



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)



PROGRAMACIÓN EN VHDL

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

VHDL

Organización y arquitectura

VHDL(Hardware Description Language) es un lenguaje orientado a la descripción o modelado de sistemas digitales ya sea para baja capacidad como un GAL o de mayor capacidad como los CPLD y FPGA.

Existen 5 tipos de unidades de diseño en VHDL:

- Declaración de entidades (entity declaration)
- Arquitectura (architecture)
- Configuración (configuration)
- Declaración del paquete (package declaration)
- Cuerpo del paquete (package body).

} Indispensables en la estructuración del programa

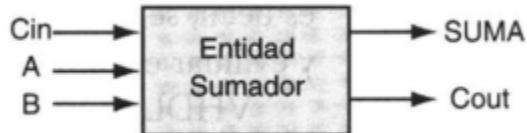
La declaración de entidad, paquete y configuración se consideran **unidades de diseño primarias**.

La arquitectura y cuerpo del paquete son **unidades de diseño secundarias**.

Explicación: Las unidades de diseño secundarias dependen de una entidad primaria que se debe analizar antes que ellas.

Entidad

Una entidad (entity) es el bloque elemental en diseño en VHDL, las entidades son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.)



Cada una de las señales de entrada y salida en una entidad son referidas como puerto, el cual es similar a una terminal 8pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un nombre, un modo y un tipo de dato.

Nombre: Forma de llamar al puerto

Modo: Permite definir la dirección que tomará la información

Tipo: define qué clase de información se transmitirá por el puerto.

Modos

Un modo puede tener uno de cuatro valores:

- In (entrada): es unidireccional y permite el flujo de datos hacia dentro de la entidad

- Out (salida): indica señales de salida de la entidad
- Inout (entrada/salida): permite declarar un puerto de forma bidireccional, permite la retroalimentación de señales dentro o fuera de la entidad.
- Buffer : permite hacer retroalimentaciones internas dentro de la entidad, el puerto declarado se comporta como una terminal de salida.

Tipos de datos

Los tipos son los valores (datos) que el diseñador establece para los puertos de entrada y salida dentro de una entidad; se asignan de acuerdo con las características de un diseño en particular.

- Bit: valor 0 y 1.
- Boolean: valores verdadero y falso en expresión
- Bit_vector: representa un conjunto de bits para cada variable de entrada o salida.
- Integer: enteros
- STD_LOGIC: standard logic
- STD_LOGIC_VECTOR: standard logic vector

Standard Logic

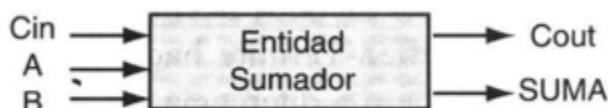
'0' (Forcing 0): Cero lógico

'1' (Forcing 1): Uno lógico

'Z' (Alta impedancia)

'-' (Don't care)

Declaración de entidades



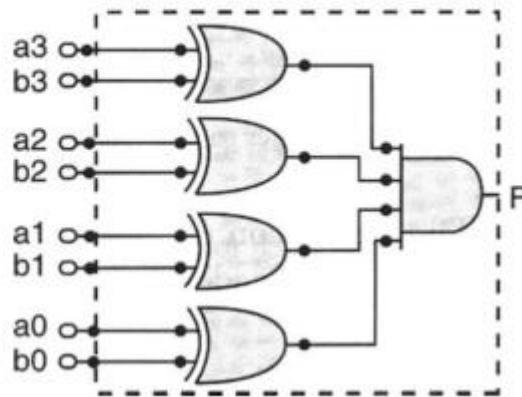
//Los – (doble guion) significan un comentario

```

1      --Declaración de la entidad de un circuito sumador
2      entity sumador is
3          port      (A, B, Cin:  in bit;
4                           SUMA, Cout: out bit);
5      end sumador;

```

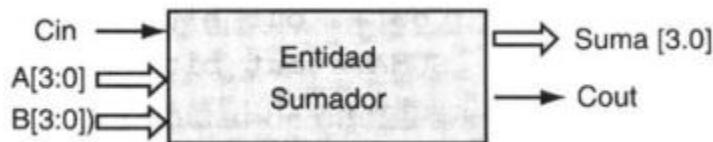
Declare la entidad del circuito lógico de la figura C2.1.



```
1 -- Declaración de la entidad
2 Entity circuito is
3     port( a3,b3,a2,b2,a1,b1,a0, b0: in bit;
4                         F: out bit);
5     end circuito;
```

Diseño de entidades mediante vectores

$$\begin{aligned} \text{vector_A} &= [\text{A}_3, \text{A}_2, \text{A}_1, \text{A}_0] \\ \text{vector_B} &= [\text{B}_3, \text{B}_2, \text{B}_1, \text{B}_0] \\ \text{vector_SUMA} &= [\text{S}_3, \text{S}_2, \text{S}_1, \text{S}_0] \end{aligned}$$



La manera de describir en VHDL una configuración que utilice vectores consiste en la utilización de la sentencia `bit_vector`.

```
port (vector_A, vector_B: in bit_vector (3 downto 0);
      vector_SUMA: out bit_vector (3 downto 0));
```

Esta declaración define a los vectores con cuatro componentes distribuidas en orden descendente por el comando:

3 downto 0 (3 hacia 0)

los cuales se agruparían de la siguiente manera.

vector_A(3) = A3	vector_B(3) = B3	vector_SUMA(3) = S3
vector_A(2) = A2	vector_B(2) = B2	vector_SUMA(2) = S2
vector_A(1) = A1	vector_B(1) = B1	vector_SUMA(1) = SI
vector_A(0) = A0	vector_B(0) = B0	vector_SUMA(0) = SO

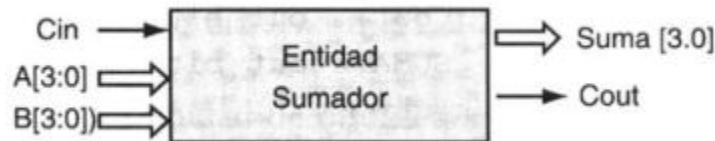
Una vez que se establezca el orden de los bits en el vector no se puede modificar en el caso de **downto** a menos que se utiliza el comando **to**, indica el orden de aparición en sentido ascendente.

Ej

0 to 3 (0 hasta 3)

EJEMPLO 1.0

Describa en VHDL la entidad del circuito sumador

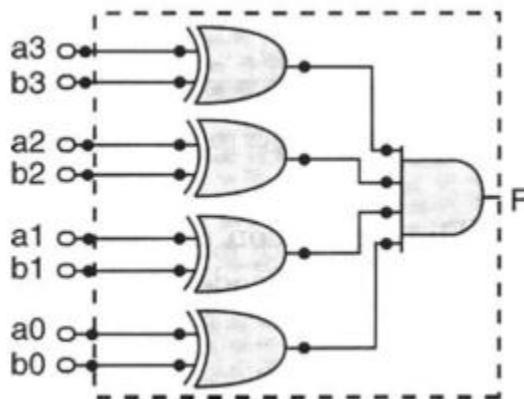


Solución

```
entity sumador is
port (A,B: in bit_vector (3 downto 0);
      Cin: in bit;
      Cout: out bit;
      SUMA: out bit_vector(3 downto 0));
end sumador;
```

EJEMPLO 1.1

Declare la entidad del circuito lógico mediante vectores



Solución

```
1 -Declaración de entidades mediante vectores
2 entity detector is
3 port (a,b: in bit_vector(3 downto 0);
4           F: out bit);
5 end detector;
```

Declaración de entidades mediante librerías y paquetes

Las librerías y paquetes permiten declarar y almacenar estructuras lógicas que nos faciliten el diseño.

Una librería o biblioteca es un lugar al que se tiene acceso para utilizar unidades de diseño predeterminadas por el fabricante de la herramienta (paquete) en VHDL se encuentran definidas dos librerías llamadas:

- IEEE
- WORK

En la librería WORK se permite almacenar el resultado de la compilación de un diseño, con el fin de utilizar en uno o en varios programas.

La librería IEEE debe ser declarada con el comando **library** mientras que WORK no es necesaria que sea declarada ya que WORK siempre está presente al desarrollar un diseño.

LIBRARY ieee;

Use IEEE.STD_LOGIC_1164.ALL;

El código es escrito en forma de Funciones (Functions), Procesos (Process), Procedimientos (Procedures) o Componentes (Components) y luego ubicados dentro de paquetes (Packages) para ser compilado dentro de la librería destino.

La librería IEEE contiene a:

- use IEEE.STD_LOGIC_1164.ALL;
- use IEEE.STD_LOGIC_ARITH.ALL;
- Especifica tipos de datos con y sin signo, operaciones aritméticas y de comparación numérica así como funciones para la conversión de datos, se encuentra (=), (<), (>) ,etc.
- use IEEE.STD_LOGIC_SIGNED.ALL;
- Permite operaciones con signo con datos tipo STD_LOGIC_VECTOR
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- Permite operaciones sin signo con datos tipo STD_LOGIC_VECTOR

STD_LOGIC

Generalmente utilizado para síntesis y simulación. Posee 3 niveles diferentes de valores.

- Nivel lógico '0'
- Nivel lógico '1'
- Alta impedancia 'Z'

STD_ULOGIC

- Señal no definida 'U'
- Desconocido 'X'
- Nivel lógico '0'
- Nivel lógico '1'
- Alta impedancia 'Z'
- Señal débil 'W' no se define como 0 o 1
- Señal débil 'L' se dirige a 0
- Señal débil 'H' se dirige a 1
- No importa '-'

Paquetes

El acceso a la información contenida de un paquete es por medio de la sentencia **use** seguida del nombre de la librería y del paquete:

```
use nombre_librería.nombre_paquete.all;
```

Por ejemplo:

```
Use IEEE.STD_LOGIC_1164.ALL;
```

- IEEE: librería
- STD_LOGIC_1164: paquete.

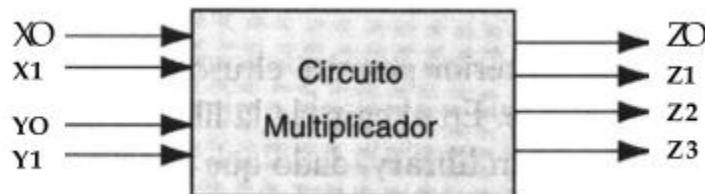
La palabra reservada all indica que se pueden usar todos los componentes almacenados en el paquete.

EJEMPLO 1.2

En la siguiente imagen se encuentra un circuito multiplicador de 2 bits.

La multiplicación de (X1,X0) y (Y1,Y0) producen la salida Z3,Z2,Z1,Z0.

Declarar la entidad del circuito utilizando la librería IEEE y el paquete STD_LOGIC_1164.ALL



Solución

1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. entity multiplicador is
4. port (X0,X1,Y0,Y1: in STD_LOGIC;
5. Z3,Z2,Z1,Z0: out STD_LOGIC);
6. end multiplicador.

Arquitecturas

Una arquitectura (architecture) se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevarán a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas.

Una ventaja de VHDL es que mediante el algoritmo de programación empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.

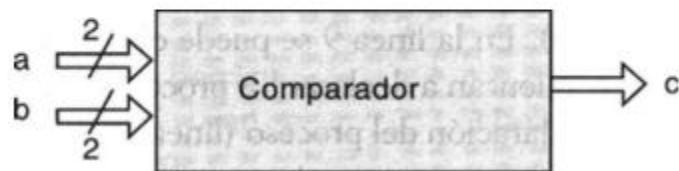
Los estilos de programación utilizados en el diseño de arquitecturas se clasifican como:

- Estilo funcional
- Estilo por flujo de datos
- Estilo estructural

Descripción funcional

Se le llama funcional porque describe la forma en que funciona el sistema, las descripciones consideran la relación que hay entre las entradas y las salidas del circuito sin importar cómo este organizado en su interior.

$$\begin{array}{ll} \text{si } & a = b \text{ entonces } c = 1 \\ \text{si } & a \neq b \text{ entonces } c = 0 \end{array}$$



Lo que se tiene en la imagen es un comparador de igualdad de dos bits

Esto se expresa en el código como:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Comp is
    Port ( a,b : in STD_LOGIC_VECTOR (1 downto 0);
           c : out STD_LOGIC);
end Comp;

architecture Funcional of Comp is

begin
process (a,b)
begin
    if a=b then
        c<='1';
    else
        c<='0';
    end if;
end process;

end Funcional;
```

Describa mediante declaraciones del tipo if-then-else el funcionamiento de la compuerta OR



a	b	f[
0	0	0
0	1	1
1	0	1
1	1	1

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity C_OR is
4     port(a,b:in std_logic;
5          c:out std_logic);
6 end C_OR;
7
8 architecture funcional of C_OR is
9 begin
10 process(a,b)
11 begin
12     if(a='0' and b='0')then
13         c<='0';
14     else
15         c<='1';
16     end if;
17 end process;
18 end funcional;
```

Descripción por flujo de datos

La descripción por flujo de datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales (if-then-else). Este tipo de descripciones permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones. En este tipo de descripción se pueden utilizar dos formatos: mediante instrucciones when-else (cuando-sino) o por medio de ecuaciones booleanas.

- a) Descripción por flujo de datos mediante when-else

Ejemplo: Realizar un comparador con flujo de datos y con sentencia when – else

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity comp is
4 port (a,b: in bit_vector (1 downto 0);
5       c: out bit);
6 end comp;
7
8 architecture f_datos of comp is
9 begin
10    c<='1' when (a=b) else '0';
11 end f_datos;
```

En VHDL se manejan dos declaraciones:

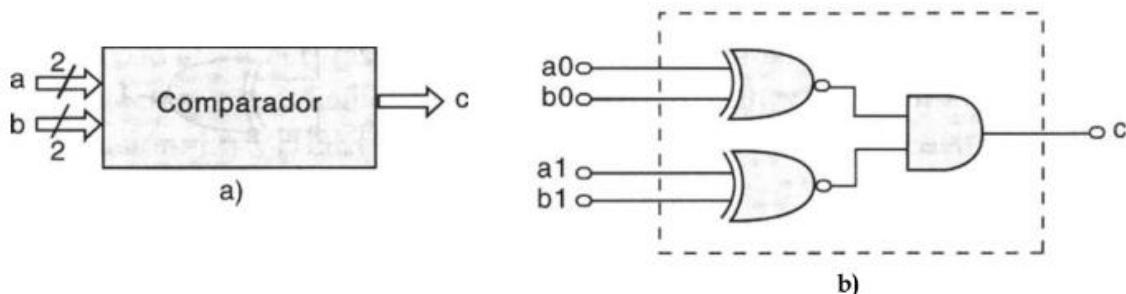
- Secuenciales: una declaración secuencial de la forma if-then-else se halla en tipo de descripción funcional dentro del proceso, donde su ejecución debe seguir un orden para evitar perder lógica.
- Concurrentes: no importa el orden en que se ejecuten como en la arquitectura de flujo de datos.

- b) Basándose en la tabla de verdad mediante declaración de when-else describa el funcionamiento de una AND.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity com_and is
4 port (a,b: in std_logic;
5       f: out std_logic);
6 end com_and;
7
8 architecture compuerta of com_and is
9 begin
10    f<='1' when (a='1' and b= '1') else '0';
11 end compuerta;
```

Descripción por flujo de datos mediante ecuaciones booleanas

Otra forma de visualizar el circuito comparador de bits es:

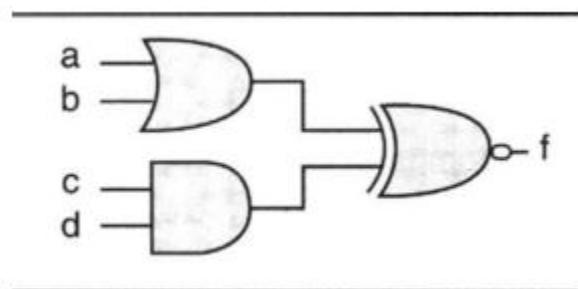


El circuito b se puede describir mediante la obtención de sus ecuaciones booleanas

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity comparador is
4 port (a,b: in bit_vector (1 downto 0);
5        c: out bit);
6 end comparador;
7 architecture booleana of comparador is
8 begin
9 c<=(a(1)xnor b(1))and (a(0) xnor b(0));
10 end booleana;
```

La forma de flujo de datos en cualquiera de sus representaciones describe el camino que los datos siguen al ser transferidos de las operaciones efectuadas entre las entradas a y b a la señal de salida c.

Ejemplo: Describe mediante ecuaciones booleanas el circuito mostrado a continuación



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity ejemplo is
4 port (a,b,c,d: in std_logic;
5       f: out std_logic);
6 end ejemplo;
7
8 architecture circuito of ejemplo is
9 begin
10 f<=((a or b) xnor (c and d));
11 end circuito;
```

Descripción estructural

Una descripción estructural basa su comportamiento en modelos lógicos establecidos (compuertas, sumadores, contadores, etc).

El usuario puede diseñar estructuras y guardarlas para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se esté utilizando.

Cada compuerta (modelo lógico) se encuentra dentro del paquete gatespkg, del cual se toman para estructurar el diseño. Este tipo de arquitecturas estándares se conoce como componentes, que al interconectarse por medio de señales internas (x0, x1) permiten proponer una solución. En VHDL esta conectividad se conoce como netlist o listado de componentes.

Para iniciar una entidad estructural, es necesario la descomposición lógica del diseño en pequeños sub módulos los cuales permiten analizar de manera práctica el circuito ya que la función de entrada/salida es conocida.

Es importante resaltar que una jerarquía en VHDL se refiere al procedimiento de dividir en bloques y no a que un bloque tenga mayor jerarquía que otro.

Diseño Lógico Combinacional mediante VHDL

Programación de estructuras básicas mediante declaraciones concurrentes

Las declaraciones concurrentes se encuentran fuera de la declaración de un proceso y suelen usarse en las descripciones de flujo de datos y estructural. Esto se debe a que en una declaración concurrente no importa el orden en que se escriban las señales, ya que el resultado para determinada función sería el mismo.

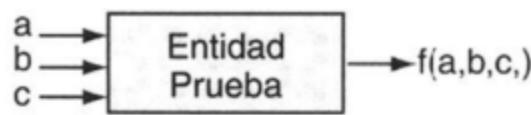
En VHDL existen 3 tipos de declaraciones concurrentes:

- Declaraciones condicionales asignadas a una señal (when-else)
- Declaraciones concurrentes asignadas a señales
- Selección de una señal (with-select-when)

Declaraciones condicionales asignadas a una señal (when-else)

La declaración when-else se utiliza para asignar valores a una señal, determinando así la ejecución de una condición propia del diseño.

Ejemplo:



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

La entidad se puede programar mediante declaraciones condicionales (when-else), debido a que este modelo permite definir paso a paso el comportamiento del sistema.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity prueba is
4 port(a,b,c: in std_logic;
5      f: out std_logic);
6 end prueba;
7
8 architecture ejemplo of prueba is
9 begin
10    f<='1' when (a='0' and b='0' and c='0') else
11        '1' when (a='0' and b='1' and c='1') else
12        '1' when (a='1' and b='1' and c='0') else
13        '1' when (a='1' and b='1' and c='1') else
14        '0';
15 end ejemplo;
```

Operadores Lógicos

Los operadores lógicos más utilizados en la descripción de funciones booleanas y definidos en los diferentes tipos de datos bit, son los operadores and, or, nand, xor, xnor y not. Las operaciones que se efectúen entre ellos (excepto not) deben realizarse con datos que tengan la misma longitud o palabra de bits.

Al momento de ser compilados los operadores lógicos presentan el siguiente orden y prioridad:

- 1) Expresiones entre paréntesis.
- 2) Complementos
- 3) Función AND
- 4) Función OR

Las operaciones XOR y XNOR son transparentes al compilador y las interpreta mediante la suma de productos correspondiente a su función.

Ejercicio:

Obtenga la tabla de verdad de la función F y realice el programa correspondiente utilizando estructuras del tipo when-else y operadores lógicos

D	C	B	A	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

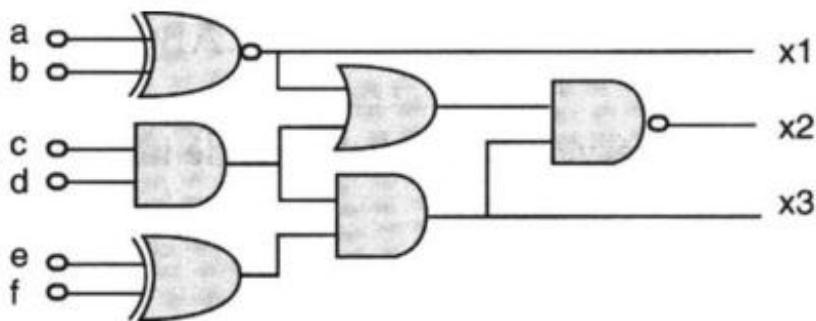
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity tabla is
4 port(D,C,B,A: in std_logic;
5       F: out std_logic);
6 end tabla;
7 architecture func of tabla is
8 begin
9     F<='1' when(A='0' and B='0' and C='1' and D='0') else
10      '1' when(A='1' and B='0' and C='1' and D='0') else
11      '1' when(A='0' and B='1' and C='1' and D='0') else
12      '1' when(A='1' and B='1' and C='1' and D='0') else
13      '0';
14 end func;

```

Declaraciones concurrentes asignadas a señales

En este tipo de declaración encontraremos las funciones de salida mediante la ecuación booleana que describe el comportamiento de cada una de las compuertas.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity logic is
4 port(a,b,c,d,e,f: in std_logic;
5      x1,x2,x3: out std_logic);
6 end logic;
7
8 architecture booleana of logic is
9 begin
10    x1<=a xnor b;
11    x2<=((c and d)or(a xnor b)) nand ((e xor f)and(c and d));
12    x3<=(e xor f)and(c and d);
13 end booleana;
```

Dada la tabla de verdad mostrada halle las ecuaciones X,Y,Z de la forma suma de productos utilizando declaraciones concurrentes asignadas a señales.

A	B	C	X	Y	Z
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0

$$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + ABC$$

$$Y = \bar{A}\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$$

$$Z = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C$$

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity tabla is
4 port(A,B,C: in std_logic;
5      X,Y,Z: out std_logic);
6 end tabla;
7
8 architecture booleana of tabla is
9 begin
10    X<=((not A and not B and not C)or(not A and not B and C)
11        or(not A and B and C)or(A and B and C));
12    Y<=((not A and not B and C)or(A and not B and C)
13        or(A and B and not C));
14    Z<=((not A and not B and not C)or(not A and B and not C)
15        or(not A and B and C));
16 end booleana;

```

Selección de una señal (with*select>when)

La declaración se utiliza para asignar un valor a una señal con base en el valor de otra señal previamente seleccionada.

a(0)	a(1)	c
0	0	1
0	1	0
1	0	1
1	1	0

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity circuito is
4 port(a: in std_logic_vector (1 downto 0);
5      c: out std_logic);
6 end circuito;
7
8 architecture selector of circuito is
9 begin
10    with a select
11        c<='1' when "00",
12            '0' when "01",
13            '1' when "10",
14            '0' when others;
15 end selector;
```

Se requiere diseñar un circuito combinacional que detecte números primos de 4 bits.

X0	X1	X2	X3	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity numeros is
4 port(X: in std_logic_vector (0 to 3);
5      F: out std_logic);
6 end numeros;
7
8 architecture primos of numeros is
9 begin
10    with X select
11        F<='1' when "0010",
12        '1' when "0011",
13        '1' when "0101",
14        '1' when "0111",
15        '1' when "1011",
16        '1' when "1101",
17        '0' when others;
18 end primos;

```

Programación de estructuras básicas mediante declaraciones secuenciales

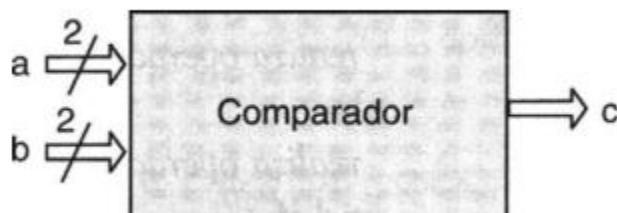
Las declaraciones secuenciales son aquellas en las que el orden que llevan puede tener un efecto significativo en la lógica descrita.

