DESCRIPCIÓN DE CIRCUITOS DIGITALES

Circuitos combinacionales

Circuitos secuenciales

Organización del diseño. Diseño genérico

Operaciones iterativas

Autores: Luis Entrena, Celia López, Mario García, Enrique San Millán, Marta Portela, Almudena Lindoso

Indice

Sentencias condicionales

Sentencias secuenciales

- ☞ **IF...THEN...ELSE**
- ☞ **CASE...IS...WHEN**

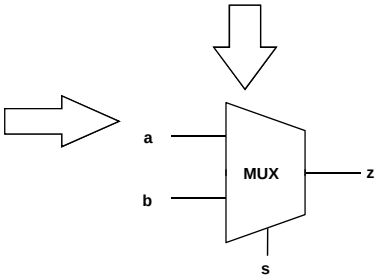
Sentencias concurrentes

- ☞ **... WHEN ...**
  (asignación condicional)
- ☞ **WITH ... SELECT**
  (asignación seleccionada)

Un multiplexor 2 a 1

```
-- Descripción secuencial
PROCESS (a, b, s)
BEGIN
        IF s = '0' THEN
                z <= a;
        ELSE
                z <= b;
        END IF;
END PROCESS;
```

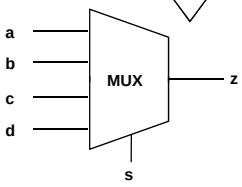
```
-- Descripción concurrente
z <= a WHEN s = '0'
        ELSE b;
```



Un multiplexor 4 a 1

```
-- Descripción secuencial
PROCESS (a, b, c, d, s)
BEGIN
    CASE s IS
        WHEN "00" =>     z <= a;
        WHEN "01" =>     z <= b;
        WHEN "10" =>     z <= c;
        WHEN OTHERS => z <= d;
        END CASE;
END PROCESS;
```

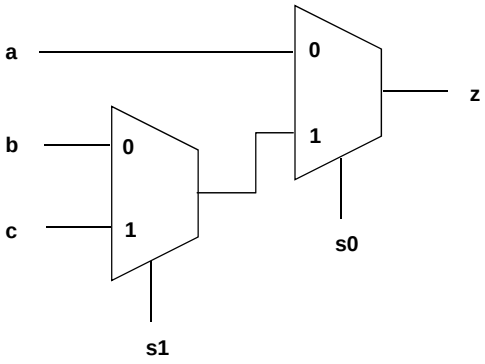
```
-- Descripción concurrente
WITH s SELECT
z <=    a WHEN "00",
        b WHEN "01",
        c WHEN "10",
        d WHEN OTHERS;
```



Diferencias entre IF y CASE

☞ IF...THEN...ELSE impone prioridades en la selección

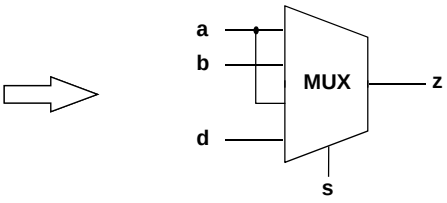
```
IF s0 = '0' THEN
        z <= a;
ELSIF s1 = '0' THEN
        z <= b;
ELSE
        z <= c;
END IF;
```



Diferencias entre IF y CASE

☞ **CASE no impone prioridades en la selección**

```
PROCESS (a, b, c, d, s)
BEGIN
    CASE s IS
        WHEN "00" | "10" =>   z <= a;
        WHEN "01" =>          z <= b;
        WHEN OTHERS =>        z <= d;
        END CASE;
END PROCESS;
```



Tablas de verdad

```
PROCESS (a, b, c)
        VARIABLE entradas: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
        entradas  := a & b & c;
        CASE entradas IS
                WHEN "000" =>       f <= '1';
                WHEN "001" =>       f <= 'X';
                WHEN "010" =>       f <= '0';
                WHEN "011" =>       f <= '1';
                WHEN "100" =>       f <= '0';
                WHEN "101" =>       f <= '1';
                WHEN OTHERS =>      f <= 'X';
        END CASE;
END PROCESS;
```

