

Diseño de Sistemas Secuenciales con VHDL

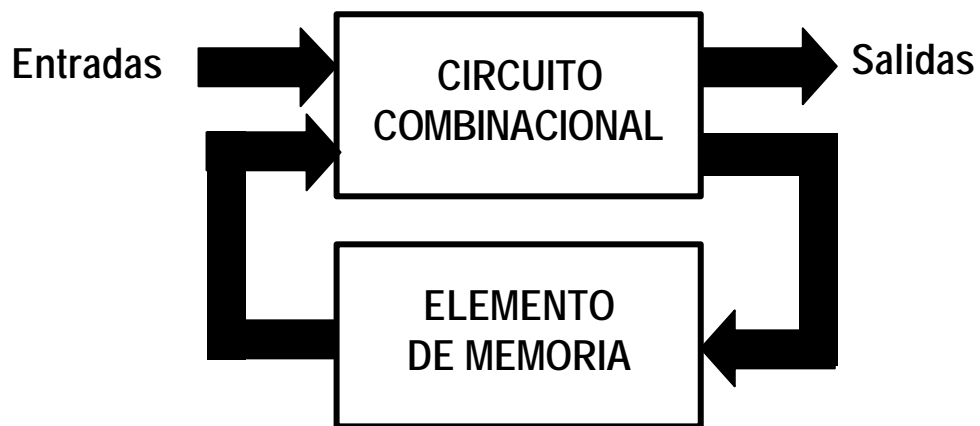
**Andres.lborra@upct.es
Juan.Suardiaz@upct.es**

Enero 2006

7.1 INTRODUCCIÓN

En la unidad anterior se han modelado sistemas combinacionales; es decir, aquellos sistemas cuyas salidas en un instante determinado sólo dependen de los valores que se encuentran presentes en sus entradas en ese momento.

Si bien un sistema secuencial puede tener también uno o más elementos combinacionales, la mayoría de los sistemas digitales que se encuentran en la práctica incluyen elementos de memoria, los cuales requieren que el sistema se describa en términos de lógica secuencial.



Un sistema secuencial está formado por un circuito combinacional y un elemento de memoria encargado de almacenar de forma temporal la historia del sistema. En esencia, la salida de un sistema secuencial no sólo depende del valor presente en las entradas en un instante determinado, sino también de la historia del sistema (se dice que los secuenciales son circuitos con memoria, mientras que los combinacionales no tienen memoria).

Básicamente hay dos tipos de sistemas secuenciales:

- ♦ Síncronos: su comportamiento se encuentra sincronizado mediante el pulso de reloj del sistema (CLK).
- ♦ Asíncrono: Su funcionamiento depende del orden y momento en el que se aplican las señales de entrada.

En esta unidad se diseñarán los circuitos secuenciales más utilizados en el diseño lógico a través del lenguaje de descripción de hardware VHDL.

7.2 DISEÑO BASADO EN ELEMENTOS MSI

Al igual que en la unidad anterior, se desarrollarán este tipo de sistemas mediante el uso del pensamiento estructurado: un circuito o sistema complejo se concibe como una colección de subsistemas más pequeños, cada uno de los cuales tiene una descripción más sencilla.

En el caso de los sistemas secuenciales, se partirá del elemento mínimo que será el biestable. Este bloque secuencial de construcción constituye el ladrillo con los que se edifican los sistemas secuenciales (equivalente a las puertas lógicas en los sistemas combinacionales). Uno de los bloques electrónicos más importantes de este tipo lo constituyen los denominados elementos MSI (*Medium Scale of Integration*) secuenciales estudiados en la asignatura de Electrónica Digital: circuitos contadores, registros de desplazamiento.... En esta unidad se describen las estructuras secuenciales más importantes desde una perspectiva de los lenguajes de descripción de hardware, en concreto usando el lenguaje VHDL.




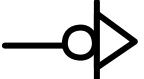
7.3 DESCRIPCIÓN VHDL DE LÓGICA SECUENCIAL

Uno de los conceptos nuevos que aparece en VHDL a la hora de describir sistemas secuenciales es la forma de describir la activación por flanco de reloj.