A diferencia de una declaración concurrente, una secuencial debe ejecutarse en el orden en que aparece y formar parte de un proceso (process).

Declaración if-then-else (si-entonces-sino). Esta declaración sirve para seleccionar una condición o condiciones basadas en el resultado de evaluaciones lógicas (falso o verdadero).

```
if la condición es cierta then
    realiza la operación 1;
else
    realiza la operación 2;
end if;
```

Si if (condición) se evalúa como verdadera, entonces (then) la instrucción indica que se ejecutará la operación 1. Por el contrario, si la condición se evalúa como falsa (else) correrá la operación 2. La instrucción que indica el fin de la declaración es end if (fin del si).

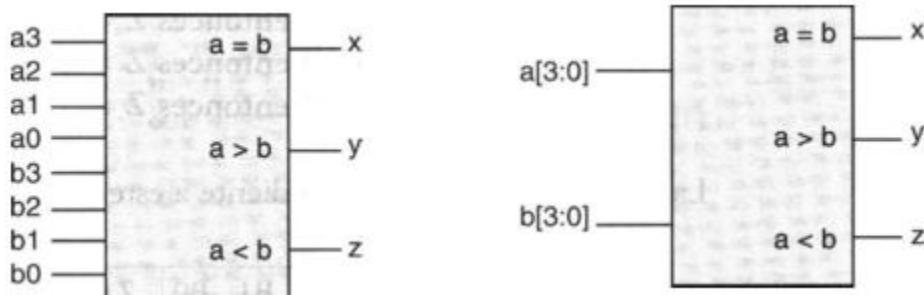


Cuando se requieren más condicionales de control, se utiliza una nueva estructura llamada elsif (sino-si), la cual permite expandir y especificar prioridades dentro del proceso. La sintaxis para esta operación es:

```
if la condición 1 se cumple then
    realiza operación 1;
elsif la condición 2 se cumple then
    realiza operación 2;
else
    realiza operación 3;
end if;
```

Comparador de magnitud de 4 bits

La forma de utilizar las declaraciones secuenciales se ilustra en el diseño de un comparador de dos números de 4 bits, el siguiente sistema tiene tres salidas que indican cuando uno de los números es mayor, igual o menor que el otro.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity comparador is
4 port(a,b: in std_logic_vector (3 downto 0);
5      x,y,z: out std_logic);
6 end comparador;
7
8 architecture a_compf of comparador is
9 begin
10 process (a,b)
11 begin
12   if(a=b) then
13     x<='1';
14   elsif(a>b) then
15     y<='1';
16   else
17     z<='1';
18 end if;
19 end process;
20 end a_compf;
```

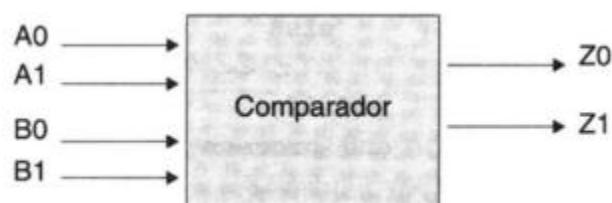
Diseñe un comparador de dos números A y B, cada número formado por dos bits (A1 A0) y (B1 B0) la salida del comparador también es de dos bits y está representado por la variable Z(Z1 Z0) de tal forma que si:

$$A = B \text{ entonces } Z = 11$$

$$A < B \text{ entonces } Z = 01$$

$$A > B \text{ entonces } Z = 10$$

A1	A0	B1	BO	Z1	Z0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1



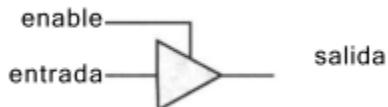
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity compara is
4 port(A,B: in std_logic_vector (1 downto 0);
5      Z: out std_logic_vector (1 downto 0));
6 end compara;
7
8 architecture tabla of compara is
9 begin
10 process(A,B)
11 begin
12     if(A=B)then
13         Z<="11";
14     elsif (A<B)then
15         Z<="01";
16     else
17         Z<="10";
18 end if;
19 end process;
20 end tabla;
```

Operadores relacionales. Los operadores relacionales se usan para evaluar la igualdad, desigualdad o la magnitud en una expresión. Los operadores de igualdad y desigualdad (= y /=) se definen en todos los tipos de datos. Los operadores de magnitud (<, <=, >, >=) lo están sólo dentro del tipo escalar. En ambos casos debe considerarse que el tamaño de los vectores en que se aplicarán dichos operadores debe ser igual.

Operador	Significado
=	Igual
/=	Diferente
<	Menor
< =	Menor o igual
>	Mayor
> =	Mayor o igual

Buffers triestado

Los registros de tres estados (buffers tri-estado) tienen diversas aplicaciones, ya sea como salidas de sistemas (modo buffer) o como parte integral de un circuito. En VHDL estos dispositivos son definidos a través de los valores que manejan (0,1 y alta impedancia 'Z').

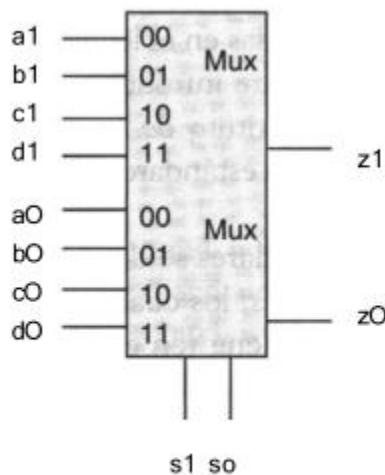


```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity triestado is
4 port(enable,entrada: in std_logic;
5          salida: out std_logic);
6 end triestado;
7
8 architecture bufftries of triestado is
9 begin
10 process(enable,entrada)
11 begin
12     if(enable='0')then
13         salida<='Z';
14     else
15         salida<= entrada;
16     end if;
17 end process;
18 end bufftries;
```

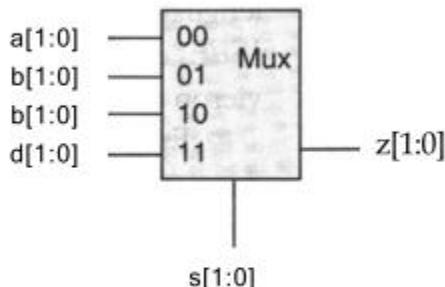
Cuando se confirma el habilitador del circuito (enable), el valor que se encuentra a la entrada del circuito se asigna a la salida; si por el contrario no se confirma enable, la salida del buffer tomará un valor de alta impedancia (Z).

Multiplexores

Los multiplexores se diseñan describiendo su comportamiento mediante la declaración with-select*when o ecuaciones booleanas.



a)



b)

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity mux is
4 port(a,b,c,d: in std_logic_vector (1 downto 0);
5      s: in std_logic_vector (1 downto 0);
6      z: out std_logic_vector (1 downto 0));
7 end mux;
8
9 architecture mux2x4 of mux is
10 begin
11 with s select
12     z<= a when "00",
13         b when "01",
14         c when "10",
15         d when others;
16 end mux2x4;
```

Tipos Lógicos Estándares

Las funciones lógicas estándares definidas en el lenguaje VHDL se crearon para evitar que cada distribuidor de software introdujera sus paquetes y tipos de datos al lenguaje. Por esta razón el Instituto de Ingenieros Eléctricos y Electrónicos, IEEE, estableció los estándares std_logic y std_logic_vector.

En cada uno de estos estándares se definen ciertos tipos de datos conocidos como tipos lógicos estándares, los cuales se pueden utilizar haciendo referencia al paquete que los contiene.

STD_LOGIC

Generalmente utilizado para síntesis y simulación. Posee 3 niveles diferentes de valores.

- Nivel lógico '0'
- Nivel lógico '1'
- Alta impedancia 'Z'

STD_ULOGIC

- Señal no definida 'U'
- Desconocido 'X'
- Nivel lógico '0'
- Nivel lógico '1'
- Alta impedancia 'Z'
- Señal débil 'W' no se define como 0 o 1
- Señal débil 'L' se dirige a 0
- Señal débil 'H' se dirige a 1
- No importa '-'

Descripción de multiplexores mediante ecuaciones booleanas

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity mux is
4 port(a,b,c,d: in std_logic_vector (1 downto 0);
5      s: in std_logic_vector (1 downto 0);
6      z: out std_logic_vector (1 downto 0));
7 end mux;
8
9 architecture muxbool of mux is
10 begin
11     z(1)<=(a(1) and not(s(1)) and not(s(0))) or
12             (b(1) and not(s(1)) and s(0)) or
13             (c(1) and s(1) and not(s(0))) or
14             (d(1) and s(1) and s(0));
15
16     z(0)<=(a(0) and not(s(1)) and not(s(0))) or
17             (b(0) and not(s(1)) and s(0)) or
18             (c(0) and s(1) and not(s(0))) or
19             (d(0) and s(1) and s(0));
20 end muxbool;

```

Sumadores

Diseño de un circuito medio sumador

Para describir el funcionamiento de los circuitos sumadores es importante recordar las reglas básicas de la adición.

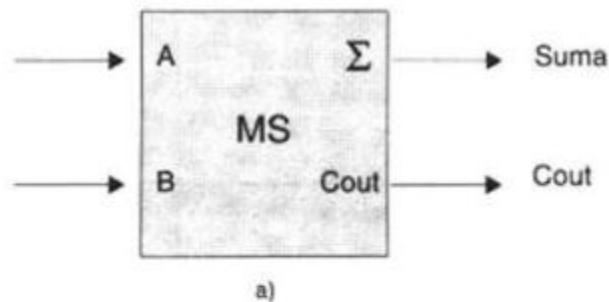
$$0+0=0$$

$$0+1=1$$

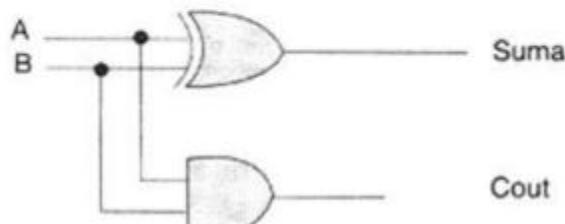
$$1+0=1$$

$$1+1=10$$

A	B	Suma	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1 (1 + 1 = 10)



a)



a)

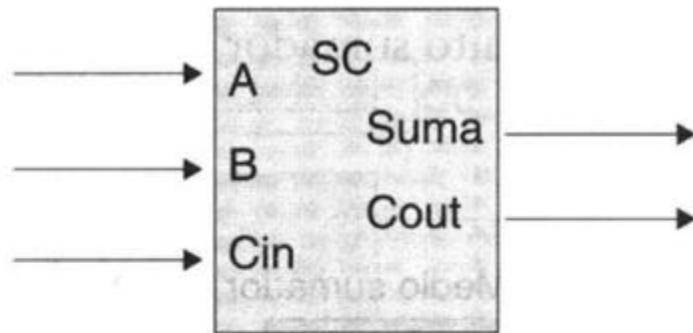
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity medio is
4 port(A,B: in std_logic;
5      SUMA,Cout: out std_logic);
6 end medio;
7
8 architecture sumador of medio is
9 begin
10    SUMA<=(A xor B);
11    Cout<=(A and B);
12 end sumador;

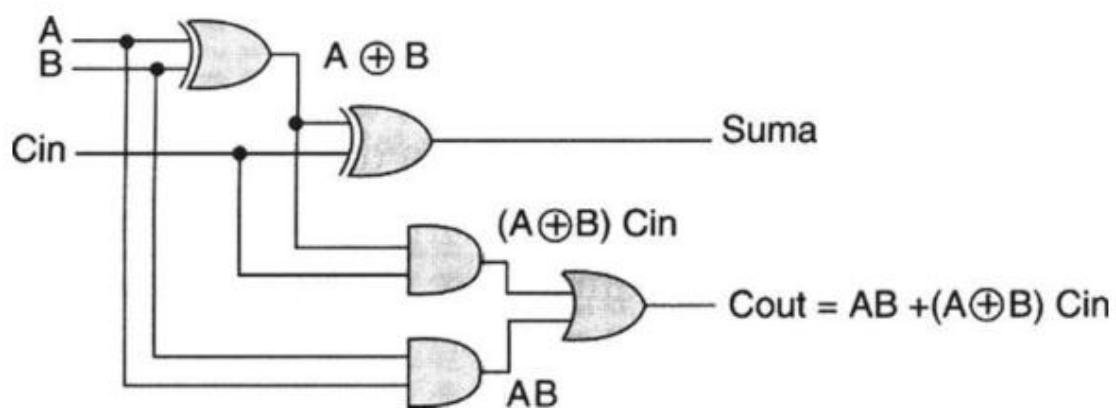
```

Diseño de un sumador completo

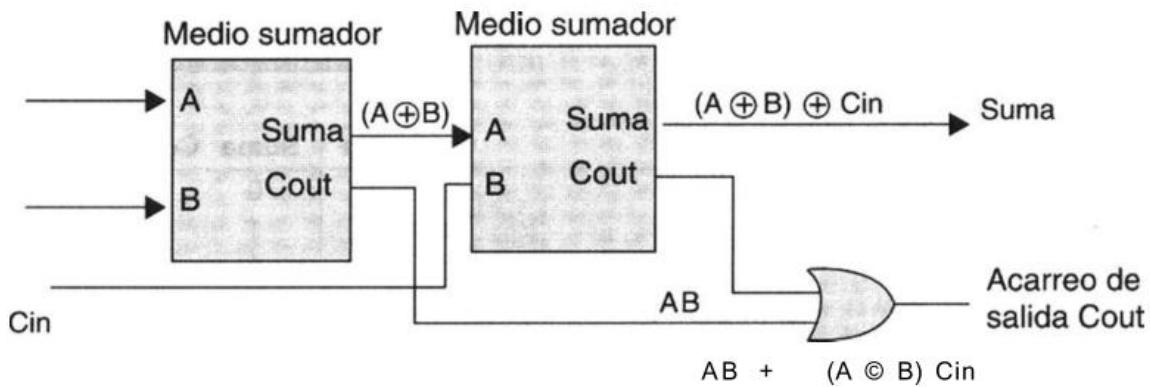
Un sumador completo (SC) a diferencia del circuito medio sumador considera un acarreo de entrada (Cin).



A	B	Cin	Suma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



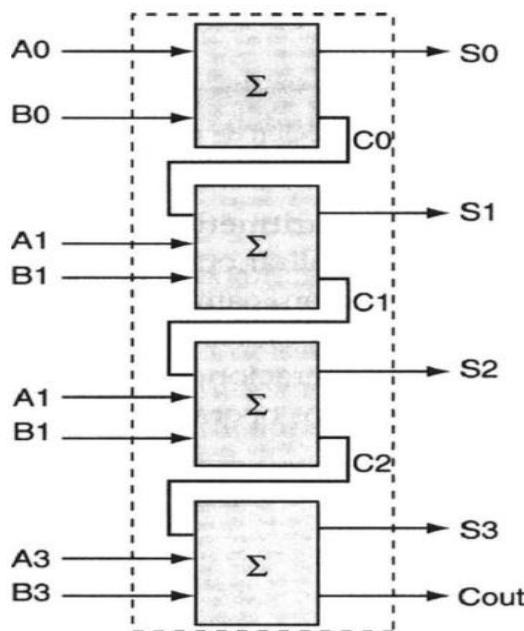
- a) Circuito sumador completo implementado por medios sumadores



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity completo is
4 port(A,B,Cin: in std_logic;
5      Suma,Cout: out std_logic);
6 end completo;
7
8 architecture sumador of completo is
9 begin
10    Suma<= ((A xor B) xor Cin);
11    Cout<= ((A and B) or ((A xor B) and Cin));
12 end sumador;
```

Sumador paralelo de 4 bits

Para realizarlo se requiere conectar en cascada un circuito medio sumador y 3 sumadores completos



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity paralelo is
4 port(A,B: in std_logic_vector (0 to 3);
5      S,C: out std_logic_vector (0 to 3);
6      Cout: out std_logic);
7 end paralelo;
8
9 architecture sumador4 of paralelo is
10 begin
11     S(0)<=(A(0) xor B(0));
12     C(0)<=(A(0) and B(0));
13     S(1)<=((A(1) xor B(1)) xor C(0));
14     C(1)<=((A(1) and B(1)) or (C(0) and (A(1) xor B(1))));
15     S(2)<=((A(2) xor B(2)) xor C(1));
16     C(2)<=((A(2) and B(2)) or (C(1) and (A(2) xor B(2))));
17     S(3)<=((A(3) xor B(3)) xor C(2));
18     Cout<=((A(3) and B(3)) or (C(2) and (A(3) xor B(3))));
19 end sumador4;
```

Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones del tipo aritmético, como suma, resta, multiplicación, división, cambio de signo, valor absoluto y concatenación.

Estos operadores suelen usarse en el diseño lógico para describir sumadores y restadores o en las operaciones de incremento y decremento de datos.

- Suma (+) y Resta (-).
- Multiplicación (*) y División (/).
- Exponencial (**).
- Valor absoluto (abs).
- Módulo (mod) y Resto (rem).

Analicemos el diseño de un circuito sumador de 4 bits que no considera el acarreo de salida

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity sumador is
5 port(A,B: in std_logic_vector (0 to 3);
6      Suma: out std_logic_vector (0 to 3));
7 end sumador;
8
9 architecture aritmetico of sumador is
10 begin
11     Suma<=(A+B);
12 end aritmetico;
```

El paquete std_arith se encuentra en la librería de trabajo work. Este paquete permite el uso de los operadores aritméticos con operaciones realizadas entre arreglos del tipo std_logic_vector.

Decodificadores

La programación de circuitos decodificadores se basa en el uso de declaraciones que permiten establecer la relación entre un código binario aplicado a las entradas del dispositivo y el nivel de salida obtenido.

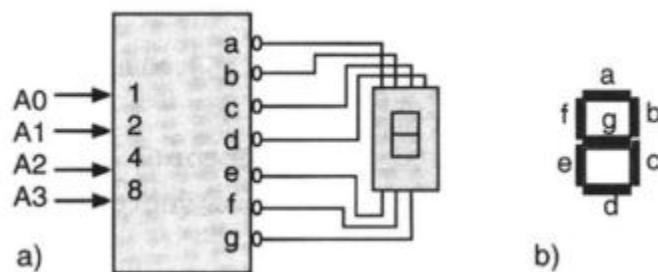
Los dos diseños más utilizado en el diseño lógico combinacional de decodificadores son: el decodificador BCD decimal y el decodificador de BCD a siete segmentos.

Decodificador BCD a decimal

Como lo dice su nombre se encarga de convertir código binario a decimal en uno de los diez dígitos decimales.

