CAPÍTULO 4
DISEÑO DE SISTEMAS DIGITALES

Programa resumido:

- Diseño intuitivo de los Sistemas Digitales — 6 horas.
- Circuitos Secuenciales Sincrónicos — 20 horas.
- Memorias RAM — 2 horas.
- Diseño formal de los Sistemas Digitales — 20 horas.
- Circuitos Secuenciales Asincrónicos — 8 horas.

INTRODUCCION

Las **MSS** estudiadas y diseñadas hasta ahora pueden ser usadas como subcircuitos o bloques de los circuitos digitales más complejos llamados **Sistemas Digitale**s.

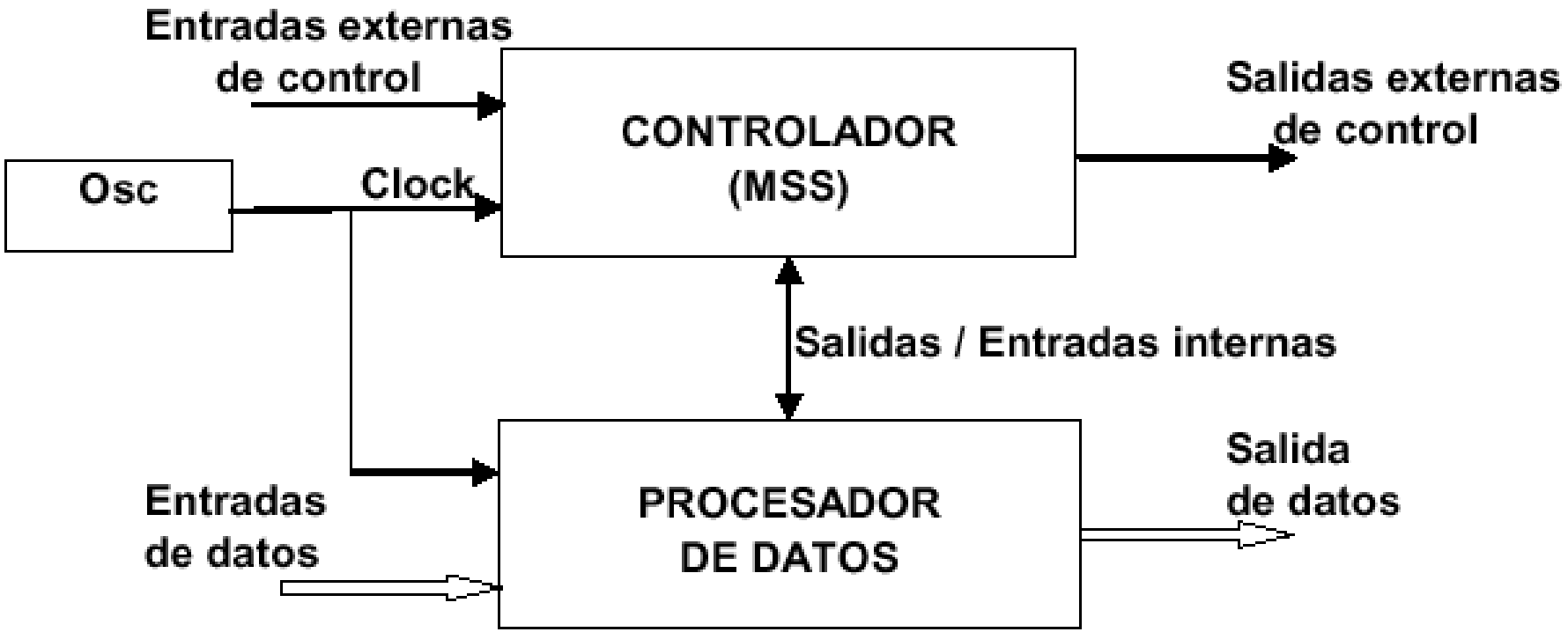
Las técnicas del diseño que estudiaremos a partir de ahora son aplicables al diseño de **Sistemas Digitales** de cualquier tamaño.

