CAPÍTULO 4
DISEÑO DE SISTEMAS DIGITALES

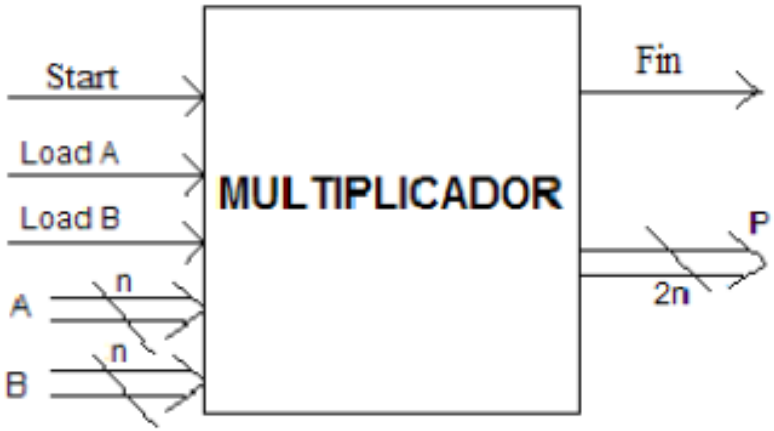
DISEÑO DE UN SISTEMA DIGITAL MULTIPLICADOR

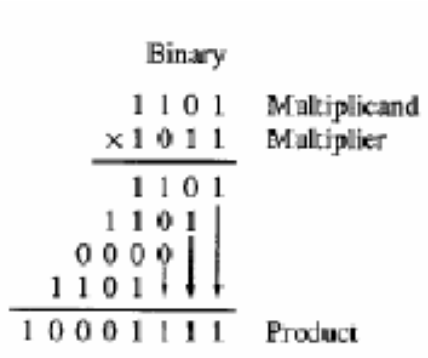
Se desea diseñar un circuito **Multiplicador** de dos números binarios sin signo, cada uno de ***n*** bits.

Utilizaremos en el diseño el mismo método de multiplicación que se utiliza en el proceso de multiplicación manual.

El **Producto** esta formado por una serie de operaciones de suma.

Para cada **i** bit del **Multiplicador_B** que es igual a **1** sumamos al **Producto** el valor de **Multiplicando_A** desplazado a la izquierda **i** veces.





En el Procesador de Datos debemos usar un registro de desplazamiento de derecha a izquierda de 2n bits para el número **Multiplicando_A**. El **Multiplicando_A** es de n bits, pero debe ser desplazado n-1 veces, por lo tanto debe tener 2n bits.

También necesitamos un registro de desplazamiento de izquierda a derecha de n bits para analizar bit por bit el número **Multiplicador_B**.

El sumador debe ser de 2n bits también y necesitamos un registro de sostenimiento de 2n bits para el **Producto_P**.

Inicialmente el registro P debe estar en 0. Ya que no podemos usar Resetn debemos cargarlo con 0. Utilizaremos un conjunto de mux de 2-a-1 de 2n bits. Cuando la entrada Sel= 0 a las entradas del registro P llegan 0 y cuando Sel=1 el registro P puede ser cargado con P+A proveniente del sumador.

También necesitamos la puerta NOR para detectar que todos los bits del **Multiplicador_B** son iguales a 0.

Bosquejo del Procesador de Datos.

