TEMA 1
Introducción al VHDL y repaso de Fundamentos de E^{ca} Digital

VHDL. Conceptos básicos.

Circuitos combinacionales y VHDL concurrente

Circuitos secuenciales y VHDL secuencial

¿Qué es VHDL y para qué sirve?

VHDL: Very High Speed Integrated Circuits Hardware Description Language

Es un lenguaje de alto nivel para describir circuitos digitales

¿Para qué sirve un HDL?

- **Sintetizar circuitos: dada una descripción en código (por ej. VHDL), obtener una implementación (por ejemplo en una FPGA) con una herramienta de síntesis**
- **Simular circuitos: comprobar la funcionalidad antes de la implementación (usando un simulador y la descripción en VHDL).**
- **Descripción: documentación de proyectos a partir de la descripción de la funcionalidad**

IEEE std 1076-1987: IEEE Standard VHDL Language Reference Manual

Lo BÁSICO (la filosofía) del VHDL

- Dos elementos fundamentales: **ENTIDAD** y **ARQUITECTURA**
  - Entity → Las **CONEXIONES** (entradas y salidas del circuito)
  - Architecture → El **COMPORTAMIENTO** o la **ESTRUCTURA** del circuito
- Dentro de la arquitectura,

  TODAS LAS SENTENCIAS SON CONCURRENTES ENTRE SI

  (o sea, es un lenguaje paralelo, como el HW)
- Dado que siempre pensamos en los sistemas complejos como sistemas interconectados, esta estructura de programación es óptima. Permite **jerarquía** estructural
- … sin embargo, algunas cosas las pensamos como algoritmos (herencia de los informáticos y los matemáticos) → Elementos secuenciales dentro de lo concurrente → ¿Cómo va el tiempo?
- **NO** hay variables (de momento), sólo **SEÑALES**
- **NO** es un ejecutable **(se simula)**

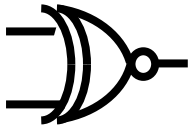
Las puertas lógicas, en VHDL

Puerta NAND

Tipo bit:
'0' ó '1'

```
entity puerta_NAND is
    Port ( A : in  bit;
           B : in  bit;
           Y : out bit
           );
end puerta_NAND;

architecture Behavioral of puerta_NAND
is

begin

 Y <= A nand B;

end Behavioral;
```

Puerta XNOR



```
entity puerta_XNOR is
    Port ( A : in  bit;
           B : in  bit;
           Y : out bit
           );
end puerta_XNOR;

architecture Behavioral of puerta_XNOR
is

begin

 Y <= ((not A) and (not B))or(A and B);

end Behavioral;
```

Aquí se está representando **comportamiento**, y no **estru**ctura
De momento, no hay paralelismo (sólo una sentencia)

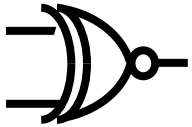
Funcionalidad mediante asignaciones condicionales

Puerta NAND

Otra forma

```
entity puerta_NAND is
    Port ( A : in  bit;
           B : in  bit;
           Y : out bit
         );
end puerta_NAND;

architecture Behavioral of puerta_NAND is

begin
 Y <= '1' when A= '0' and B = '0' else
      '1' when A ='0' and B = '1' else
      '1' when A ='1' and B = '0' else
      '0';

end Behavioral;
```

Puerta XNOR



```
entity puerta_XNOR is
    Port ( A : in  bit;
           B : in  bit;
           Y : out bit
         );
end puerta_XNOR;

architecture Behavioral of puerta_XNOR is

begin

 Y <= '1' when A= '0' and B = '0' else
      '0' when A ='0' and B = '1' else
      '0' when A ='1' and B = '0' else
      '1';

end Behavioral;
```

(Vaya rollo, cada vez escribo más….)

Ejemplo 1. Funciones lógicas

A	B	C	D	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

```
entity funcion is
    Port ( A : in  bit;
           B : in  bit;
           C : in  bit;
           D : in  bit;
           f : out bit
         );
end funcion;

architecture Behavioral of funcion is

Begin

 f <= ‘1’ when A='0'and B='0‘and C =‘0’ and D =‘0’ else
      ‘0’ when A='0'and B='0‘and C =‘0’ and D =‘1’ else
      ‘0’ when A='0'and B='0‘and C =‘1’ and D =‘0’ else
      ‘0’ when A='0'and B='0‘and C =‘1’ and D =‘1’ else
       …                    .
                           .
                           .
      '1' when A=‘1'and B=‘1’ and C =‘1’ and D =‘0’ else
      ‘1’;
end Behavioral;
```

Habrá otras formas para escribir menos (don’t panic).