Un **Sistema Digital** puede ser dividido en dos partes principales llamados **Procesador de Datos** y circuito **Controlado**r.

El Procesador de Datos es usado para almacenar y manipular datos y para transferir datos de una parte del **Sistema Digital** al otro. El **procesador de Datos** incluye bloques como registros de almacenamiento, registros de desplazamiento, contadores, multiplexores, decodificadores, sumadores, etc.

El circuito **Controlador** es una **MSS** que controla la operación de **Procesador de Datos**.

Diagrama de bloques general de un Sistema Digital.

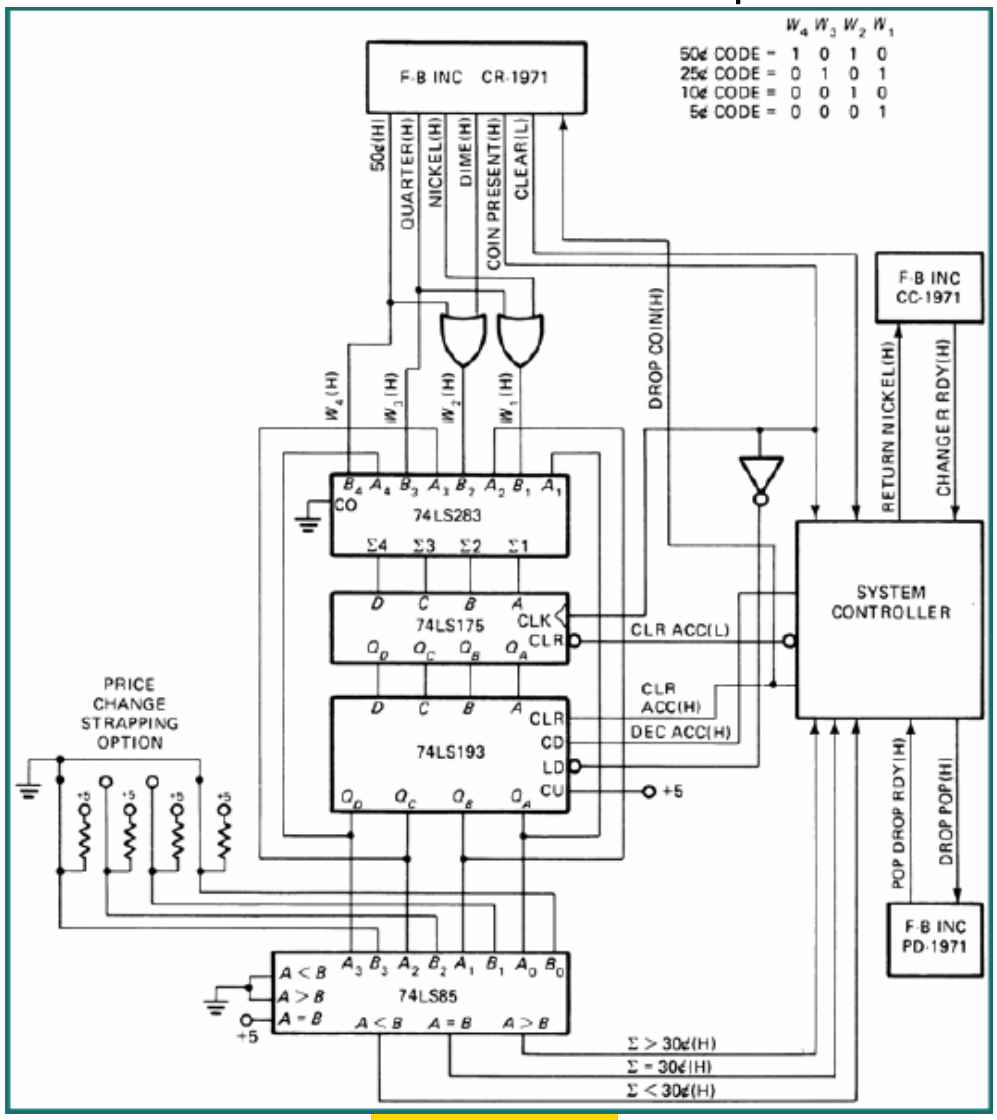


Procedimiento general para el diseño formal de un Sistema Digital.