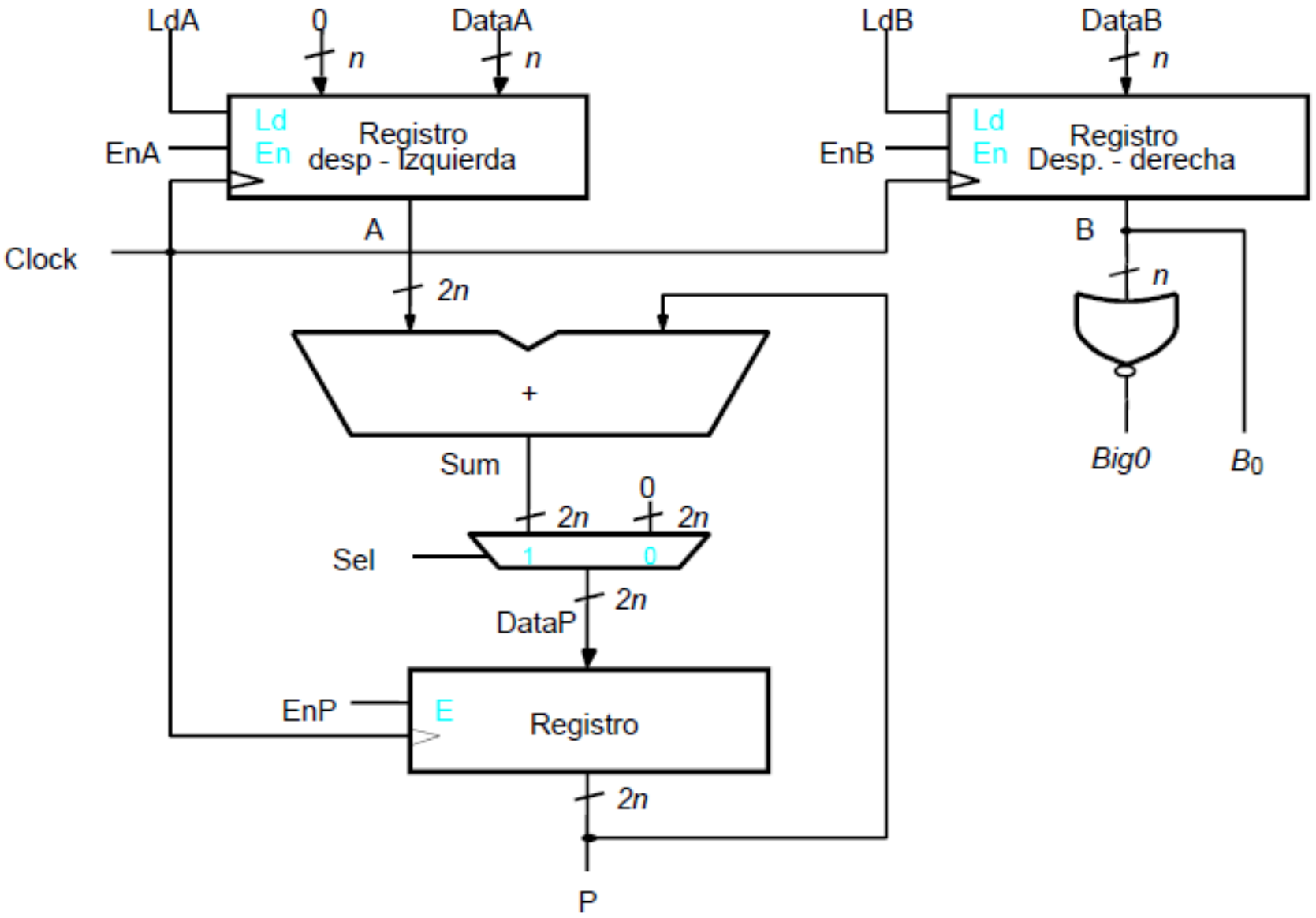
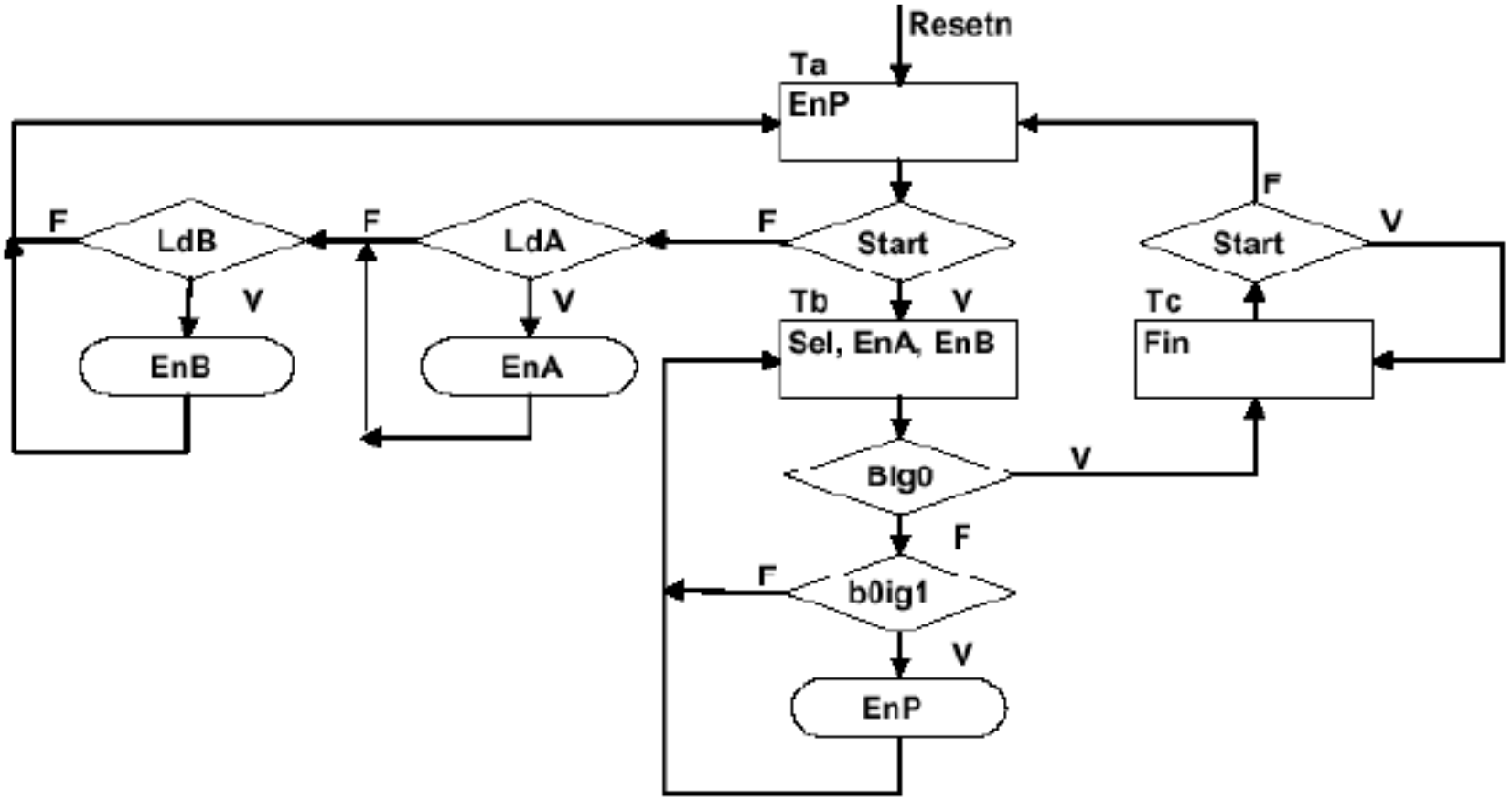