Ejemplo 2. Valores a 'X' (don't care)

	e_3	e_2	e_1	e_0	f
X	0	0	0	0	X
E	0	0	0	1	1
F	0	0	1	0	0
M	0	0	1	1	1
A	0	1	0	0	0
M	0	1	0	1	1
J	0	1	1	0	0
J	0	1	1	1	1
A	1	0	0	0	1
S	1	0	0	1	0
O	1	0	1	0	1
N	1	0	1	1	0
D	1	1	0	0	1
X	1	1	0	1	X
X	1	1	1	0	X
X	1	1	1	1	X

¿Qué pasa con los valores X (*don't care*) ?

El tipo de datos 'bit' no cubre la realidad de la electrónica digital en todos sus aspectos → necesitamos nuevos tipos de datos

Tipo std_logic: '1', '0', 'Z', 'X', 'U', '-', 'L', 'H', 'W'

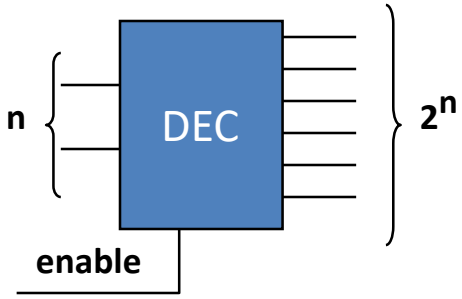
Z: alta impedancia	L: '0' débil
X: conflicto	H: '1' débil
U: undefined	'W': conflicto débil
- : don't care	

Ejemplo 2. Código en VHDL

```
entity funcion is
    Port ( A : in std_logic;
           B : in std_logic;
           C : in std_logic;
           D : in std_logic;
           f : out std_logic
         );
end funcion;

architecture Behavioral of funcion is

Begin

 f <= ‘-’ when A='0'and B='0‘and C =‘0’ and D =‘0’ else
      ‘1’ when A='0'and B='0‘and C =‘0’ and D =‘1’ else
      ‘0’ when A='0'and B='0‘and C =‘1’ and D =‘0’ else
      ‘1’ when A='0'and B='0‘and C =‘1’ and D =‘1’ else
                    .
       …                .
                       .
      ‘-';

end Behavioral;
```

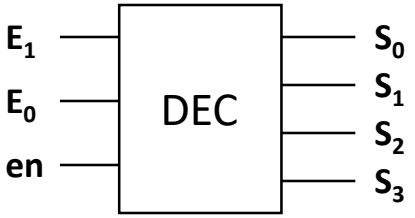
Decodificadores (repaso)

Decodificador



Se activa la salida correspondiente al número binario codificado en la entrada

***Ejemplo*:**
Decod 2 entradas con enable



en	E_1	E_0	S_0	S_1	S_2	S_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Funciones lógicas

$$S_0 = e_n \cdot \overline{E_1}\,\overline{E_0}$$
$$S_1 = e_n \cdot \overline{E_1} E_0$$
$$S_2 = e_n \cdot E_1 \overline{E_0}$$
$$S_3 = e_n \cdot E_1 E_0$$

Decodificador de 2 a 4 con enable en VHDL

```
entity DEC2a4 is
    port ( E   : in  std_logic_vector (1 downto 0);
           S   : out std_logic_vector (3 downto 0);
           ena : in  std_logic
         );
end DEC2a4;

architecture Behavioral of DEC2a4 is
   signal interna: std_logic_vector (3 downto 0);
Begin

  interna <= "0001" when E = "00" else
             "0010" when E = "01" else
             "0100" when E = "10" else
             "1000" when E = "11" else
             "----" when others;

  S <= interna when ena = '1' else "0000";

end Behavioral;
```

Un std_logic_vector representa un BUS (un conjunto de señales numeradas que se usan generalmente juntas)

Dentro de la arquitectura se pueden declarar señales internas

Estas dos asignaciones son concurrentes (como lo es el hardware), y por tanto da igual el orden)