Reglas para el diseño de circuitos combinacionales

- **Condiciones para que un proceso infiera un circuito combinacional correctamente:**
  - ***Si una señal se asigna dentro de un proceso o en una sentencia de asignación concurrente, entonces debe garantizarse que recibe un valor bajo todas las posibles condiciones o casos. En otras palabras, el valor de la señal debe estar definido en todos los casos.***
  - ***La lista de sensibilidad de un proceso que modela lógica combinacional debe contener todas las señales que son leídas en el proceso, es decir, todas las señales cuyo valor se utiliza dentro del proceso y que por tanto actúan como entradas en el circuito resultante de la síntesis.***
  - ***Si se utilizan variables dentro de un proceso, estas deben de ser utilizadas propiamente como variables intermedias, es decir, que deben ser escritas antes de ser leídas.***

Reglas para el diseño de circuitos combinacionales

- **Algunas consecuencias de las reglas anteriores:**
  - **Para cubrir todos los casos, es conveniente que toda sentencia IF o asignación condicional lleve una claúsula ELSE, y toda sentencia CASE o asignación seleccionada lleve una claúsula WHEN OTHERS**
  - **No se deben crear referencias circulares, es decir, salidas que dependen de sí mismas, ya que se generarían lazos de realimentación asíncronos**
  - **Se aconseja el uso de señales en lugar de variables cuando sea posible**

Ejemplos

```
PROCESS( s, a)
BEGIN
    IF s = '0' THEN
        z <= a;
    END IF;
END PROCESS;
```

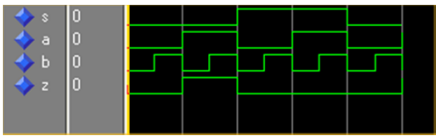
```
PROCESS( s, a)
BEGIN
    IF s = '0' THEN
        z <= a;
    ELSE
        z <= z;
    END IF;
END PROCESS;
```

```
PROCESS( s, a)
BEGIN
    IF s = '0' THEN
        z <= a;
    ELSE
        z <= b;
    END IF;
END PROCESS;
```

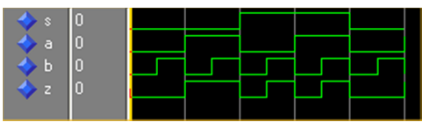
```
PROCESS( s, a, b)
BEGIN
    IF s = '0' THEN
        z <= a;
    ELSE
        z <= b;
    END IF;
END PROCESS;
```


Efecto de la lista de sensibilidad en la simulación

- Lista de sensibilidad incorrecta (s, a)



- Lista de sensibilidad correcta (s, a, b)



Ejercicio

- Diseñar un decodificador de 2 a 4
  - de forma secuencial
  - de forma concurrente
- Declaración de la entidad

```
ENTITY decodificador IS
        PORT (  a: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                z: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END decodificador;
```


Ejercicio

- Diseñar un comparador de 4 bits
  - de forma secuencial
  - de forma concurrente
- Declaración de la entidad

```
ENTITY comparador IS
        PORT (  a, b: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                menor, mayor, igual: OUT STD_LOGIC);
END comparador;
```

Ejercicio

- Diseñar un codificador con prioridad de 2 bits
  - de forma secuencial
  - de forma concurrente
- Declaración de la entidad

```
ENTITY codificador IS
        PORT (  a: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                z: OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
                gs: OUT STD_LOGIC);
END decodificador;
```


Solución al ejercicio: decodificador

Solución al ejercicio: comparador

Solución al ejercicio: codificador

DESCRIPCIÓN DE CIRCUITOS DIGITALES

Circuitos combinacionales

Circuitos secuenciales

Organización del diseño. Diseño genérico

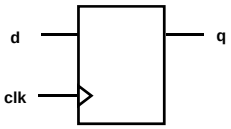
Operaciones iterativas

Autor: Luis Entrena Arrontes

Indice

Biestables

```
-- Biestable D
-- activo por flanco de subida
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        q <= d;
    END IF;
END PROCESS;
```




```
-- Biestable D
-- activo por nivel (latch)
PROCESS (enable, d)
BEGIN
    IF enable = '1' THEN
        q <= d;
    END IF;
END PROCESS;
```