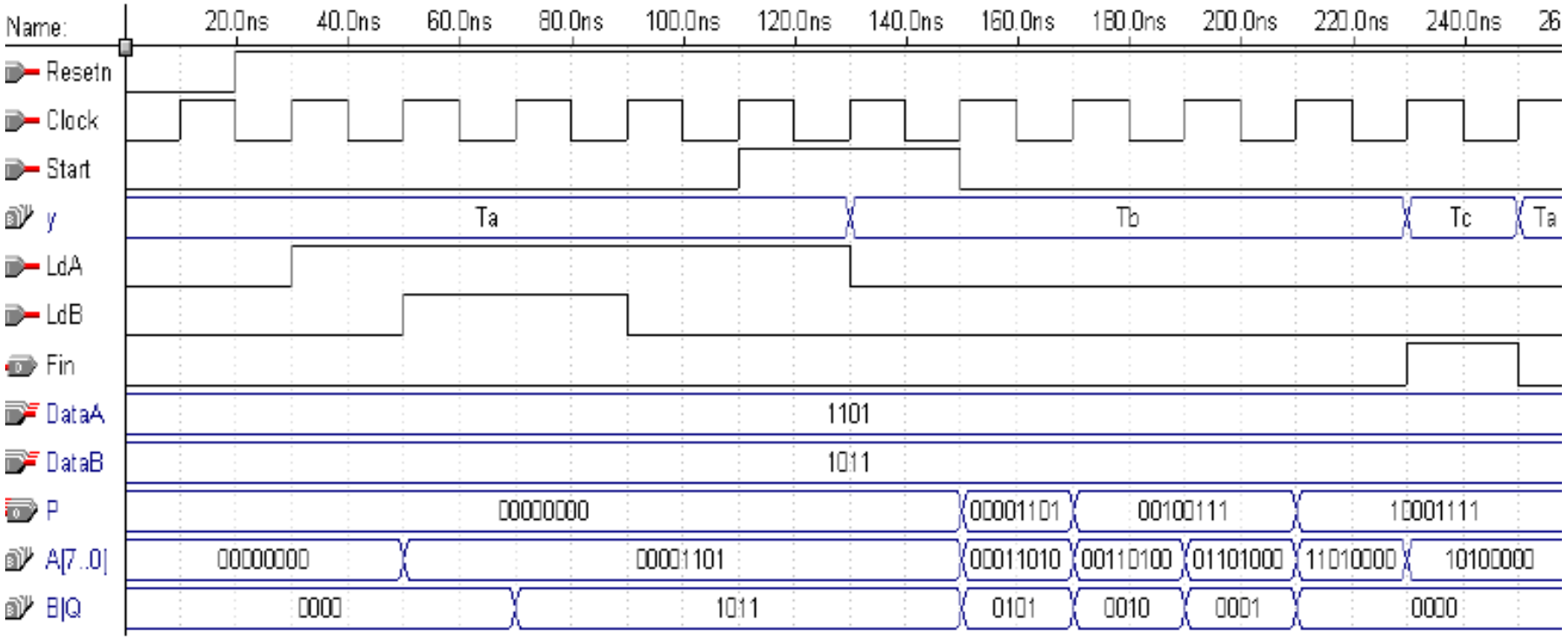