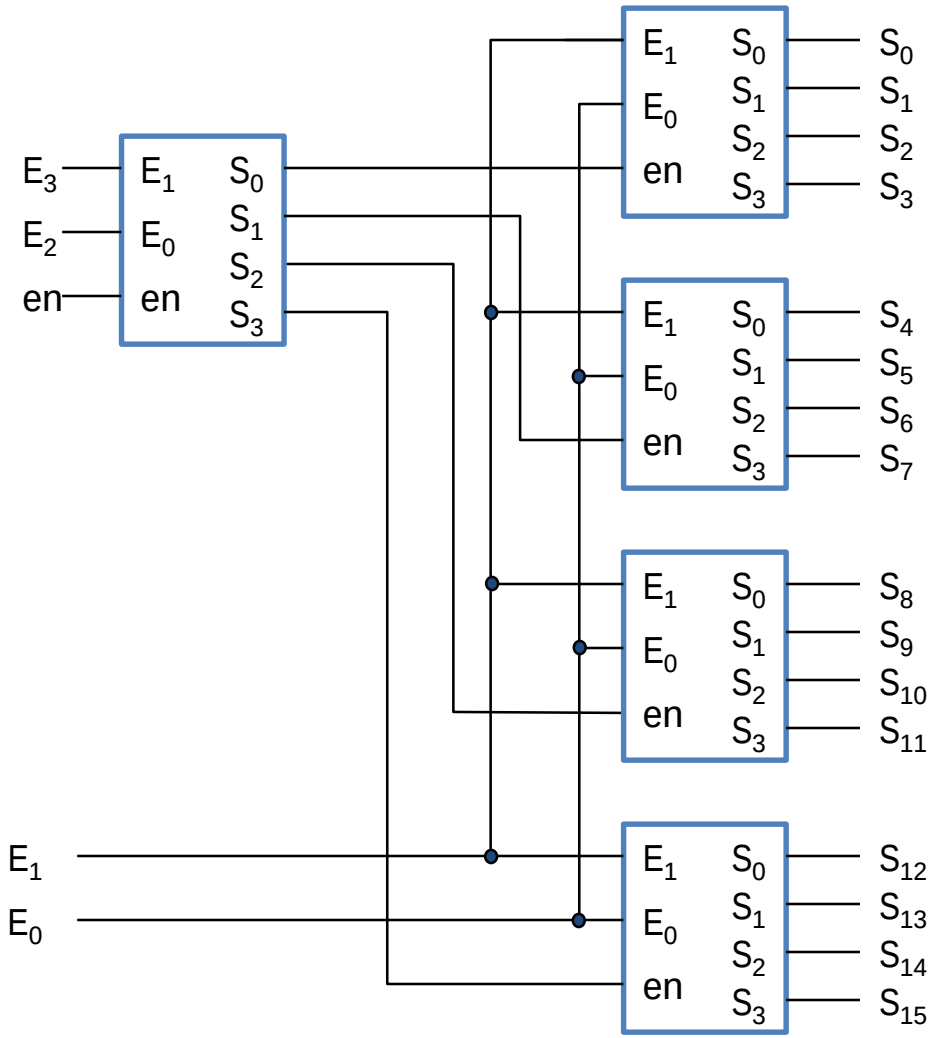
Alternativa

```
with E select
 S <= "0001" when "00",
      "0010" when "01",
      "0100" when "10",
      "1000" when "11",
      "----" when others;
```