- **Entender exactamente lo que debe hacer el Sistema Digital.** Esto incluye examinar detenidamente las especificaciones para obtener el cuadro total de relaciones del Sistema Digital con el mundo externo. Definir todas las entradas y salidas externas que debe tener el Sistema.
- **Desarrollar una Partición Funcional del Sistema Digital.** Una Partición funcional es una ilustración detallada de todos los subcircuitos y subsistemas que debe tener el Procesador de Datos del sistema además del circuito Controlador. El circuito Controlador es el único subcircuito de la Partición Funcional que no está detallado.
- **Definir el algoritmo de control del circuito Controlador.** Se debe desarrollar un Diagrama de Estados o Diagrama ASM. En realidad, la Partición Funcional y el algoritmo de control deben desarrollarse simultáneamente. Se requiere mucha intuición, creatividad y conocimientos para decidir cuales funciones debe realizar el Controlador y cuales deben realizar los subcircuitos o subsistemas del Procesador de Datos.
- **Diseñar el circuito Controlador** seleccionando el método y la arquitectura deseada.

Ejemplo de una Partición Funcional.

Partición Funcional del Sistema de Control de una maquina vendedora de colas.



Para la implementación usando dispositivos lógicos programables (PLD) se debe:

- Ingresar el Sistema Digital en un software de diseño y simulación (por ejemplo, Max+plusII de Altera) utilizando cualquier de los editores (Editor Grafico, Editor de Formas de Onda, Editor de texto / VHDL).
- Compilar el diseño, simular y revisar los resultados de simulación analizando los Diagramas de Tiempo generados por el simulador.

COMENTARIOS IMPORTANTES:

- El circuito Controlador de un Sistema Digital maneja solamente señales de control de un bit. Recibe las señales de entrada de un bit que pueden ser externas (Start, Inicio, etc.) o internas provenientes de algunos subcircuitos Procesadores de Datos (AmenorB, LSBigual0, etc.).

También, genera señales de salida de un bit que pueden ser externas (Fin, Error,etc.) o internas que van a controlar las operaciones de subcircuitos que forman parte de Procesador de Datos.

- Las señales de entrada de datos de múltiples bits no pueden nunca entrar al circuito Controlador. Deben entrar en algún subcircuito del Procesador de Datos. También, las señales de salida de múltiples bits deben ser generadas por algún circuito Procesador de Datos. Pueden existir casos en los que las señales externas de control de un bit, también entran o salen del Procesador de Datos. Pero nunca las señales de múltiples bits pueden entrar o salir del Controlador.

Estructura general del código VHDL en el diseño de un Sistema Digital.

Luego de estructurar la **Partición Funcional** se debe decidir que tipo de descripción se desea usar para cada subcircuito del Sistema Digital y para el Sistema completo.

Un subcircuito o subsistema puede ser representado con un solo componente (registro, multiplexor, comparador) o con varios componentes como, por ejemplo, el circuito **Acumulador** (sumador + registro de sostenimiento).

Cada subcircuito en un diseño del **Sistema Digital** con **VHDL** representa una ***entity***.
En caso que la ***entity*** represente un **Sistema Digital** completo, sus terminales de entrada y salida le permiten interactuar con el mundo exterior.

La declaración de ***entity*** puede incluir también la keyword ***generi***c. Mediante **VHDL,** la instrucción ***generic*** puede declarar y especificar parámetros constantes útiles para crear bloques parametrizados, en los que los valores de algunos parámetros pueden ser variados.