SOLO DISEÑO SÍNCRONO

Un registro

- Si q es un vector, se crea un registro de tantos biestables como indique la dimensión del vector

```
-- Registro de 8 bits
SIGNAL q, d: STD_LOGIC_VECTOR (7 DOWNTO 0);
...
PROCESS (clk)
BEGIN
        IF clk'EVENT AND clk = '1' THEN
            q <= d;
        END IF;
END PROCESS;
```

Biestable con entradas asíncronas

- Las entradas asíncronas deben figurar en la lista de sensibilidad
- Nota: la precedencia de las señales de control viene dada por el orden IF ... ELSIF ... ELSIF ...

```
PROCESS (preset, clr, clk)
BEGIN
    IF preset = '1' THEN
        q <= '1';
    ELSIF clr = '1' THEN
        q <= '0';
    ELSIF clk'EVENT AND clk = '1' THEN
        q <= d;
    END IF;
END PROCESS;
```

preset tiene precedencia sobre el clr

Biestable con entradas de control síncronas

- Las entradas síncronas no figuran en la lista de sensibilidad

```
PROCESS (reset, clk)
BEGIN
    IF reset = '1' THEN
        q <= '0';
    ELSIF clk'EVENT AND clk = '1' THEN
        IF enable = '1' THEN
          q <= d;
        END IF;
    END IF;
END PROCESS;
```

parte asíncrona

parte síncrona

Reglas para el diseño de circuitos secuenciales síncronos

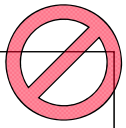
- Deben describirse mediante procesos
- Debe haber una condición de flanco de reloj
  - ✓ IF clk'EVENT AND clk = '1' THEN
  - ✓ IF clk'EVENT AND clk = '0' THEN
  - ❑ Detrás de dicha condición no puede haber ELSE
  - ❑ Si se asigna una señal, no es necesario cubrir todos los casos: en los casos no cubiertos, la señal mantiene su valor
- Lista de sensibilidad: reloj + entradas asíncronas (reset)
  - ❑ Las entradas de datos y las entradas síncronas no aparecen en la lista de sensibilidad del proceso

Reglas para el diseño de circuitos secuenciales síncronos

- "Plantilla" para el diseño de circuitos secuenciales síncronos

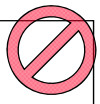
```
PROCESS( <reloj + entradas asíncronas>)
BEGIN
    IF <condición de inicialización asíncrona> THEN
        <inicialización asíncrona>
    ELSIF <flanco activo de reloj> THEN
        -- Comportamiento síncrono
    END IF;
END PROCESS;
```

Ejemplos



```
PROCESS (reset, clk)
BEGIN
    IF reset = '0' THEN
        q <= "0101";
    ELSIF clk'EVENT AND clk = '1' AND enable = '1' THEN
        q <= d;
    END IF;
END PROCESS;
```

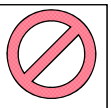

Ejemplos



```
PROCESS (reset, clk)
BEGIN
    IF reset = '0' THEN
        q <= "0101";
    ELSIF enable = '1'  THEN
        IF clk'EVENT AND clk = '1' THEN
            q <= d;
        END IF;
    END IF;
END PROCESS;
```

Ejemplos

```
PROCESS (reset, clk)
BEGIN
    IF reset = '1' THEN
        IF clk'EVENT AND clk = '1' THEN
            IF enable = '1' THEN
                    q <= d;
            END IF;
        END IF;
    ELSE
        q <= "0101";
    END IF;
END PROCESS;
```



Inferencia de registros y biestables

- Todas las señales que se asignan entre *IF clk'EVENT* ... y *END IF* infieren un biestable
- Todas las variables que se asignan antes de ser leídas entre *IF clk'EVENT* ... y *END IF* infieren un biestable

Ejemplos

```
-- Ejemplo 1 (recomendado)
PROCESS (clk)
    VARIABLE v: STD_LOGIC;
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        v := d;
        q <= v;
    END IF;
END PROCESS;
```

```
-- Ejemplo 2 (no recomendado)
PROCESS (clk)
    VARIABLE v: STD_LOGIC;
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        q <= v;
        v := d;
    END IF;
END PROCESS;
```


