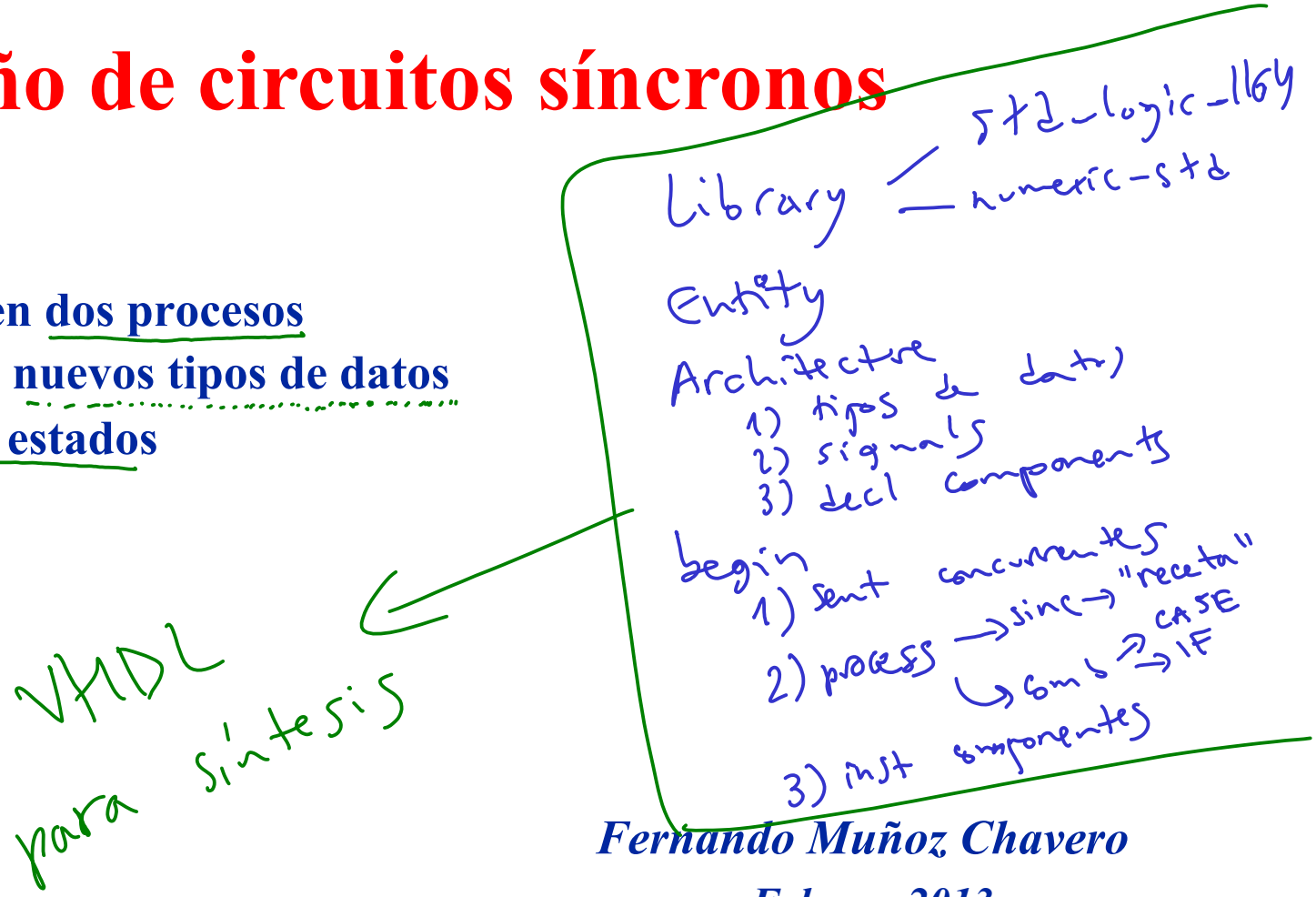
Clase 3:

Diseño de circuitos síncronos

3.1 Descripción en dos procesos
3.2 Definición de nuevos tipos de datos
3.3 Máquinas de estados
3.4 Atributos



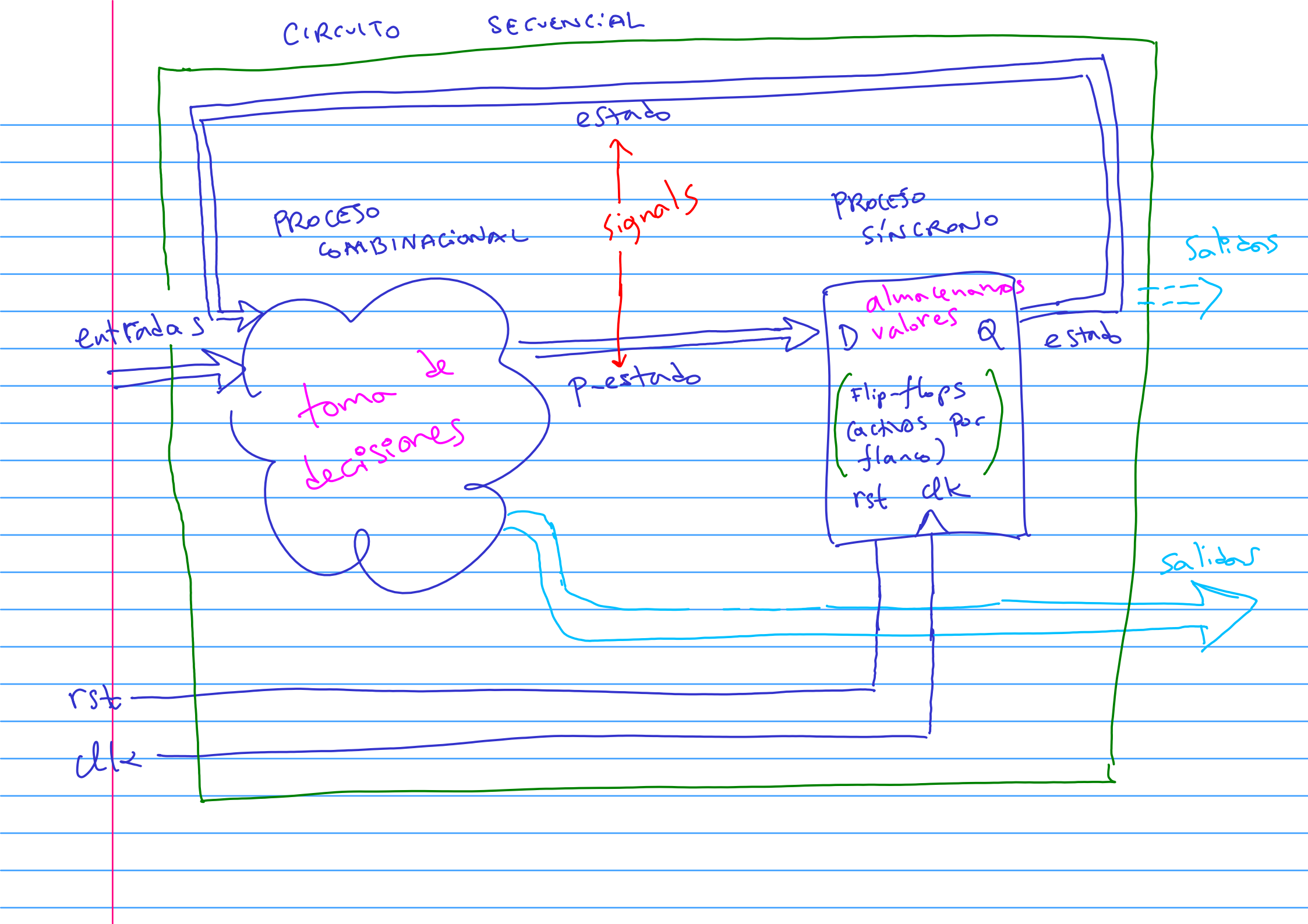
Fernando Muñoz Chavero

Febrero 2013

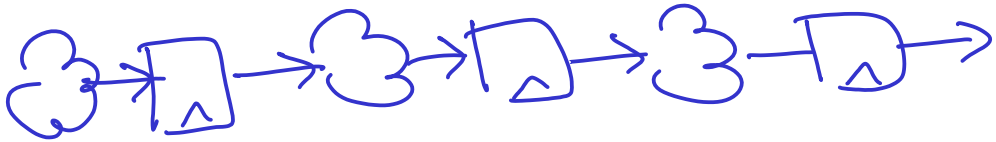
Clase 3:

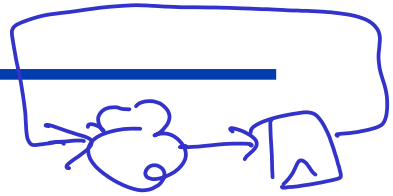
Diseño de circuitos síncronos

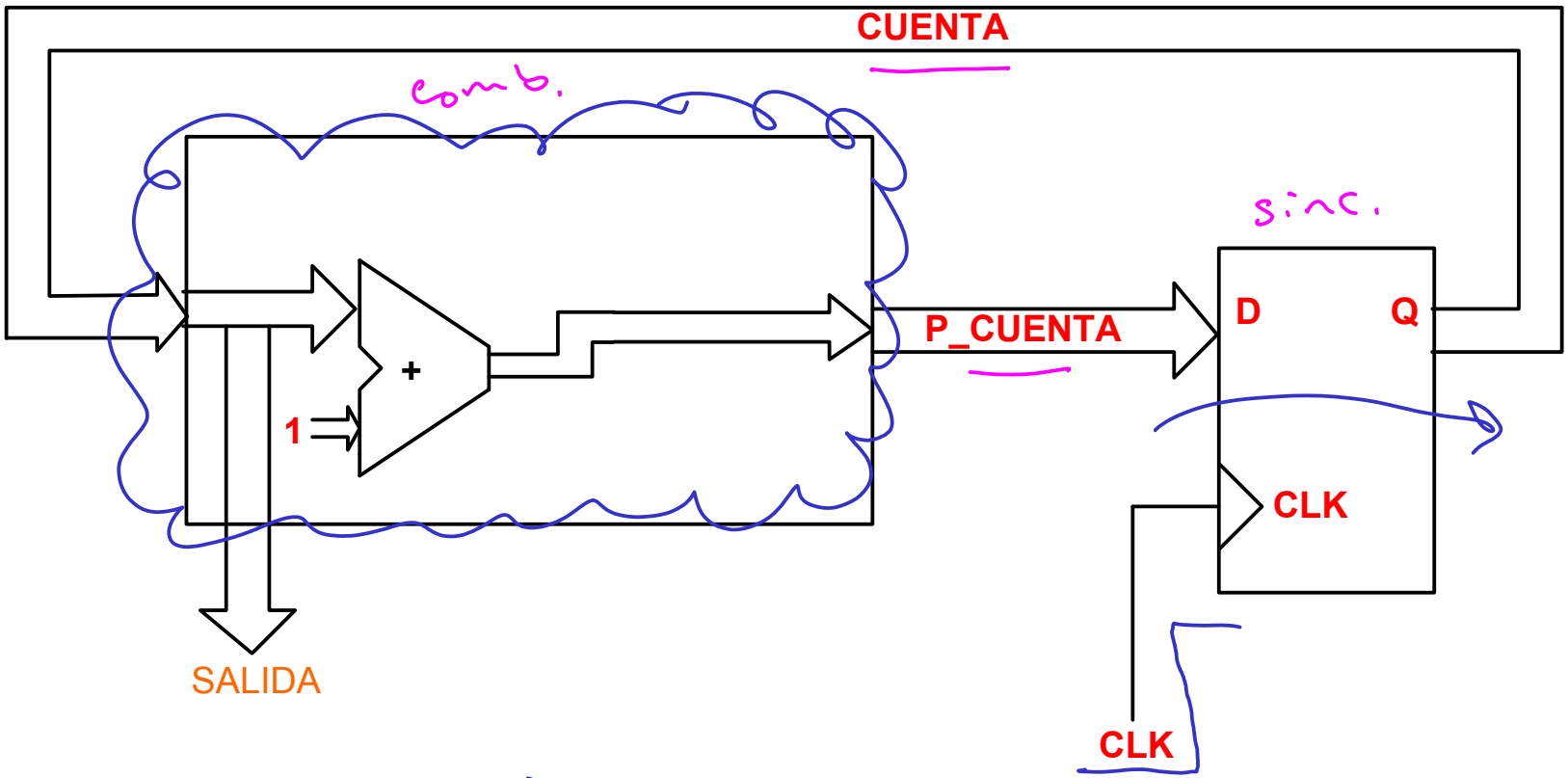
3.1 Descripción en dos procesos
3.2 Definición de nuevos tipos de datos
3.3 Máquinas de estados
3.4 Atributos

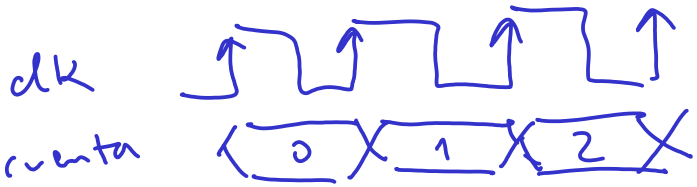

CIRCUITO SECUENCIAL
estado
PROCESO COMBINACIONAL
signals
PROCESO SÍNCRONO
salidas
entradas
toma de decisiones
p_estado
almacenamos valores
D
Q
estado
Flip-flops (activos por flanco)
rst
clk
salidas
rst
clk

Contador

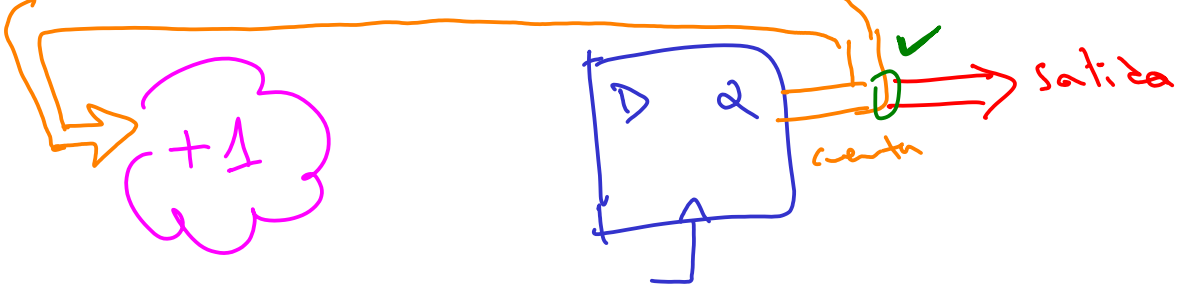








Contador

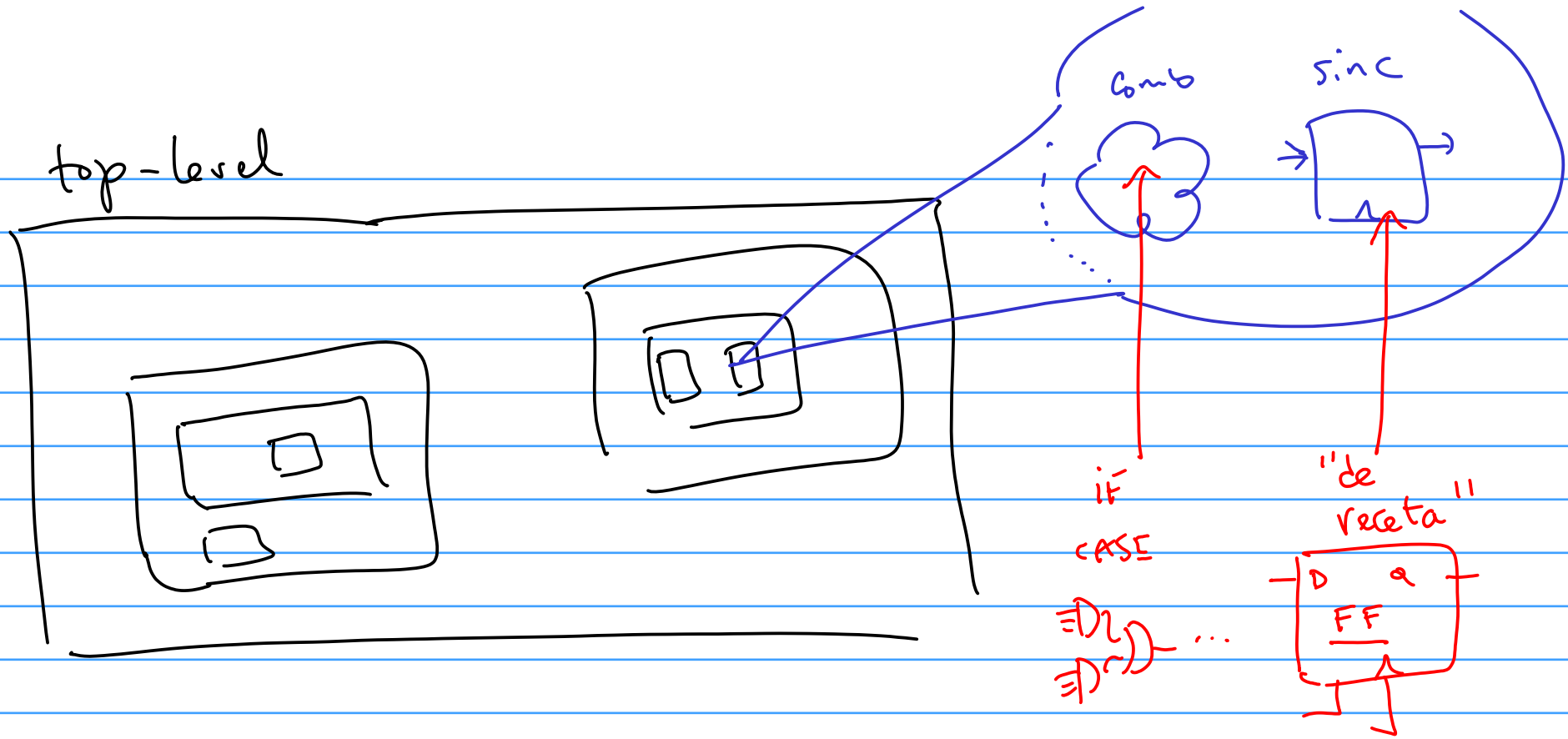



```
Architecture cont_arch of cont is
  signal cuenta, p_cuenta: std_logic_vector(7 downto 0);
  begin

  comb:process(cuenta)
  begin
    p_cuenta<=cuenta+1;
  end process;
  salida<=cuenta;

  sinc:process(clk)
  begin
  if(clk='1' and clk'event) then
    cuenta <= p_cuenta;
  end if;
  end process;
end cont_arch;
```