- **Atributo 'event.**

En el lenguaje VHDL los atributos sirven para definir características que se pueden asociar con cualquier tipo de datos, objeto o entidades. El atributo 'event (evento, donde ' indica que se trata de un atributo) se utiliza para describir un hecho u ocurrencia de una señal particular.

Considerando una señal de reloj (CLK), la sentencia CLK'event es cierta sólo cuando ocurre un cambio de valor (paso de '0' a '1' o de '1' a '0'). Combinándola con una sentencia de comprobación de igualdad del nuevo valor es posible definir la activación por el tipo de flanco que se desee:

Flanco de Subida	Flanco de Bajada
 	 
IF(CLK'event and CLK='1')	IF(CLK'event and CLK='0')

7.3.1 BIESTABLES

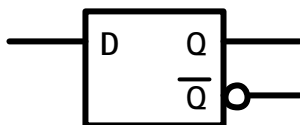
Se trata de un dispositivo de almacenamiento temporal de dos estados, que pueden permanecer en cualquiera de sus dos estados gracias a la capacidad de realimentación, lo que consiste en conectar las salidas con las entradas.

Es posible agrupar los biestables en dos grupos:

- **Asíncronos:** La salida cambia de estado cuando cambian las entradas.
- **Síncronos:** La salida cambia de estado (en función de las entradas) de forma acompasada con una señal de reloj. Este tipo de dispositivos síncronos a su vez se clasifican en:
 - ♦ **Activos por nivel (alto o bajo):** La salida cambia de estado sólo cuando la señal de reloj se encuentra en el estado lógico '1' (nivel alto) ó '0' (nivel bajo).
 - ♦ **Activos por flanco (subida o bajada):** La salida cambia de estado sólo cuando la señal de reloj pasa de un nivel lógico de '0' a '1' (flanco de subida) o de '1' a '0' (flanco bajada).

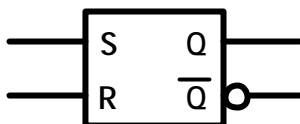
- **Tipos de biestables.**

Independientemente del momento de activación, es posible catalogar los biestables en los siguientes tipo:



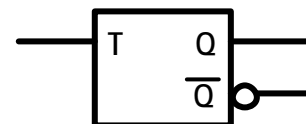
BIESTABLE D

D	Q*
0	0
1	1



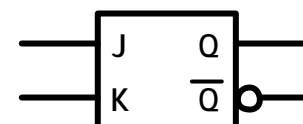
BIESTABLE S-R

S	R	Q*
0	0	Q
0	1	0
1	0	1
1	1	No Valido



BIESTABLE T

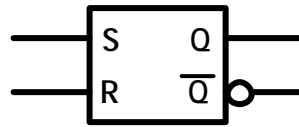
T	Q*
0	Q
1	\bar{Q}



BIESTABLE J-K

J	K	Q*
0	0	Q
0	1	0
1	0	1
1	1	\bar{Q}

- Ej_1: Biestable S-R asíncrono.



BIESTABLE S-R

S	R	Q*
0	0	Q
0	1	0
1	0	1
1	1	No Valido

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

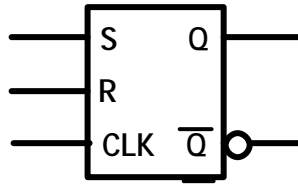
```
ENTITY BiestableSR IS
  PORT ( sS, sR: IN std_logic;
        sQ, sQn: INOUT std_logic);
  -- Por realimentación INOUT
END BiestableSR;
```

```
ARCHITECTURE BiestableSRArch OF
  BiestableSR IS
  BEGIN
    -- Descripción mediante las ecuaciones
    -- características (Realimentación)
    sQ <= sS nor sQn;
    sQn <= sR nor sQ;
  END BiestableSRArch;
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY BiestableSR IS
  PORT ( sS, sR: IN std_logic;
        sQ, sQn: INOUT std_logic);
END BiestableSR;
-- Versión secuencial
ARCHITECTURE BiestableSRArch2 OF
  BiestableSR IS
  BEGIN
    PROCESS(sS, sR, sQ, sQn)
    BEGIN
      IF (sS = '0' and sR = '0') THEN
        sQ <= sQ; sQn <= sQn;
      ELSIF (sS = '0' and sR = '1') THEN
        sQ <= '0'; sQn <= '1';
      ELSIF (sS = '1' and sR = '0') THEN
        sQ <= '1'; sQn <= '0';
      ELSE sQ <= '-'; sQn <= '-';
      -- Se usa '-' para el caso no válido
      END IF;
    END PROCESS;
  END BiestableSRArch2;
```