Decodificador de BCD a display de siete segmentos

Acepta código BCD en sus entradas y proporciona salidas capaces de excitar un display de siete segmentos que indica el dígito decimal seleccionado.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity decoder is
4 port(A: in std_logic_vector (3 downto 0);
5      d: out std_logic_vector (6 downto 0));
6 end decoder;
7 --Programa para display en cátodo común
8 architecture segments of decoder is
9 begin
10 process(A)
11 begin
12 case A is
13     when "0000"=> d <= "1111110";
14     when "0001"=> d <= "0110000";
15     when "0010"=> d <= "1101101";
16     when "0011"=> d <= "1111001";
17     when "0100"=> d <= "0110011";
18     when "0101"=> d <= "1011011";
19     when "0110"=> d <= "1011111";
20     when "0111"=> d <= "1110001";
21     when "1000"=> d <= "1111111";
22     when "1001"=> d <= "1111011";
23     when others=> d <= "0000000";
24 end case;
25 end process;
26 end segments;
```

En la instrucción case-when podemos observar el uso de asignaciones dobles ($=> d \leq$), las cuales permiten que una señal adopte un determinado valor de acuerdo con el cumplimiento de una condición especificada.

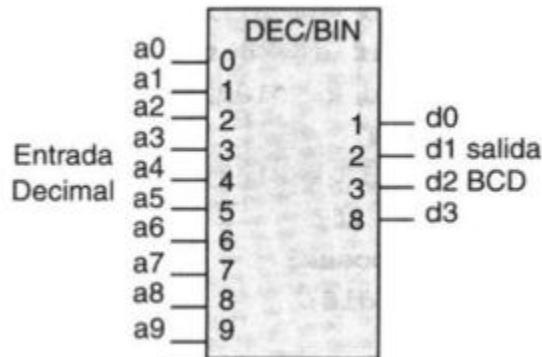
Ahora será con ánodo común

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity decoder is
4 port(A: in std_logic_vector (3 downto 0);
5      d: out std_logic_vector (6 downto 0));
6 end decoder;
7 --Programa para display en ànodo comùn
8 architecture segments of decoder is
9 begin
10 process(A)
11 begin
12 case A is
13 when "0000"=> d <= "0000001";
14 when "0001"=> d <= "1001111";
15 when "0010"=> d <= "0010010";
16 when "0011"=> d <= "0000110";
17 when "0100"=> d <= "1001100";
18 when "0101"=> d <= "0100100";
19 when "0110"=> d <= "0100000";
20 when "0111"=> d <= "0001110";
21 when "1000"=> d <= "0000000";
22 when "1001"=> d <= "0000100";
23 when others=> d <= "1111111";
24 end case;
25 end process;
26 end segments;
```

Codificadores

Codificador decimal-BCD

Este tipo de codificador tiene diez entradas, una para cada código decimal, y cuatro salidas que corresponden al código BCD, es un codificador básico de 10 líneas a 4 líneas

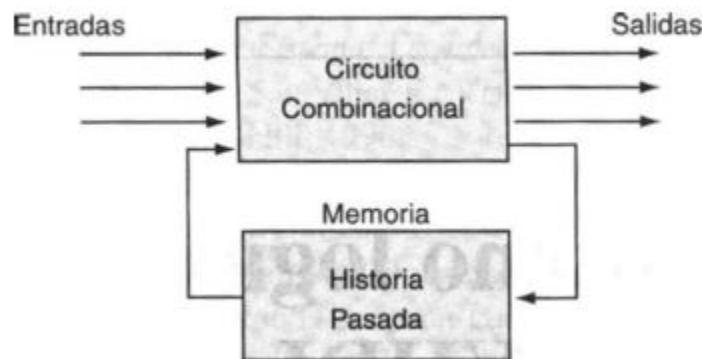


Dígito decimal	Código BCD			
	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity codificador is
5 port(a: in integer range 0 to 9;
6      d: out std_logic_vector (3 downto 0));
7 end codificador;
8
9 architecture arqcodif of codificador is
10 begin
11 process (a)
12 begin
13     if a=0 then
14         d<="0000";
15     elsif a=1 then
16         d<="0001";
17     elsif a=2 then
18         d<="0010";
19     elsif a=3 then
20         d<="0011";
21     elsif a=4 then
22         d<="0100";
23     elsif a=5 then
24         d<="0101";
25     elsif a=6 then
26         d<="0110";
27     elsif a=7 then
28         d<="0111";
29     elsif a=8 then
30         d<="1000";
31     else
32         d<="1001";
33 end if;
34 end process;
35 end arqcodif;
```

Diseño Lógico 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.



Básicamente hay dos tipos de sistemas secuenciales: síncronos y asíncronos.

El comportamiento de los sistemas secuenciales síncronos se encuentra sincronizado mediante el pulso de reloj del sistema, mientras que los asíncronos depende del orden y momento en el cual se aplican sus señales de entrada, por lo que requieren un pulso de reloj para sincronizar sus acciones.

Flip-flops

El elemento de memoria utilizado indistintamente en el diseño de los sistemas síncronos o asíncronos se conoce como flip-flop o celda binaria.

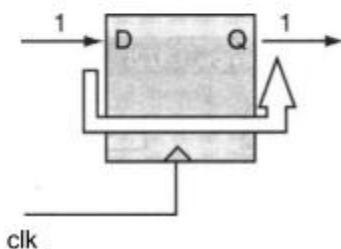
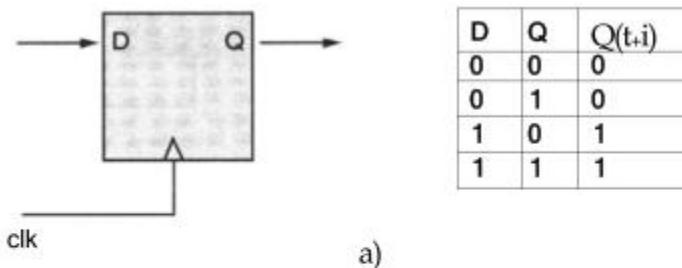
La característica principal de un flip flop es mantener o almacenar un bit de manera indefinida hasta que a través de un pulso o una señal cambie de estado. Los flip flop mas conocidos son los tipos SR, JK, T Y D.

El significado de la notación Q y Q(t+1)

Q= estado presente o actual

Q(t+1)= estado futuro o siguiente

Por ejemplo consideramos el funcionamiento del flip flop tipo D



La ejecución del proceso es sensible a los cambios en clk (pulso de reloj), cuando clk cambia de valor de una transición de 0 a 1 el valor de D se asigna a Q y se conserva hasta que se genera un nuevo pulso.

Si clk no puede presentar dicha transición el valor de Q se mantiene igual.

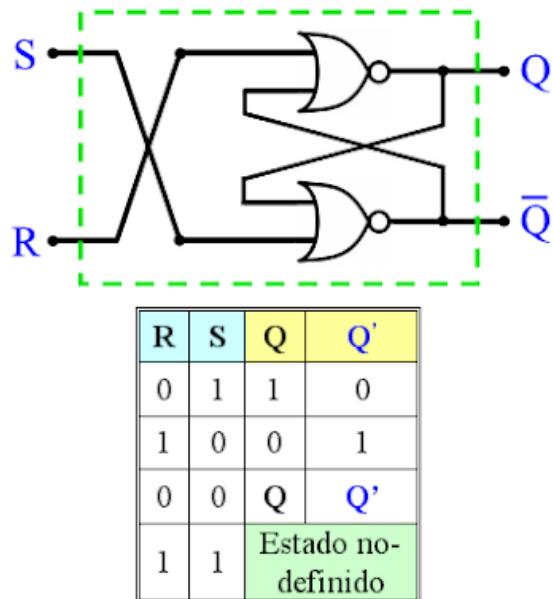
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity D is
4 port(D,clk: in std_logic;
5      Q: out std_logic);
6 end D;
7
8 architecture flipflop of D is
9 begin
10 process(clk)
11 begin
12   if (clk 'event and clk='1') then
13     Q<=D;
14 end if;
15 end process;
16 end flipflop;
```

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' se utiliza para describir un hecho u ocurrencia de una señal en particular.

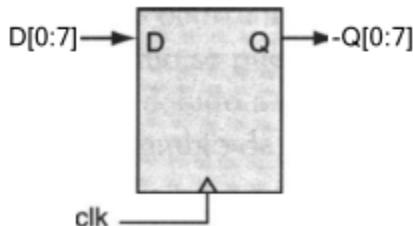
- a) Escriba un programa que describa el funcionamiento de un flip-flop SR con base en la siguiente tabla de verdad.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity RS is
4 port(S,R,clk: in std_logic;
5      Q,Qn: inout std_logic);
6 end RS;
7
8 architecture flipflop of RS is
9 begin
10 process(clk,S,R)
11 begin
12     if (clk'event and clk= '1')then
13         if (S='0' and R='1')then
14             Q<='0';
15             Qn<='1';
16         elsif (S='1' and R='0')then
17             Q<='1';
18             Qn<='0';
19         elsif (S='0' and R='0') then
20             Q<=Q;
21             Qn<=Qn;
22         else
23             Q<='-';
24             Qn<='-';
25     end if;
26 end if;
27 end process;
28 end flipflop;
```

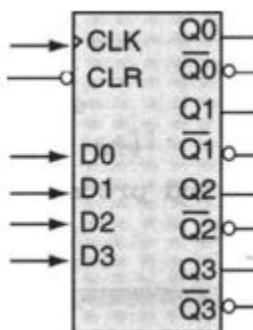
Registros

En la siguiente figura se presenta la estructura de un registro de 8 bits con entrada y salida de datos en paralelo. El diseño es muy similar al flip-flop anterior, la diferencia radica en la utilización de vectores de bits en lugar de un solo bit.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity ochobits is
4 port (D: in std_logic_vector (0 to 7);
5        clk: in std_logic;
6        Q: out std_logic_vector (0 to 7));
7 end ochobits;
8
9 architecture registro of ochobits is
10 begin
11 process(clk)
12 begin
13     if (clk'event and clk='1')then
14         Q<=D;
15     end if;
16 end process;
17 end registro;
```

Escriba un programa de un registro de 4 bits, utilice instrucciones if-then-else y procesos



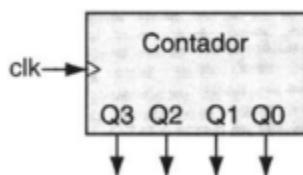
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity fourbits is
4 port(D: in std_logic_vector (3 downto 0);
5      CLK,CLR: in std_logic;
6      Q,Qn: inout std_logic_vector (3 downto 0));
7 end fourbits;
8
9 architecture regis of fourbits is
10 begin
11 process(CLK,CLR)
12 begin
13     if(CLK'event and CLK='1')then
14         if(CLR='1')then
15             Q<=D;
16             Qn<= not Q;
17         else
18             Q<="0000";
19             Qn<="1111";
20     end if;
21 end if;
22 end process;
23 end regis;
```

Las variables sensitivas que determinan el comportamiento del circuito se encuentran dentro del proceso (CLR y CLK). Para transferir los datos de entrada hacia la salida es necesario generar un pulso de reloj a través del atributo event (CLK'event and CLK='1').

Contadores

Los contadores entidades muy utilizadas en el diseño lógico. La forma usual para describirlos en VHDL es mediante operaciones de incremento, decremento de datos o ambas.

La siguiente imagen representa un contador ascendente de 4 bits



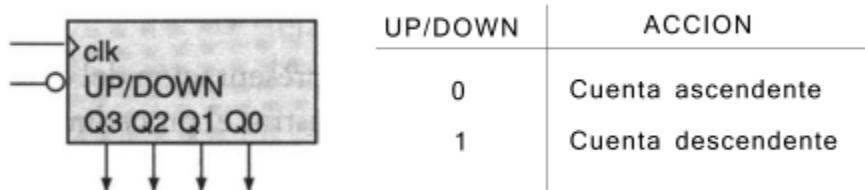
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity cuatrobites is
5 port(clk: in std_logic;
6       Q: inout std_logic_vector (3 downto 0));
7 end cuatrobites;
8
9 architecture contador of cuatrobites is
10 begin
11 process(clk)
12 begin
13   if(clk'event and clk='1')then
14     Q<=Q+1;
15   end if;
16 end process;
17 end contador;
```

Cuando requerimos la retroalimentación de una señal ($Q \leq Q+1$), ya sea dentro o fuera de la entidad, utilizamos el modo inout

Inout (entrada/salida): permite declarar un puerto de forma bidireccional, permite la retroalimentación de señales dentro o fuera de la entidad.

El funcionamiento del contador se define básicamente en un proceso, en el cual se llevan a cabo los eventos que determinan el comportamiento del circuito. Una transición de 0 a 1 de clk provoca que se ejecute el proceso, lo cual incrementa en 1 el valor asignado a la variable Q. Cuando esta salida tiene el valor de 15 ("1111") y si el pulso de reloj se sigue aplicando, el programa empieza a contar nuevamente de 0.

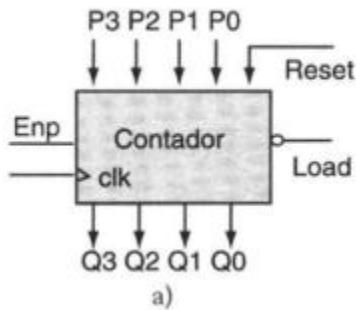
- a) Elabore un programa que describa el funcionamiento de un contador de 4 bits. Realice el diseño una señal de control (Up/Down) que determine el sentido del conteo: ascendente o descendente.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity fourbits is
5 port(clk,UP: in std_logic;
6       Q: inout std_logic_vector(3 downto 0));
7 end fourbits;
8
9 architecture contador of fourbits is
10 begin
11 process(UP,clk)
12 begin
13   if(clk'event and clk='1')then
14     if(UP='0')then
15       Q<=Q+1;
16     else
17       Q<=Q-1;
18   end if;
19 end if;
20 end process;
21 end contador;
```

Contador con reset y carga en paralelo (load)

La entidad de diseño siguiente es un ejemplo de un circuito contador síncrono de 4 bits. Este contador tiene varias características adicionales respecto al anterior.



a)

Enp	Load	Accion
0	0	Carga
0	1	Mantiene Estado
1	0	Carga
1	1	Cuenta

b)

El propósito de describir este ejemplo radica en subrayar el uso de varias condiciones, de tal manera que no sea posible evaluar una instrucción si las condiciones predeterminadas no se cumplen.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity contpar is
5 port(P: in std_logic_vector(3 downto 0);
6       clk,ENP,RESET,LOAD: in std_logic;
7       Q: inout std_logic_vector(3 downto 0));
8 end contpar;
9
10 architecture contador of contpar is
11 begin
12 process(clk,ENP,RESET,LOAD)
13 begin
14   if(RESET='1')then
15     Q<="0000";
16   elsif(clk'event and clk='1')then
17     if(LOAD='0' and ENP='-' )then
18       Q<=P;
19     elsif(LOAD='1' and ENP='0')then
20       Q<=Q;
21     elsif (LOAD ='1' and ENP='1')then
22       Q<=Q+1;
23   end if;
24 end if;
```

```
25 end process;  
26 end contador;
```

Diseño de sistemas secuenciales síncronos

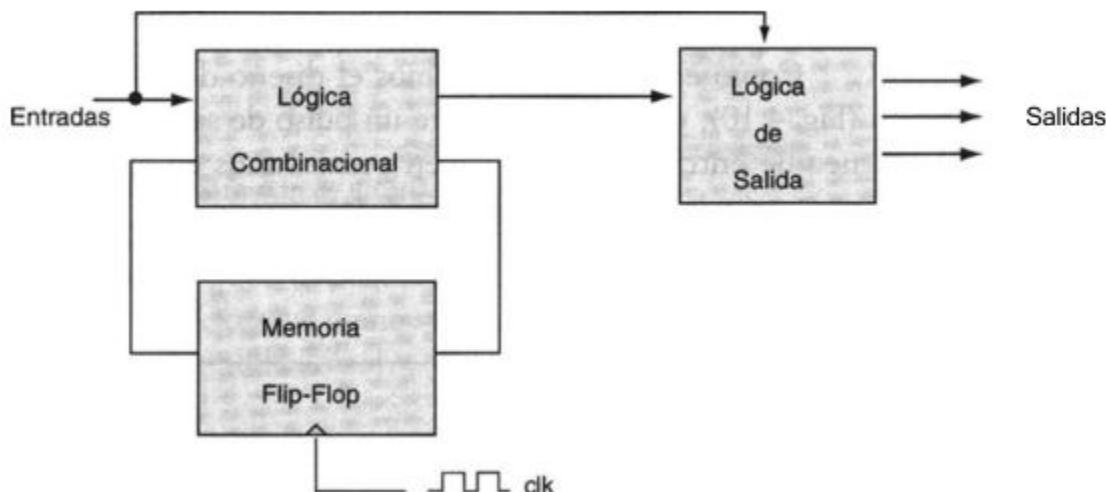
La estructura de los sistemas secuenciales síncronos basa su funcionamiento en los elementos de memoria conocidos como flip-flops. La palabra sincronía se refiere a que cada uno de estos elementos de memoria que interactúan en un sistema se encuentran conectados a la misma señal de reloj, de forma tal que sólo se producirá un cambio de estado en el sistema cuando ocurra un flanco de disparo o un pulso en la señal de reloj.

Existe una división en el diseño de los sistemas secuenciales que se refiere al momento en que se producirá la salida del sistema.

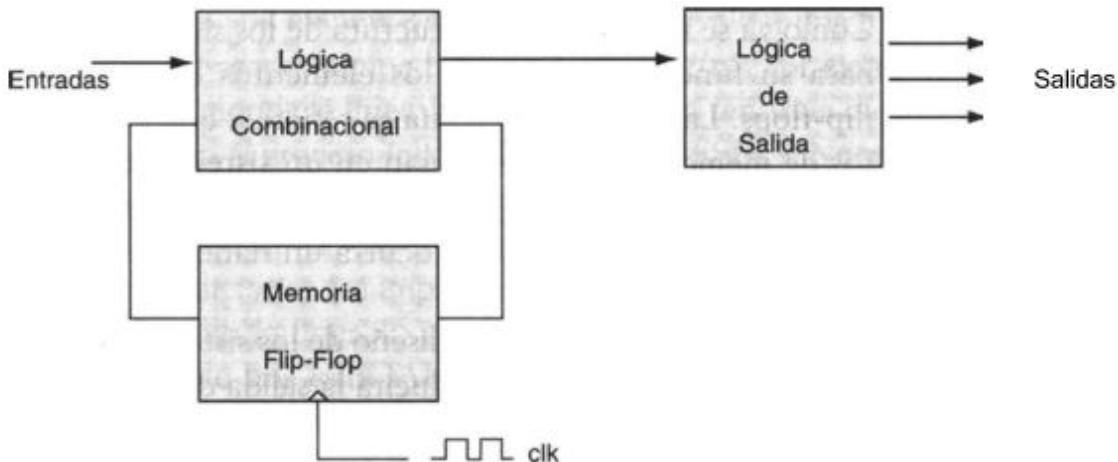
En la estructura de Mealy las señales de salida dependen tanto del estado en que se encuentra el sistema, como de la entrada que se aplica en determinado momento.

En la estructura de Moore la señal de salida sólo depende del estado en que se encuentra.

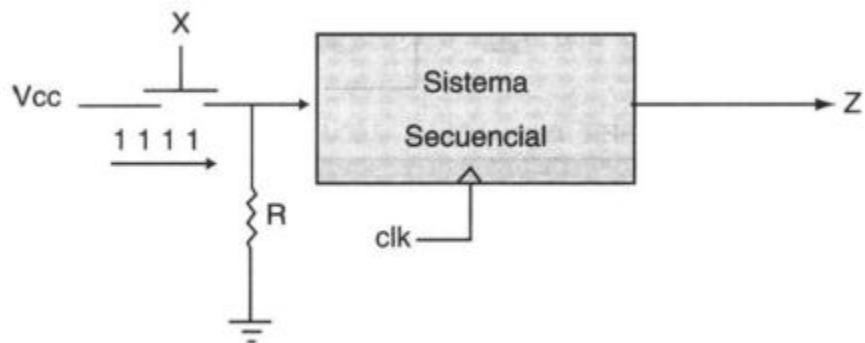
a) Arquitectura secuencial de Mealy



b) Arquitectura secuencial de Moore

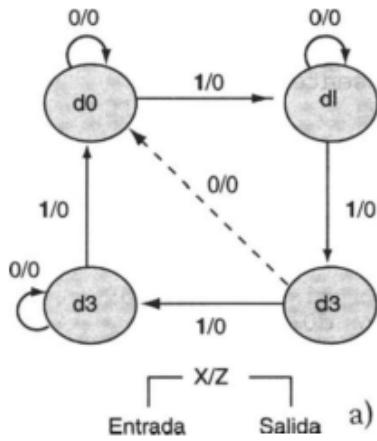


Un sistema secuencial se desarrolla a través de una serie de pasos generalizados que comprenden el enunciado del problema, diagrama de estados, tabla de estados, asignación de estados, ecuaciones de entrada a los elementos de memoria y diagrama electrónico del circuito.



El uso de diagramas de estados en la lógica programable facilita de manera significativa la descripción de un diseño secuencial, ya que no es necesario seguir la metodología tradicional de diseño. En VHDL se puede utilizar un modelo funcional en que sólo se indica la transición que siguen los estados y las condiciones que controlarán el proceso.

La siguiente figura nos permite ver que el sistema secuencial se puede representar por medio del diagrama de estados: arquitectura Mealy.

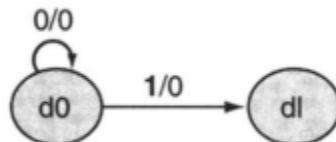


Edo. presente	Edo. futuro X=0 X=1	Salida Z X=0 X=1
d0	d0 d1	0 0
d1	d1 d2	0 0
d2	d2 d3	0 0
d3	d3 d0	0 1

b)

En este diagrama se advierte que el sistema cuenta con una señal de entrada denominada X y una señal de salida Z.

Cuando se está en el estado d0 y la señal de entrada X es igual a uno, se avanza al estado d1 y la salida Z durante esta transición es igual a cero; en caso contrario, cuando la entrada X es igual a cero, el circuito se mantiene en el estado d0 y la salida también es 0.



Este diagrama se puede codificar con facilidad mediante una descripción de alto nivel en VHDL. Esta descripción supone el uso de declaraciones case-when las cuales determinan, en un caso particular, el valor que tomará el siguiente estado. Por otro lado, la transición entre estados se realiza por medio de declaraciones if- then'else, de tal forma que éstas se encargan de establecer la lógica que seguirá el programa para realizar la asignación del estado.

Como primer paso en nuestro diseño, consideramos los estados d0, d1, d2 y d3. Para poder representarlos en código VHDL hay que definirlos dentro de un tipo de datos enumerado (se llaman tipos enumerados porque en ellos se agrupan o enumeran elementos que pertenecen a un mismo género) mediante la declaración type. Observamos la forma en que se listan los identificadores de los estados, así como las señales utilizadas para el estado actual (estado presente) y siguiente (estado futuro).

```
type estados is (d0, d1, d2,d3) ;
signal edo_presente, edo_futuro : estados;
```

El siguiente paso consiste en la declaración del proceso que definirá el comportamiento del sistema. En este debe considerarse que el estado futuro depende del valor del estado presente y de la entrada X.

