, '0', "0010", rojo, etc.
- Un rango de valores : 0 TO 9, rojo TO verde, etc.
- Una serie de valores separados por el delimitador | : 2|3|7|8, rojo|amarillo, etc.
- La palabra reservada OTHERS, equivalente a "todo lo no especificado antes".

Algunos ejemplos de la sentencia case :

```
CASE dato_triestado IS
  WHEN '0' => entrada <= '0';
  WHEN '1' | 'Z' => entrada <= '1';
END CASE;
```

```
CASE display_bcd IS
  WHEN "0000"      => display <= '0';
  WHEN "0001"      => display <= '1';
  .
  WHEN "1001"      => display <= '9';
  WHEN OTHERS => display <= '_'
END CASE;

CASE x + y + z IS
  WHEN 0          => suma <= " cero ";
  WHEN 1|2        => suma <= " bajo ";
  WHEN 9|10       => suma <= " alto ";
  WHEN (3 to 8)   => suma <= " medio ";
  WHEN OTHERS    => NULL;
END CASE;
```

3.3 Sentencias NULL

Es una sentencia secuencial que puede utilizarse para indicar que no debe realizarse ninguna acción, como se ve en el ejemplo anterior.

3.4 Sentencias LOOP

Se emplean cuando se necesita repetir una acción que, en la mayoría de los casos, ocurre mientras se mantiene o cumple una determinada condición.

Los LOOP pueden anidarse. Existen tres tipos de sentencias LOOP con los formatos genéricos que se indican :

3.4.1 LOOP SIMPLE

[etiqueta :]	[esquema repetitivo] LOOP	LOOP
	secuencia de sentencias	y := x**2;
	END LOOP [etiqueta] ;	EXIT WHEN x > 10 ;
		x := x + 1;
		END LOOP;

Si no se especifica un esquema iterativo, el LOOP entraría en un proceso repetitivo infinito. Para evitar este caso, que sería rechazado por el compilador, debe incluirse alguna sentencia para salir del bucle en función de alguna condición. Es típico el uso de sentencias EXIT.

3.4.2 LOOP FOR

```
[etiqueta :] FOR nombre de índice IN rango discreto del índice LOOP
  secuencia de sentencias
END LOOP [etiqueta] ;
```

No es necesario declarar el índice, ya que se considera una variable local declarada al inicializar el rango de valores entre los que se mueve. Su valor, por tanto, no es exportable.

Por la misma razón, si en la región donde opera el LOOP existe otra variable de igual nombre pero de distinto tipo, no interaccionarán entre ellas. Pero, en general, no es buena práctica *sobrecargar* si no es necesario. El VHDL no se confundirá, pero el modelador puede hacerlo. El índice puede ser un rango de valores de un tipo enumerado:

```
FOR dia IN lun TO sab LOOP          FOR i IN 1 to 10 LOOP
    color := negro;                  y := x**2;
END LOOP;                           END LOOP;
```

3.4.3 LOOP WHILE

```
[etiqueta :] WHILE condición booleana LOOP
    secuencia de sentencias a ejecutar si la condición es cierta
    sentencia de alteración del valor relacionado a la condición booleana
END LOOP [etiqueta] ;
```

Dentro del bucle no hay regiones declarativas. Las variables que intervengan en la condición booleana deben ser declaradas previamente para que el programa pueda evaluar la condición la primera vez que la encuentra. Las variables se alteran dentro del bucle y, al no ser locales a él, es necesario tener presente si el valor con que quedan puede afectar a otras secciones del área secuencial donde está el bucle y actuar en consecuencia. El ejemplo sería:

```
    x := 1;
    WHILE x <= 10 LOOP
        y := x**2;
        x:= x + 1;
    END LOOP;
```

3.5 Sentencias NEXT

Se emplean dentro de LOOPS para saltar la ejecución de las sentencias que siguen dentro del paso iterativo en curso o, simplemente, para continuar en el inicio del siguiente ciclo iterativo si la variable de iteración sigue dentro del rango del bucle, ya que si no finalizará el bucle. Su formato es :[SINTAXIS](#)

```
NEXT [etiqueta_de_LOOP] [WHEN condición] ;
```

y un ejemplo, relacionado con bucles LOOP de distinto tipo anidados :

```
bucle_1 : FOR i IN 1 TO 100 LOOP
    j := 150;
    bucle_2: LOOP
        j := j - i;
        NEXT bucle_1 WHEN j < (i * i);
    END LOOP bucle_2;
END LOOP bucle_1;
```

Nótese que el hecho de poner la etiqueta de un bucle permite actuar con la sentencia NEXT dentro de bucles anidados. Si no se incluye la etiqueta, por defecto se reinicia el bucle interno donde está la sentencia NEXT cuando se cumple la condición adjunta a WHEN.