- Ej_2: Biestable S-R síncrono activo en nivel alto.



```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY BiestableSRNivel IS
    PORT ( sS, sR: IN std_logic;          --Señales de entrada.
           sCLKH: IN std_logic;          --Señal de reloj.
           sQ, sQn: INOUT std_logic);    --Salida
END BiestableSRNivel;

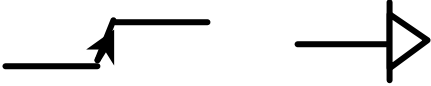
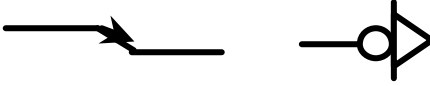
ARCHITECTURE BiestableSRNivelArch OF BiestableSRNivel IS
BEGIN
    PROCESS(sS, sR, sCLKH, sQ, sQn)
    BEGIN
        -- Se cambia de estado sólo si el reloj está a nivel alto '1'
        IF (sCLKH = '1') THEN
            IF (sS = '0' and sR = '0') THEN
                sQ <= sQ; sQn <= sQn;
            ELSIF (sS = '0' and sR = '1') THEN
                sQ <= '0'; sQn <= '1';
            ELSIF (sS = '1' and sR = '0') THEN
                sQ <= '1'; sQn <= '0';
            ELSE sQ <= '-'; sQn <= '-';
            END IF;
        ELSE NULL;
        END IF;
    END PROCESS;
END BiestableSRNivelArch;

```

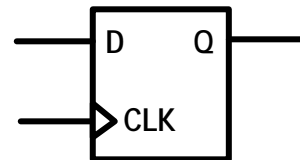
◆ Modelado VHDL de la activación por flanco.

Tal y como se ha comentado anteriormente, la principal opción que ofrece VHDL para modelar la activación por flanco en los circuitos secuenciales es mediante el atributo 'event. Sin embargo, es posible modelarlos también mediante dos formas adicionales:

- Dentro de un proceso con la sentencia WAIT UNTIL.
- Usando las macros VHDL'93 rising_edge(señal) y falling_edge(señal);

Flanco de Subida	Flanco de Bajada
	
IF(CLK'event and CLK='1')	IF(CLK'event and CLK='0')
WAIT UNTIL CLK = '1'	WAIT UNTIL CLK = '0'
Rising_edge(CLK)	Falling_edge(CLK)

• Ej_3: Flip-Flop D activo en flanco de subida.



```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FlipFlopD IS
  PORT ( sD, CLK: IN std_logic;
         sQ: OUT std_logic);
END FlipFlopD;

--Arquitectura con eventos.
ARCHITECTURE FlipFlopDArch OF
FlipFlopD IS
BEGIN
  PROCESS (CLK) BEGIN
    IF (CLK'event and CLK = '1') THEN
      sQ <= sD;
    END IF;
  --Hubiera sido equivalente a usar

```

```

--la macro rising_edge poniendo:
--IF (rising_edge(CLK)) THEN
--sQ <= sD;
--END IF;
END PROCESS;
END FlipFlopDArch;

--Arquitectura con WAIT
--Método no recomendado.
ARCHITECTURE FlipFlopDArch2 OF
FlipFlopD IS
BEGIN
  PROCESS BEGIN
    WAIT UNTIL CLK = '1';
    sQ <= sD;
  END PROCESS;
END FlipFlopDArch2;

```


◆ Modelización VHDL de las señales de inicialización.