De esta manera la lista sensitiva del proceso quedaría de la siguiente forma:

```
procesol: process (edo_presente, X)
```

Dentro del proceso se describe la transición del estado presente al estado futuro.

```
procesol: process (edo_presente, X) begin
    case edo_presente is
        when d0 => Z<= '0';
            if X = 1' then
                edo_futuro <= di;
            else
                edo_futuro <= d0;
            end if;
```

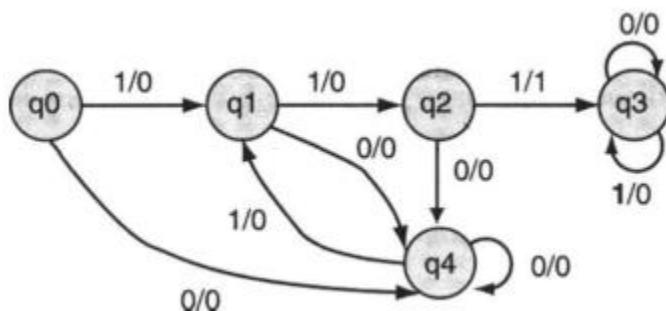
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity states is
4 port(clk,x: in std_logic;
5          z: out std_logic);
6 end states;
7
8 architecture diagram of states is
9 type estados is (d0,d1,d2,d3);
10 signal edo_presente,edo_futuro:estados;
11 begin
12 proceso1:process(edo_presente,x)
13 begin
14     case edo_presente is
15         when d0=> z <='0';
16             if x='1' then
17                 edo_futuro<=d1;
18             else
19                 edo_futuro<=d0;
20             end if;
21         when d1=> z <='0';
22             if x='1' then
23                 edo_futuro<=d2;
24             else

```

```
25         edo_futuro<=d1;
26      end if;
27      when d2=> z <='0';
28          if x='1' then
29             edo_futuro<=d3;
30          else
31             edo_futuro<=d0;
32          end if;
33      when d3=>
34          if x='1' then
35             edo_futuro<=d0;
36             z<='1';
37          else
38             edo_futuro<=d3;
39             z<='0';
40          end if;
41      end case;
42 end process proceso1;
43
44 proceso2:process(clk)
45 begin
46     if(clk'event and clk='1')then
47        edo_presente<=edo_futuro;
48     end if;
49 end process proceso2;
50 end diagram;
```

Ejemplo programa el siguiente diagrama de estados



```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity estado is
4 port(clk,x: in std_logic;
5          z:out std_logic);
6 end estado;
7
8 architecture diagrama of estado is
9 type estados is(q0,q1,q2,q3,q4);
10 signal edo_presente,edo_futuro:estados;
11 begin
12 proces01:process(edo_presente,x)
13 begin
14     case edo_presente is
15         when q0=> z <='0';
16             if x='0' then
17                 edo_futuro<=q4;
18             else
19                 edo_futuro<=q1;
20             end if;
21         when q1=> z <='0';
22             if x='0' then
23                 edo_futuro<=q4;
24             else

```

```
25         edo_futuro<=q2;
26      end if;
27  when q2=>
28      if x='0' then
29         edo_futuro<=q4;
30          z<='0';
31      else
32         edo_futuro<=q3;
33          z<='1';
34      end if;
35  when q3=> z <='0';
36      if x='0' then
37         edo_futuro<=q3;
38      else
39         edo_futuro<=q3;
40      end if;
41  when q4=> z <='0';
42      if x='0' then
43         edo_futuro<=q4;
44      else
45         edo_futuro<=q1;
46      end if;
47  end case;
48 end process proceso1;
49 proceso2:process(clk)
50 begin
51     if(clk'event and clk='1')then
52        edo_presente<=edo_futuro;
53     end if;
54 end process proceso2;
55 end diagrama;
```

Integración de entidades en VHDL

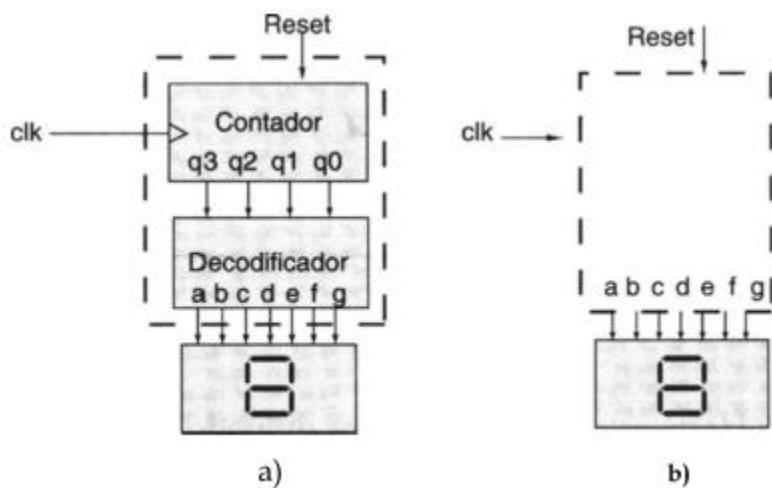
Esquema básico de integración de entidades

La integración de entidades puede realizarse mediante el diseño individual de cada bloque lógico a través de varios procesos internos que posteriormente pueden unirse mediante un programa común.

Otra posibilidad es observar y analizar de manera global todo el sistema evaluando su comportamiento sólo a través de sus entradas y salidas.

El caso 1 es el que ha estado manejando, desarrollar una entidad por sistema, el inconveniente principal con este caso es el número excesivo de terminales utilizadas en el dispositivo, debido a que al diseñar entidades individuales, se tendrían que declarar las terminales de entrada-salida de cada entidad.

En el segundo caso, cuando observamos el sistema como un todo el número de terminales de entrada y salida disminuye, por lo que nuestro trabajo es desarrollar no sólo un algoritmo interno capaz de interpretar el funcionamiento de cada bloque, sino también cómo conectar cada uno de ellos.



Caso 1 Programación de entidades individuales

Consideremos la programación del contador y decodificador de la figura anterior mediante la programación de entidades individuales.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity display is
5 port(clk,reset: in std_logic;
6       d: inout std_logic_vector(6 downto 0);
7       q: inout std_logic_vector(3 downto 0));
8 end display;
9
10 architecture contydec of display is
11 begin
12 process(clk,reset)
13 begin
14     if(clk'event and clk='1')then
15         q<=q+1;
16         if(reset='1' or q="1001") then
17             q<="0000";
18         end if;
19         end if;
20 end process;
21
22 process (q)
23 begin
24     case q is
25         when "0000"=> d <="1111110";
26         when "0001"=> d <="0110000";
27         when "0010"=> d <="1101101";
28         when "0011"=> d <="1111001";
29         when "0100"=> d <="0110011";
30         when "0101"=> d <="1011011";
31         when "0110"=> d <="1011111";
32         when "0111"=> d <="1110001";
33         when "1000"=> d <="1111111";
34         when "1001"=> d <="1111011";
35         when others=> d <="0000000";
36     end case;
37 end process;
38 end contydec;
```

Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	1	3
Clock/Inputs	1	2
I/O Macrocells	11	32
Buried Macrocells	0	32
PIM Input Connects	9	156
	22	/ 225 = 9 %

Programación mediante relación entradas/salidas

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity display2 is
5 port(clk,reset: in std_logic;
6       d: out std_logic_vector (6 downto 0));
7 end display2;
8
9 architecture relinout of display2 is
10 signal q: std_logic_vector (3 downto 0);
11 begin
12 process(clk,reset)
13 begin
14   if(clk'event and clk='1')then
15     q<=q+1;
16     if(reset='1' or q="1001")then
17       q<="0000";
18     end if;
19   end if;
20 end process;
21
22 process(q)
23 begin
24   case q is
```

```

25      when "0000"=> d <="1111110";
26      when "0001"=> d <="0110000";
27      when "0010"=> d <="1101101";
28      when "0011"=> d <="1111001";
29      when "0100"=> d <="0110011";
30      when "0101"=> d <="1011011";
31      when "0110"=> d <="1011111";
32      when "0111"=> d <="1110001";
33      when "1000"=> d <="1111111";
34      when "1001"=> d <="1111011";
35      when others=> d <="0000000";
36  end case;
37 end process;
38 end relinout;

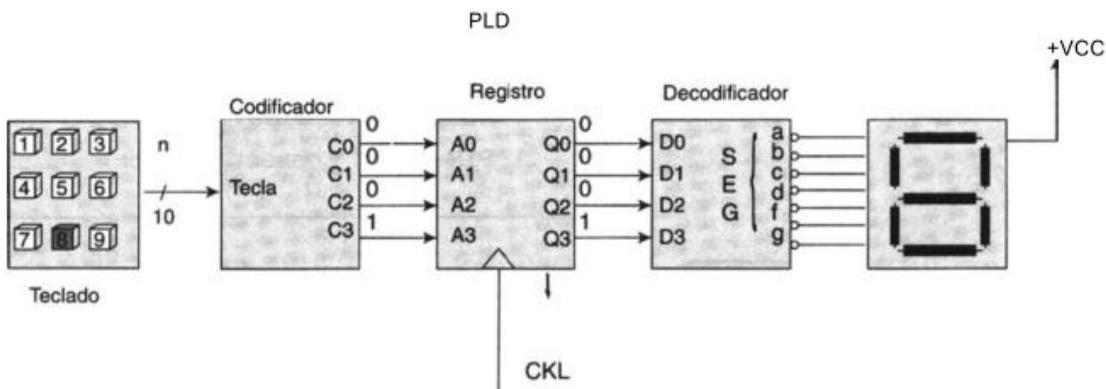
```

Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs 1 3		
Clock/Inputs 1 2		
I/O Macrocells 7 32		
Buried Macrocells 4 32		
PIM Input Connects 5 156		
18 / 225 = 8 %		

Programación de tres entidades individuales

Se muestra un circuito lógico formado por los siguientes subsistemas: teclado, codificador, registro, decodificador de siete segmentos activo en bajo y un display de siete segmentos. La finalidad de este sistema electrónico es observar en el display el número decimal equivalente al de la tecla presionada.



Para comenzar el usuario presiona la tecla decimal correspondiente al número que se desea que aparezca en el display. El codificador convierte en código BCD el equivalente al número decimal presionado.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity sistema is
4 port(clk: in std_logic;
5      tecla: in std_logic_vector(0 to 8);
6      --Salida del codificador
7      C: inout std_logic_vector (3 downto 0);
8      --Entrada del registro
9      A: inout std_logic_vector (3 downto 0);
10     --Salida del registro
11     Q: inout std_logic_vector (3 downto 0);
12     --Entrada del decodificador
13     D: inout std_logic_vector (3 downto 0);
14     --Salida del decodificador
15     seg: out std_logic_vector (0 to 6));
16 end sistema;
17
18 architecture tresentis of sistema is
19 begin
20     C<="0001" when (tecla="10000000") else
21         "0010" when (tecla="01000000") else
22         "0011" when (tecla="00100000") else
23         "0100" when (tecla="00010000") else
24         "0101" when (tecla="00001000") else
```

```

25      "0110" when (tecla="000001000") else
26      "0111" when (tecla="000000100") else
27      "1000" when (tecla="000000010") else
28      "1001";
29      A<=C;
30 process(clk,A,D)
31 begin
32     if(clk'event and clk='1')then
33         Q<=A;
34         D<=Q;
35     end if;
36     case d is
37         when "0000"=> seg <= "0000001";
38         when "0001"=> seg <= "1001111";
39         when "0010"=> seg <= "0010010";
40         when "0011"=> seg <= "0000110";
41         when "0100"=> seg <= "1001100";
42         when "0101"=> seg <= "0100100";
43         when "0110"=> seg <= "0100000";
44         when "0111"=> seg <= "0001110";
45         when "1000"=> seg <= "0000000";
46         when others => seg <= "0000100";
47     end case;
48 end process;
49 end tresentis;

```

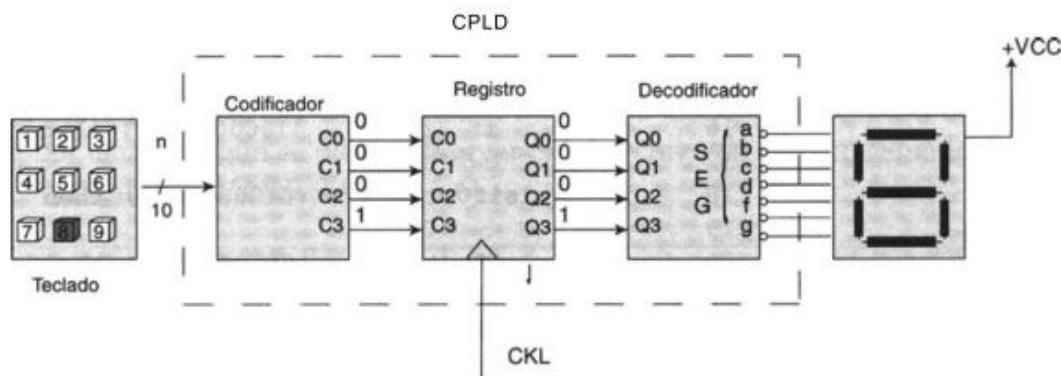
Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	28	32
Buried Macrocells	0	32
PIM Input Connects	24	156

57 / 225 = 25 %

Programación de entidades individuales mediante asignación de señales

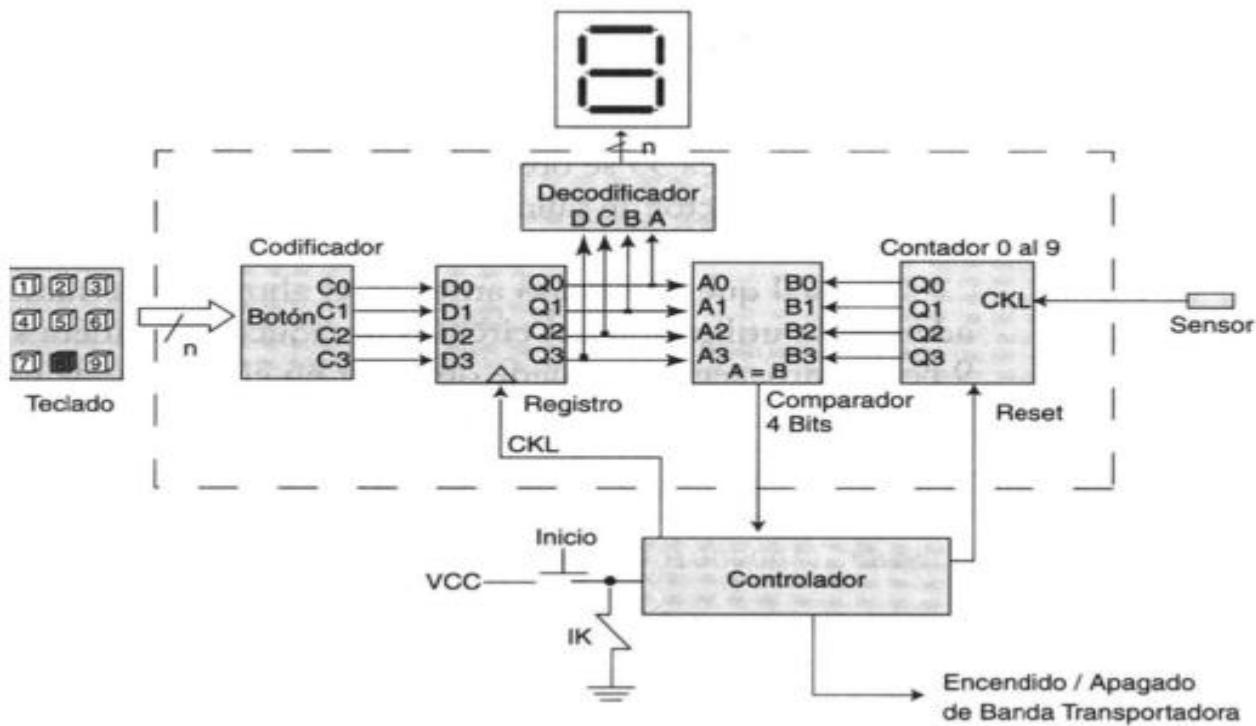
Imaginemos que los bloques de la siguiente figura se pueden ver como un solo módulo. En esta última figura se aprecia que las salidas de una entidad funcionan como las entradas de otra entidad. Esta forma de programación es posible siempre y cuando los bloques individuales se relacionen mediante señales internas.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity control is
4 port(clk: in std_logic;
5      --Entradas por el teclado
6      tecla: in std_logic_vector(0 to 8);
7      --Salidas del decodificador
8      seg: out std_logic_vector (0 to 6));
9 end control;
10
11 architecture asignation of control is
12 signal C: std_logic_vector(3 downto 0);
13 signal Q:std_logic_vector(3 downto 0);
14 begin
15 process (clk,tecla,C,Q)
16 begin
17     if(clk'event and clk='1')then
18         Q<=C;
19     end if;
20     if(tecla="100000000") then
21         C<="0001";
22     elsif(tecla="010000000") then
23         C<="0010";
24     elsif(tecla="001000000") then
```

```
25          C<="0011";
26      elsif(tecla="000100000") then
27          C<="0100";
28      elsif(tecla="000010000") then
29          C<="0101";
30      elsif(tecla="000001000") then
31          C<="0110";
32      elsif(tecla="000000100") then
33          C<="0111";
34      elsif(tecla="000000010") then
35          C<="1000";
36      else
37          C<="1001";
38      end if;
39      case Q is
40          when "0000"=> seg <="0000001";
41          when "0001"=> seg <="1001111";
42          when "0010"=> seg <="0010010";
43          when "0011"=> seg <="0000110";
44          when "0100"=> seg <="1001100";
45          when "0101"=> seg <="0100100";
46          when "0110"=> seg <="0100000";
47          when "0111"=> seg <="0001110";
48          when "1000"=> seg <="0000000";
49          when others=> seg <="0000100";
50      end case;
51  end process;
52 end asignation;
```

Ejemplo Realice la programación en VHDL para integrar los diferentes bloques lógicos de la siguiente figura

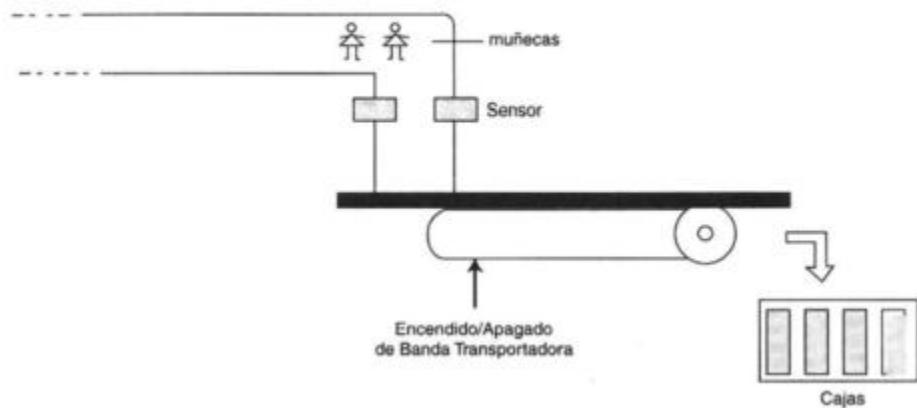


El circuito de la siguiente figura se utiliza para automatizar el proceso de empaquetamiento de muñecas de porcelana. Considere que las muñecas se pueden empaquetar en forma individual o con un máximo de nueve unidades por paquete.

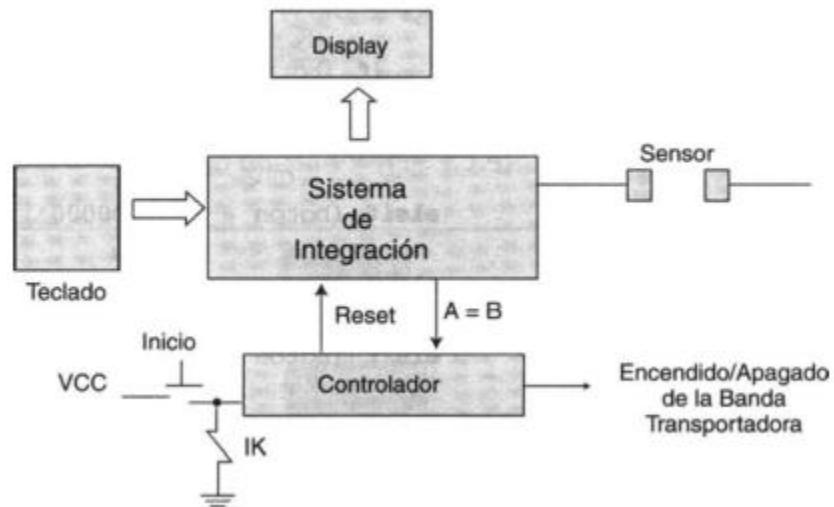
En un inicio el operador selecciona mediante el teclado decimal la cantidad de piezas que va a empaquetar.

Después de seleccionar la cantidad de muñecas por empaquetar, el operador presiona el botón de inicio, que desencadena una serie de acciones controladas por el bloque denominado controlador. Comienza con una señal de salida llamada reset, que coloca el contador binario en el estado cero, enseguida envía la señal de arranque al motor que controla el avance de la banda transportadora.

Cada vez que una de las muñecas colocadas sobre la banda transportadora pasa por el "sensor" se origina un impulso eléctrico (pulso) que hace que el contador aumente en uno su conteo. Este procedimiento continúa incrementando el contador en una unidad, hasta el momento en que la cifra actual del contador B es igual a A ($A=B$) con lo cual este último envía una señal al bloque controlador que detiene la banda transportadora (Apagado) y marca el fin del proceso.



En la siguiente figura se observa el intercambio de señales que se realiza entre el controlador y cada uno de los bloques lógicos del sistema.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity control is
5 port(clk,reset: in std_logic;
6      boton: in std_logic_vector (0 to 8);
7      sensor: in std_logic;
8      dec: out std_logic_vector (0 to 6);
9      compara: out std_logic);
10 end control;
11
12 architecture sistema of control is
13 signal Q,C,R: std_logic_vector(3 downto 0);
14 begin
15 proces01:process(sensor,reset,Q)
16 begin
17     if(sensor'event and sensor='1')then
18         Q<="0000";
19         Q<=Q+1;
20     if(reset='1' or Q="1001")then
21         Q<="0000";
22     end if;
23     end if;
24 end process proces01;
```

```
25 proceso2:process(clk,boton,R)
26 begin
27     if(clk'event and clk='1')then
28         R<=C;
29     end if;
30     if(boton="100000000")then
31         C<="0001";
32     elsif(boton="010000000")then
33         C<="0010";
34     elsif(boton="001000000")then
35         C<="0011";
36     elsif(boton="000100000")then
37         C<="0100";
38     elsif(boton="000010000")then
39         C<="0101";
40     elsif(boton="000001000")then
41         C<="0110";
42     elsif(boton="000000100")then
43         C<="0111";
44     elsif(boton="000000010")then
45         C<="1000";
46     else
47         C<="1001";
48     end if;
```

```

49    case R is
50        when "0000" => dec <= "00000001";
51        when "0001" => dec <= "1001111";
52        when "0010" => dec <= "0010010";
53        when "0011" => dec <= "0000110";
54        when "0100" => dec <= "1001100";
55        when "0101" => dec <= "0100100";
56        when "0110" => dec <= "0100000";
57        when "0111" => dec <= "0001110";
58        when "1000" => dec <= "0000000";
59        when others => dec <= "0000100";
60    end case;
61 end process proceso2;
62 compara <='1' when Q=R else '0';
63 end sistema;

```

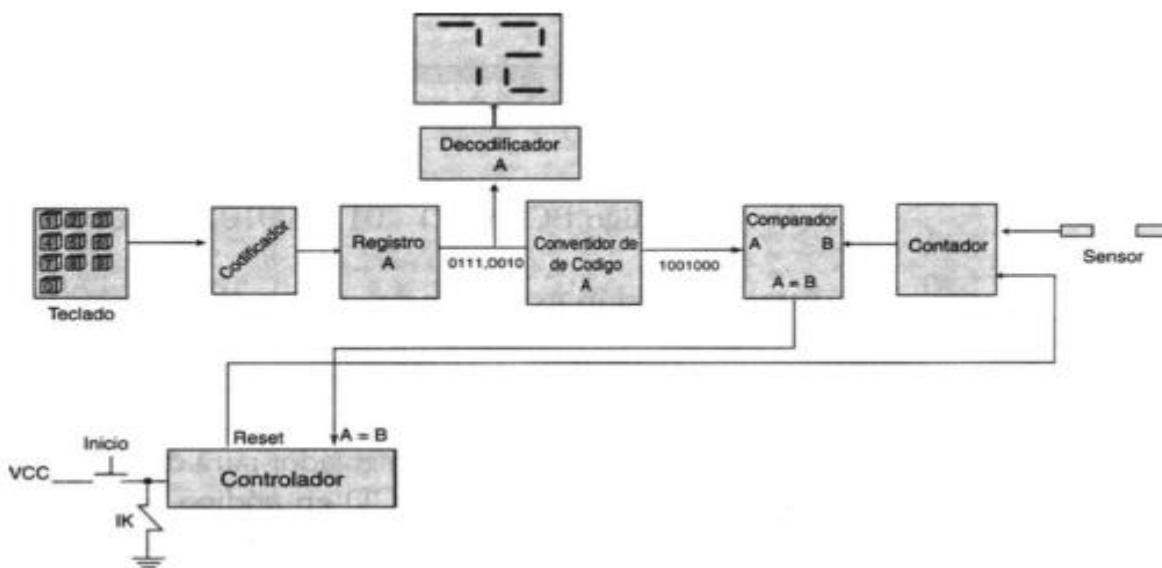
Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	15	32
Buried Macrocells	8	32
PIM Input Connects	18	156
<hr/>		
	46	/ 225 = 20 %

Observe la integración de entidades que forman el sistema sólo precisó dos procesos: el primero (línea 15), para describir el conteo de las muñecas mediante la variable de entrada del sensor; el segundo (línea 25), para integrar el codificador, registro y decodificador de siete segmentos.