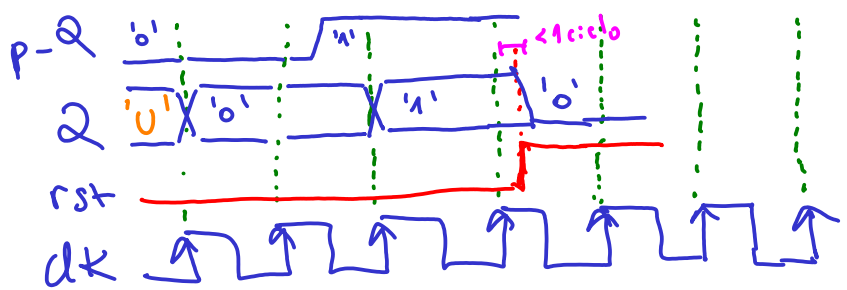
estado p_estado unsigned

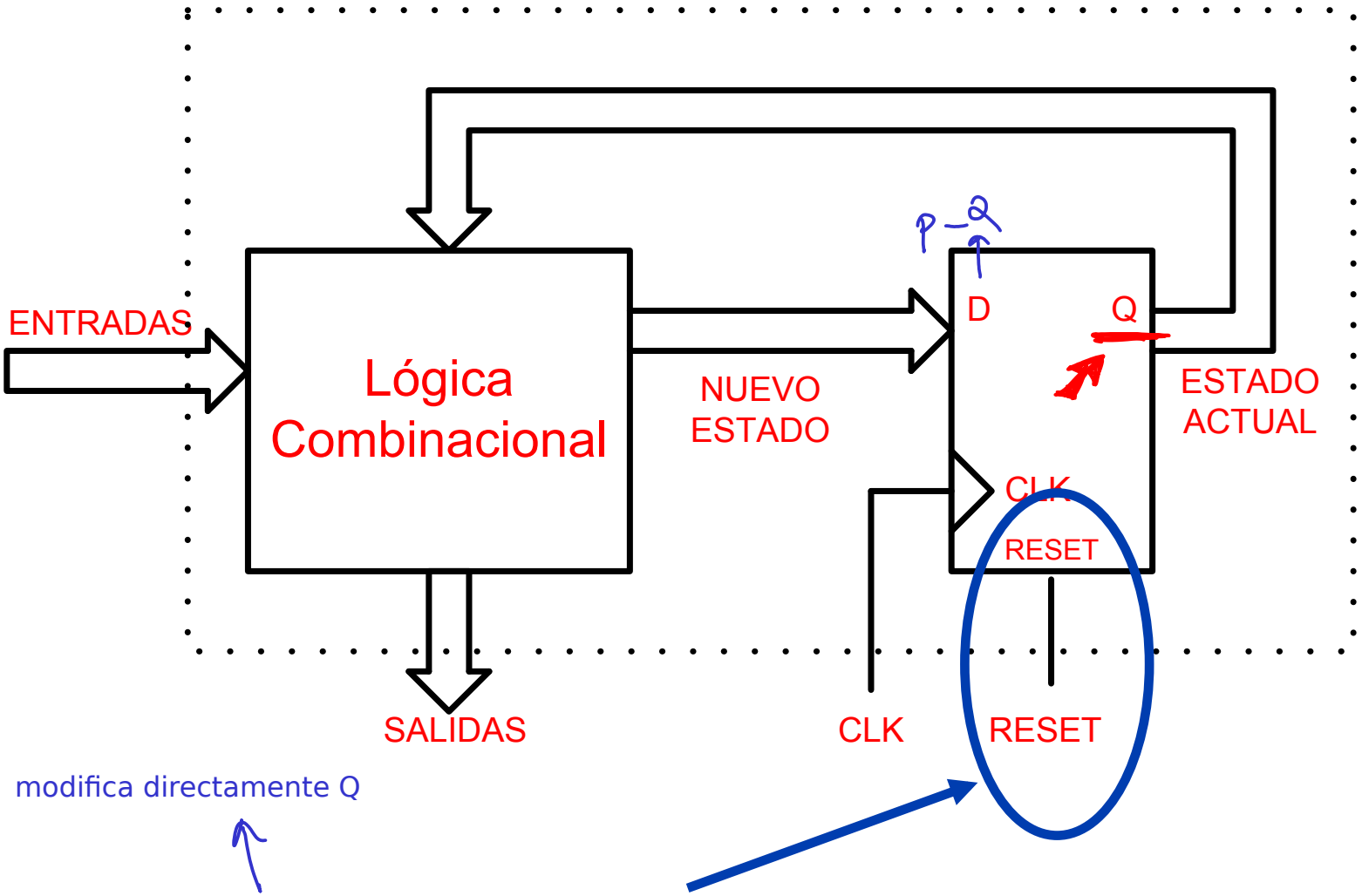
Sentencia concurrente

```
sinc: process (clk)
begin
  if rising_edge(clk) then
   cuenta <= p_cuenta;
  end if;
end process;
```


top-level
comb
sinc
if
case
...
"de
receta"
D
Q
FF

Reset asíncrono



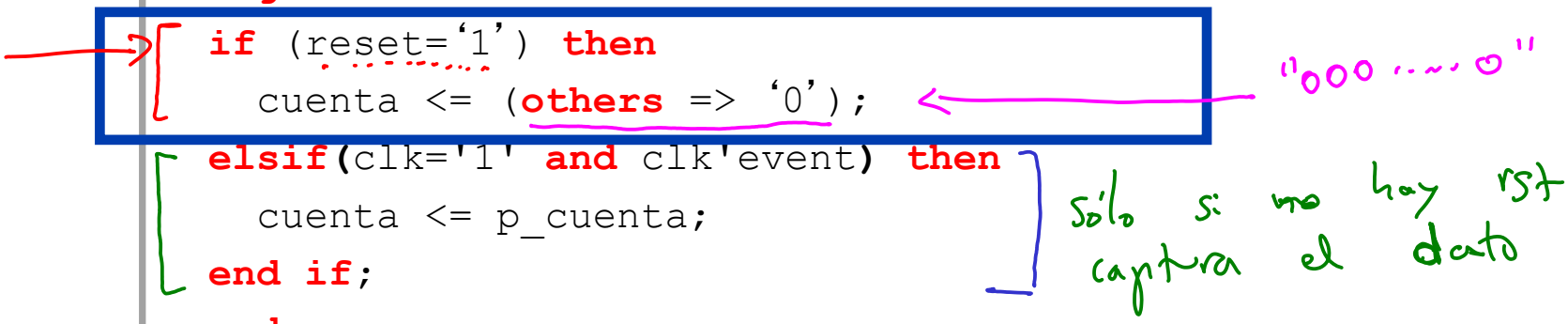


El reset asíncrono se define en el proceso síncrono

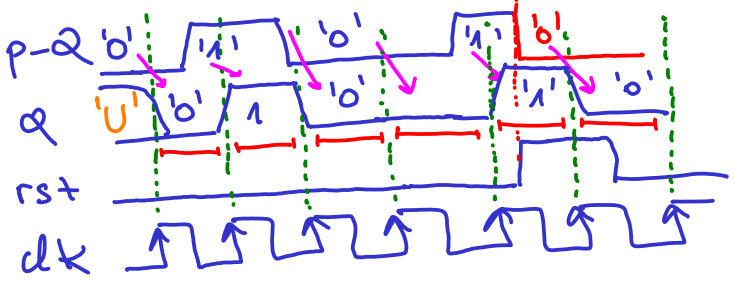
(debe actuar de independientemente al flanco de clk)