Si se omite la condición, la sentencia es incondicional y se ejecuta siempre que la secuencia llega a ella.

3.6 Sentencias EXIT

Su uso, formato y comportamiento son muy similares a los de la secuencia NEXT:

EXIT [etiqueta_de_LOOP] [WHEN condición] ;

solo que en lugar de saltar a la siguiente iteración, salta fuera del bucle a que se refiere la etiqueta opcional o, por defecto, fuera del bucle que incluye la sentencia EXIT, continuando el programa en la sentencia que sigue a la sentencia END LOOP del bucle del que se sale al cumplirse la condición fijada a la cláusula WHEN . Si la condición no existe, la salida del bucle se ejecuta siempre que se accede a EXIT.

El ejemplo siguiente es aplicable para EXIT o NEXT

```
bucle_1:   FOR i IN 1 TO 10 LOOP                -- i se autodeclara aquí
            secuencia de sentencias 1ª ;

            bucle_2 : WHILE k <= 100 LOOP        -- k estará declarada previamente
                secuencia de sentencias 2ª ;
                NEXT bucle_1 WHEN condición 1ª ;
                NEXT bucle_2 WHEN condición 2ª ;
                secuencia de sentencias 3ª ;      -- k se modifica aquí
            END LOOP bucle_2;

        END LOOP bucle_1;
```

y otro ejemplo con bucles anidados :

```
bucle_1 : LOOP
            i := i + 1;
            j := 200;

            bucle_2: LOOP
                IF j < ( i * i ) THEN
                    EXIT bucle_2;
                END IF;
                j := j - i;
            END LOOP bucle_2;

            EXIT bucle_1 WHEN i > 10;
        END LOOP bucle_1;
```

3.7 Sentencias ASSERT

Durante los procesos de simulación del modelo, el diseñador necesita información de la evolución de señales o variables para conocer el comportamiento del modelo. En VHDL se dispone de la sentencia **ASSERT** para emitir mensajes cuyo contenido está prefijado por el diseñador en función del comportamiento que desea verificar automáticamente.

La sentencia **ASSERT** verifica el resultado booleano de una expresión y si ésta es cierta, si se cumple la condición prefijada, la sentencia **ASSERT** *no actúa*. Por el contrario, si el resultado de la verificación es falso, se emite un mensaje. El formato es :

```

ASSERT condición;      -- expresión booleana a evaluar .
[ REPORT "mensaje" ]   -- emitir si el resultado de evaluar la condición es falso.
[ SEVERITY nivel ]     -- tipo SEVERITY del empaquetamiento STANDARD

```

En bastantes casos, la condición que se fija va precedida de una cláusula **NOT**. El sentido de esto no es otro que fijar la actuación de la sentencia cuando la “condición negada” es cierta. La decisión para utilizar uno u otro modo de condición es simplemente reducir el tamaño o número de las condiciones a verificar: Puede ser más fácil y corto verificar que **NO** ocurre algo incorrecto, que comprobar que ocurren los demás casos posibles que son correctos.

```

ASSERT NOT ( set = '1' AND rst = '1')      -- verifica si (set = rst = '1') es TRUE
REPORT " set y reset son '1' "             -- mensaje del fallo a evitar detectado
SEVERITY ERROR;                            -- y detiene la simulación

```

El “mensaje”, que debe ir entre comillas, reportará un aviso cuyo significado conoce el diseñador. En cuanto al nivel del tipo **SEVERITY** enviado, determinará si la simulación debe continuar o no.

Normalmente la simulación se detiene con los niveles **ERROR y FAILURE** y será el creador del modelo quien deba prejuzgar la gravedad del posible error o condición que quiera verificar.

El nivel **NOTE** puede considerarse como un simple aviso, tal vez solo para avisar de que se ha cubierto una fase o etapa de la verificación.

El nivel **WARNING** tiene un nivel de importancia más alto. No detiene la simulación porque, por ejemplo, los resultados pendientes de obtener siguen siendo significativos e inafectados, tal vez, por la causa que motivó el mensaje de nivel **WARNING**. No obstante, el mensaje indica que hay algo que debe subsanarse. En cualquier caso la acción que se deriva de un nivel de severidad, está relacionado a la herramienta CAD en que se hace la simulación y , posiblemente, a los parámetros que determinen su acción.

Existe también una sentencia **ASSERT** concurrente, que actúa cuando alguna señal referida en la condición a verificar tiene un evento o cambio de valor. El formato es idéntico al de la sentencia secuencial, pero su ocurrencia viene determinada por el contexto de ejecución dentro del área concurrente donde se ubica.