En la línea 62 se asigna a la variable compara el valor de 1 cuando el valor binario de los vectores Q y R son iguales. Esto con objeto de detener la banda transportadora.

Ahora se considera el caso en el que el número máximo de muñecas que se puede empaquetar pasó de 9 a 99.



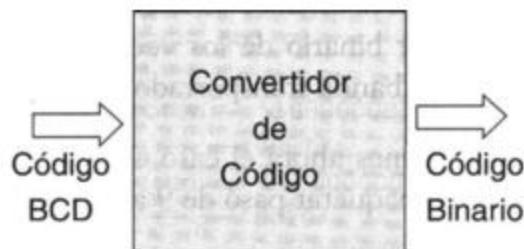
El operador selecciona mediante el teclado decimal la cantidad de muñecas por empaquetar. Como sabemos este número decimal se convierte en código BCD mediante el circuito codificador y luego pasa vía el registro hacia los decodificadores de siete segmentos para observar en el display el valor del número seleccionado. En este caso el operador puede elegir un número de dos dígitos.

Como ejemplo el operador teclea el número decimal 72 mediante la pulsación del 7 y el 2, estos números codificados en BCD proporcionan el siguiente código:

$$\text{BCD}(7)=0111$$

$$\text{BCD}(2)=0010$$

El vector de entrada será de 8 bits (01110010). Este valor pasa vía registro a los decodificadores de siete segmentos para visualizar en el display las cifras correspondientes a este valor. Note que el número BCD(72) es introducido en un nuevo bloque denominado convertidor de código que tiene como función convertir código BCD en código binario.



Esta conversión es necesaria, ya que el contador incrementa su valor en forma binaria cada vez que sensa una muñeca. Con esto se intuye que la función del comparador es relacionar el valor binario proveniente del teclado y el valor que proporciona el contador; cuando estos números son iguales ($A=B$) se envía un pulso al controlador para detener la banda transportadora.

La conversión de código BCD en código binario puede realizarse a través de circuitos sumadores. Básicamente para obtener un número binario a partir de un número BCD hay que sumar los números binarios que representan los pesos de los bits del número BCD.

Por ejemplo, consideramos que tenemos el número BCD (72)

B3	B2	B1	B0	
0	1	1	1	7

A3	A2	A1	A0	
0	0	1	0	2

El grupo de 4 bits a la izquierda representa 70, y el grupo de 4 bits a la derecha representa el número 2. Esto quiere decir que el grupo a la izquierda tiene un peso de 10 y el de la derecha tiene un peso de 1. En cada grupo el peso binario es el siguiente:

Digito de las decenas				
Peso	80	40	20	10
Des. Bit	B3	B2	B1	B0

Digito de las unidades				
Peso	8	4	2	1
Des. Bit	A3	A2	A1	A0

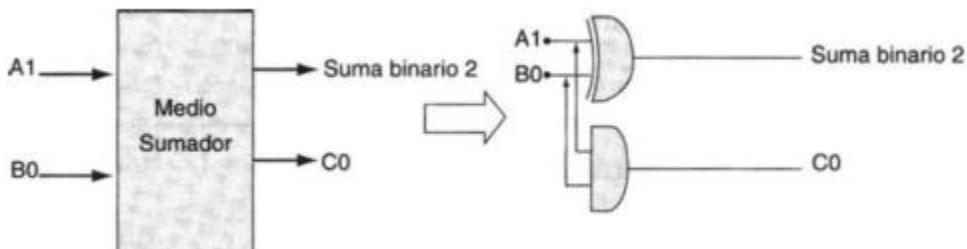
El equivalente binario de cada bit BCD es un número binario que representa el peso de cada bit dentro del número BCD completo.

En la siguiente tabla se muestra que el valor del bit BCD A0 tiene un peso de 1, por lo cual su valor binario representado por 7 bits es 0000001. De igual manera el bit B3 tiene un peso de 80 por lo que el valor binario correspondiente representado en 7 bits es 1010000. Observe que este valor se obtiene mediante la suma de (64+16) en notación binaria.

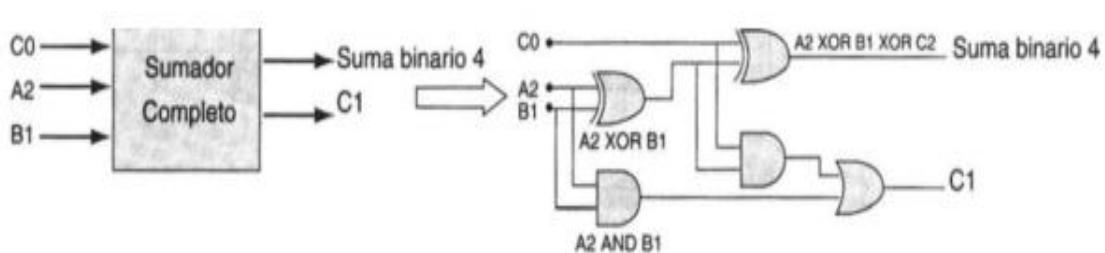
Bit BCD	Peso BCD	Representación binaria						
		64	32	16	8	4	2	1
A0	1	0	0	0	0	0	0	1
A1	2	0	0	0	0	0	1	0
A2	4	0	0	0	0	1	0	0
A3	8	0	0	0	1	0	0	0
B0	10	0	0	0	1	0	1	0
B1	20	0	0	1	0	1	0	0
B2	40	0	1	0	1	0	0	0
B3	80	1	0	1	0	0	0	0

Se puede construir un circuito lógico mediante sumadores que realicen la suma correspondiente a los unos necesarios para la transformación de un número BCD en binario.

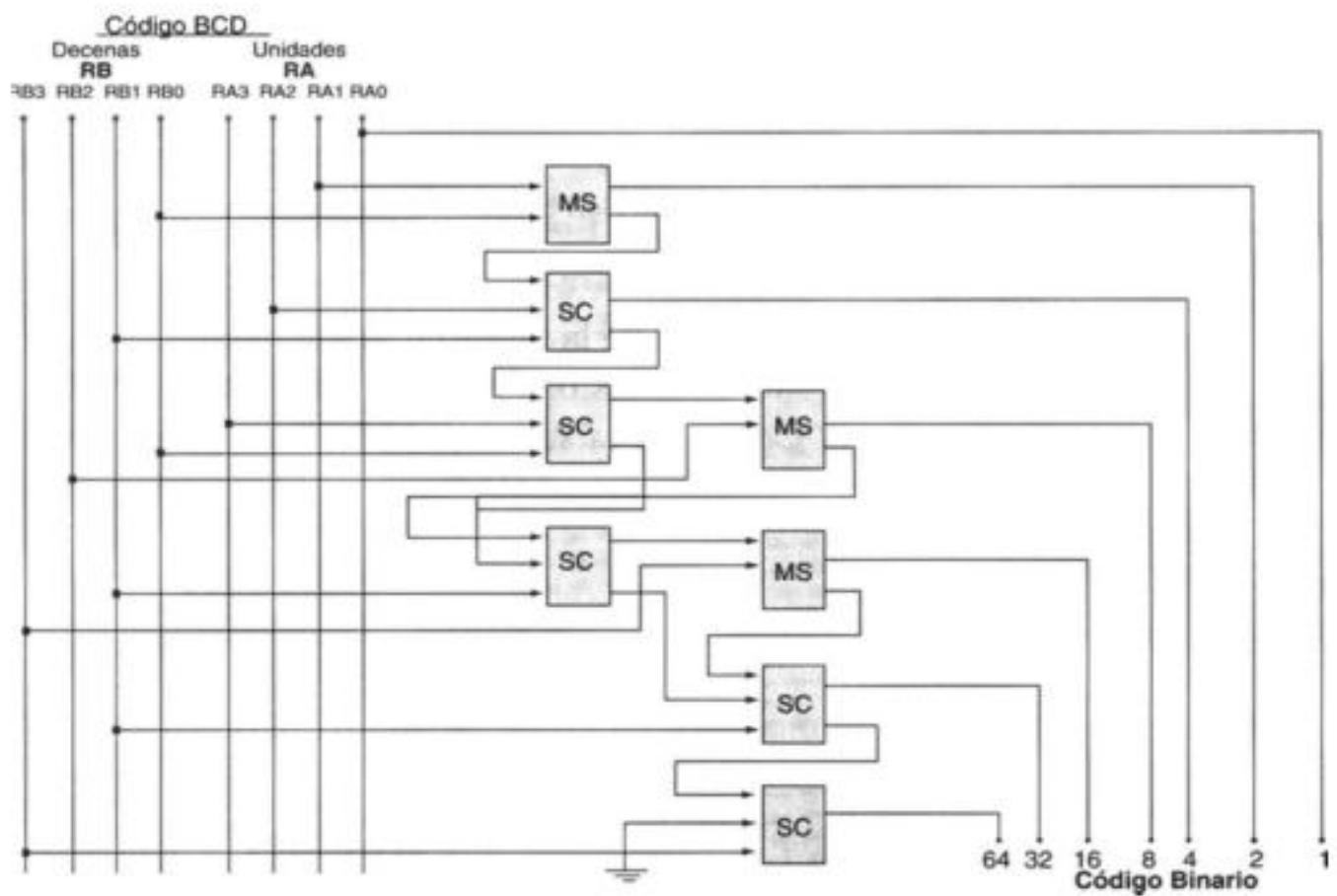
En la columna “2” de la representación binaria debe realizarse la suma de los bits A1 Y B0 del número BCD. Esta suma puede efectuarse a través de un circuito medio sumador.



En la columna “4” de la representación binaria, la posible ocurrencia de dos unos ocurre sumando los bits A2 y B1 del número BCD y el acarreo de la suma anterior. La suma tiene lugar a través de un sumador completo.



El sistema sumador que realiza esta conversión de BCD a binario es:



La secuencia de acciones que realiza un controlador puede describirse mediante los diagramas de estado o a través de un diagrama diseñado específicamente para definir algoritmos de hardware denominado carta ASM. La ventaja principal de una carta ASM con respecto a los diagramas de estado es que permite controlar y especificar el flujo de la información al mismo tiempo.

Hay que recordar que un algoritmo describe paso a paso del comportamiento de un sistema, tomando en consideración las siguientes características:

- El algoritmo debe ser finito. Debe tener un número determinado de estados.
- Tiene que ser definido. En cada estado debe establecerse por completo todas las acciones que se llevan a cabo. Esto incluye las entradas, salidas y decisiones que conducen a ese estado.

Algoritmo de la máquina de estado (ASM)

El algoritmo o carta ASM utiliza 3 símbolos básicos para describir el comportamiento de un sistema:

- Bloque de estado
- Bloque de decisión
- Bloque de salida condicional

Bloque de estado

El bloque de estado representa el “estado” de una maquina secuencial y debe contener la siguiente información:



- Nombre del estado: Números o letras
- Código del estado (“xxxx”). Se refiere al código binario asignado al estado.
- Lista de salidas: señales de salida asignadas al estado y que sólo se encuentran activas durante el tiempo que permanezca el sistema en ese estado.

Bloque de decisión

El rombo o bloque de decisión se refiere a las variables de entrada al sistema y contienen la siguiente información:



- Una variable de entrada: en este rombo se indica el nombre de la variable de entrada
- Una salida verdadera
- Una salida falsa

Bloque de salidas condicionales

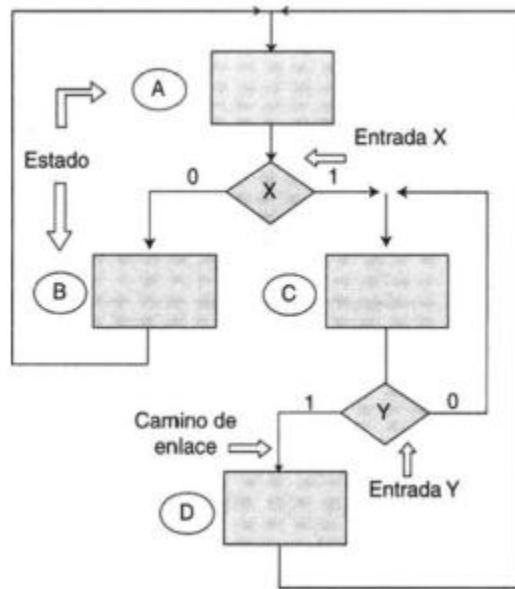
El bloque de salidas condicionales se utiliza para activar señales de salida que sólo se encuentran disponibles para ciertas condiciones de entrada. La información contenida en dicho bloque es la siguiente:



- Una lista de salidas condicionales que dependen de cierta condición de entrada.

Estructura de una carta ASM

Una carta ASM consiste de uno o más bloques ASM interconectados de una manera consistente



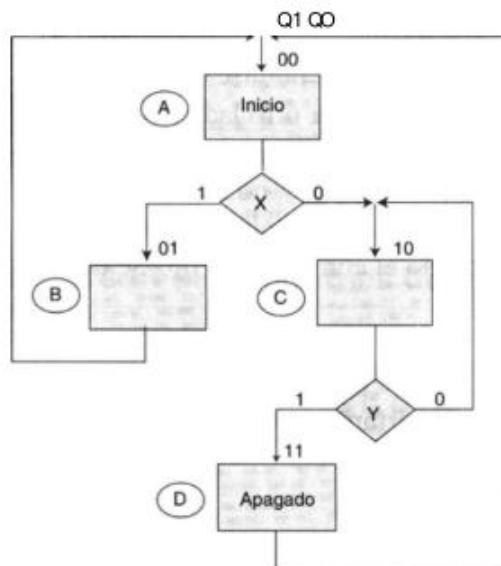
En la figura se aprecian 4 estados (A,B,C,D) y dos estradas (X,Y).

La transición de un bloque a otro se realiza a través de líneas denominadas caminos de enlace.

En las cartas ASM, a cada bloque le corresponde una unidad de tiempo y en este lapso se ejecutan todos los bloques de decisión y de salidas condicionales que estén asociados con el mismo estado.

Descripción de una carta ASM

Para describir de manera general los atributos de una carta ASM consideremos la siguiente figura



La carta ASM está formada por cuatro estados (A,B,C,D) cada uno con un código binario respectivo.

A=00

B=01

C=10

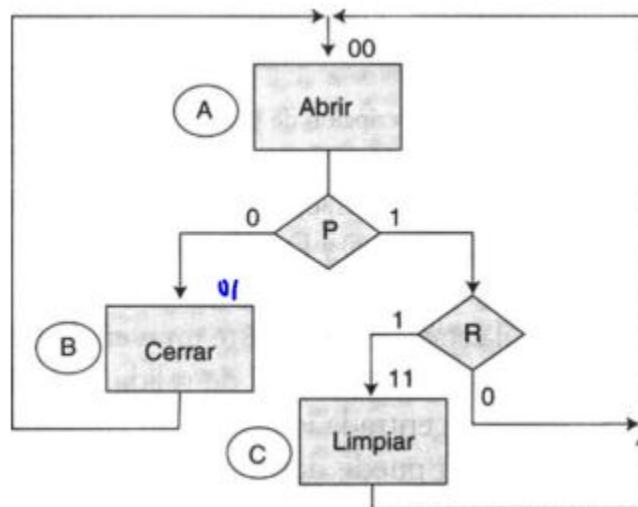
D=11

El estado A tiene una señal de salida denominada Inicio, mientras que en el estado D la señal de salida se define como Apagado. La carta tiene dos señales de entrada identificadas por las variables X y Y.

Esta carta puede describirse mediante la tabla:

Estado presente Q1 Q0			Entradas X Y	Estado Futuro (Q1, Q0) T + 1	Salidas
1	A	0 0	1 *	0 1	Inicio
2		0 0	0 *	1 0	Inicio
3	B	0 1	* *	0 0	
4	C	1 0	* 0	1 0	
5		1 0	* 1	1 1	
6	D	1 1	* *	0 0	Apagado

Observa la carta ASM de la siguiente figura y elabora una tabla que describa su comportamiento

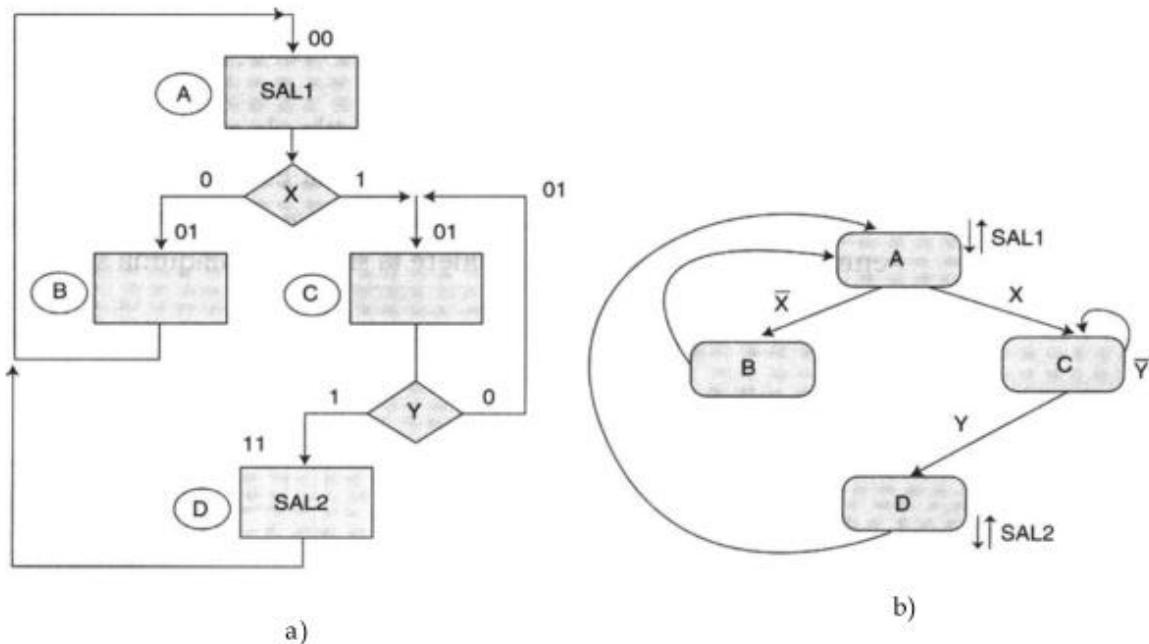


Estado presente				Entradas	Estado Futuro		Salidas
	Q1	Q0		P	R	(Q1, Q0) T + 1	
1	A	0	0	0	*	0 1	Abrir
2		0	0	1	0	0 0	
3		0	0	1	1	1 1	
4	B	0	1	*	*	0 1	Cerrar
5	C	1	1	*	*	0 0	Limpiar

Cartas ASM en comparación con las máquinas de estado

Es relativamente fácil pasar de una carta ASM a un diagrama (máquina) de estados y viceversa.

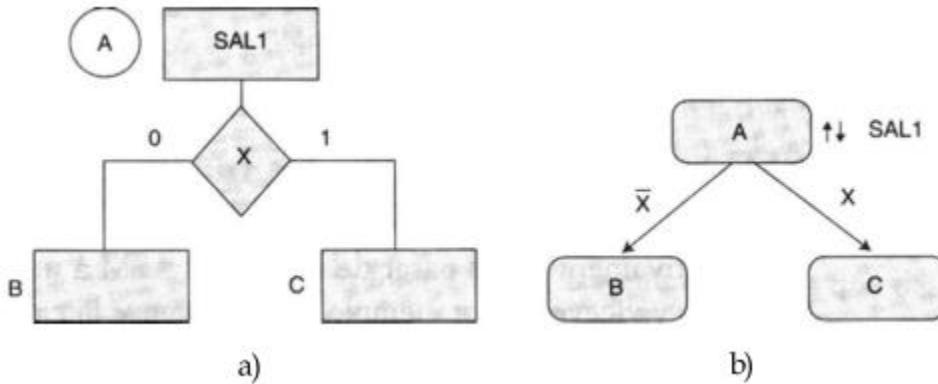
Consideremos las siguientes



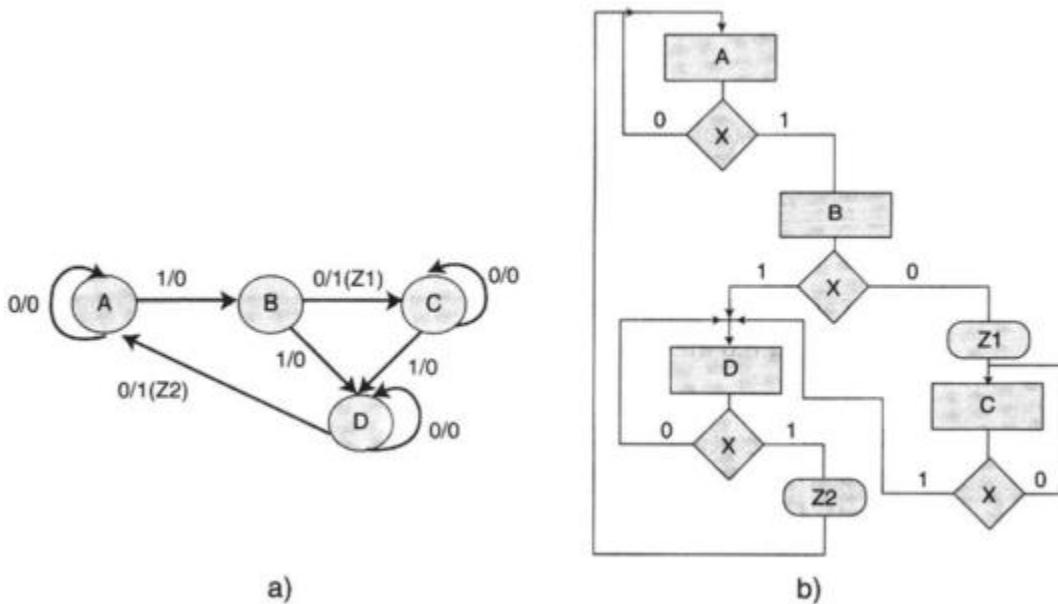
Esta carta ésta carta está formada por cuatro estados (A,B,C,D) y dos salidas (SAL1, SAL2).

Estas señales de salida se encuentran activas sólo mientras el sistema se encuentre en el estado presente (estructura de Moore).

En la figura b) se muestra la máquina de estados correspondiente, según se aprecia, existen también cuatro estados y la unión entre ellos se presenta a través de sus líneas de entrada (X y Y). La negación de la variable indica la condición cuando las entradas son igual a 0 o a 1. Observe las siguientes figuras y cómo se denotan las señales de salida

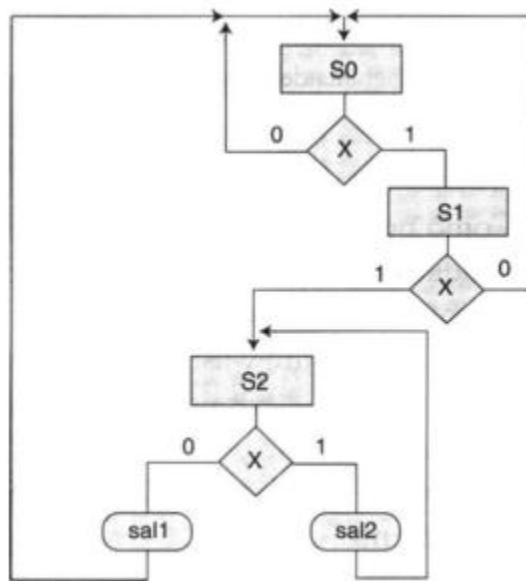
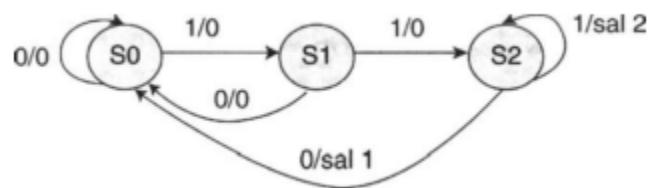


Hasta este momento no se ha especificado el uso de las salidas condicionadas en la carta ASM; éstas se usan en la estructura de Mealy cuando la salida depende no sólo del estado en el que se encuentra, sino también de sus entradas.