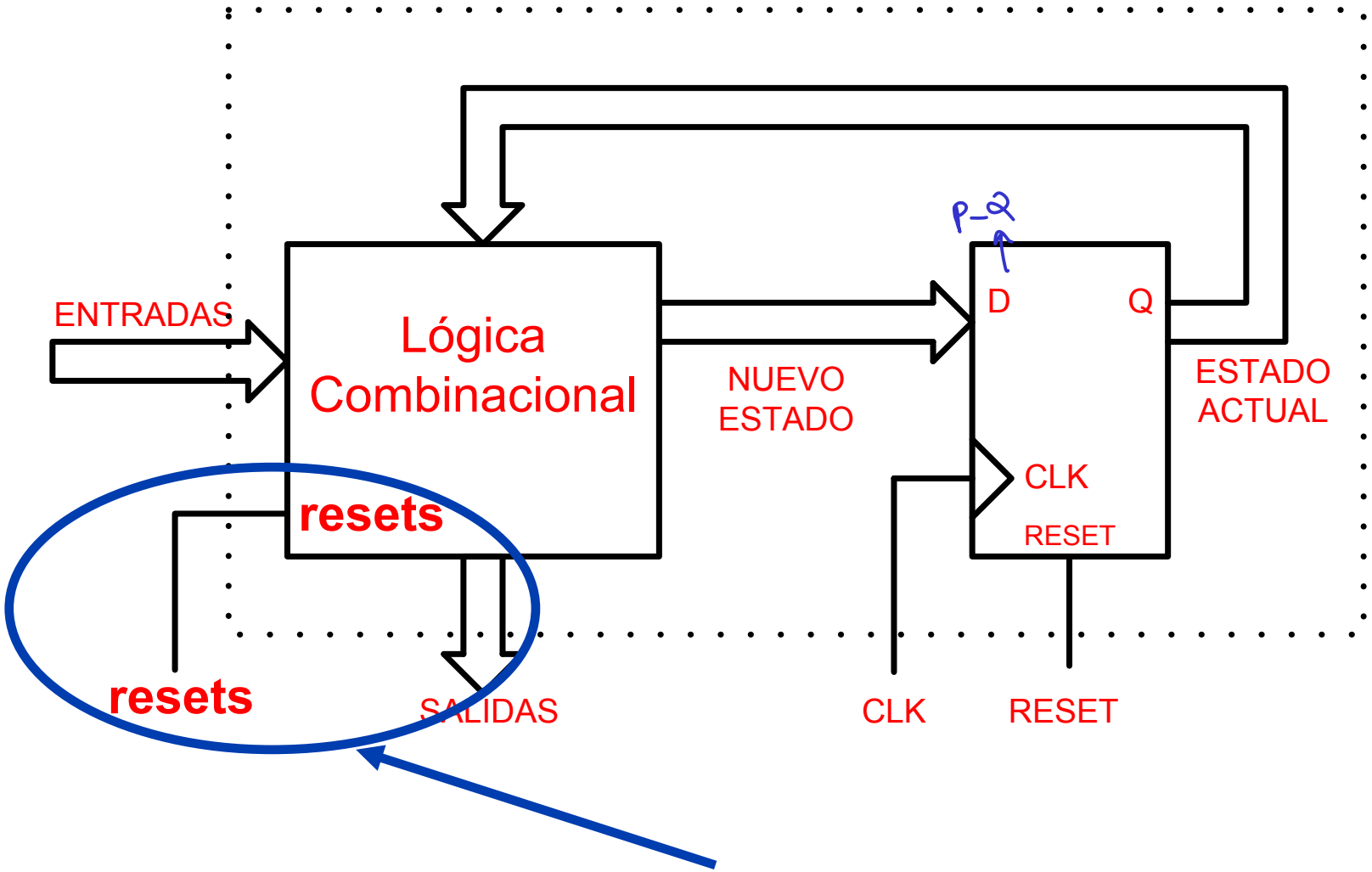
Contador

```
Architecture cont_arch of cont is
  signal cuenta, p_cuenta: std_logic_vector(7 downto 0);
  begin
  comb:process(cuenta)
  begin
    p_cuenta<=cuenta+1;
end process;
 salida<=cuenta;
  sinc:process(clk, reset)
  begin
  if (reset='1') then
    cuenta <= (others => '0');
  elsif(clk='1' and clk'event) then
    cuenta <= p_cuenta;
  end if;
  end process;
End cont_arch;
```



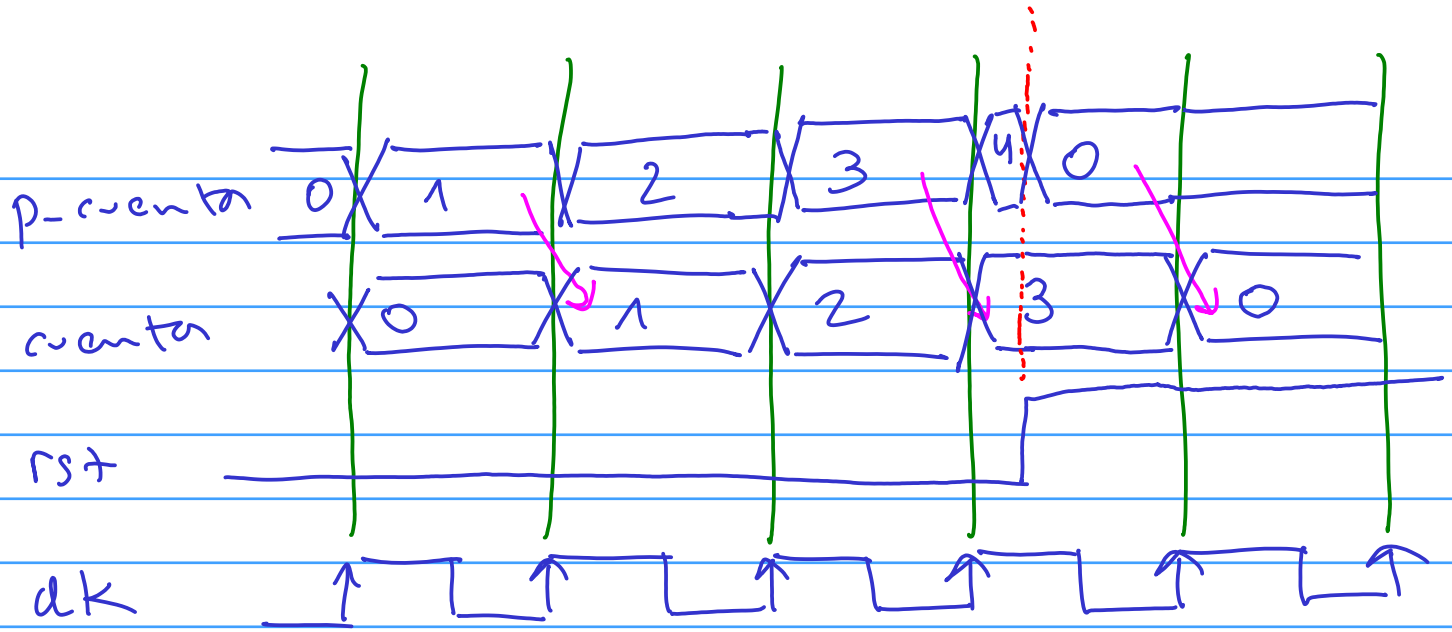
Reset síncrono (clear)