3.8 Sentencias WAIT

Por su relación con el parámetro tiempo, las sentencias WAIT son elementos clave en las descripciones VHDL, empleándose en procesos y subprogramas en los que la suspensión temporal de su ejecución permite modelar retardos, sincronismos, etc. El formato más complejo de la secuencia es :

WAIT [ON lista] [UNTIL condición] [FOR tiempo] ;

Las opciones pueden combinarse, pero siempre debe existir al menos una de ellas. Si la palabra WAIT está aislada la simulación se detiene indefinidamente. Los formatos elementales de la sentencia son:

3.8.1 WAIT ON señales

Suspende la ejecución de un proceso hasta que en alguna de las señales de la lista se produce un evento o cambio en el valor de la misma. Esta opción de WAIT se utiliza como alternativa a la lista de sensibilidad a señales de un proceso, siempre que no haya otro WAIT interno. Se verá al tratar las sentencias PROCESS.

3.8.2 WAIT UNTIL condición

La condición tiene resultado booleano y suspende la ejecución de un proceso hasta que la condición se hace cierta. La condición se evalúa cuando cualquiera de las señales implicadas en la condición cambia de valor. Debe tenerse en cuenta el hecho de que esta opción de WAIT espera un evento en una señal para continuar. Así pues, cuando la condición de espera es una expresión en la que intervienen varios objetos, al menos uno de ellos debe ser señal y en ella deberá producirse el evento que hace cierta la condición de espera y que activa el proceso suspendido

WAIT UNTIL ((permiso = TRUE) OR (clk = '1')) ;

El proceso continúa en la sentencia que sigue a WAIT. Esta opción de WAIT es utilizada para descripción de sincronismos y flancos de señal. Es la opción normalmente soportada para modelar sincronismo en herramientas de síntesis.

3.8.3 WAIT FOR especificación de tiempo

Suspende la ejecución hasta que transcurre el tiempo especificado en la sentencia.

Los ejemplos siguientes muestran empleos combinados de las opciones WAIT :

WAIT;	-- Espera indefinida e incondicional
WAIT FOR 50 NS ;	-- Tiempo fijo de espera
WAIT ON x,y,z;	-- Espera hasta el cambio de alguna señal
WAIT UNTIL y = '1';	-- Espera hasta que y = '1'
WAIT ON x,y,z FOR 1ms;	-- Espera máxima 1ms hasta cambio de x,y o z
WAIT ON x,z UNTIL y = '1' ;	-- Espera un cambio en x, z, o y, siendo y = '1'
WAIT UNTIL x=y FOR 2 ms;	-- Espera máxima de 2 ms hasta que x = y

El uso de WAIT FOR es frecuente para modelar retardos máximos o “time out”.

Combinado con otras opciones de WAIT puede proporcionar resultados confusos, ya que la espera puede finalizar por un “time out” y no por la causa correcta que se espera en el modelo. El ejemplo siguiente muestra un posible empleo de la opción para no detener la simulación por espera de un evento que no se produce, pero que es necesario y, por tanto, no debe ser enmascarado por un “time out” :

```

WAIT until ok = '1';           -- si la señal “ok” no cambia, la simulación se
    bien <= '1' after 20 ns;    -- detiene indefinidamente y no puede simularse
WAIT until ok = '0';           -- las secciones relacionadas a la señal “bien”.
    bien <= '0' after 20 ns;

```

Para no condicionar la simulación del resto del modelo a la señal “ok”, se fuerza “time out” y se hace uso de ASSERT para poner de manifiesto la ausencia de cambio en “ok” :

```

WAIT until ( ok = '1' ) FOR 1 ms ;
    ASSERT ( ok = '1' )
    REPORT “ ok = '1' no se produjo”
    SEVERITY WARNING;
    bien <= '1' after 20 ns;
WAIT until ( ok = '0' ) FOR 1 ms ;
    ASSERT ( ok = '0' )
    REPORT “ ok = '0' no se produjo”
    SEVERITY WARNING;
    bien <= '0' after 20 ns;

```

Las sentencias WAIT son utilizadas frecuentemente para la descripción algorítmica de relojes, osciladores o generadores de ondas. Un ejemplo es el siguiente :

```

ENTITY generador IS
    GENERIC ( tiempo_uno, tiempo_cero : TIME);
    PORT ( sincro : IN BIT ; pulsos : OUT BIT );
END generador;

ARCHITECTURE algoritmica OF generador IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL sincro = '1';
        WHILE sincro = '1' LOOP
            pulsos <= '0';
            WAIT FOR tiempo_uno;
            pulsos <= '1';
            WAIT FOR tiempo_cero;
        END LOOP;
    END PROCESS;
END algoritmica;

```