Ejemplo:

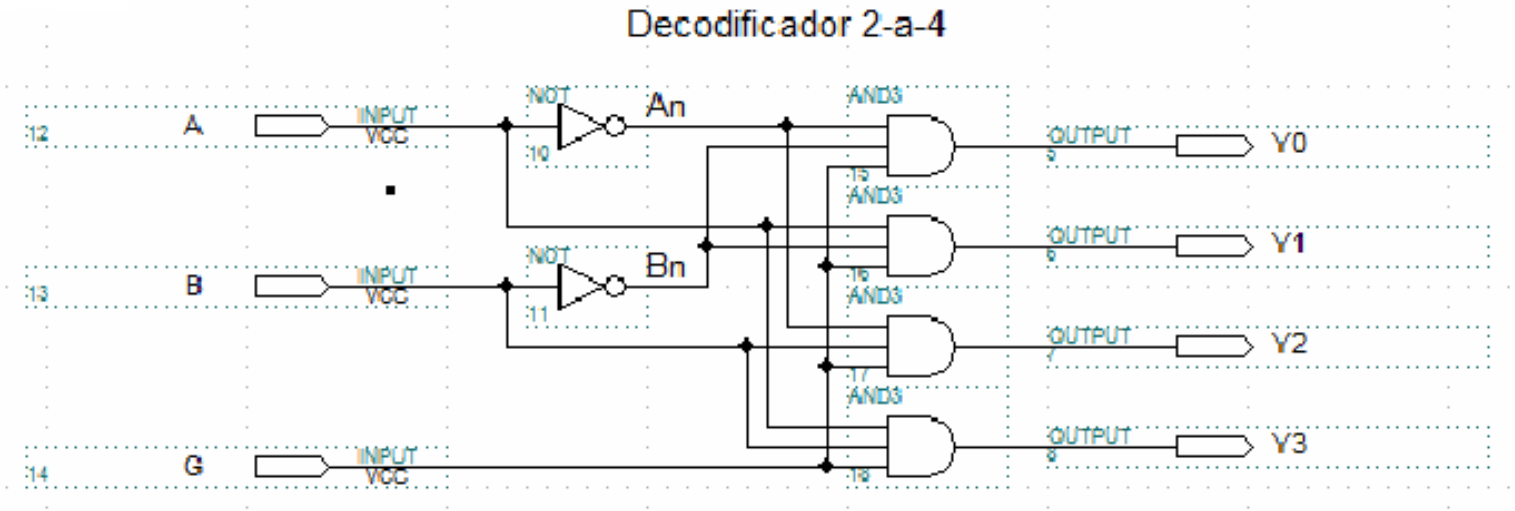
```
library ieee;
use ieee.std_logic_1164.all;

entity reg is
    generic (n: integer := 4);
    port (R              : in std_logic_vector (n-1 downto 0);
          Rin, Clock     : in std_logic;
          Q              : out std_logic_vector (n-1 downto 0));
```

La declaración de entidad solo proporciona la información necesaria para conectar el subcircuito representado por esta entity con otros circuitos.

La declaración architecture describe el funcionamiento interno de una entity. VHDL permite crear más de una alternativa para la architecture.

VHDL soporta tres tipos diferentes de descripción de architecture (o niveles de abstracción) de los Sistemas Digitales:

- **Descripción estructural** (nivel lógico de abstracción) en la que se especifican los componentes que forman el sistema y sus interconexiones.



Para describir en VHDL el funcionamiento de este circuito utilizando la descripción estructural hay que definir los componentes que forman parte del circuito (si es que no están disponibles en la biblioteca library).

```
entity inversor is
     port (e : in bit;
           s : out bit);
end inversor;
architecture fl_de_dat of inversor is
begin
     s <= not (e);
end fl_de_dat;
```

```
entity p_and is
     port (e0, e1, e2: in bit;
                  s   : out bit);
end p_and;
architecture fl_de_dat of p_and is
begin
     s <= e0 and e1 and e2;
end fl_de_dat;
```

Luego se debe hacer la descripción lógica del circuito declarando previamente las señales (***signals)*** internas de interconexión **An** y **Bn** y los componentes a utilizar (inversor y puerta AND).