Diagrama ASM del Controlador



Para implementar en VHDL nuestro diseño, primero desarrollamos individualmente cada componente del procesador de datos:

Diagrama de tiempo del circuito MULTIPLICADOR



Código VHDL para el registro de desplazamiento_d_i

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

entity registro_d_i is
    generic (n  : integer := 4);
    port (  Resetn, Clock   : in std_logic;
            En, Ld, L       : in std_logic;
            Entpar          : in std_logic_vector (n-1 downto 0);
            Q               : buffer std_logic_vector (n-1 downto 0));
end registro_d_i;

architecture comportamiento of registro_d_i is
begin
    process (Resetn, Clock)
    begin
        if Resetn ='0' then Q <= (others => '0');
        elsif (Clock'event and Clock ='1') then
        if En ='1' then
            if Ld ='1' then Q <= Entpar;
            else desplazamiento: for i in 1 to n-1 loop
                Q(i) <= Q(i-1);
            end loop;
            Q(0) <= L;
            end if; end if; end if;
    end process;
end comportamiento;
```


01101010010010100110000101101110

Código VHDL para el registro de desplazamiento_i_d

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

entity registro_i_d is
    generic (   n            : integer:= 4);
    port     (   Resetn, Clock       : in std_logic;
                 En, Ld, R           : in std_logic;
                 Entpar              : in std_logic_vector (n-1 downto 0);
                 Q                   : buffer std_logic_vector (n-1 downto 0));
end registro_i_d;

architecture comportamiento of registro_i_d is
begin
    process (Resetn, Clock)
    begin
        if Resetn ='0' then Q <= (others =>'0');
        elsif (Clock'event and Clock ='1') then
        if En ='1' then
            if Ld ='1' then Q <= Entpar;
            else desplazamiento: for i in 0 to n-2 loop
                Q(i) <= Q(i+1);
            end loop;
            Q(n-1) <= R;
            end if; end if; end if;
    end process;
end comportamiento;
```