Muchos casos se acortan como `'-' when others;`

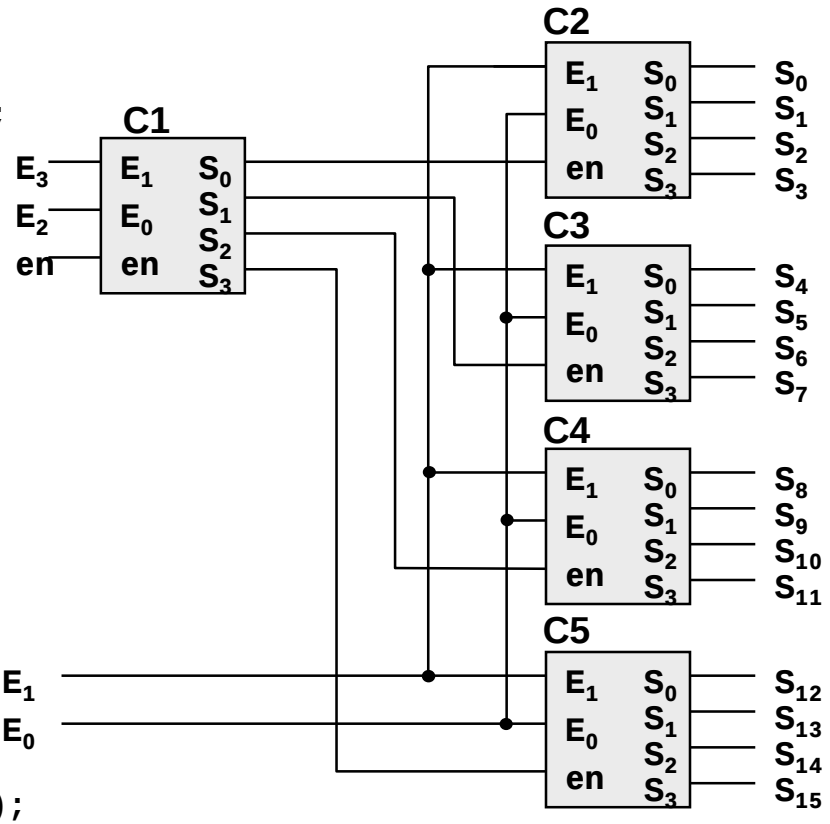
Decodificadores grandes (Repaso)

A partir de decodificadores de 2 entradas, construir un decodificador de 4 entradas

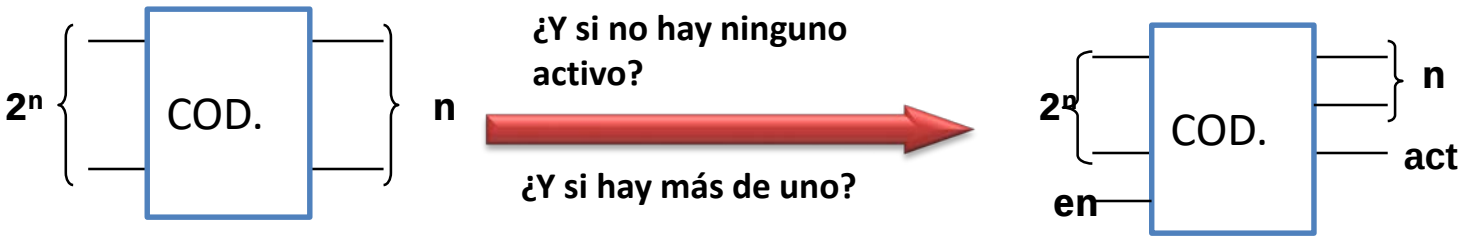


El decodificador anterior con VHDL estructural

```
entity DEC4a16 is
    Port ( E   : in  std_logic_vector (3 downto 0);
           S   : out std_logic_vector (15 downto 0);
           ena : in  std_logic
         )
end DEC4a16;

architecture Structural of DEC4a16 is

   component DEC2a4 is
    port ( E   : in  std_logic_vector (1 downto 0);
           S   : out std_logic_vector (3 downto 0);
           ena : in  std_logic
         );
   end component;

   signal enaint: std_logic_vector (3 downto 0);

Begin

   C1: DEC2a4 port map ( E(3 downto 2), enaint, ena);
   C2: DEC2a4 port map ( E(1 downto 0), S(3 downto 0,    enaint (0));
   C3: DEC2a4 port map ( E(1 downto 0), S(7 downto 4,    enaint (1));
   C4: DEC2a4 port map ( E(1 downto 0), S(11 downto 8,   enaint (2));
   C5: DEC2a4 port map ( E(1 downto 0), S(15 downto 12), enaint (3));

end Structural;
```



Codificadores (Repaso)



Se codifica en binario sobre la salida el número de entrada que esté activa

Codificador prioritario al más alto

en	E_0	E_1	E_2	E_3	A_1	A_0	act	
0	x	x	x	x	0	0	0	deshabilitado
1	0	0	0	0	0	0	0	inactivo
1	x	x	x	1	1	1	1	activo
1	x	x	1	0	1	0	1	
1	x	1	0	0	0	1	1	
1	1	0	0	0	0	0	1	