Tanto las señales internas como los componentes deben ser declarados en el cuerpo de ***architecture*** antes del ***begin*** de la descripción.

```
entity decoder_2_a_4 is
     port (G, B, A     : in bit;
     Y3, Y2, Y1, Y0    : out bit);
end decoder_2_a_4;

architecture estructural of decoder_2_a_4 is
     signal Bn, An    : bit;

     component inversor is
          port (e : in bit; s : out bit);
     end component;

     component puerta_and is
          port (e0, e1, e2: in bit; s : out bit);
     end component;
```

component tiene una estructura similar a la declaración ***entit***y, pero mientras la ***entity*** proporciona la interfaz del circuito con el mundo exterior, la declaración ***component*** proporciona la interfaz entre los componentes del circuito.

Cada ***component*** se conecta con los demás con ayuda de la **keyword "*port map*".**

Las señales en la declaración ***port map*** deben aparecer en el mismo orden como ellas fueron declaradas en ***componen*t.**

```
begin
inversor0: inversor port map (A, An);
inversor1: inversor port map (B, Bn);

puerta_and0: puerta_and port map (An, Bn, G, Y0);
puerta_and1: puerta_and port map (A, Bn, G, Y1);
puerta_and2: puerta_and port map (An, B, G, Y2);
puerta_and3: puerta_and port map (A, B, G, Y3);
end estructural;
```

Se puede ver claramente que la descripción estructural es una descripción del diagrama esquemático del circuito y que es muy extensa, incluso para los circuitos relativamente pequeños.

Sin embargo, la descripción estructural se utiliza para la descripción de los diseños más grandes.

Descripción a nivel de transferencia entre registros o flujo de datos (nivel RTL de abstracción) en la que se especifica el comportamiento de las señales de salida a partir de las señales de entrada.

Esta descripción tiene un cierto grado de abstracción con respecto al hardware, pero es necesario describir las distintas señales que interactúan en el circuito y su comportamiento en función de las señales de entradas por medio de ecuaciones lógicas y declaraciones de asignación.

Nota: el operador <= indica asignación a señal.

Se puede ver que en esta descripción se debe hacer la declaración de todas las señales internas y se debe utilizar los operadores lógicos.

Los operadores lógicos que pueden ser usados en VHDL son definidos en la biblioteca (library) estándar del lenguaje y son los siguientes: not, and, or, nand, nor, xor y xnor.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_2_a_4_rtl is
    port (G, B, A   : in bit;
    Y3, Y2, Y1, Y0  : out bit);
end decoder_2_a_4_rtl;

architecture flujo_de_datos of decoder_2_a_4_rtl is
    signal An, Bn   : bit;
begin
    An <= not (A);
    Bn <= not (B);
    Y0 <= An and Bn and G;
    Y1 <= A and Bn and G;
    Y2 <= An and B and G;
    Y3 <= A and B and G;
end flujo_de_datos;
```

Al igual que otros lenguajes de programación, **VHDL**, utiliza diferentes operadores que pueden ser utilizados con cualquier tipo de señal (bit, bit_vector, std_logic, std_logic_vector).

Operadores	Definidos en el lenguaje VHDL para los tipos:
Lógicos: AND, OR, XOR, NOT, NAND, NOR y XNOR	Bit y Boolean
De relación: = , /= , < , > , >= , <=	Integer, Bit y Bit_vector
Aritméticos: + , - , * (multiplicación sólo por 2)	Integer
Concatenación: &	Bit, Bit_vector y para las cadenas

Por lo tanto, no se puede usar estos operadores como los nombres de las señales de entrada o salida de los sistemas digitales diseñados.

Por ejemplo, para indicar las salidas del comparador no se puede usar A>B, A<B y A=B. Se puede asignar AmayorB o AgtB, etc.

Se puede también utilizar declaraciones concurrentes condicionales when – else.