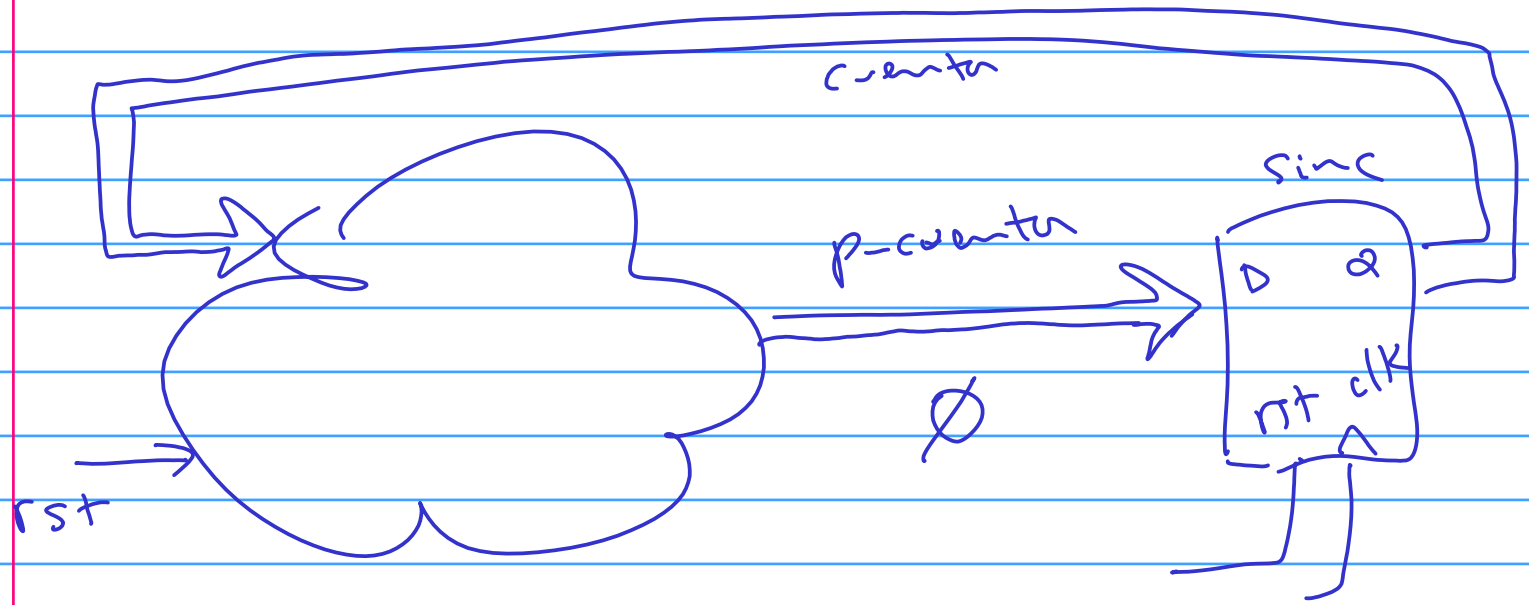




El reset síncrono se define en el proceso combinacional

(debe actuar en el flanco de clk)






```
if rst = '1' then
  p_cuenta <= 0;
else
  -- funcionamiento normal
  p_cuenta <= cuenta + 1;
```

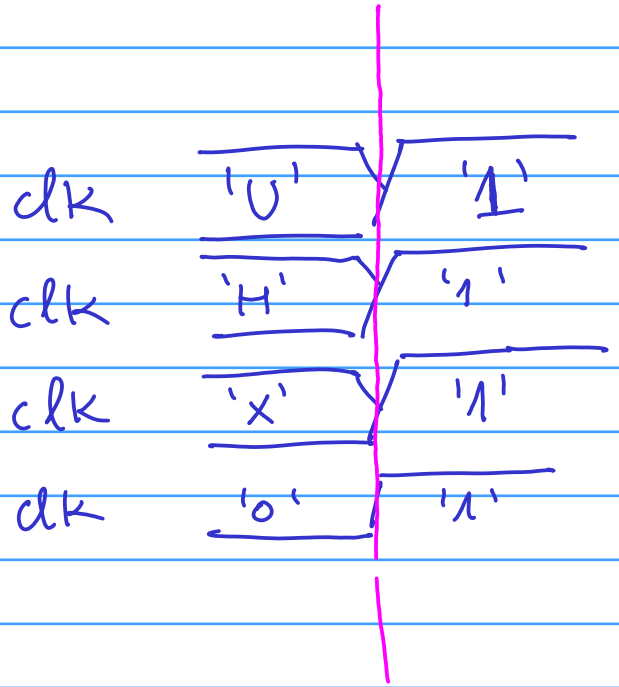
Contador

```
Architecture cont_arch of cont is
  signal cuenta, p_cuenta: std_logic_vector(7 downto 0);
  begin
  comb:process(cuenta,resets)
  begin
if (resets='1') then
      p_cuenta <= (others => '0');
   else
      p_cuenta<=cuenta+1;
   end if;
  end process;
 salida<=cuenta;
  sinc:process(clk,reset)
  begin
  if (reset='1') then
    cuenta <= (others => '0');
  elsif(clk='1' and clk'event) then
    cuenta <= p_cuenta;
  end if;
  end process;
End cont_arch;
```

asynchronous

mejor rising_edge(clk)

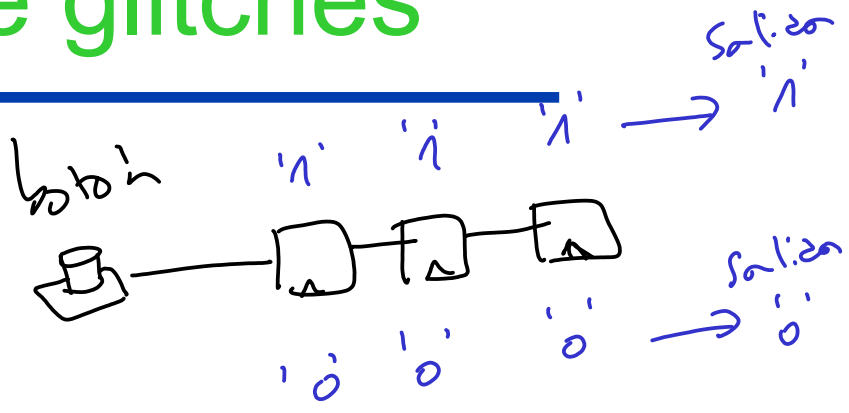
rising_edge(clk) es verdadero sólo si
clk = '1' and clk'event and clk'last_value='0'



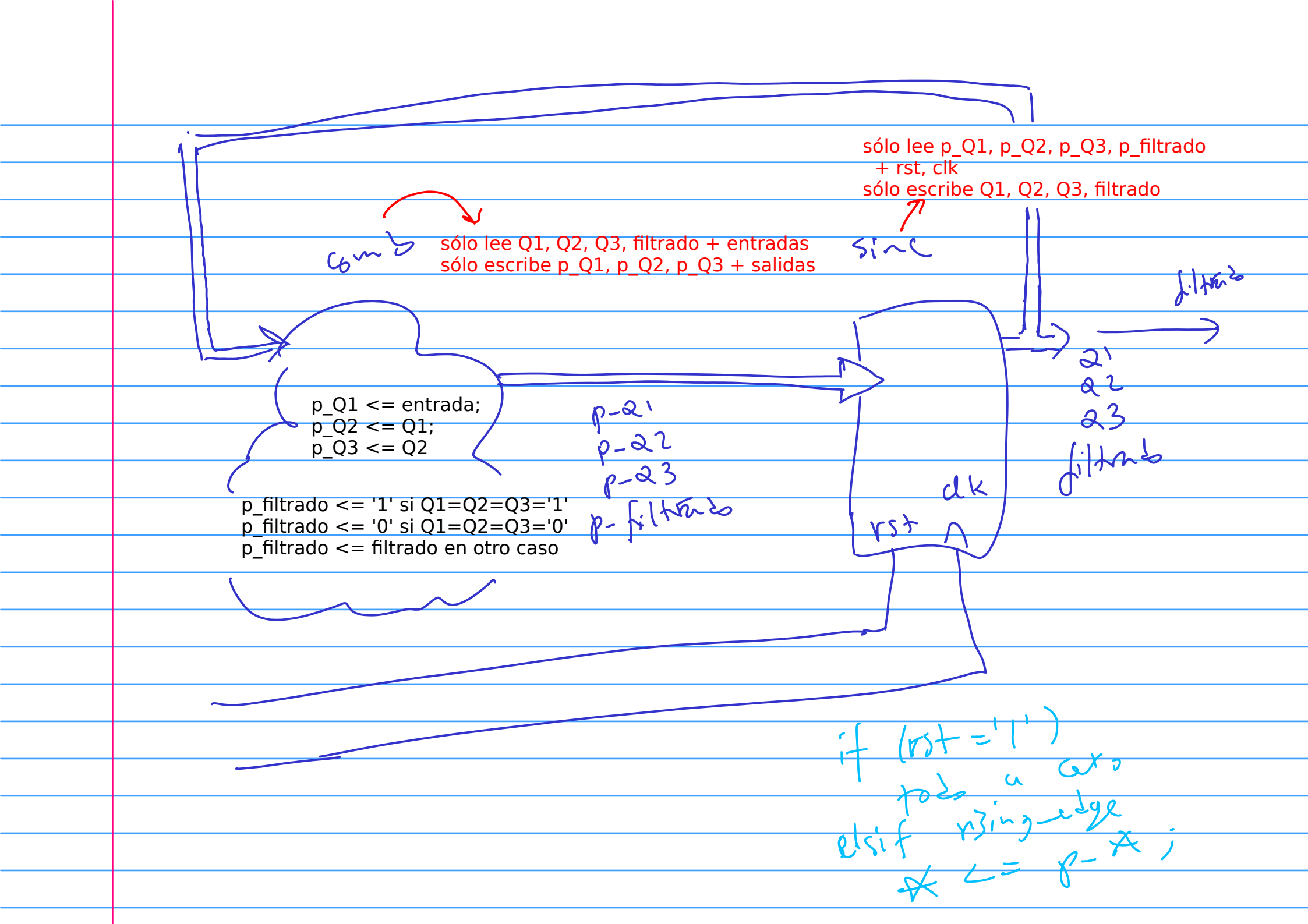
	clk='1' and clk'event	rising_edge(clk)
clk 'U' '1'	true	false
clk 'H' '1'	true	false
clk 'X' '1'	true	false
clk '0' '1'	true	true