También existe el codificador prioritario al más bajo

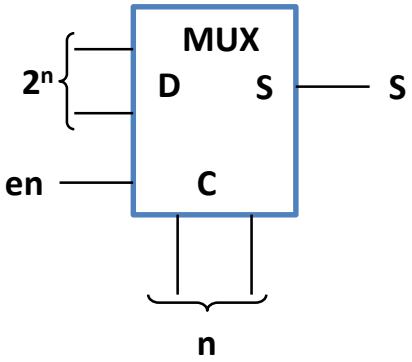
Codificador en VHDL

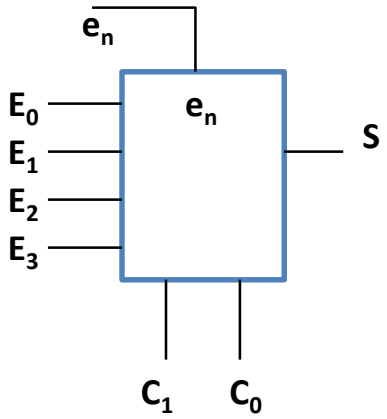
```
entity COD4a2 is
    port ( E   : in  std_logic_vector (3 downto 0);
           S   : out std_logic_vector (1 downto 0);
           ena : in  std_logic;
           act : out std_logic;
         );
end CUD4a2;

architecture Behavioral of COD4a2 is
   signal interna : std_logic_vector (1 downto 0);
Begin

  interna <= "11" when E(3) = '1' else
             "10" when E(2) = '1' else
             "01" when E(1) = '1' else
             "00";

  S <= interna when ena = '1' else "00";

  act <= '1' when ena = '1' and E /= "0000" else '0';

end Behavioral;
```

La prioridad va marcada por el orden de interpretación de los else

Multiplexores (Repaso)

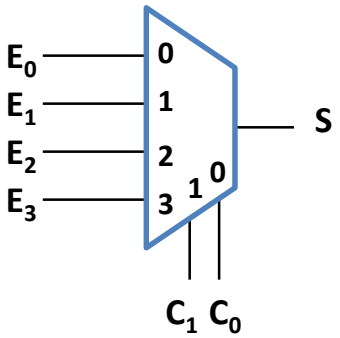
Multiplexor (MUX)





La entrada de datos correspondiente al número codificado en binario en las señales de control se conecta a la salida

Símbolo propio:

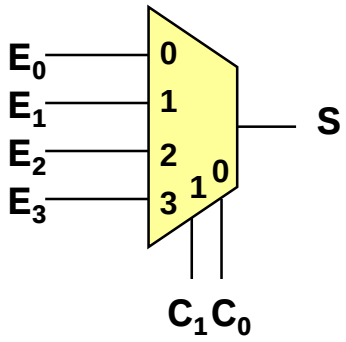


en	E_0	E_1	E_2	E_3	C_1	C_0	S
0	X	X	X	X	X	X	0
1	D	X	X	X	0	0	D
1	X	D	X	X	0	1	D
1	X	X	D	X	1	0	D
1	X	X	X	D	1	1	D

Ejemplo de MUX en VHDL

```
entity MUX4a1 is
    port ( E   : in  std_logic_vector (3 downto 0);
           C   : in  std_logic_vector (1 downto 0);
           S   : out std_logic;
           ena : in  std_logic
         );
end MUX4a1 ;

architecture Behavioral of MUX4a1 is
   signal interna : std_logic;

begin
  with C select
    interna <= E(0) when “00”,
               E(1) when “01”,
               E(2) when “10”,
               E(3) when “11”,
               ‘-’ when others;

    S <= interna when ena = ‘1’ else ‘0’;

end Behavioral;
```

MUX de 4 a 1

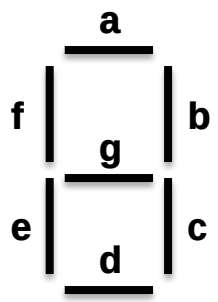


Con with … select se acorta la escritura de funciones lógicas

Convertidores de código (Repaso y VHDL)

Convertidor BCD a 7 segmentos

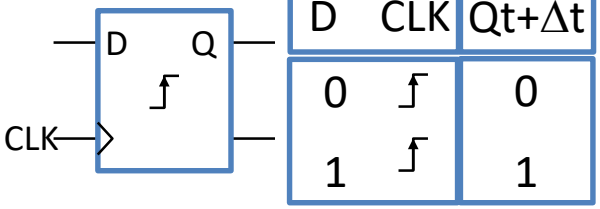
E_3	E_2	E_1	E_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
resto				x	x	x	x	x	x	x