01101010011001010110000101101110

Código VHDL para el registro de sostenimiento

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

entity registro_sost is
 generic (n: integer:=4);
 port ( Resetn, Clock   : in std_logic;
        En              : in std_logic;
        Entrada         : in std_logic_vector(n-1 downto 0);
        Q               : out std_logic_vector (n-1 downto 0));
end registro_sost;

architecture comportamiento of registro_sost is
begin
    process (Resetn, Clock)
    begin
        if Resetn ='0' then Q <= (others => '0');
        elsif (Clock'event and Clock ='1') then
            if En ='1' then
                Q <= Entrada;
            end if; end if;
 end process;

end comportamiento;
```

Código VHDL para un BUS MUX 2 a 1

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

entity busmux2a1 is
    generic (n: integer:=4);
    port (  sel          : in std_logic;
            ent0, ent1   : in std_logic_vector(n-1 downto 0);
            s            : out std_logic_vector(n-1 downto 0));
end busmux2a1;

architecture comportamiento of busmux2a1 is
begin
    with sel select
        s <=    ent0 when '0',
                ent1 when others;
end comportamiento;
```

Creación del paquete COMPONENTES

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

package componentes is
component registro_d_i
    generic (n  : integer := 4);
    port (  Resetn, Clock   : in std_logic;
            En, Ld, L       : in std_logic;
            Entpar          : in std_logic_vector (n-1 downto 0);
            Q               : buffer std_logic_vector (n-1 downto 0));
end component;

component registro_i_d
    generic (n  : integer:= 4);
    port(   Resetn, Clock   : in std_logic;
            En, Ld, R       : in std_logic;
            Entpar          : in std_logic_vector (n-1 downto 0);
            Q               : buffer std_logic_vector (n-1 downto 0));
end component;

component registro_sost
    generic (n: integer:-4);
    port (  Resetn, Clock   : in std_logic;
            En              : in std_logic;
            Entrada         : in std_logic_vector(n-1 downto 0);
            Q               : out std_logic_vector (n-1 downto 0));
end component;
```

Creación del paquete COMPONENTES

```
component busmux2a1
    generic (n: integer:=4);
    port (  sel          : in std_logic;
            ent0, ent1   : in std_logic_vector(n-1 downto 0);
            s            : out std_logic_vector(n-1 downto 0));
end component;

end componentes;
```

Código VHDL del circuito total Multiplicador de dos palabras de n bits.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.componentes.all;

-- multiplicador de dos numeros de n bits
entity multiplicador is
    generic (n: integer := 4; nn: integer := 8);
    port (Resetn, Clock      : in std_logic;
          LdA, LdB, Start    : in std_logic;
          DataA, DataB       : in std_logic_vector(n-1 downto 0);
          P                  : buffer std_logic_vector(nn-1 downto 0);
          Fin                : out std_logic);
end multiplicador;
```

El numero de bits de los datos DataA y DataB es establecido por el parámetro genérico n.
Ya que los registros A y P deben ser de 2n bits se define un segundo parámetro genérico nn para representar el tamaño 2 x n.
Cambiando los valores de los parámetros genéricos se puede usar el mismo código VHDL para números de cualquier tamaño.

```
architecture comportamiento of multiplicador is
    type estado is (Ta,Tb,Tc);
    signal y: estado;
    signal Sel, EnA, EnB, EnP, Big0, Zero: std_logic;
    signal B, n_zeros: std_logic_vector (n-1 downto 0);
    signal Ain, A, DataP, Sum: std_logic_vector (nn-1 downto 0);
begin
```

La entrada de carga en paralelo del Multiplicando_A es de 2n bits, pero el DataA es solo de n bits. Por lo tanto se usa la señal interna n_zeros para completar con ceros los 2n bits.
La señal Ain concatena estos bits con DataA para cargarlos en el registro Multiplicando_A.
Los procesos MSS_transiciones y MSS_salidas definen las transiciones de estados y generación de salidas del circuito Controlador.