Se puede observar que cuando la máquina se encuentra en el estado B, si la señal de entrada vale 0 ($0/1(Z1)$) se transfiere al estado C y durante este enlace se activa la señal de salida $Z1=1$. Igual sucede cuando se halla en el estado D y se dirige hacia el estado A, la señal de salida se activa en $Z2=1$ cuando $Z2=1$, la carta ASM correspondiente a esta máquina se encuentra en la figura b) de la figura anterior.

Convierte la siguiente máquina de Mealy en una carta ASM

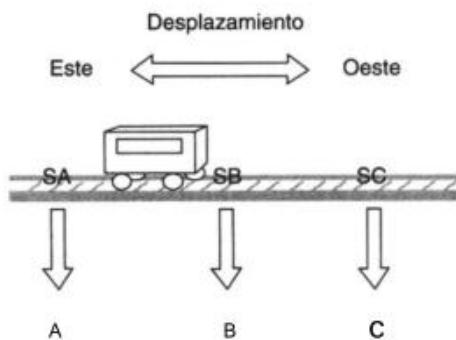


Diseño de controladores mediante cartas ASM

El diseño de cartas ASM tiene mucho que ver con la forma y sentido común de interpretar un problema, para luego visualizar un camino que permita encontrar la solución óptima.

Situación: se nos solicita diseñar un controlador que permita automatizar el funcionamiento de un tren que debe desplazarse de una estación a otra. En cada estación se han colocado sensores que detectan cuando el tren se aproxima al andén y envían al vagón una señal denominada PARO.

Al recibir dicha señal, el vagón activa su sistema de frenado y el tren comienza a detenerse en forma automática hasta detenerse y colocarse en los límites de la estación.



En la figura anterior se muestran tres estaciones denominadas A, B y C con sendos sensores cada una (SA,SB y SC). Consideremos que cuando el tren se encuentra detenido, abre sus puertas automáticamente y así las conserva durante 15 segundos para permitir la entrada y salida de pasajeros; luego las cierra y continúa su camino hacia la siguiente estación.

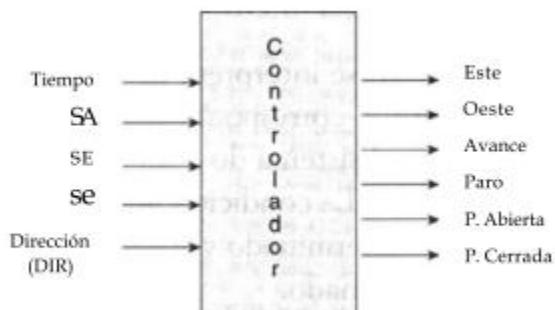
Con base en la descripción anterior podemos inferir algunos aspectos de funcionamiento:

- 1) El tren debe poder moverse de la estación A a la C y viceversa, no obstante, no se establece con precisión dónde inicia su recorrido. Si la trayectoria se realiza de A a C y viceversa se produciría un algoritmo con bastantes estados, por lo cual sería conveniente situarse en la estación B y de ahí desplazarse hacia la dirección ESTE (estación A} o dirección OESTE (estación C).
- 2) También se interpreta que cuando el tren se aproxima a una estación, el sensor correspondiente envía al vagón una señal de PARO que activa el sistema de frenado y detiene el tren justo en los límites de la estación. La condición establece que el sistema de frenado está predeterminado y sólo basta la señal PARO para iniciar su secuencia de frenado.
- 3) En la descripción del problema también puede interpretarse que el tiempo de ascenso/descenso de pasajeros se marca mediante un controlador de tiempo independiente al controlador del sistema.

En el diseño de controladores es importante identificar con claridad la parte de control y los subsistemas periféricos que interactúan con él.

Desde esta perspectiva es posible ubicar el controlador como una caja negra que permite identificar con claridad el intercambio de señales de entrada y salida que se realizan entre él y sus subsistemas.

En la siguiente figura se puede ver



Se observan las señales de entrada/salida que interactúan en el controlador. La función de estas señales se explica a continuación:

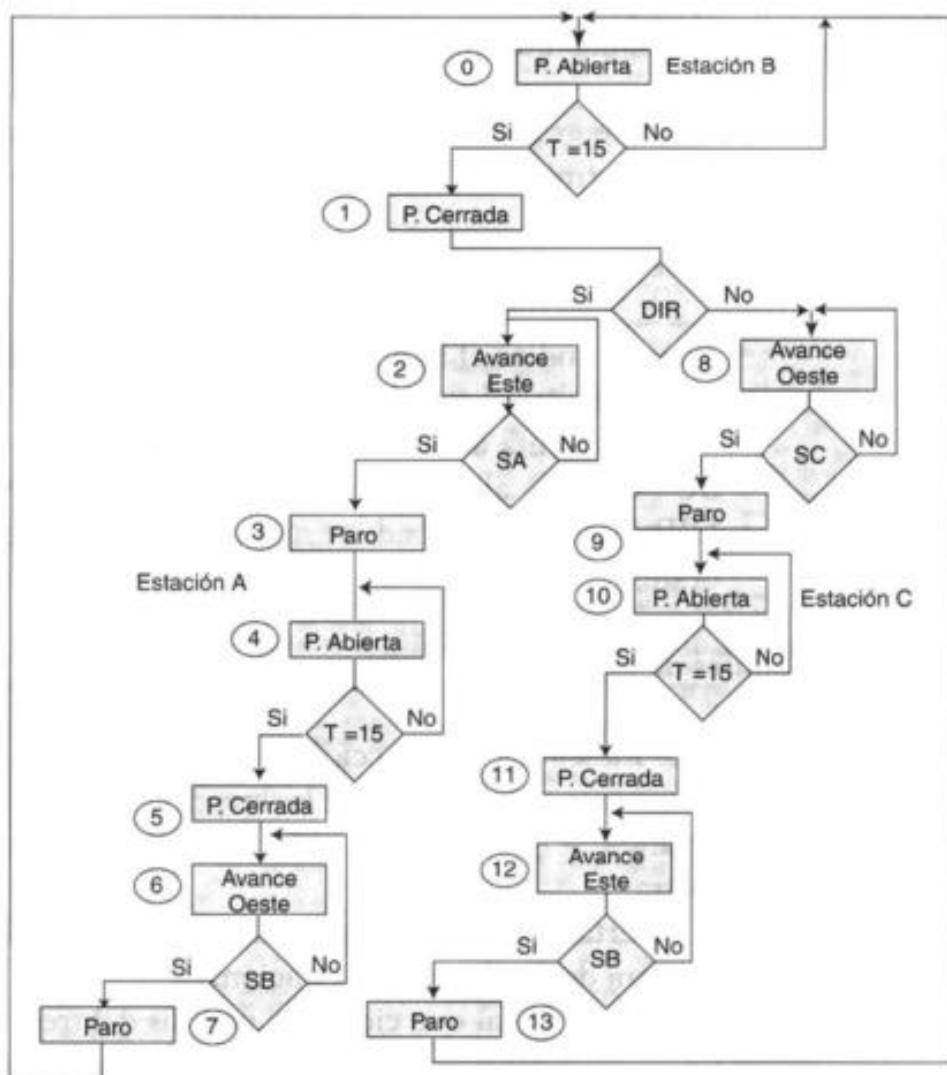
Señales de entrada

- Tiempo indica el lapso destinado al ascenso o descenso de pasajeros.
- SA Simboliza al sensor colocado en la estación A.
- SB representa al sensor ubicado en la estación B.
- SC simboliza al sensor colocado en la estación C.
- DIR indica hacia donde se moverá el tren (este u oeste).

Señales de salida

- Este. Indica que la dirección del tren será hacia el este.
- Oeste. Esta señal indica que la dirección que tomará el tren será al oeste
- Avance. Se envía para que el tren realice su recorrido de una estación a otra.
- P.abierta. Señal que permite abrir las puertas del tren cuando se encuentra detenido en una estación.
- P. cerrada. Señal que cierra las puertas del tren una vez que el tiempo de ascenso/descenso de pasajeros ha finalizado.
- Paro. Señal que al ser recibida por el tren activa su sistema de frenado para que se detenga lentamente en los límites de una estación.

Un posible algoritmo de control “carta ASM” que permite detallar de manera didáctica el comportamiento del sistema es el siguiente



Descripción de algoritmos de control

Estado 0. En un inicio el controlador se encuentra en la estación B y la puerta del vagón se mantiene abierta. Si la entrada de tiempo($T=15$ s) no ha alcanzado su valor final., la puerta continúa abierta ($T=\text{No}$), si ya alcanzo el valor de $T=15$ s, la puerta se cierra y el controlador se transfiere al estado 1.

Estado 1. Con la puerta cerrada se elige la dirección a través de la entrada DIR, es decir,

Si $\text{DIR}=\text{Si}$ el vagón se dirige hacia el este (estación A)

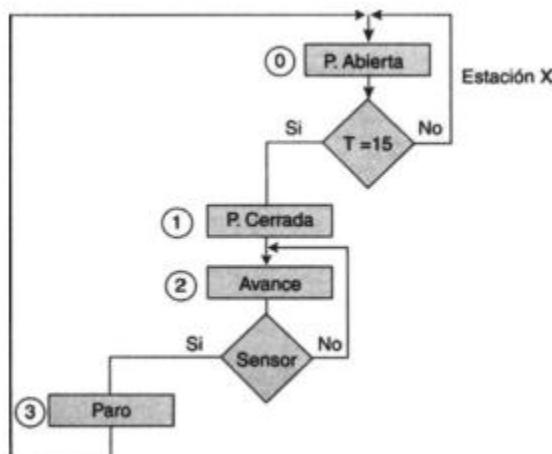
Si $\text{DIR}=\text{No}$ el vagón se dirige hacia el Oeste (estación B)

Estado 2. En el estado 2 se da la orden para que el tren continúe su movimiento a través de la señal de avance. Si el sensor SA no detecta la presencia del vagón la señal de avance permanece inalterable; si el sensor SA detecta al vagón, el controlador ($\text{SA}=\text{si}$) envía una señal de paro (estado 3), que detiene de forma automática el tren justo en los límites del andén de la estación A.

Estado 4. Con el vagón detenido en la estación A, el controlador envía la señal de puerta abierta, que permite la entrada y salida de pasajeros mientras el tiempo $T=15s$ no se cumpla, el tren permanece en la estación A con las puertas abiertas, en caso contrario las cierra y continúa su avance a la estación B.

Se puede deducir con relativa facilidad la descripción general de la carta, en la cual se observaron 14 estados, del estado 0 al estado 13.

En el diseño de algoritmos de control ASM tiene mucho que ver el sentido común y la experiencia del diseñador. Por ejemplo, en la carta anterior se puede considerar que los sensores SA, SB y SC actúan de forma similar, por lo cual en lugar de representar tres sensores basta utilizar solo uno (S), dado que el vagón es incapaz de saber en qué estación se encuentra, en consecuencia, sólo podrá tenerse un estado (X) que representa la estación actual. Al efectuar estos arreglos, la carta anterior podría simplificarse de la siguiente manera.



Diseño de cartas ASM mediante VHDL

Desarrollar un programa en VHDL a través de una carta ASM es relativamente fácil; sin embargo no es posible recomendar, un procedimiento determinado para programar.

Descripción:

Se requiere diseñar una máquina despachadora de refrescos, la cual está formada por tres módulos (subsistemas) independientes. Cada módulo realiza una función predeterminada, pero hay que diseñar el sistema controlador que gobierne y sincronice cada acción de los subsistemas. A continuación se describe cada subsistema:

- a) Módulo de recolección de monedas: Recibe las monedas que el cliente introduce en la máquina para obtener el refresco.

Características:

- Acepta monedas de \$5, \$10 y \$20
- Cuenta con un mecanismo que rechaza monedas defectuosas.
- Posee un mecanismo de detección de valor de la moneda, es capaz de discriminar el valor de la moneda Menor que precio (MP) e igual a precio (Precio).

El sistema recibe dos señales de entrada denominadas Limpieza y Captura. La primera limpia el sistema y lo deja en condiciones de Inicio. La segunda recolecta las monedas que ingresó el cliente.

- b) Módulo de devolución de monedas: Proporciona el cambio al cliente cuando introduce monedas cuyo monto excede el precio del producto.

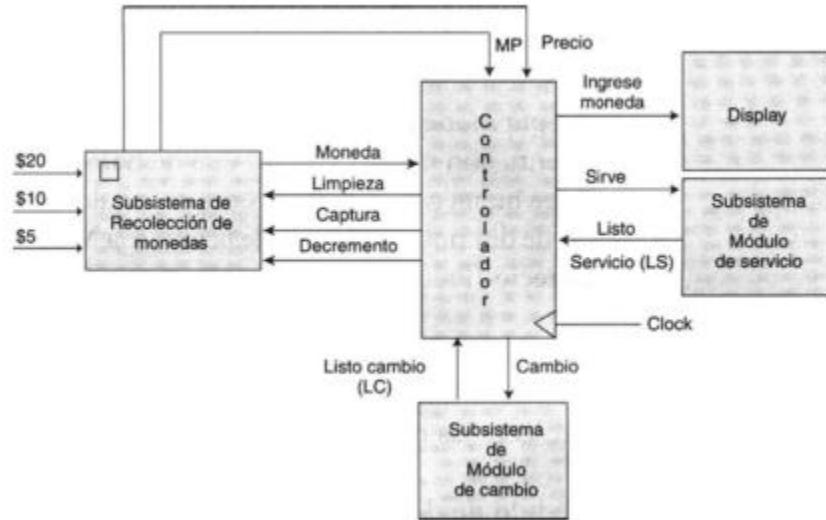
Características

- Activa una señal de salida denominada Cambio, cuya función es entregar cambio al cliente en monedas de \$5.
- Recibe una señal denominada Listo cambio (LC), que indica cuándo se han dado \$5 de cambio.
- c) Módulo de servicio: Su función es entregar el refresco al cliente; sin embargo, el producto sólo se libera cuando la cantidad que proporcionó el cliente es igual al valor del refresco.

Características

- Cuenta con una señal de salida denominada sirve que activa el sensor correspondiente para que el refresco se pueda servir y entregar.
- Posee una señal de entrada denominada Listo servicio (LS) que indica cuando se entregó el refresco.
- d) Controlador: Esta unidad sincroniza las acciones de los diferentes módulos para automatizar el funcionamiento de la máquina despachadora de refrescos

En la siguiente figura se observa cada módulo y la señalización de entrada/salida que existe entre cada subsistema.



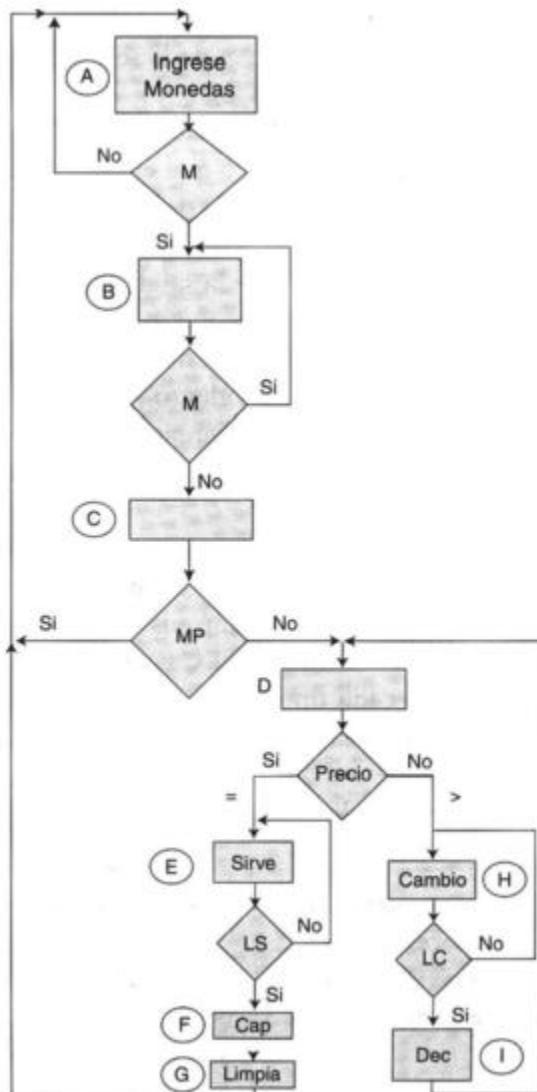
Un posible algoritmo ASM es:

Estado A: Se tiene una salida denominada Ingrese monedas, que no es sino una señal que indica al cliente a través de un display que la máquina se encuentra funcionando. Si la señal de entrada Moneda es igual a “No” ($M=no$), la máquina permanece en el estado A; si la señal Moneda es igual a “si” la carta conduce al estado B.

Estado B: La señal de entrada moneda (M) se utiliza para confirmar que el sensor encargado de esta detección ha realizado toda su rutina.

El sensor detector de monedas se encuentra en estado de reposo, para luego pasar a su estado de activación y regresar al reposo de nuevo.

Estado C: La señal de entrada Menor que precio (MP) se utiliza para indicar al cliente que debe suministrar más monedas; si $MP=yes$ se vuelve al estado A y se solicita al cliente que ingrese monedas, si $MP=no$ se pasa al estado D, donde se pregunta si la cantidad ingresada es igual o mayor que el precio.



Estado D: En este estado se pregunta si la cantidad ingresada es igual al precio si lo es (Precio=sí) se pasa al estado E donde se activa la salida, Sirve, que otorga el servicio. En el estado E la señal Listo servicio (LS) se utiliza para indicar al controlador que el sensor ha detectado que ya se entregó el refresco: (LS=Sí), en caso contrario (LS=No), se repite la petición de servicio. Si éste ya se brindó, la carta se transfiere al estado F para capturar el dinero (Cap) y luego el estado G para limpiar el sistema (Limpia).

Al volver al estado D se observa que si Precio= No equivale a decir que el precio es mayor al valor del refresco; en consecuencia, el algoritmo debe devolver cambio a través de la señal Cambio en el estado H.

Estado H: La señal Listo cambio (LC) se utiliza para indicar al controlador si el sensor ha detectado que el cambio se dio (LC=Sí) ; en caso contrario (LC=No), se repite la petición del cambio.

Estado I: La función de salida Decrementa (Dec) es indicar al módulo de recolección de monedas que se entregaron \$5 de cambio, con el objeto de que compare y determine si con el decremento de \$5 la cantidad abonada por el producto ya es igual al precio o sigue siendo mayor. Observe cómo el camino de enlace de estado I retroalimenta al estado D en que se pregunta si el pago por el producto es igual o mayor al precio.

Como puede observarse, para describir controladores se requiere conocer a la perfección el funcionamiento de cada uno de sus subsistemas así como la función que realizan sus sensores.

La instrucción case en VHDL puede ayudar a simplificar lo que sucede en cada uno de los estados; de igual manera, el uso de if o elsif puede auxiliar a manejar con simplicidad los rombos de decisión.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity maquina is
5 port(CLK,MONEDA,MP,PRECIO,LC,LS: in std_logic;
6      CAP,LIMPIA,SIRVE,CAMBIO,DEC: out std_logic);
7 end maquina;
8
9 architecture despacha of maquina is
10 type estados is(A,B,C,D,E,F,G,H,I);
11 signal edo_pre,edo_fut: estados;
12 begin
13 p_estados: process(edo_pre,MONEDA,MP,PRECIO,LC,LS)
14 begin
15     case edo_pre is
16         when A=> CAP <='0';
17             LIMPIA<='0';
18             SIRVE<='0';
19             CAMBIO<='0';
20             DEC<='0';
21             if MONEDA='1' then
22                 edo_fut<=B;
23             else
24                 edo_fut<=A;
```

```
25           end if;
26       when B=> CAP <='0';
27           LIMPIA<='0';
28           SIRVE<='0';
29           CAMBIO<='0';
30           DEC<='0';
31           if MONEDA='0' then
32              edo_fut<=C;
33           else
34              edo_fut<=B;
35           end if;
36       when C=> CAP <='0';
37           LIMPIA<='0';
38           SIRVE<='0';
39           CAMBIO<='0';
40           DEC<='0';
41           if MP='0' then
42              edo_fut<=D;
43           else
44              edo_fut<=A;
45           end if;
46       when D=> CAP <='0';
47           LIMPIA<='0';
48           DEC<='0';
```

```
49          if PRECIO='0' then
50             edo_fut<=H;
51             CAMBIO<='1';
52          else
53             edo_fut<=E;
54             SIRVE<='1';
55          end if;
56      when E=> LIMPIA <='0';
57          CAMBIO<='0';
58          DEC<='0';
59          if LS='0' then
60             edo_fut<=E;
61             SIRVE<='1';
62          else
63             edo_fut<=F;
64             CAP<='1';
65          end if;
66      when F=> LIMPIA <='0';
67          SIRVE<='0';
68          CAMBIO<='0';
69          DEC<='0';
70         edo_fut<=G;
71          CAP<='1';
72      when G=> CAP <='0';
```

```
73          SIRVE<='0';
74          CAMBIO<='0';
75          LIMPIA<='1';
76          DEC<='0';
77          edo_fut<=A;
78      when H=> CAP <='0';
79          SIRVE<='0';
80          LIMPIA<='1';
81          if LC='0' then
82              edo_fut<=H;
83              CAMBIO<='1';
84          else
85              edo_fut<=I;
86              DEC<='1';
87          end if;
88      when I=> CAP <='0';
89          SIRVE<='0';
90          LIMPIA<='1';
91          CAMBIO<='0';
92          edo_fut<=D;
93          DEC<='1';
94      end case;
95 end process p_estados;
96 p_reloj:process(CLK)
```