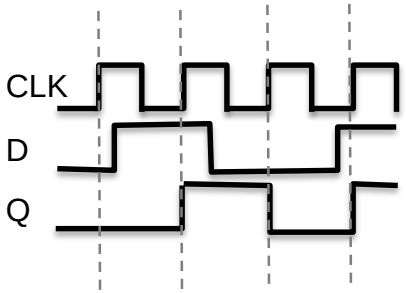

```
with BCD_IN select
  SSEG <= "0000001" when "0000", -- 0
          "1001111" when "0001", -- 1
          "0010010" when "0010", -- 2
          "0000110" when "0011", -- 3
          "1001100" when "0100", -- 4
          "0100100" when "0101", -- 5
          "0100000" when "0110", -- 6
          "0001111" when "0111", -- 7
          "0000000" when "1000", -- 8
          "0000100" when "1001", -- 9
          "-------" when others;
```

Circuitos secuenciales. Biestables síncronos

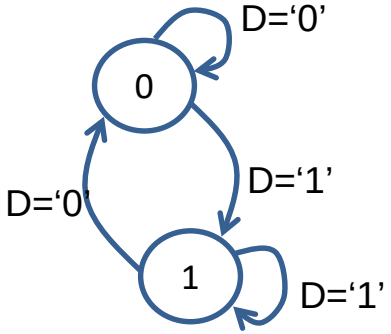
Biestable D (*dato*)



D	CLK	Qt+Δt
0	↑	0
1	↑	1

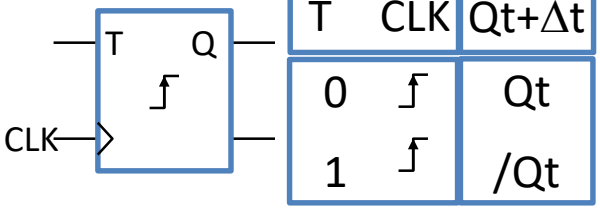


La entrada D se muestrea en el flanco activo de CLK
Q sólo puede cambiar en ese flanco

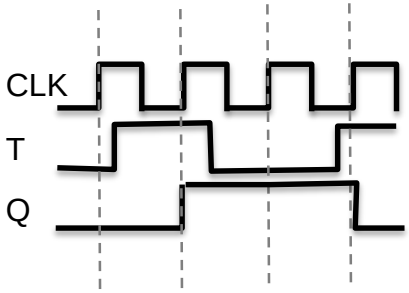


Puede ser activo por flanco de subida ↑ o de bajada

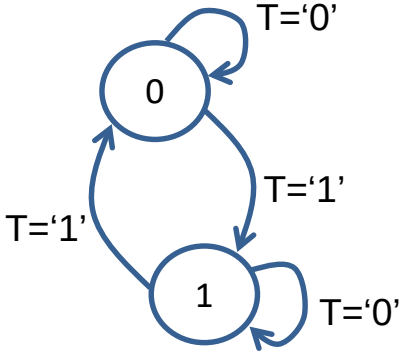
Biestable T (*toggle*)



T	CLK	Qt+Δt
0	↑	Qt
1	↑	/Qt



La entrada T se muestrea en el flanco activo de CLK
Q sólo puede cambiar en ese flanco



Puede ser activo por flanco de subida ↑ o de bajada

Otras señales de los biestables

Inicialización de biestables

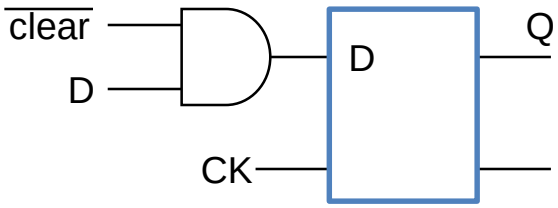
	0	1
ASÍNCRONOS	reset	preset
SÍNCRONOS	clear	set

No hay acuerdo en el uso de esta terminología

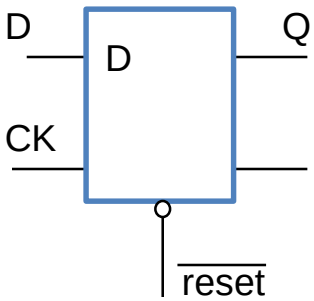
La inic. asíncrona es inmediata, mientras que la síncrona se espera al primer flanco activo de reloj