Los procesos **MSS_transiciones** y **MSS_salidas** definen las transiciones de estados y generación de salidas del circuito **Controlador**.

```
-- Circuito Controlador
    MSS_transiciones: process (Resetn, Clock)
    begin
        if Resetn ='0' then y <= Ta;
        elsif (Clock'event and Clock ='1') then
            case y is
                when Ta => if Start ='0' then y <= Ta; else y <= Tb; end if;
                when Tb => if Big0 ='0' then y <= Tb; else y <= Tc; end if;
                when Tc => if Start ='1' then y <= Tc; else y <= Ta; end if;
            end case;
        end if;
    end process;

    MSS_salidas: process (y, start, LdA, LdB, Big0, b(0))
    begin
        EnA <= '0'; EnB <='0'; EnP <='0'; Fin <='0'; Sel <='0';
        case y is
            when Ta => EnP <='1';
                if (start='0' and LdA ='1') then EnA <='1'; else EnA <='0'; end if;
                if (start='0' and LdB ='1') then EnB <='1'; else EnB <='0'; end if;
            when Tb => EnA <='1'; EnB <='1'; Sel <='1';
                if (Big0='0' and b(0) ='1') then EnP <='1'; else EnP <='0'; end if;
            when Tc => Fin <='1';
        end case;
    end process;
```

El Procesador de Datos esta descrito por el siguiente código VHDL:

```
-- Procesador de datos

    Zero <='0';
    n_zeros <= (others =>'0'); nn_zeros <= (others =>'0');
    Ain <= n_zeros & DataA;

    multiplicando_A: registro_d_i generic map (n => nn)
        port map    (Resetn, Clock, EnA, LdA, Zero, Ain, A);

    multiplicador_B:    registro_i_d generic map (n => n)
        port map    (Resetn, Clock, EnB, LdB, Zero, DataB, B);

    Big0 <='1'  when B = n_zeros else '0';
    Sum <= A + P;

    Multiplexor: busmux2a1 generic map (n => nn)
        port map    (Sel, nn_zeros, Sum, DataP);

    producto_P: registro_sost generic map (n => nn)
        port map (Resetn, Clock, EnP, DataP, P);

end comportamiento;
```