```
architecture flujo_de_datos of decoder_2_a_4_rtl_a is
begin
    Y0 <= '1' when (A='0' and B='0' and G='1') else '0';
    Y1 <= '1' when (A='1' and B='0' and G='1') else '0';
    Y2 <= '1' when (A='0' and B='1' and G='1') else '0';
    Y3 <= '1' when (A='1' and B='1' and G='1') else '0';
    end flujo_de_datos;
```

Se puede también utilizar asignación selectiva with – select – when.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_2_a_4_rtl_b is
    port (G, B, A   : in bit;
                Y   : out bit_vector (3 downto 0));
end decoder_2_a_4_rtl_b;
architecture flujo_de_datos of decoder_2_a_4_rtl_b is
    signal GBA  : bit_vector (2 downto 0);
begin
    GBA <= G & B & A;
    with GBA select
        Y <= "0001" when "100",
             "0010" when "101",
             "0100" when "110",
             "1000" when "111",
             "0000" when others;
end flujo_de_datos;
```

En la declaración with – select – when se evalúa la expresión que acompaña a la declaración with. Cuando el valor de esta expresión coincide con una de las alternativas dadas después de “when”, el valor correspondiente a esta alternativa se asigna a la señal de salida.

La expresión que debe evaluarse en este ejemplo es GBA que se declara como una señal de tres bits. Mediante la concatenación (&) de las tres entradas (G, A y B) se forma una string de tres bits.

Las alternativas dadas después de when pueden ser varias o una sola.

Por ejemplo, en vez de escribir “0000” when others podemos escribir lo siguiente: “0000” when “000” | ”001” | ”010” | ”011”;

La barra “ | ” tiene el mismo significado que el operador lógico or.

Descripción funcional o por comportamiento (nivel algorítmico de abstracción) en la que se especifica el funcionamiento del sistema.

Esta es la descripción con mayor grado de abstracción. Solo se describe el comportamiento del sistema, sin preocuparse por las señales o componentes internos.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_2_a_4_behavior is
    port (G, B, A   : in bit;
                Y   : out bit_vector (0 to 3));
end decoder_2_a_4_behavior;