Las señales de inicialización suelen ser activas por nivel bajo (la acción se produce cuando la señal es 0)

Ejemplo: Biestable D con clear



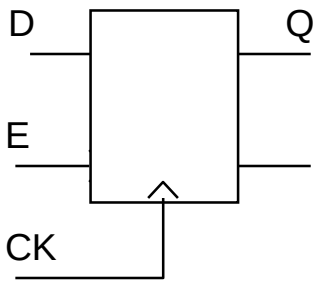
Ejemplo: Biestable D con reset

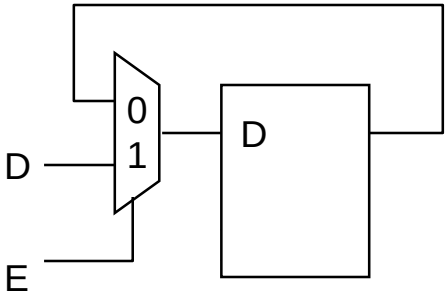


La inic. síncrona se puede considerar como parte de la funcionalidad

Otras señales de los biestables

Señal de enable (carga) en biestables D





E	D	$Q_{t+\Delta t}$
0	X	Qt
1	D	D

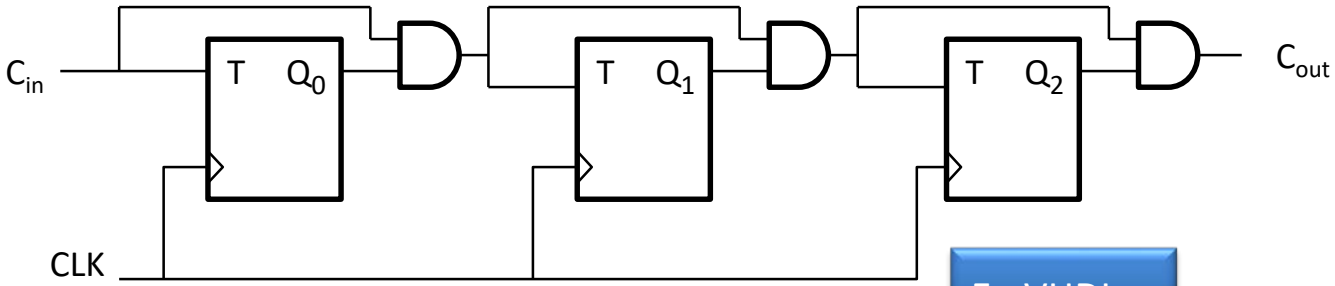
No confundir este enable con el combinacional

Ejemplo: Describir un biestable T con clear, reset, carga y enable

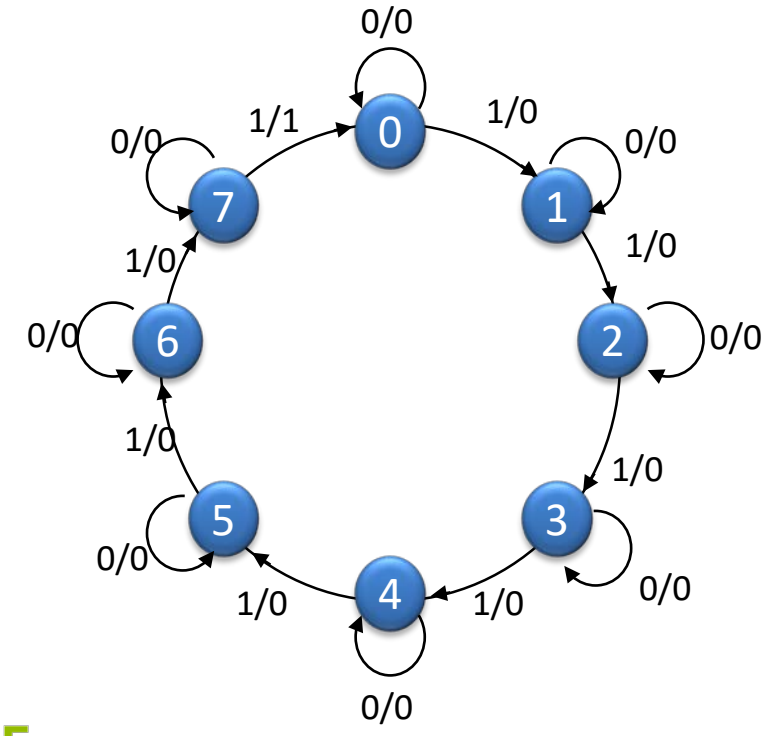
Orden: reset > clear > enable > carga > T