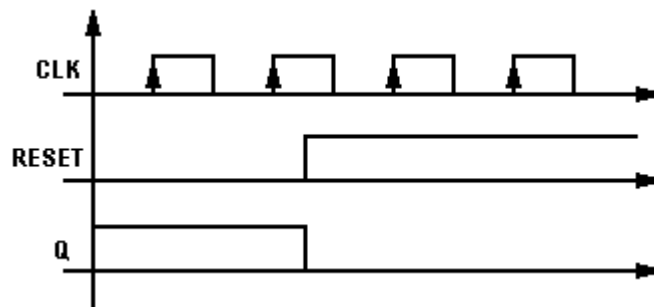
Cuando los biestables se alimentan, pueden adoptar un valor indeterminado "a priori", dependiendo de la forma en que lo haya implementado el fabricante.

Suele ser habitual el uso de señales de inicialización, que fuerzan la salida del biestable a un valor conocido. Típicamente podemos clasificar esta clase de señales en dos grupos:

- Señales de inicialización asíncrona.
 - * En cuanto se activan la salida del biestable pasa automáticamente a tomar el valor correspondiente.
 - * Pueden ser activas a nivel alto o bajo.

RESET \rightarrow **Q** = '0'

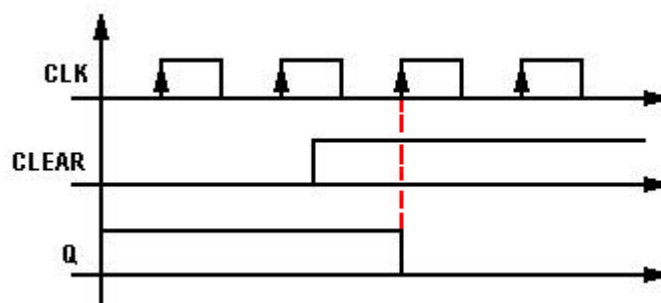
SET \rightarrow **Q** = '1'



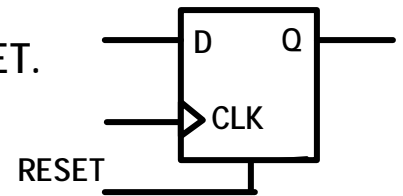
- Señales de inicialización síncrona.
 - * Cuando se activan la salida del biestable espera la señal de reloj para tomar el valor correspondiente.
 - * Pueden ser activas a nivel alto o bajo.

CLEAR \rightarrow **Q** = '0'

PRESET \rightarrow **Q** = '1'



- Ej_4: Flip-Flop D activo en flanco de subida con RESET.



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

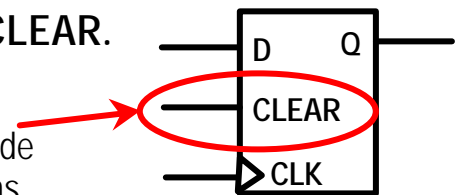
```
ENTITY FlipFlopD IS
  PORT ( sD, CLK: IN std_logic;
         sResetH: IN std_logic;
         sQ: OUT std_logic);
END FlipFlopD;
```

```
ARCHITECTURE FlipFlopDArch OF
  FlipFlopD IS
  BEGIN
```

```
  PROCESS (CLK, sResetH) BEGIN
    IF (sResetH = '1') THEN
      sQ <= '0';
    ELSIF (CLK'event and CLK = '1') THEN
      sQ <= sD;
    END IF;
  END PROCESS;
END FlipFlopDArch;
```

- Ej_5: Flip-Flop D activo en flanco de subida con CLEAR.

El CLEAR y PRESET se suelen colocar junto a las entradas a fin de indicar que son señales síncronas



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY FlipFlopD IS
  PORT ( sD, CLK: IN std_logic;
         sClearH: IN std_logic;
         sQ: OUT std_logic);
END FlipFlopD;
```

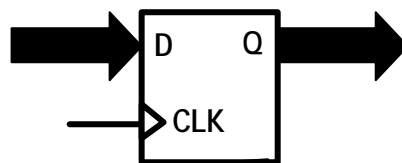
```
ARCHITECTURE FlipFlopDArch OF
  FlipFlopD IS
  BEGIN
```

```
  PROCESS (CLK, sClearH) BEGIN
    IF (CLK'event and CLK = '1') THEN
      IF (sClearH = '1') THEN sQ <= '0';
      ELSE sQ <= sD;
      END IF;
    END IF;
  END PROCESS;
END FlipFlopDArch;
```

7.3.2 REGISTROS DE ALMACENAMIENTO

Presentan una estructura similar a los flip-flops. La diferencia radica en que almacenan el estado de un vector de bits en lugar de un solo bit.

- Ej_6: Registro de almacenamiento de N bits.