Ejemplos

```
-- Sumador combinacional
PROCESS (a, b)
BEGIN
    s <= a + b;
END PROCESS;
```

```
-- Sumador seguido de un registro
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        s <= a + b;
    END IF;
END PROCESS;
```

Ejemplos

```
-- Sumador con entradas y salidas registradas
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        a1 <= a;
        b1 <= b;
        s <= a1 + b1;
    END IF;
END PROCESS;
```

```
-- Sumador con entradas y salidas registradas
-- Registros de entrada y salida con reset
PROCESS (reset, clk)
BEGIN
    IF reset = '1' THEN
        a1 <= (OTHERS => '0');
        b1 <= (OTHERS => '0');
        s <= (OTHERS => '0');
    ELSIF clk'EVENT AND clk = '1' THEN
        a1 <= a;
        b1 <= b;
        s <= a1 + b1;
    END IF;
END PROCESS;
```


Ejemplos

- Un registro puede no tener reset, pero no puede hacer reset de una señal que no sea un registro!

```
-- Sumador seguido de un registro
-- Inicializacion incorrecta!
PROCESS (clk)
BEGIN
    IF reset = '1' THEN
        a <= (OTHERS => '0');
        b <= (OTHERS => '0');
        s <= (OTHERS => '0');
    ELSIF clk'EVENT AND clk = '1' THEN
        s <= a + b;
    END IF;
END PROCESS;
```

Ejercicios

- Diseñar un biestable JK activo por flanco de bajada

```
ENTITY biestable_jk IS
    PORT (  reset:   IN STD_LOGIC;
            clk:     IN STD_LOGIC;
            j, k:    IN STD_LOGIC;
            q:       OUT STD_LOGIC);
END biestable_jk;
```

- Diseñar un registro de desplazamiento

```
ENTITY reg_desp IS
    PORT (  reset:   IN STD_LOGIC;
            clk:     IN STD_LOGIC;
            es:      IN STD_LOGIC;
            q:       OUT STD_LOGIC_VECTOR (0 to 3));
END reg_desp;
```


Solución: Biestable JK

Solución: Registro de desplazamiento

Contadores

```
PROCESS (reset, clk)
BEGIN
    IF reset = '1' THEN
        q <= "00000000";
    ELSIF clk'EVENT AND clk = '1' THEN
        IF enable = '1' THEN
                q <= q + 1;
        END IF;
    END IF;
END PROCESS;
```

Contadores

```
PROCESS (reset, clk)
    VARIABLE c: INTEGER RANGE 0 TO 255;
BEGIN
    IF reset = '1' THEN
        c := 0;
    ELSIF clk'EVENT AND clk = '1' THEN
        IF enable = '1' THEN
            IF c = 255 THEN
                c := 0;
            ELSE
                c := c + 1;
            END IF;
        END IF;
    END IF;
    q <= c;
END PROCESS;
```

Necesario para evitar el desbordamiento en simulación

q declarado como entero
Si q es un std_logic_vector, utilizar
q <= CONV_STD_LOGIC_VECTOR(c, 8);

Ejercicio

☞ Diseñar un contador ascendente de 4 bits con salida de acarreo

```
ENTITY cont4 IS
    PORT (  reset:      IN STD_LOGIC;
            clk:        IN STD_LOGIC;
            q:          OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            carry_out:  OUT STD_LOGIC);
END cont4 ;
```

Ejercicio

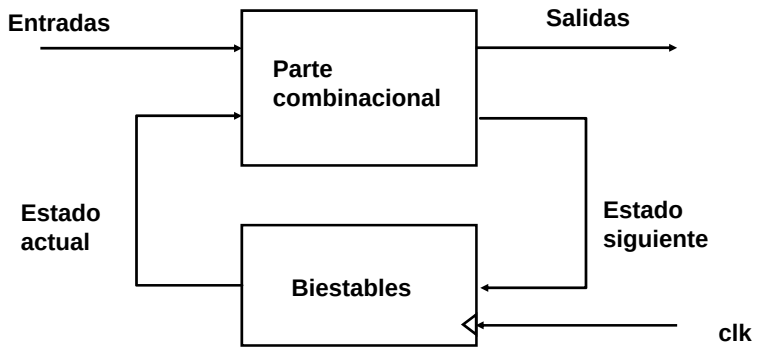
- Diseñar un contador ascendente/descendente con entradas de habilitación y de precarga

```
ENTITY cont_up_down IS
    PORT (  reset:      IN STD_LOGIC;
            clk:        IN STD_LOGIC;
            load:       IN STD_LOGIC;
            d:          IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            ud:         IN STD_LOGIC;
            enable:     IN STD_LOGIC;
            q:          OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END cont_up_down;
```