Contadores

Ejemplo: Contador síncrono de 3 bits

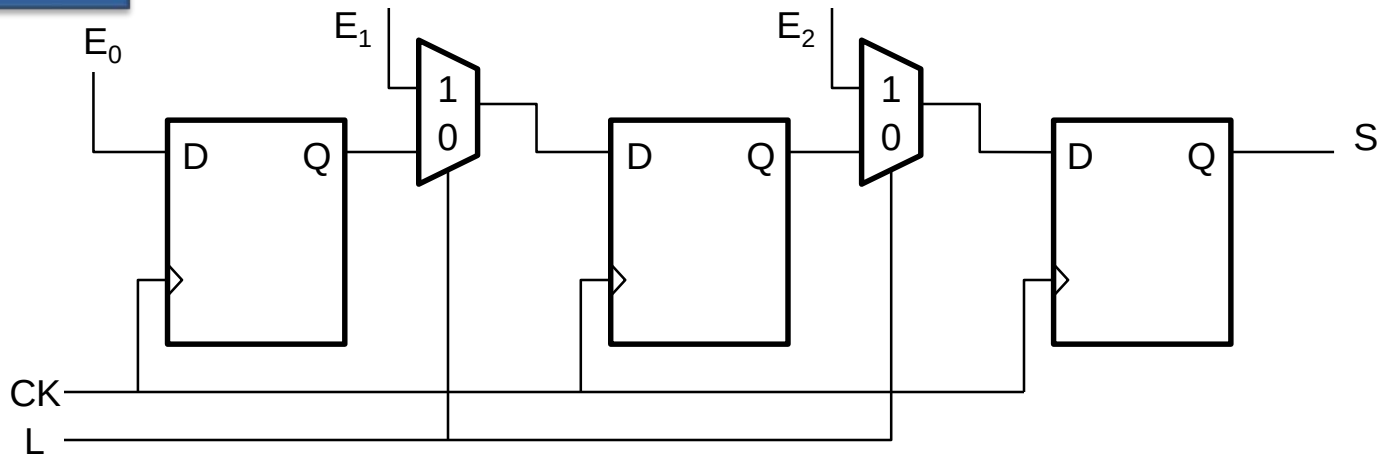


En VHDL:



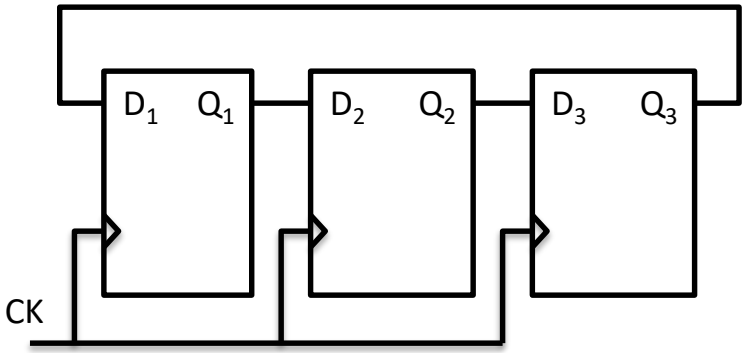
Registros

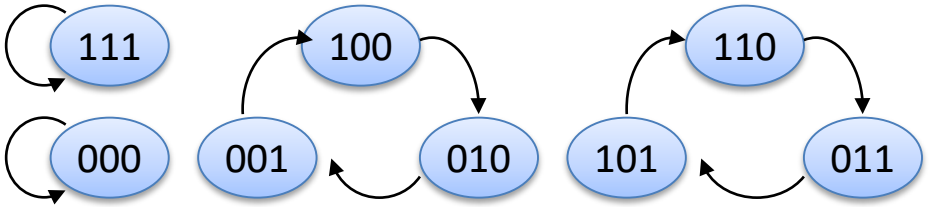
Registro PISO



Contadores con registros de desplazamiento

Contador en anillo





Contador Johnson