```
97 begin
98     if(CLK'event and CLK='1')then
99         edo_pre<=edo_fut;
100    end if;
101 end process p_reloj;
102 end despacha;
```

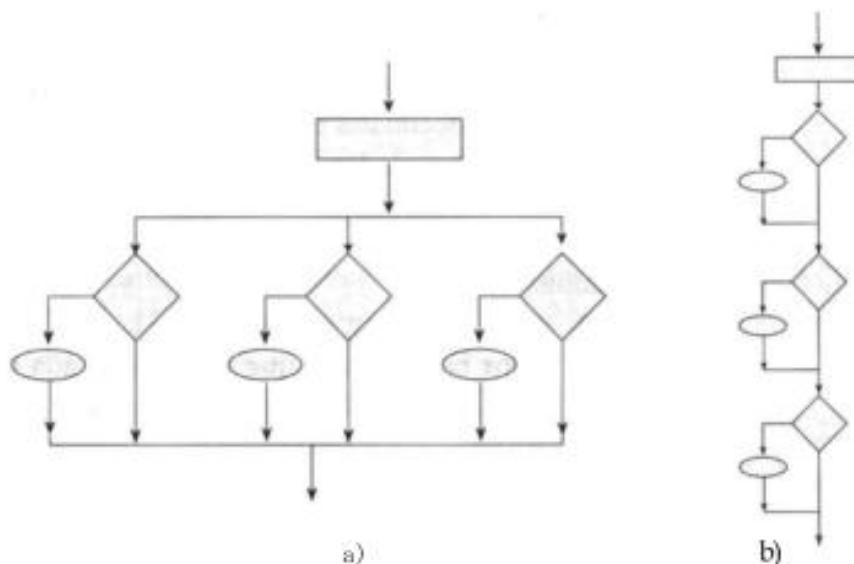
Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	3	3
Clock/Inputs	2	2
I/O Macrocells	7	32
Buried Macrocells	3	32
PIM Input Connects	13	156
	28	225
	= 12 %	

En el diseño de cartas ASM podemos puntualizar los siguientes aspectos:

1. Para cada combinación válida de variables de entrada, sólo puede existir una salida; esto es, no es posible presentar dos veces la misma situación, debido a que habría dos salidas diferentes.
2. Si es necesario retroalimentar una salida, esto debe hacerse antes del estado y en éste.
3. Un bloque puede tener varios caminos en paralelo que lleven a la misma salida y más de uno puede estar activo al mismo tiempo.

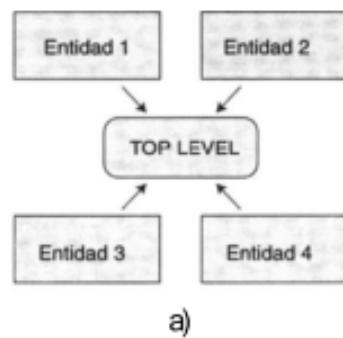
Cuando se tienen caminos en paralelo, se pueden dibujar en forma de serie sin afectar el resultado; sin embargo, un diagrama en paralelo puede hacer a la carta más compacta mientras que un diagrama en serie tiene menos probabilidades de llevar a transiciones de estado ambiguas.



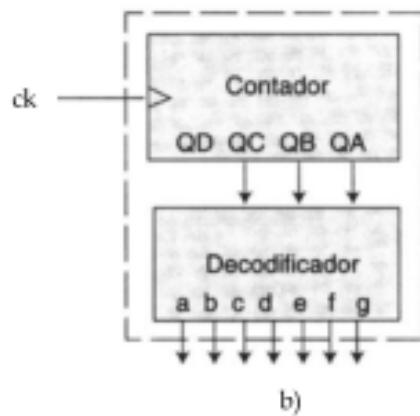
La figura a) es una estructura en paralelo de una carta ASM y la figura b) es una estructura en serie de una carta ASM.

Diseño Jerárquico en VHDL

El diseño jerárquico es una herramienta de apoyo que permite la programación de extensos diseños mediante la unión de pequeños bloques; es decir, un diseño jerárquico agrupa varias entidades electrónicas, las cuales se pueden analizar y simular de manera individual con facilidad, para luego relacionarlas a través de un algoritmo de integración llamado Top Level



La conceptualización del diseño top level difiere de la programación de los sistemas digitales, integrados en una entidad. En este caso no se trata de integrar dos o más módulos electrónicos individuales.



Si no diseñar cada módulo por separado y luego coordinar su funcionamiento a través del algoritmo de programación Top Level.

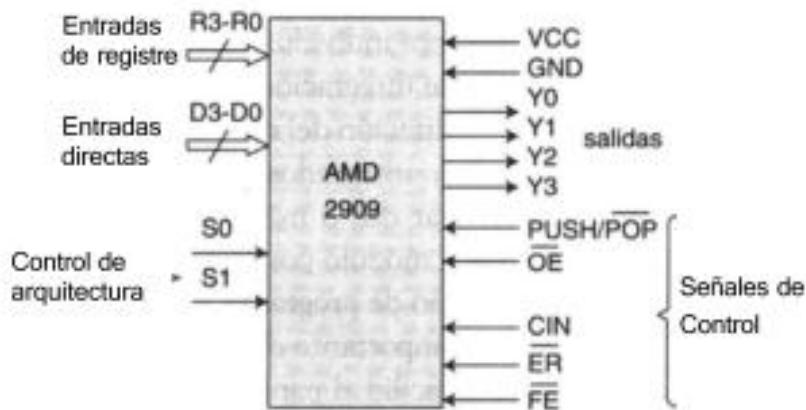
Una ventaja importante del diseño jerárquico en la programación de grandes diseños es la facilidad para trabajar al mismo tiempo con otros diseñadores (paralelismo), ya que mientras uno puede diseñar una parte del sistema, otro puede desarrollar un bloque distinto para unirlos en un solo proyecto más tarde.

Metodología de diseño de estructuras jerárquicas

Una metodología que se recomienda al programar extensos diseños es la siguiente:

- 1) Analizar con detalle el problema y descomponer en bloques individuales la estructura global.
- 2) Diseñar y programar módulos individuales (componentes).
- 3) Crear un paquete de componentes.
- 4) Diseñar el programa de alto nivel (Top Level).

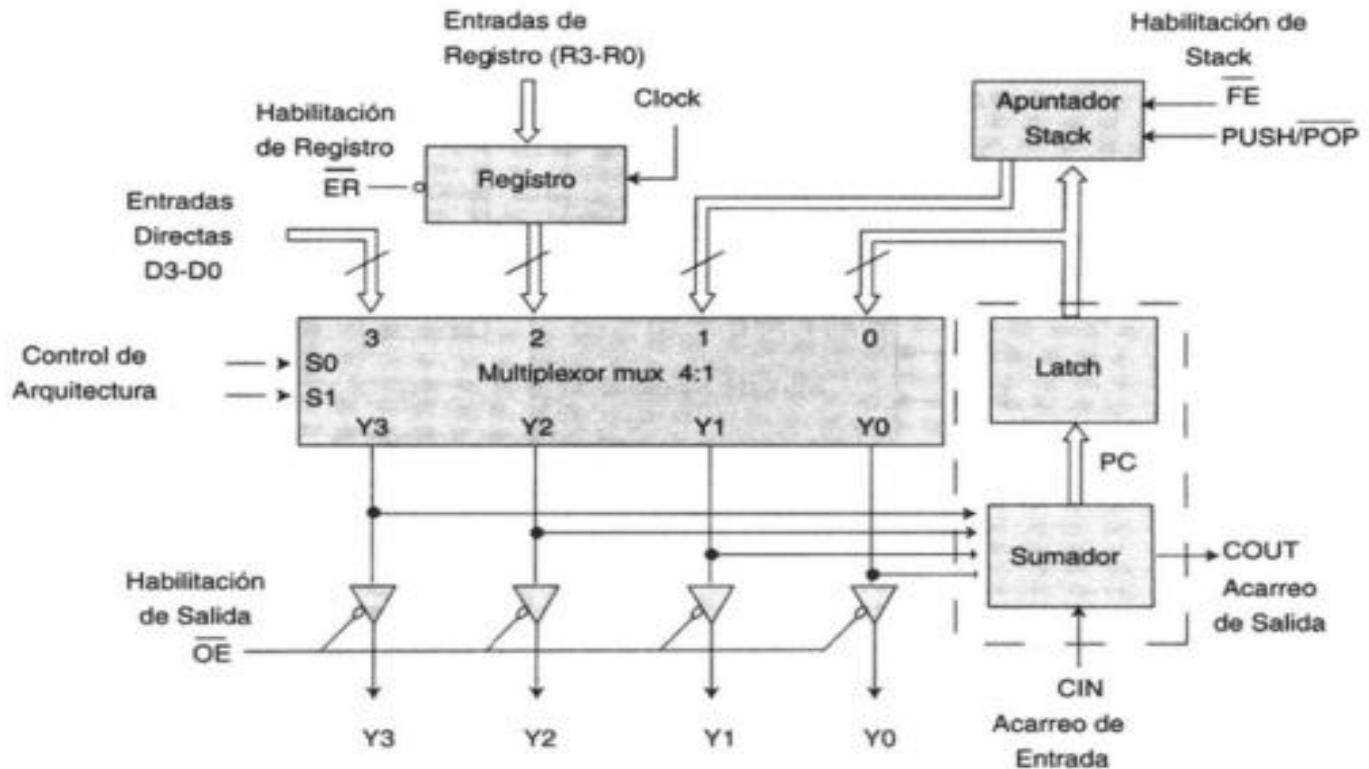
Con el fin de describir y detallar la metodología empleada en el diseño de estructuras jerárquicas, consideremos el circuito AMD2909. Este dispositivo es un secuenciador de 4 bits desarrollado por la compañía Advanced Micro Devices, cuya función es transferir a su bus de salida (Y) una de entre cuatro fuentes internas y externas de datos.



Terminal	Función
Entradas de registro R3-R0	Conservan el estado presente en una función de <i>hold</i> o retén.
Entradas directas D3-D0	Se usan como entradas del secuenciador para indicar un cambio de dirección en la lógica del programa.
Entrada /ER	Habilitación del registro R.
Entrada /FE	Habilitación del puntero de pila (stack pointer: ST).
Entrada CIN	Acarreo de entrada.
Entrada /OE	Habilitación de salidas.
Entrada PUSH/POP	Señales para brincos y retornos de subrutina.
Entradas SO y SI	Lineas de selección que determinan una de cuatro fuentes diferentes de entrada.
Salidas Y3-Y0	Salidas del secuenciador.
Salida COUT	Acarreo de salida.
Vcc y Gnd	Alimentación del circuito.

Descripción del circuito AMD2909

El secuenciador de 4 bits tiene entrada para un bus de datos externos D (D3-D0) u registro R (R3-R0) un apuntador de pila de una palabra ST y un sumador de 4 bits, que funciona como contador de programa.

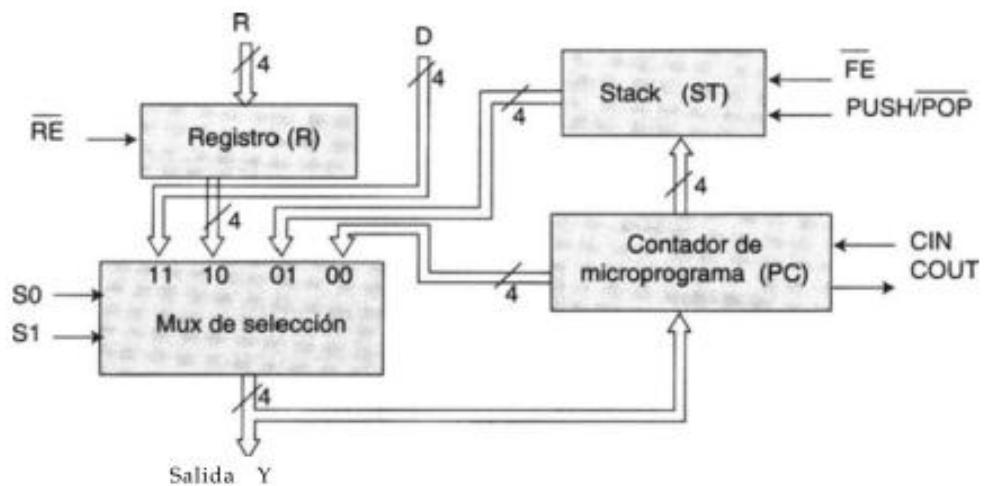


Las entradas directas D se utilizan para canalizar las direcciones de carga desde una memoria de programa y/o control hacia el secuenciador de microprograma. En algunas de las arquitecturas, las entradas R sirven para almacenar el estado presente en una función de hold o retén. El contador de microprograma, PC, incrementa la salida en PC+1, siempre y cuando el acarreo de entrada CIN sea igual a 1. Esto permite realizar una instrucción de cuenta o almacenar en la pila la siguiente dirección cuando se hace un llamado a subrutina, por lo que al salir de ésta la dirección se obtiene e la pila y se canaliza hacia el bus de salida. El circuito contiene 4 multiplexores de 4:1 que seleccionan una de sus cuatro entradas PC, ST, R y D, a través de sus líneas de selección S0 y S1.

Análisis del problema y descomposición en bloques individuales de la estructura global.

El diseño jerárquico basa su fortaleza en la descomposición o división de un diseño. Lo anterior permite analizar los diferentes subsistemas que lo forman y unirlos más tarde a través de un programa denominado Top Level.

En la siguiente figura se muestra la interconexión interna de cada uno de los bloques y las señales de control que interactúan en cada uno de los subsistemas (multiplexor de selección, contador de microprograma, registro y apuntador de pila) del circuito AMD2909.



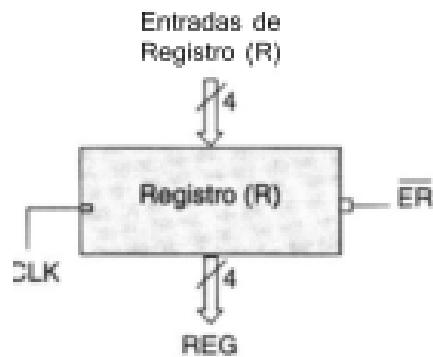
Diseño y programación de componentes o unidades del circuito

El primer paso de diseño consiste en programar de manera individual cada uno de los componentes y/o unidades del circuito.

Un componente es la parte de un programa que define un elemento físico, el cual se puede usar en otros diseños o entidades.

Haremos la programación de cada uno de los bloques que forman el circuito AMD2909.

Diseño del registro {R}



El siguiente es un **registro de 4 bits**

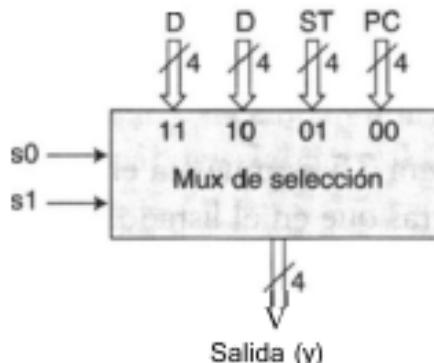
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity registro is
4 port(R: in std_logic_vector (3 downto 0);
5      ER,CLK: in std_logic;
6      REG: inout std_logic_vector (3 downto 0));
7 end registro;
8
9 architecture fourbts of registro is
10 begin
11 process(CLK,ER,REG,R)
12 begin
13     if(CLK'event and CLK='1')then
14         if ER='0' then
15             REG<=R;
16         else
17             REG<=REG;
18         end if;
19     end if;
20 end process;
21 end fourbts;

```

Diseño del Multiplexor

Es un multiplexor cuádruple de 4:1 con dos líneas de selección (S0 y S1), cuatro entradas (R,ST,D,PC) y una salida (Y) de 4 bits.



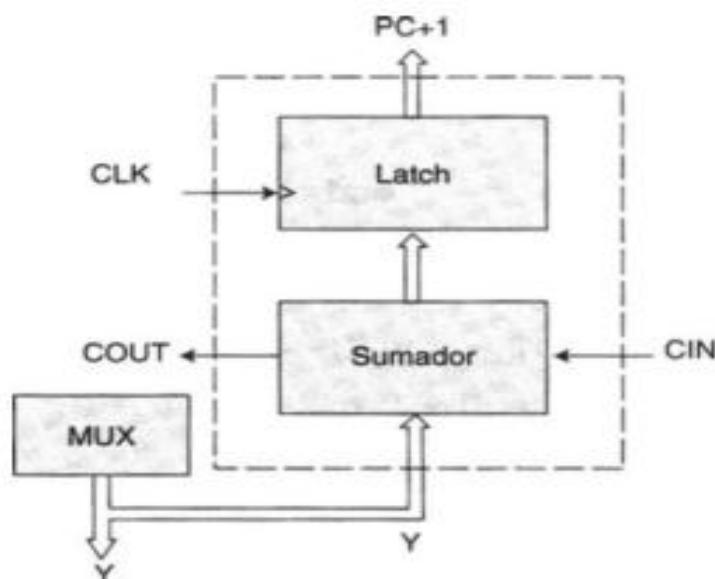
```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity mux_4 is
4 port(D,R,ST,PC: in std_logic_vector (3 downto 0);
5      S: in std_logic_vector (1 downto 0);
6      Y: out std_logic_vector (3 downto 0));
7 end mux_4;
8
9 architecture arq_mux of mux_4 is
10 begin
11     with S select
12         Y<= PC when "00",
13                 ST when "01",
14                 R when "10",
15                 D when others;
16 end arq_mux;

```

Contador de microprograma (PC)

La función de este bloque es incrementar la dirección de entrada cuando CIN es igual a 1. El diagrama correspondiente se encuentra en la siguiente figura.



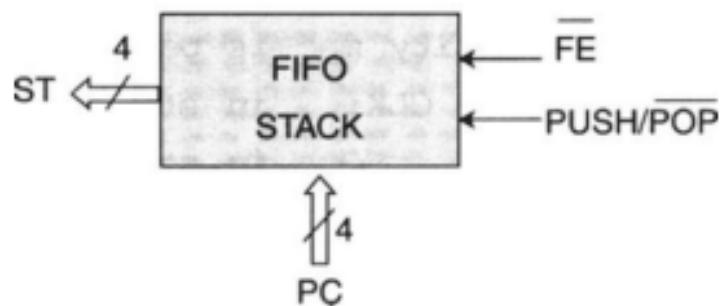
La manera más sencilla de programar el contador de microprograma es a través de la arquitectura funcional, considerando el circuito sumador y el contador como una entidad de diseño, con entradas y salidas generales. El funcionamiento es muy simple: cuando el acarreo de entrada (CIN) tiene el valor de '1' y el reloj del sistema está en transición de '0' a '1'. La dirección Y se incrementa en uno y su valor pasa al bus de salida PC. En caso contrario cuando CIN='0', se asigna a PC el valor de Y sin cambios.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity mpc is
5 port(CIN,CLK: in std_logic;
6       Y: in std_logic_vector (3 downto 0);
7       COUT: inout std_logic;
8       PC: inout std_logic_vector (3 downto 0));
9 end mpc;
10
11 architecture arq_mpc of mpc is
12 begin
13 process (CLK,Y,CIN)
14 begin
15   if(CLK'event and CLK='1')then
16     if(CIN='1')then
17       PC<=Y+1;
18     else
19       PC<=Y;
20     end if;
21   end if;
22 end process;
23 COUT<=(CIN and Y(0) and Y(1) and Y(2) and Y(3));
24 end arq_mpc;
```

Apuntador de pila (Stack Pointer)

La pila (stack) está diseñada para almacenar un dato de 4 bits, de modo que cuando en un programa se hace un llamado a subrutina, la siguiente dirección ($PC+1$) se almacena en la pila, por lo que al salir de la subrutina la dirección se toma de la pila y se envía al multiplexor de selección, el cual canaliza al bus de salida (Y).

Para poder introducir y almacenar un dato en la pila, se debe habilitar primero la señal FE (habilitación de pila) y la señal PUSH ($FE=0'$ y $PUSH=T$). De esta forma, cuando la transición del reloj (CLK) sea de '0' a '1' el dato que se encuentra en PC se introduce y almacena en la pila hasta que la señal PUSH se desabilite ($PUSH'0'$). Para sacar el dato se habilita la señal POP ($POP=0'$), con lo que el dato se canaliza al bus de salida ST.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity stack is
5 port(CLK,FE,PUSH,POP: in std_logic;
6      PC: inout std_logic_vector(3 downto 0);
7      ST: inout std_logic_vector (3 downto 0));
8 end stack;
9
10 architecture pointer of stack is
11 signal var: std_logic_vector (3 downto 0);
12 begin
13 process(FE,CLK,PUSH,POP,PC)
14     variable x: std_logic_Vector (3 downto 0);
15 begin
16     if(CLK'event and CLK='1')then
17         if(FE='0')then
18             if(PUSH='1')then
19                 x:=PC;
20                 var<=x;
21             elsif(POP='0')then
22                 ST<=var;
23             else
24                 ST<=ST;
25         end if;
26     end if;
27 end if;
28 end process;
29 end pointer;
```

Creación de un paquete de componentes

Una vez que se ha diseñado cada módulo que forma la arquitectura general, se crea un programa que contenga los componentes de cada una de las entidades de diseño descritas con anterioridad. Para esto es necesario identificar primero el paquete en que se almacenarán los diseños.

En nuestro ejemplo el paquete recibe el nombre de comps_sec (EL USUARIO ESCOGE EL NOMBRE DE SU PAQUETE).

Cada componente se declara como una entidad de diseño, con la omisión de la palabra reservada is y agregando la cláusula component.