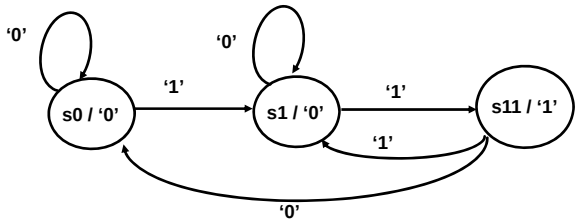

Solución: contador ascendente de 4 bits

Solución: contador ascendente/descendente con habilitación y precarga

Máquinas de estados



Ejemplo: máquina de Moore



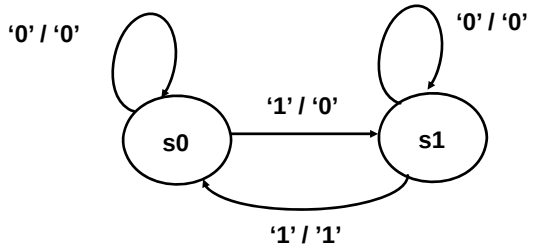

```
ENTITY maquina is
    PORT(  clk:   in std_logic;
           reset: in std_logic;
           a:     in std_logic;
           z :    out std_logic);
END maquina;
```


Ejemplo: Máquina de Moore

```
ARCHITECTURE moore OF maquina IS
   TYPE estado IS (s0, s1, s11);
   SIGNAL estado_actual, estado_siguiente: estado;
BEGIN
   PROCESS (clk, reset)
   BEGIN
      IF reset = '1' THEN
         estado_actual <= s0;
      ELSIF clk'EVENT AND clk = '1' THEN
         estado_actual <= estado_siguiente;
      END IF;
   END PROCESS;
....
```

```
   PROCESS (estado_actual, a)
   BEGIN
      CASE estado_actual IS
      WHEN s0 =>  z <= '0';
            IF a = '0' THEN
                  estado_siguiente <= s0;
            ELSE
                  estado_siguiente <= s1;
            END IF;
       WHEN s1 => z <= '0';
            IF a = '0' THEN
                  estado_siguiente <= s1;
            ELSE
                  estado_siguiente <= s11;
            END IF;
       WHEN s11 => ...
       ....
      END CASE;
   END PROCESS;
END moore;
```

Ejemplo: máquina de Mealy




```
ENTITY maquina is
    PORT(  clk:   in std_logic;
           reset: in std_logic;
           a:     in std_logic;
           z :    out std_logic);
END maquina;
```


Ejemplo: Máquina de Mealy

```
ARCHITECTURE mealy OF maquina IS
   TYPE estado IS (s0, s1);
   SIGNAL estado_actual, estado_siguiente: estado;
BEGIN
   PROCESS (clk, reset)
   BEGIN
      IF reset = '1' THEN
         estado_actual <= s0;
      ELSIF clk'EVENT AND clk = '1' THEN
         estado_actual <= estado_siguiente;
      END IF;
   END PROCESS;
....
```

```
   PROCESS (estado_actual, a)
   BEGIN
      CASE estado_actual IS
      WHEN s0 =>  z <= '0';
            IF a = '0' THEN
               estado_siguiente <= s0;
            ELSE
               estado_siguiente <= s1;
            END IF;
       WHEN s1 =>
            IF a = '0' THEN
               z <= '0';
               estado_siguiente <= s1;
            ELSE
               z <= '1';
               estado_siguiente <= s11;
            END IF;
      END CASE;
   END PROCESS;
...
```