0110000101110011011000010110111001111010011000011

INTERFASES

Interfas de Entrada INTERFASES

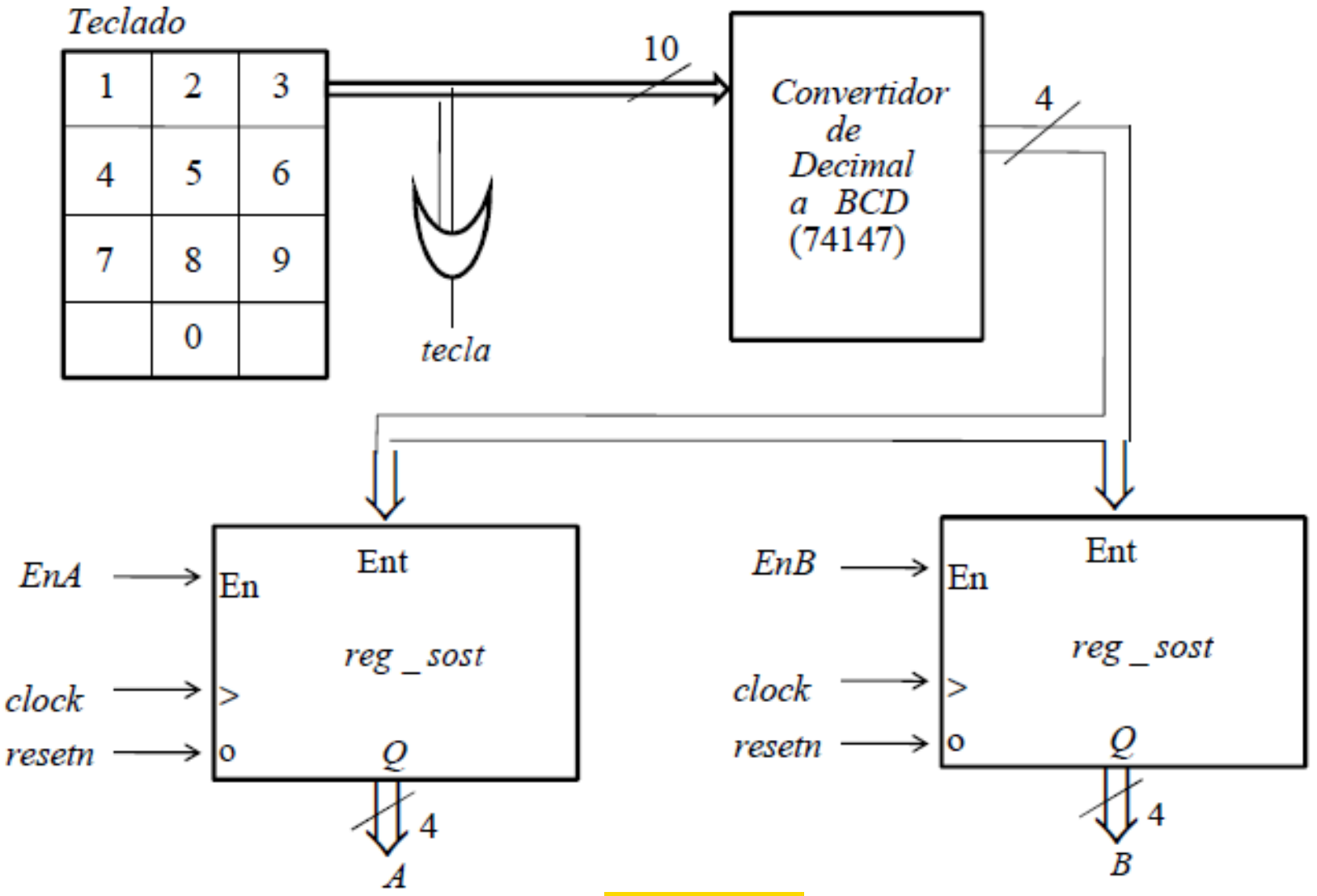
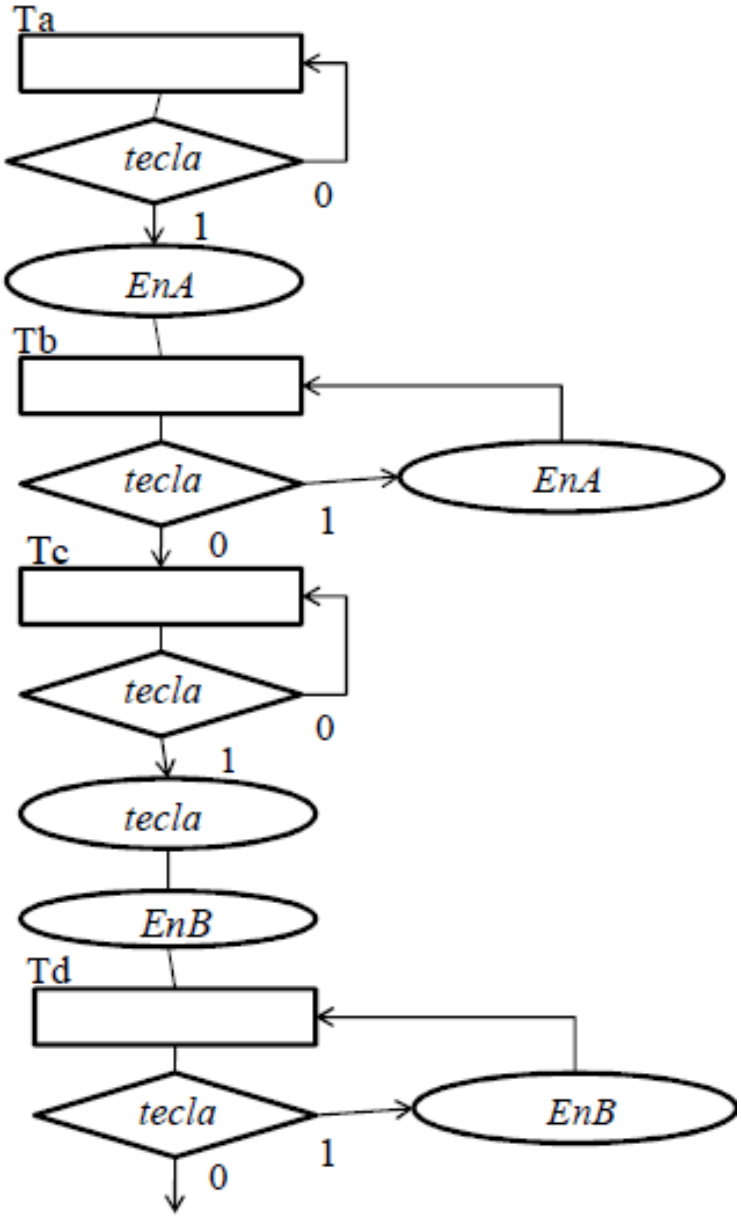