architecture comportamiento of decoder_2_a_4_behavior is
begin
    process (A, B, G)
    begin
        if G='0' then
            Y <= "0000";
        elsif A='0' and B='0' then
            Y <= "1000";
        elsif A='1' and B='0' then
            Y <= "0100";
        elsif A='0' and B='1' then
            Y <= "0010";
        elsif A='1' and B='1' then
            Y <= "0001";
        end if;
    end process;
end comportamiento;
```


01101010010010100110000101101110

Dentro de la arquitectura (architecture) aparece un proceso (process) que es la declaración que representa un conjunto de declaraciones que se ejecutan en secuencia.

La declaración process puede ser usada tanto en la descripción de circuitos combinatoriales como secuenciales.

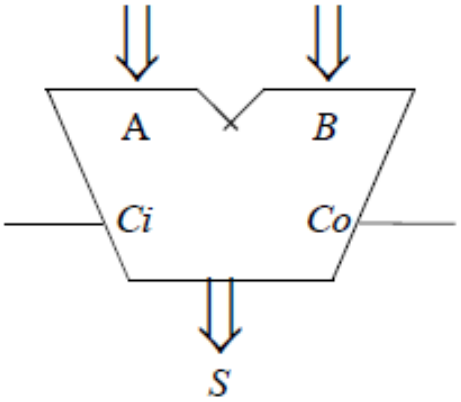
Los tres tipos de descripción de un Sistema Digital en VHDL pueden mezclarse en un mismo diseño.

Si ya tenemos estructurada la Partición Funcional, se puede adoptar un modelo de diseño top-down (de arriba hacia abajo).

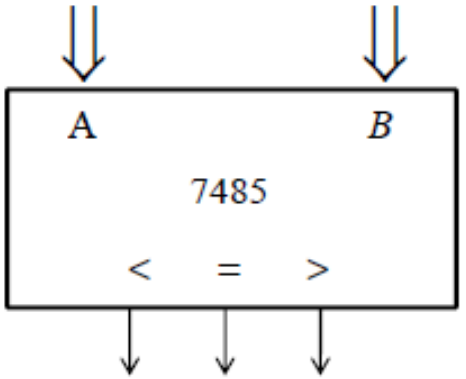