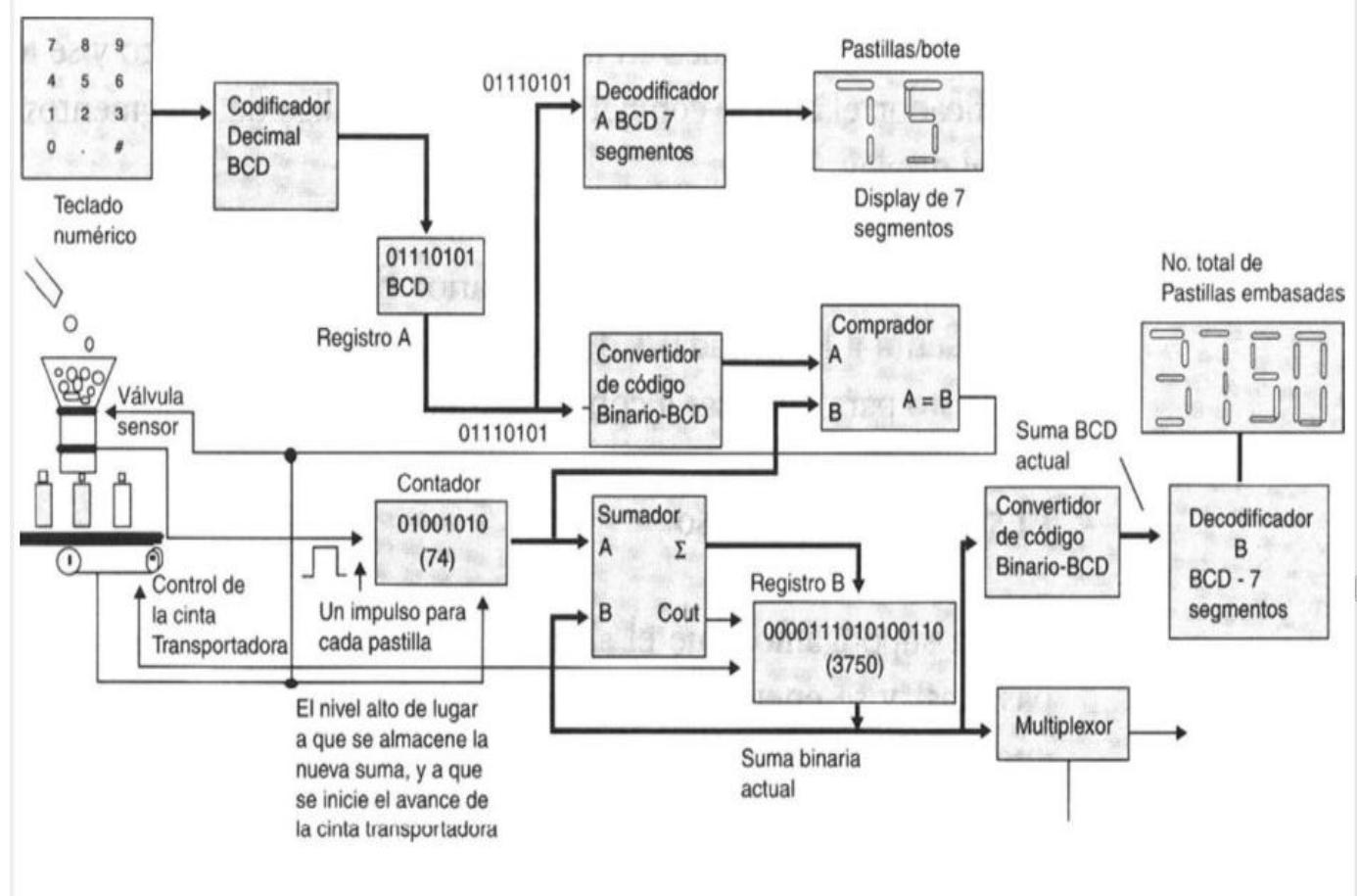
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 package comps_sec is
4 component registro
5 port(R: in std_logic_vector(3 downto 0);
6      ER, CLK: in std_logic;
7      REG: inout std_logic_vector(3 downto 0));
8 end component;
9
10 component mpc
11 port(CIN,CLK: in std_logic;
12      Y: in std_logic_vector(3 downto 0);
13      COUT: inout std_logic;
14      PC: inout std_logic_vector(3 downto 0));
15 end component;
16
17 component stack
18 port(CLK,FE,PUSH,POP: in std_logic;
19      PC: inout std_logic_vector(3 downto 0);
20      ST: inout std_logic_vector(3 downto 0));
21 end component;
22
23 component mux_4
24 port(D,R,ST,PC: in std_logic_vector(3 downto 0);
```

```
25      S: in std_logic_vector(1 downto 0);
26      Y: buffer std_logic_vector(3 downto 0));
27 end component;
28 end comps_sec;
```

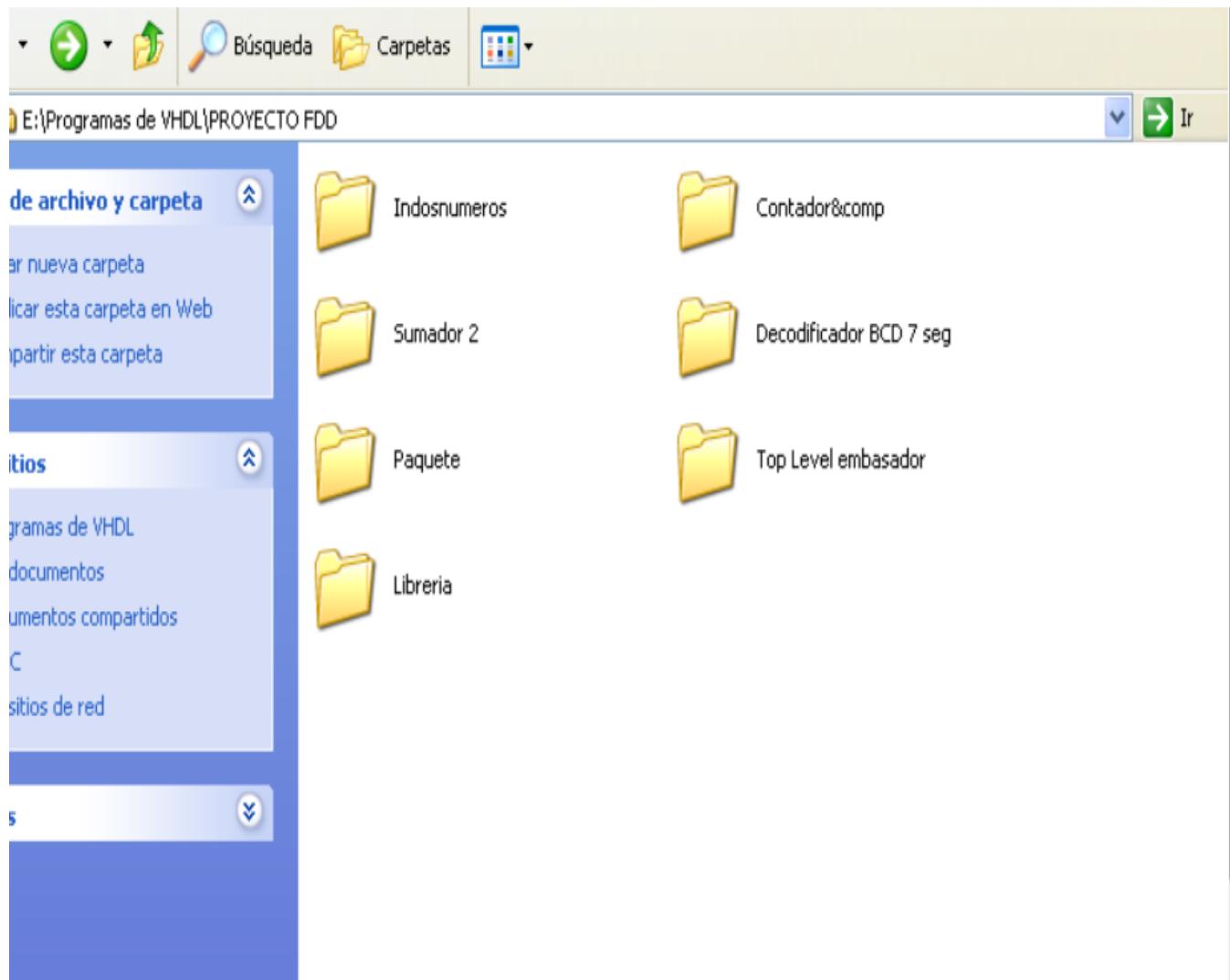
Desarrollo de proyecto:

Envásador de pastillas a partir de la introducción en el teclado matricial.

Se busca aplicar lo aprendido en FDD respecto al lenguaje VHDL en un sistema que permita envasar y contar pastillas.



El proyecto se dividió en distintos módulos y gracias a un empaquetado y al diseño de librerías se logró unirlo en una modalidad denominada Top Level.



A continuación se muestran los códigos de cada módulo.

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesús
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee;
10 use ieee.std_logic_1164.all;
11 use work.std_arith.all;
12 entity module is
13 port(boton: in std_logic_vector (0 to 9);--teclado matricial
14 --Displays de cátodo común
15     dispa,dispb: out std_logic_vector(0 to 6);
16     SUM: out std_logic_vector(6 downto 0));
17 end module;
18
19 architecture arq_module of module is
20 --Salida del codificador C
21 --Registro A(RA) Registro B(RB)
22 signal C,RA,RB: std_logic_vector (3 downto 0);
23 signal tecla: std_logic_vector (1 downto 0);
24 --Acarreo de la suma
```

```
25 signal CARRY: std_logic_vector (0 to 6);
26 signal x: std_logic_vector (0 to 1);
27 begin
28 teclado: process(boton,C,RA,RB,tecla)
29 variable i: std_logic_vector (1 downto 0) := "00";
30 begin
31     if(boton= "1000000000") then
32         C<="0000"; tecla<=i;
33     elsif(boton= "0100000000") then
34         C<="0001"; tecla<=i;
35     elsif(boton= "0010000000") then
36         C<="0010"; tecla<=i;
37     elsif(boton= "0001000000") then
38         C<="0011"; tecla<=i;
39     elsif(boton= "0000100000") then
40         C<="0100"; tecla<=i;
41     elsif(boton= "0000010000") then
42         C<="0101"; tecla<=i;
43     elsif(boton= "0000001000") then
44         C<="0110"; tecla<=i;
45     elsif(boton= "0000000100") then
46         C<="0111"; tecla<=i;
47     elsif(boton= "0000000010") then
48         C<="1000"; tecla<=i;
```

```
49      else
50          C<="1001"; tecla<=i;
51      end if;
52
53      if(tecla="00")then
54          RB<=C;
55      else
56          RA<=C;
57      end if;
58
59      i:=i+1;
60
61      if(tecla="10")then
62          i:="00";
63      end if;
64 end process teclado;
65 --El manejo de los displays serà de catodo comun
66 display:process(RA,RB,dispa,dispb)
67 begin
68 --Display menos significativo
69     case RA is
70         when "0000" => dispa <= "1111110";
71         when "0001" => dispa <= "0110000";
72         when "0010" => dispa <= "1101101";
```

```
73      when "0011" => dispa <= "1111001";
74      when "0100" => dispa <= "0110011";
75      when "0101" => dispa <= "1011011";
76      when "0110" => dispa <= "1011111";
77      when "0111" => dispa <= "1110001";
78      when "1000" => dispa <= "1111111";
79      when others => dispa <= "1111011";
80  end case;
81 --display más significativo
82  case RB is
83      when "0000" => dispb <= "1111110";
84      when "0001" => dispb <= "0110000";
85      when "0010" => dispb <= "1101101";
86      when "0011" => dispb <= "1111001";
87      when "0100" => dispb <= "0110011";
88      when "0101" => dispb <= "1011011";
89      when "0110" => dispb <= "1011111";
90      when "0111" => dispb <= "1110001";
91      when "1000" => dispb <= "1111111";
92      when others => dispb <= "1111011";
93  end case;
94 end process display;
95
96 convertidor: process(RA,RB,CARRY,x,SUM)
```

```
97 variable z: std_logic:='0';
98 begin
99     SUM(0)<=RA(0);
100    SUM(1)<=RA(1) xor RB(0);
101    CARRY(0)<=RA(1) and RB(0);
102    SUM(2)<=((RA(2) xor RB(1)) xor CARRY(0));
103    CARRY(1)<=((RA(2) and RB(1)) or (CARRY(0) and (RA(2) xor
104                                RB(1))));
105    x(0)<=((RA(3) xor RB(0)) xor CARRY(1));
106    CARRY(2)<=((RA(3) and RB(0)) or (CARRY(1) and (RA(3)
107                                xor RB(0))));
108    SUM(3)<=X(0) xor RB(2);
109    CARRY(3)<=X(0) and RB(2);
110    x(1)<=((CARRY(2) xor RB(1)) xor CARRY(3));
111    CARRY(4)<=((CARRY(2) and RB(1)) or (CARRY(3) and (CARRY(2)
112                                xor RB(1))));
113    SUM(4)<= X(1) xor RB(3);
114    CARRY(5)<= X(1) and RB(3);
115    SUM(5)<=((CARRY(4) xor RB(2)) xor CARRY(5));
116    CARRY(6)<=((CARRY(4) and RB(2)) or (CARRY(5) and (CARRY(4)
117                                xor RB(2))));
118    SUM(6)<=((z xor RB(3)) xor CARRY(6));
119 end process convertidor;
120 end arq_module;
```

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesùs
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee;
10 use ieee.std_logic_1164.all;
11 use work.std_arith.all;
12 entity contador is
13 port(SUM:inout std_logic_vector(7 downto 0);
14       sensor,reset: in std_logic;
15       compara: out std_logic_vector (7 downto 0);
16       Q: inout std_logic_vector(7 downto 0));
17 end contador;
18
19 architecture arq_compcont of contador is
20 begin
21 contador:process(sensor,reset,Q)
22 begin
23   if(sensor'event and sensor='1')then
24     Q<="00000000";
```

```
25      Q<=Q+1;
26      if(reset='1' or Q="01100011") then
27          Q<="00000000";
28      end if;
29      end if;
30 end process contador;
31 compare: process (Q,SUM,compara)
32 begin
33     if (Q=SUM) then
34         compara<=Q;
35     else
36         compara<=SUM;
37     end if;
38 end process compare;
39 end arq_compcont;
```

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesús
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee;
10 use ieee.std_logic_1164.all;
11 use work.std_arith.all;
12 entity sumador2 is
13 port(Cout: inout std_logic_vector (0 to 7);
14       RC,RD: in std_logic_vector(3 downto 0);
15       SUM2: out std_logic_vector (0 to 7));
16 end sumador2;
17
18 architecture arq_sumdos of sumador2 is
19 signal y: std_logic_vector(1 downto 0);
20 begin
21 process (RC,RD,Cout,SUM2,y)
22 variable n: std_logic:='0';
23 begin
24     SUM2(0)<=RC(0);
```

```
25    SUM2(1)<=RC(1) xor RD(0);
26    Cout(0)<=RC(1) and RD(0);
27    SUM2(2)<=((RC(2) xor RD(1)) xor Cout(0));
28    Cout(1)<=((RC(2) and RD(1)) or (Cout(0) and (RC(2) xor
29                  RD(1)))); 
30    y(0)<=((RC(3) xor RD(0)) xor Cout(1));
31    Cout(2)<=((RC(3) and RD(0)) or (Cout(1) and (RC(3)
32                  xor RD(0)))); 
33    SUM2(3)<=y(0) xor RD(2);
34    Cout(3)<=y(0) and RD(2);
35    y(1)<=((Cout(2) xor RD(1)) xor Cout(3));
36    Cout(4)<=((Cout(2) and RD(1)) or (Cout(3) and (Cout(2)
37                  xor RD(1)))); 
38    SUM2(4)<= y(1) xor RD(3);
39    Cout(5)<= y(1) and RD(3);
40    SUM2(5)<=((Cout(4) xor RD(2)) xor Cout(5));
41    Cout(6)<= ((Cout(4) and RD(2)) or (Cout(5) and (Cout(4)
42                  xor RD(2)))); 
43    SUM2(6)<=((n xor RD(3)) xor Cout(6));
44 end process;
45 end arq_sumdos;
```

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesús
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee;
10 use ieee.std_logic_1164.all;
11 entity decodifdos is
12 port(RA,RB,RC,RD: in std_logic_vector(3 downto 0);
13       dispC,dispD,dispE,dispF: out std_logic_vector(6 downto 0));
14 end decodifdos;
15
16 architecture arq_decos of decodifdos is
17 begin
18 process(RA,RB,RC,RD)
19 begin
20 --Display de Unidades
21 case RA is
22     when "0000" => dispC <= "1111110";
23     when "0001" => dispC <= "0110000";
24     when "0010" => dispC <= "1101101";
```

```
25      when "0011" => dispc <= "1111001";
26      when "0100" => dispc <= "0110011";
27      when "0101" => dispc <= "1011011";
28      when "0110" => dispc <= "1011111";
29      when "0111" => dispc <= "1110001";
30      when "1000" => dispc <= "1111111";
31      when others => dispc <= "1111011";
32  end case;
33 --display decenas
34  case RB is
35      when "0000" => dispd <= "1111110";
36      when "0001" => dispd <= "0110000";
37      when "0010" => dispd <= "1101101";
38      when "0011" => dispd <= "1111001";
39      when "0100" => dispd <= "0110011";
40      when "0101" => dispd <= "1011011";
41      when "0110" => dispd <= "1011111";
42      when "0111" => dispd <= "1110001";
43      when "1000" => dispd <= "1111111";
44      when others => dispd <= "1111011";
45  end case;
46 --Display centenas
47  case RC is
48      when "0000" => dispe <= "1111110";
```

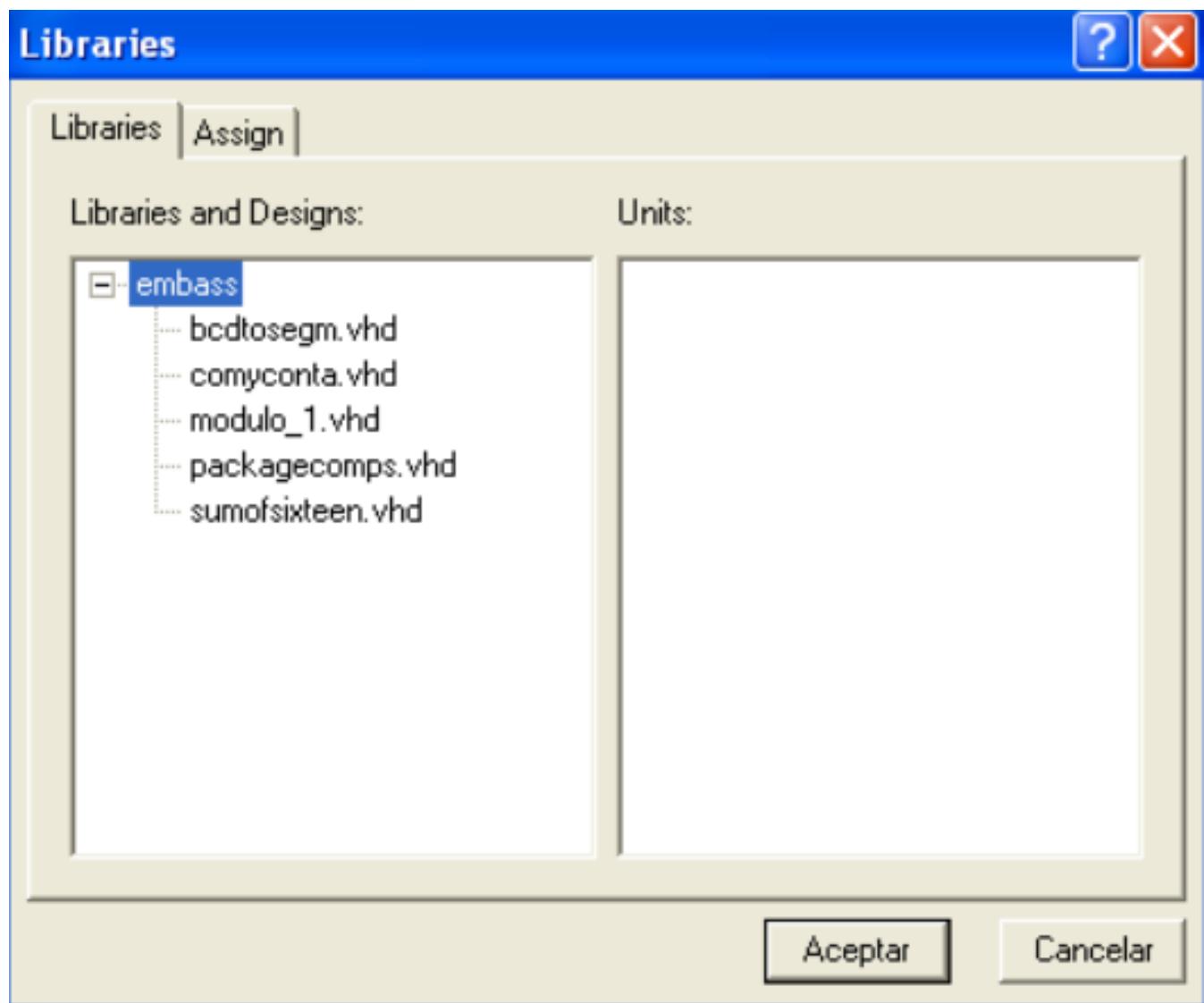
```
49      when "0001" => dispe <= "0110000";
50      when "0010" => dispe <= "1101101";
51      when "0011" => dispe <= "1111001";
52      when "0100" => dispe <= "0110011";
53      when "0101" => dispe <= "1011011";
54      when "0110" => dispe <= "1011111";
55      when "0111" => dispe <= "1110001";
56      when "1000" => dispe <= "1111111";
57      when others => dispe <= "1111011";
58  end case;
59 --Display millares
60  case RD is
61      when "0000" => dispf <= "1111110";
62      when "0001" => dispf <= "0110000";
63      when "0010" => dispf <= "1101101";
64      when "0011" => dispf <= "1111001";
65      when "0100" => dispf <= "0110011";
66      when "0101" => dispf <= "1011011";
67      when "0110" => dispf <= "1011111";
68      when "0111" => dispf <= "1110001";
69      when "1000" => dispf <= "1111111";
70      when others => dispf <= "1111011";
71  end case;
72 end process;
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

Paquete de Componentes

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesùs
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee;
10 use ieee.std_logic_1164.all;
11 package embasador is
12 component module
13 port(boton: in std_logic_vector (0 to 9);
14      dispa,dispb: out std_logic_vector(0 to 6);
15      SUM: out std_logic_vector(6 downto 0));
16 end component;
17
18 component contador
19 port(SUM: inout std_logic_vector(7 downto 0);
20       sensor: in std_logic;
21       reset: in std_logic;
22       compara: out std_logic_vector(7 downto 0);
23       Q: inout std_logic_vector(7 downto 0));
24 end component;
```

```
25
26 component sumador2
27 port(Cout: inout std_logic_vector (0 to 7);
28     RC: in std_logic_vector(3 downto 0);
29     RD: in std_logic_vector (3 downto 0);
30     SUM2: out std_logic_vector (0 to 7));
31 end component;
32
33 component decodifdos
34 port(RA:in std_logic_vector(3 downto 0);
35     RB:in std_logic_vector(3 downto 0);
36     RC:in std_logic_vector(3 downto 0);
37     RD:in std_logic_vector(3 downto 0);
38     dispc:out std_logic_vector (6 downto 0);
39     dispd:out std_logic_vector (6 downto 0);
40     dispe:out std_logic_vector (6 downto 0);
41     dispf:out std_logic_vector (6 downto 0));
42 end component;
43 end embasador;
```

La librería se desarrolló en Warp Cypress aclarando que de la bibliografía hubo diferencias respecto a la versión de Galaxy.



En la imagen se observan los módulos que se desarrollaron y que se unieron para aplicar la librería y poderla llamar en el programa en que se quiera utilizar

El programa Top Level es el siguiente:

```
1 --PROYECTO FUNDAMENTOS DE DISEÑO DIGITAL
2 --CONTADOR DE PASTILLAS
3 --Equipo 4
4 --Integrantes:           Grupo:2CV3
5 --*Alcantara Gomez Roberto
6 --*Santos Mèndez Ulises Jesús
7 --*Velasco Aguilar Carlos Eduardo
8 --*Villavicencio Quintero Kevin Enrique
9 library ieee,embass;
10 use ieee.std_logic_1164.all;
11 use work.std_arith.all;
12 use embass.embasador.all;
13 entity sistema is
14 port(boton: in std_logic_vector (0 to 9);
15       dispa,dispb: out std_logic_vector(0 to 6);
16       SUM: out std_logic_vector(6 downto 0);
17       SUM: inout std_logic_vector(7 downto 0);
18       sensor: in std_logic;
19       reset: in std_logic;
20       compara: out std_logic_vector(7 downto 0);
21       Q: inout std_logic_vector(7 downto 0);
22       Cout: inout std_logic_vector (0 to 7);
23       RC: in std_logic_vector(3 downto 0);
24       RD: in std_logic_vector (3 downto 0);
--      . . . . .
```