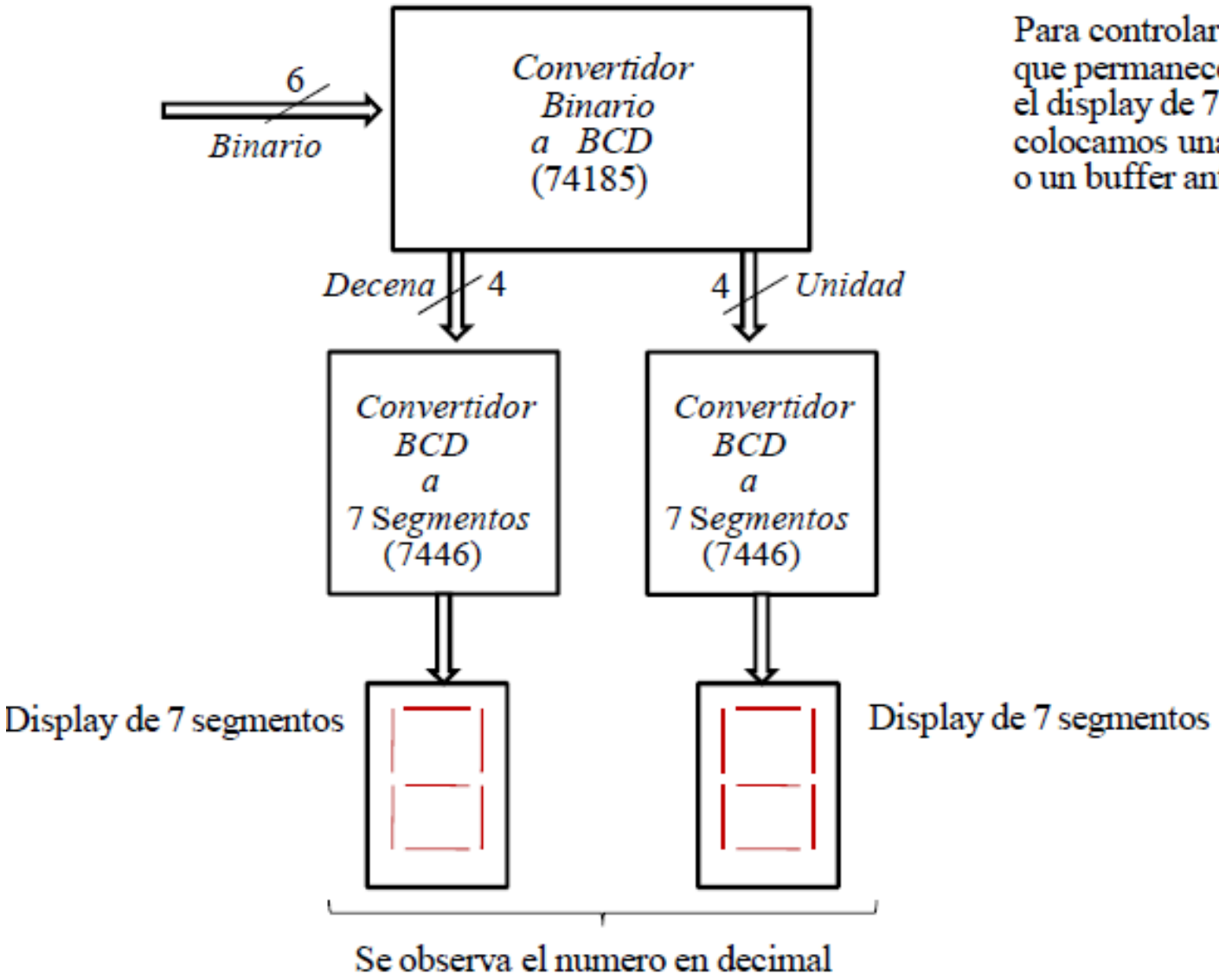


Diagrama ASM



Interfaz de Salida



Para controlar el tiempo que permanece encendido el display de 7 segmentos colocamos una puerta logica o un buffer antes del convertidor.