En industria se recomienda el uso de rising_edge(clk)
En síntesis no hay diferencias
En simulación no hay diferencias si nuestro clk no toma valores distintos de '0' y '1'
(sólo toma valores 'raros' en diseños complejos con múltiples relojes)

Ejemplo -- Un filtro de glitches



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY filtro IS
   PORT (
      entrada: IN STD_LOGIC;
      filtrado: OUT STD_LOGIC;
      clk: IN STD_LOGIC;
      rst: IN STD_LOGIC
   );
END filtro;
ARCHITECTURE filtro_arch OF filtro IS
SIGNAL Q1, Q2, Q3: std_logic; -- muestras
SIGNAL Q1p, Q2p, Q3p: std_logic; -- proximas
SIGNAL filtrado_p, filtrado_a: std_logic; -- estado
BEGIN
```


sólo lee p_Q1, p_Q2, p_Q3, p_filtrado
+ rst, clk
sólo escribe Q1, Q2, Q3, filtrado
comb
sólo lee Q1, Q2, Q3, filtrado + entradas
sólo escribe p_Q1, p_Q2, p_Q3 + salidas
sinc
p_Q1 <= entrada;
p_Q2 <= Q1;
p_Q3 <= Q2
p_filtrado <= '1' si Q1=Q2=Q3='1'
p_filtrado <= '0' si Q1=Q2=Q3='0'
p_filtrado <= filtrado en otro caso
p_Q1
p_Q2
p_Q3
p_filtrado
Q1
Q2
Q3
filtrado
clk
rst
if (rst='1')
todo a cero
elsif rising_edge
* <= p_*;

Proceso combinacional

Filtrado_a es el "estado", Q1, Q2, Q3 las ultimas entradas

```
comb: PROCESS(entrada, Q1, Q2, Q3, filtrado_a)
  BEGIN
-- si estamos en estado filtrado_a = '0', solamente cambiamos después
-- de ver tres '1's
  IF (Q1 = '1' AND Q2 = '1' AND Q3 = '1' AND filtrado_a = '0') THEN
    filtrado_p <= '1';
   -- Lo mismo para cambiar de  filtrado_a = '1' a filtrado_a = '0'
  ELSIF (Q1 = '0' AND Q2 = '0' AND Q3 = '0' AND filtrado_a = '1') THEN
    filtrado_p <= '0';
  ELSE
    filtrado_p <= filtrado_a; -- no cambiar nada
  END IF;

  Q1p <= entrada; -- "registros de desplazamiento"
  Q2p <= Q1;
  Q3p <= Q2;
 END PROCESS;
```

Proceso sincrono

```
sync: PROCESS(rst, clk)
  BEGIN
   IF (rst = '1') THEN
     Q1 <= '0';
     Q2 <= '0';
     Q3 <= '0';
     filtrado_a <= '0';
   ELSIF (clk = '1' AND clk'EVENT) THEN
     Q1 <= Q1p;
     Q2 <= Q2p;
     Q3 <= Q3p;
     filtrado_a <= filtrado_p;
   END IF;
  END PROCESS;

  filtrado <= filtrado_a;
```

mejor rising-edge

estado <= p_estado

conecta el estado interno a la salida

¿Cuándo se usa cada reset?

El reset asíncrono se utiliza para llevar a TODO el circuito a un estado conocido al principio de la operación (~ tiempo 0)
(se suele hablar de "global reset")

El reset síncrono se utliza cuando durante la operación de un circuito necesitamos llevar a uno de sus submódulos a un estado conocido de manera ordenada y sin posibles problemas de temporización

Clase 3:

Diseño de circuitos síncronos

3.1 Descripción en dos procesos
3.2 Definición de nuevos tipos de datos
3.3 Máquinas de estados
3.4 Atributos

Definición de tipo de datos

- VHDL permite introducir nuevos tipos de datos.

```
Type nombre_del_tipo is definición_del_tipo;
```

```
Type Dia_mes is integer range 1 to 31;
```

- Una vez definidos, cualquier objeto puede ser declarados con ese nuevo tipo.

```
Signal dia_hoy, dia_manyana: Dia_mes;
```

- No está permitido realizar asignaciones entre objetos de tipos diferentes. Es necesario utilizar funciones de conversión de tipos.

```
Dia_manyana <= dia_hoy+1;
```

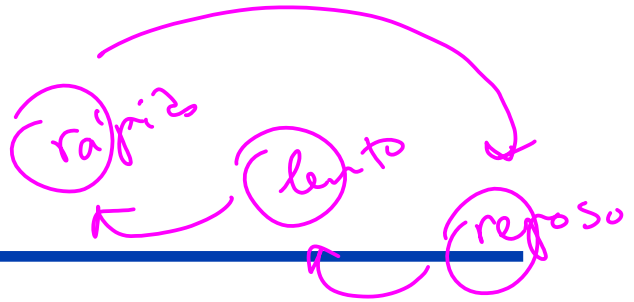
→ ***Deben ser del mismo tipo***

integer

```
Type otro_tipo is range 1 to 31;
Signal otro_dia: otro_tipo;
…….
Otro_dia<=dia_hoy;
```

No se puede realizar la asignación, aunque los tipos sean idénticos.

Tipos enumerados



- Consisten en una lista de valores que podrá tomar el objeto (señal, variable …), normalmente **se utilizan para implementar máquinas de estado.**

```
Type estado is (reposo, estado1, estado2, estado3);
```

- VHDL le asigna un número a cada posible valor, por orden de declaración. Por ejemplo, podemos decir que `estado2>reposo`.
- El std_ulogic de la librería del IEEE es un tipo enumerado.