El Diseño top-down comprende la construcción de cada subcircuito o subsistema usando la descripción por comportamiento o por flujo de datos y luego, conectándolos en un Sistema Digital utilizando la descripción estructural.

COMPONENTES MAS IMPORTANTE S DE LA PARTICION FUNCIONAL

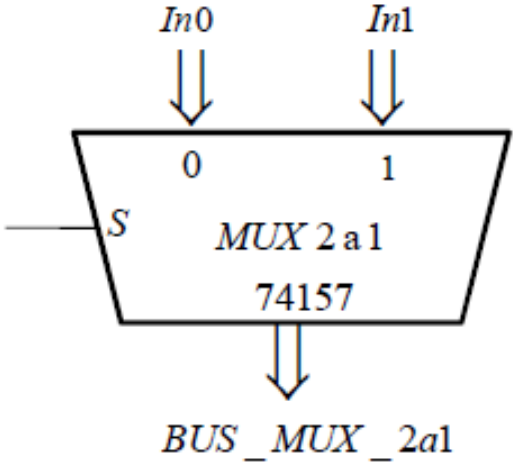
Sumadores

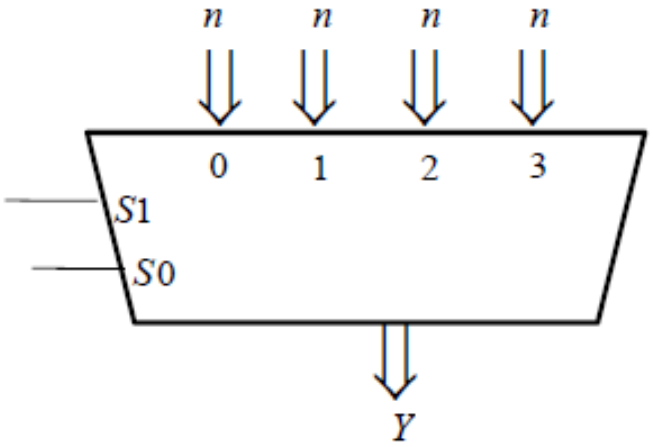


Comparadores

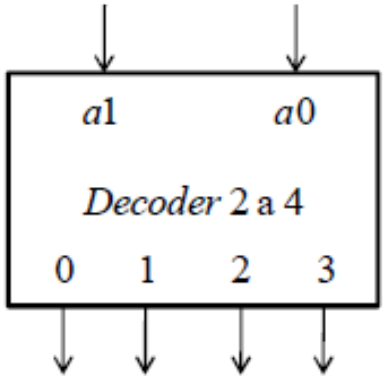


MUX

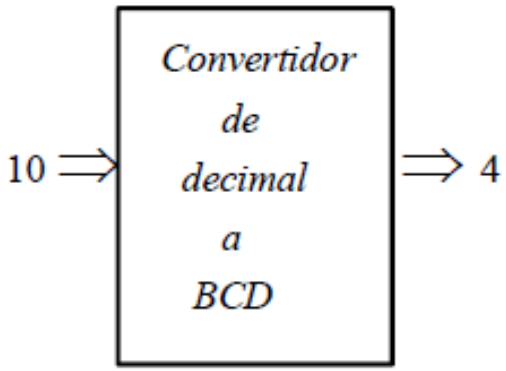




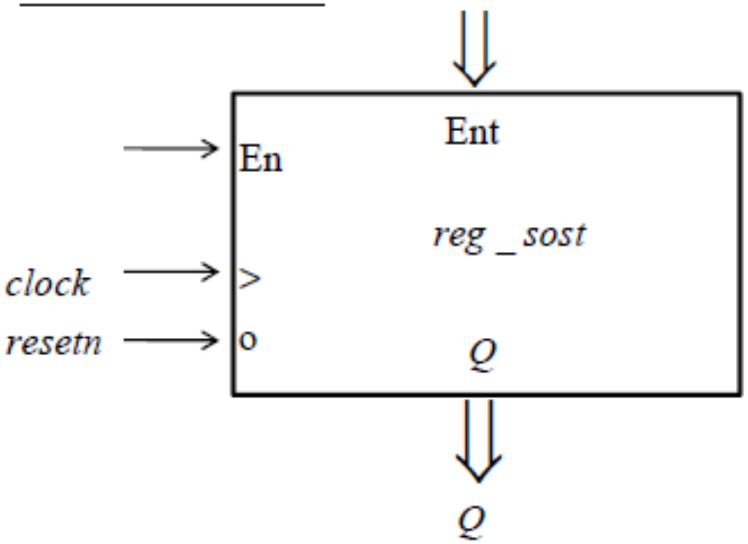
Decoder



Conv. Codigo

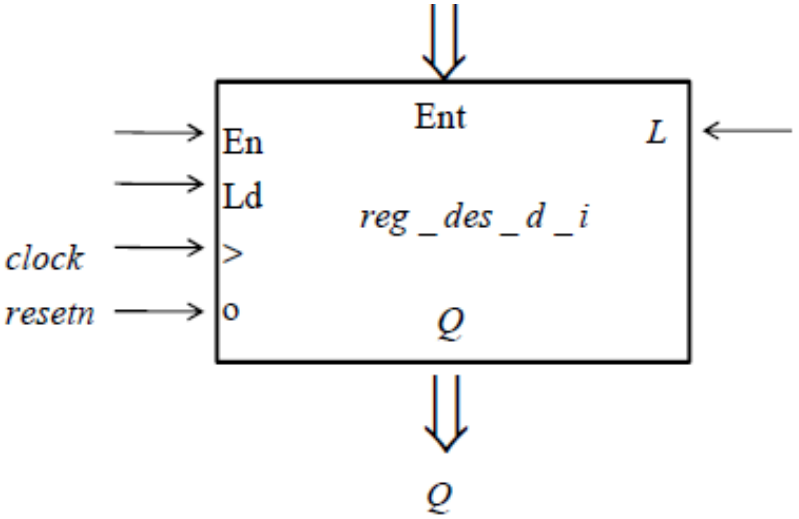


Reg. Sostenimiento



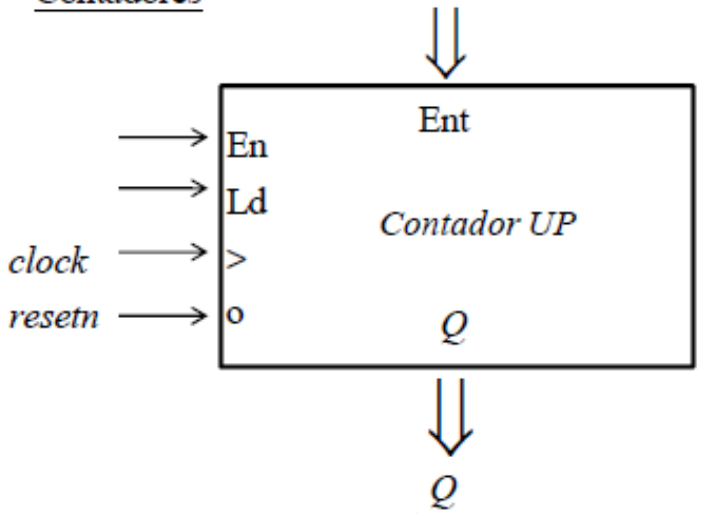
Si Resetn = '0' Q = 000...0
Si clock ≠ Q = Qn (Hold)
Si clokk =
Si En = 1 ⇒ Q = Ent
Si En = 0 ⇒ Q = Qn (Hold)

Reg. Desplazamiento



Si Resetn = '0' Q = 000...0
Si clock ≠ Q = Qn (Hold)
Si clokk =
Si En = 0 ⇒ Q = Qn (Hold)
Si En = 1
Si Ld = 1 ⇒ Q = Ent
Si Ld = 0 ⇒ reg_des_d_i

Contadores



Si Resetn = '0' Q = 000...0
Si clock ≠ Q = Qn (Hold)
Si clokk =
Si En = 0 ⇒ Q = Qn (Hold)
Si En = 1
Si Ld = 1 ⇒ Q = Ent
Si Ld = 0 ⇒ cuenta hacia arriba