```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

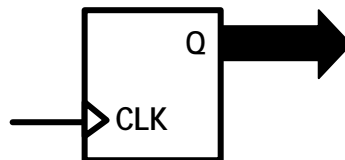
ENTITY Registro IS
  GENERIC(iNAncho: integer := 8);
  PORT ( svD: IN std_logic_vector(iNAncho-1 DOWNT0 0); --Vector Entrada.
         CLK: IN std_logic;          --Señal de reloj.
         svQ: OUT std_logic_vector(iNAncho-1 DOWNT0 0)); --Vector Salida
END Registro;

ARCHITECTURE RegistroArch OF Registro IS
BEGIN
  PROCESS(CLK)
  BEGIN
    -- Uso de la MACRO rising_edge
    IF (rising_edge(CLK)) THEN
      svQ <= svD;
    END IF;
  END PROCESS;
END RegistroArch;
  
```

7.3.3 CONTADORES

Se trata de dispositivos que en su salida presentan una cuenta que se actualiza mediante la señal de reloj.

- Ej_7: Contador de N bits.



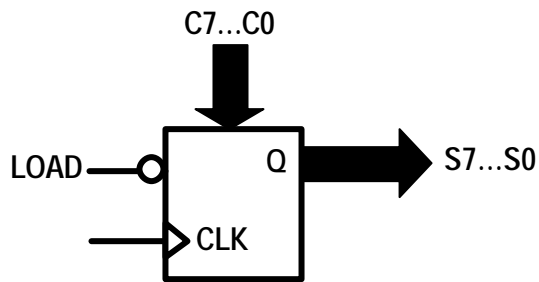
```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ContadorNb IS
  GENERIC(iAncho: integer := 8);
  PORT (CLK: IN std_logic;          --Señal de reloj.
        svQ: INOUT std_logic_vector(iAncho-1 DOWNT0 0)); --Vector Salida
END ContadorNb;

ARCHITECTURE ContadorNbArch OF ContadorNb IS
BEGIN
  PROCESS(CLK)
  BEGIN
    -- Uso de la MACRO rising_edge
    IF (rising_edge(CLK)) THEN
      svQ <= svQ + 1; -- Debido a esta realimentación se pone svQ como INOUT.
    END IF;
  END PROCESS;
END ContadorNbArch;
  
```

- Ej_8: Contador de N bits con RESET y carga paralelo síncrona (LOAD#).



Si
 LOAD = 0 Los biestables se cargan con los valores C7...C0 y se muestra en la salida.
 LOAD = 1. Se produce una cuenta ascendente sobre S7...S0.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ContadorLoad IS
  GENERIC(iAncho: integer := 8);
  PORT (CLK: IN std_logic;           --Señal de Reloj.
        sLoadL: IN std_logic;       -- Señal de Carga.
        sResetH: IN std_logic;      -- Señal de Reset.
        svC: IN std_logic_vector(iAncho-1 DOWNT0 0); --Vector de Carga
        svQ: INOUT std_logic_vector(iAncho-1 DOWNT0 0)); --Vector Salida
END ContadorLoad;

ARCHITECTURE ContadorLoadArch OF ContadorLoad IS
BEGIN
  PROCESS(CLK, sResetH, sLoadL)
  BEGIN
    -- Análisis de la posibilidad de RESET
    IF (sResetH = '1') THEN svQ <= (OTHERS => '0');
    -- Uso de la MACRO rising_edge
    ELSIF (rising_edge(CLK)) THEN
      IF (sLoadL = '0') THEN svQ <= svC;
      ELSE svQ <= svQ+1;
      END IF;
    END IF;
  END PROCESS;
END ContadorLoadArch;
  
```