```
Type std_ulogic is ('U','X','0','1','Z','W','L','H','-');
```

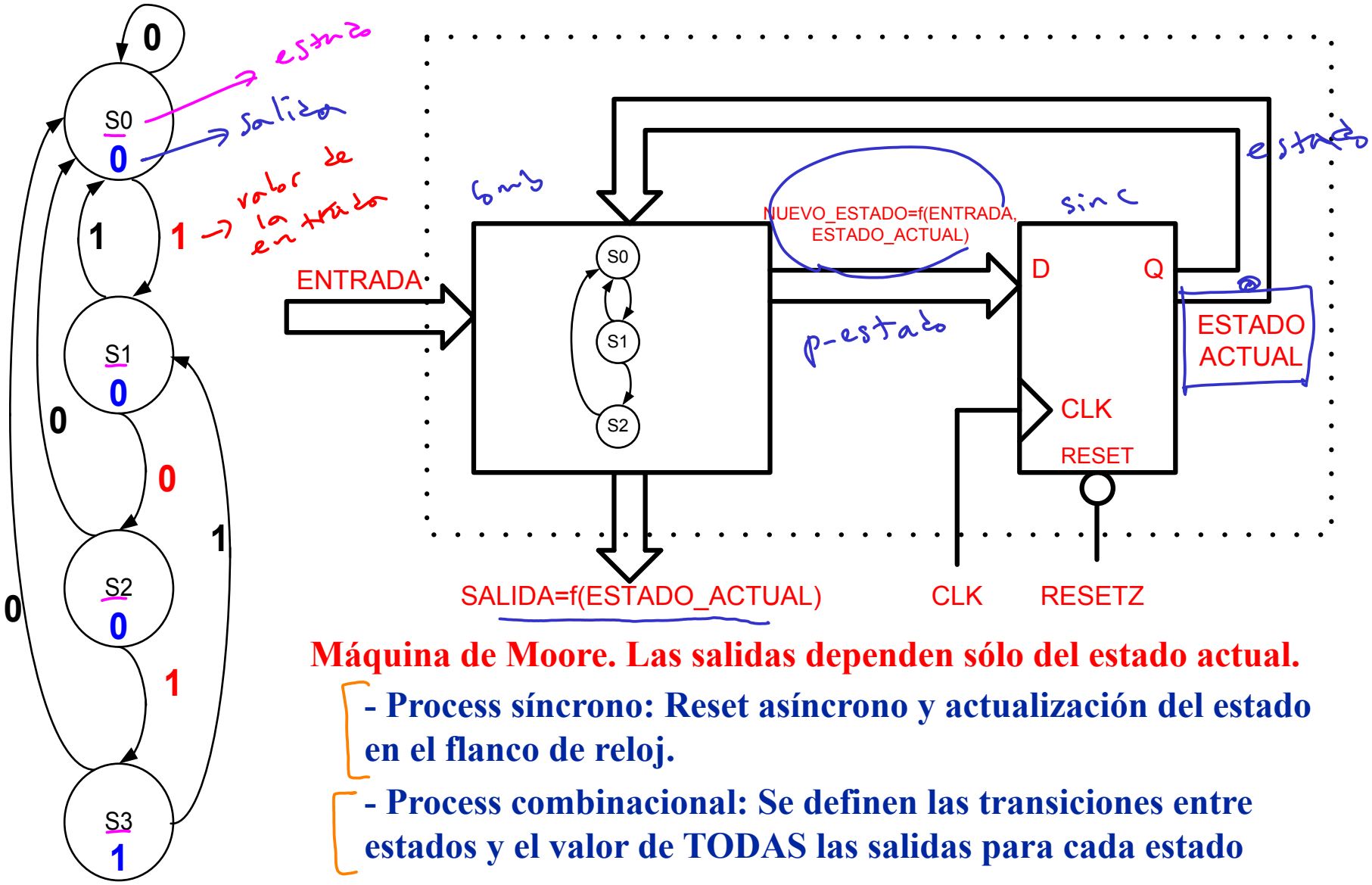
type t_estado is (reposo, marcha, lento, rapido, ...);
signal estado, p_estado: t_estado;

Clase 3:

Diseño de circuitos síncronos

3.1 Descripción en dos procesos
3.2 Definición de nuevos tipos de datos
3.3 Máquinas de estados
3.4 Atributos

Máquina de estados

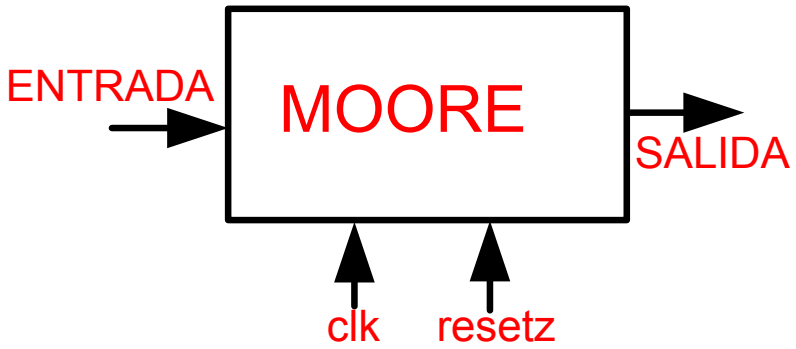


Máquina de Moore. Las salidas dependen sólo del estado actual.

- **Process síncrono: Reset asíncrono y actualización del estado en el flanco de reloj.**
- **Process combinacional: Se definen las transiciones entre estados y el valor de TODAS las salidas para cada estado**

Máquina de Estados

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MOORE is
  port(entrada: in std_logic;
       clk: in std_logic;
       resetz: in std_logic;
       salida: out std_logic);
end MOORE;
```




```
architecture A of MOORE is
  type estado is (S0, S1, S2, S3);
  signal estado_actual: estado;
  signal  estado_nuevo: estado;
begin
  process(resetz,clk)
  begin
    if(resetz= '0') then
      estado_actual <= S0;
    elsif (clk='1' and clk'event) then
      estado_actual<= estado_nuevo;
    end if;
  end process;
```

Comb:

opción 1 (no es la mejor)

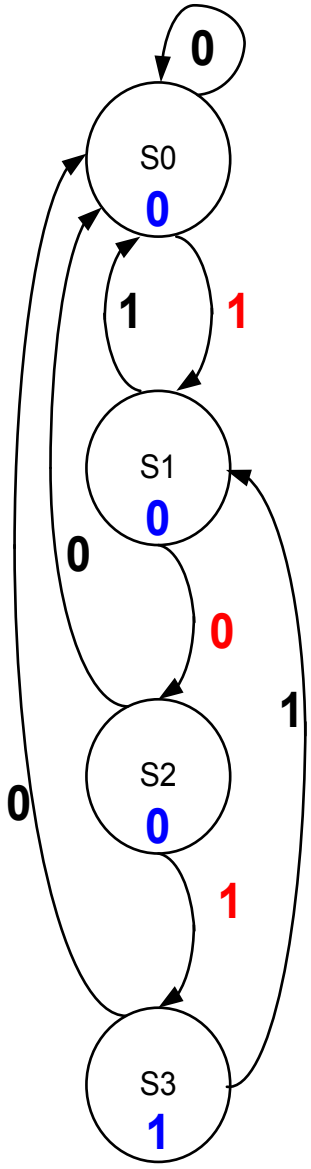
```
if estado = S0 then
   ==========
elsif estado = S1 then
   ==========
   if(...)
   :
```

opción 2: ✓

```
case estado is
   when S0 =>
   (  ==========
   when S1 =>
   (  ==========
   (  if(....)
   :
   when others =>

end case;
```

Máquina de Estados

```
process(estado_actual, entrada)
 begin
case estado_actual is
  when S0 =>
      salida <= '0';
      if entrada = '0' then
       estado_nuevo <= S0;
      else
        estado_nuevo <= S1;
      end if;
  when S1 =>
      salida <= '0';
      if entrada = '1' then
        estado_nuevo <= S0;
      else
        estado_nuevo <= S2;
      end if;
```



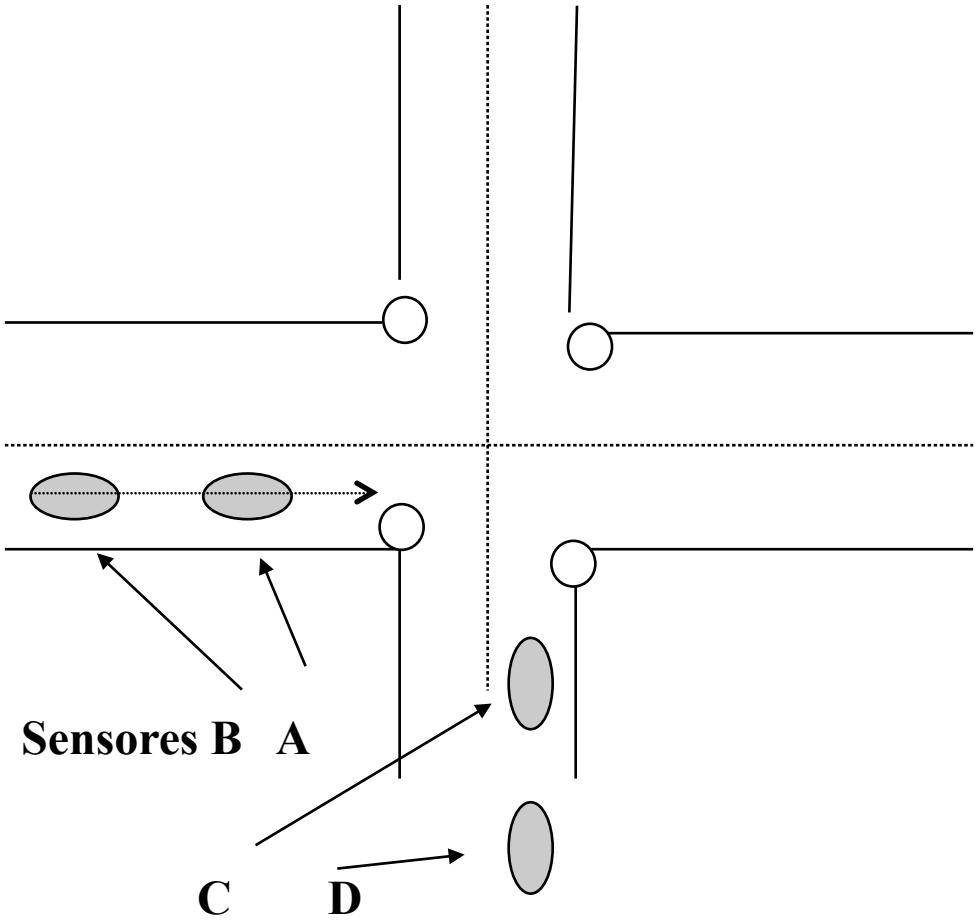

```
  when S2 =>
      salida <= '0';
      if entrada = '0' then
        estado_nuevo <= S0;
      else
        estado_nuevo <= S3;
      end if;
  when S3 =>
      salida <= '1';
      if entrada = '0' then
        estado_nuevo <= S0;
      else
        estado_nuevo <= S1;
      end if;
end case;
end process;
end A;
```

Problema

Implementa un sistema de control de semáforos que modifica el periodo de las luces según el estado de 4 sensores (A.B,C,D) debajo de dos de las calles.

Durante trafico normal los semáforos cambian cada 1000 ciclos de reloj. Pero cuando cada sensor de la calle en rojo detecta un coche estacionado (señalizado con un valor de lógica alta), la cuenta se reduce en 200 ciclos.

Situación de los sensores



Entity

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_unsigned.all;
ENTITY sf IS
   PORT (
      clk: IN STD_LOGIC;
      rst: IN STD_LOGIC;
      S_a: IN STD_LOGIC; -- sensores A
      S_b: IN STD_LOGIC; -- B
      S_c: IN STD_LOGIC; -- C
      S_d: IN STD_LOGIC; -- D
      rav0: OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
       -- Luces rojo, amarillo, verde de calle 0
      rav1: OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
       -- Luces de calle 1     );
END sf;
```

Architecture I – contador de ciclos

```
ARCHITECTURE sf_arch OF sf IS
SIGNAL fin_cuenta, cero_cuenta: std_logic;
SIGNAL cuenta, prox_cuenta: integer range 0 TO 1023;
TYPE mi_estado IS (rojo0, ra0, rr0, rojo1, ra1, rr1);
SIGNAL estado, prox_estado : mi_estado;
BEGIN

 -- un contador con reset _sincrono_
  cont: PROCESS(cuenta, cero_cuenta)
  BEGIN -- contado con reset sincrono
    IF (cero_cuenta = '1') THEN
      prox_cuenta <= 0;
    ELSE
      prox_cuenta <= cuenta + 1;
   END IF;
  END PROCESS;
```

Architecture II – control de tiempo

```
fin_estado: PROCESS(S_a, S_b, S_c, S_d, estado, cuenta)
  VARIABLE max_cuenta : integer range 0 TO 1023;
  BEGIN
    max_cuenta := 999;
    IF (estado = rojo0) THEN
      IF (S_a = '1') THEN
        max_cuenta := 799;
        IF (S_b = '1') THEN
          max_cuenta := 599;
        END IF;
      END IF;
    ELSE -- otro caso de importancia es rojo1
      IF (S_c = '1') THEN
        max_cuenta := 799;
        IF (S_d = '1') THEN
          max_cuenta := 599;
        END IF;
      END IF;
    END IF;
```

```
IF cuenta >= max_cuenta THEN
    fin_cuenta <= '1';
  ELSE
    fin_cuenta <= '0';
  END IF;
 END PROCESS;
```

Architecture III – La máquina

```
fsm: PROCESS(cuenta, estado,
     fin_cuenta)
  BEGIN
  prox_estado <= estado;
  cero_cuenta <= '0';
  CASE estado IS
    WHEN  rojo0 =>
      IF fin_cuenta = '1' THEN
        prox_estado <= ra0;
      END IF;
      rav0 <= "100"; rav1 <= "001";
    WHEN ra0 =>
      prox_estado <= rr0;
      rav0 <= "100";  rav1 <= "010";
    WHEN rr0 =>
      prox_estado <= rojo1;
      rav0 <= "100";  rav1 <= "100";
      cero_cuenta <= '1';
```

```
WHEN rojo1 =>
      IF fin_cuenta = '1' THEN
        prox_estado <=ra1;
      END IF;
      rav0 <= "001";  rav1 <= "100";
    WHEN ra1 =>
      prox_estado <= rr1;
      rav0 <= "010";  rav1 <= "100";
    WHEN rr1 =>
      prox_estado <= rojo0;
      rav0 <= "100";  rav1 <= "100";
      cero_cuenta <= '1';
  END CASE;
 END PROCESS;
```

Architecture IV – Los registros de estado

```
sync: PROCESS(rst, clk)
  BEGIN
  IF (rst = '0') THEN
    cuenta <= 0;
    estado <= rojo0;
  ELSIF (clk = '1' AND clk'EVENT) THEN
    cuenta <= prox_cuenta;
    estado <= prox_estado;
  END IF;
  END PROCESS;
END sf_arch;
```

Clase 3:

Diseño de circuitos síncronos

3.1 Descripción en dos procesos
3.2 Definición de nuevos tipos de datos
3.3 Máquinas de estados
3.4 Atributos

Atributos

clk'event
clk'last_value

- Un atributo es una característica asociada a un elemento (tipo de dato, señal, entidad,…) que proporciona información adicional.
- **Atributo ≠ Valor**
  - Un objeto tiene un solo valor y puede tener múltiples atributos.
- VHDL proporciona una serie de atributos predefinidos.

```
Nombre_elemento'Nombre_atributo
```

```
Byte_a'range
Signal_a'event
```

Atributos de rango vectores

Nombre del atributo	Significado
LEFT	Identifica el valor definido a la izquierda del intervalo de índices
~~RIGTH~~ RIGHT	Identifica el valor definido a la derecha del intervalo de índices
HIGH	Identifica el valor máximo del intervalo de índices
LOW	Identifica el valor mínimo del intervalo de índices
LENGTH	Identifica el valor total del intervalo de índices
RANGE	Copia el intervalo de índices
REVERSE_RANGE	intervalo de índices pero en sentido inverso.

Atributos de rango de vectores. Ejemplo

```
Signal Mi_senal: std_logic_vector(7 downto 2);
```

Nombre del atributo	Ejemplo
LEFT	Mi_senal' **left** toma el valor **7**
~~RIGTH~~ right	Mi_senal' **rigth** ht toma el valor **2**
HIGH	Mi_senal' **high** toma el valor **7**
LOW	Mi_senal' **low** toma el valor **2**
LENGTH	Mi_senal' **length** toma el valor **6**
RANGE	Mi_senal' **range** toma el valor **7 downto 2**
REVERSE_RANGE	Mi_senal' **reverse_range** toma el valor **2 to 7**

Permite la descripción de modelos más generales que facilitan la actualización del código

A downto B (A ≥ B)

C to D (C ≤ D)

7 downto ∅

∅ to 7

Atributos de rango de vectores. Loops

```
Signal Mi_senal: std_logic_vector(7 downto 2);
```

Bucles en VHDL

```
for i in Mi_senal'range
loop
   Mi_senal(i) <= '0'
end loop;
```

for i in 7 downto 2

Pon un vector de tamaño desconocido a cero

```
for i in Mi_senal'high to 1
loop
  Mi_senal(i) <= Mi_senal(i-1)
end loop;
```

Desplaza a la izquierda

i no es ni señal ni variable, los bucles son siempre estaticos

Atributos de las señales

- DELAYED(t). Valor de la señal retrasada t unidades de tiempo.
- STABLE(t), verdadero si la señal permanece invariable durante t unidades de tiempo.
- QUIET(t), verdadero si la señal no ha recibido ninguna asignación en t unidades de tiempo.
- TRANSACTION, tipo bit, a '1' cuando hay una asignación a la señal.
- EVENT, verdadero si ocurre un cambio en la señal en el paso de simulación.
- ACTIVE, verdadero si ocurre una asignación a la señal en el paso de simulación.
- LAST_EVENT, unidades de tiempo desde el último evento.
- LAST_ ACTIVE, unidades de tiempo desde la última asignación.
- LAST_VALUE, valor anterior de la señal.
- DRIVING, verdadero si el proceso actual determina el valor de la señal.
- DRIVING_VALUE, valor que toma la señal tras el proceso.