LABORATORIO DE CIRCUITOS DIGITALES
(2005-II)
CUARTA CLASE DE VHDL

✓ *CIRCUITOS SECUENCIALES*

- Procesos asíncronos (Latches)
- Procesos síncronos (flip flops, contadores y registros)

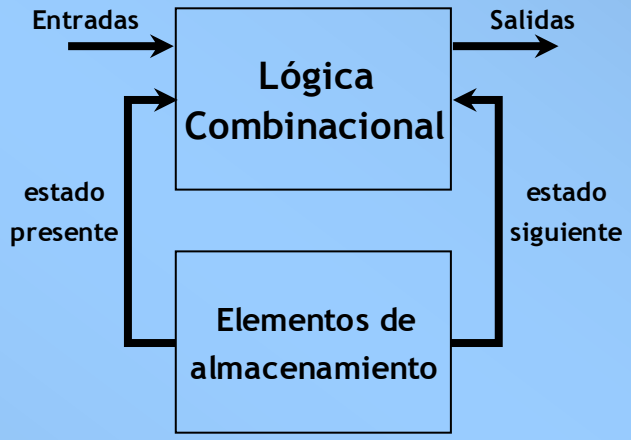
Instructor: Daniel Llamocca

✓ *CIRCUITOS SECUENCIALES*

- En los circuitos combinacionales, la salida depende sólo de los valores de entrada.
- Pero existe otra clase de circuitos lógicos cuyas salidas dependen tanto de los valores presentes en las entradas como del estado pasado del circuito. Estos circuitos incluyen elementos de almacenamiento para guardar los valores de las señales.
- El contenido de estos elementos de almacenamiento representa el ***estado*** del circuito. Al cambiar las entradas del circuito, puede suceder que el circuito se quede en un estado o cambie a otro. Con el tiempo, el circuito pasa por una secuencia de estados como resultado del cambio en las entradas. Los circuitos con este comportamiento se llaman circuitos secuenciales.

CIRCUITO COMBINACIONAL

CIRCUITO SECUENCIAL

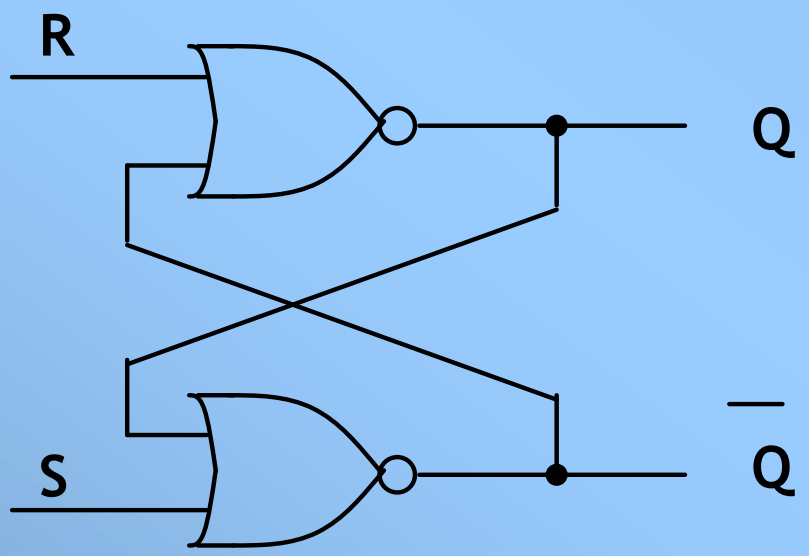


✓ *CIRCUITOS SECUENCIALES*

- Si bien los circuitos combinacionales pueden describirse con sentencias concurrentes o secuenciales, **los circuitos secuenciales sólo pueden describirse con sentencias secuenciales**.
- Los circuitos secuenciales pueden ser síncronos o asíncronos.
- El circuito secuencial asíncrono básico es el Latch.
- Los circuitos secuenciales síncronos básicos son: flip flops, contadores y registros.

✓ PROCESOS ASÍNCRONOS (LATCHES)

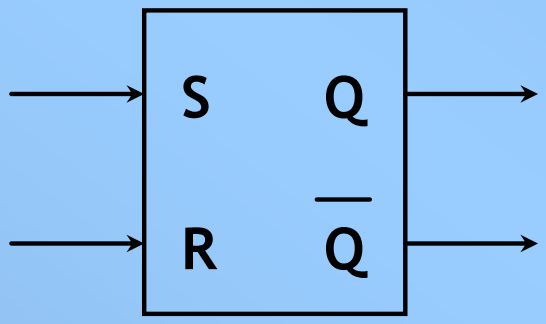
- **Latch SR**
- Un Latch (Retenedor) SR en base a compuertas NOR resulta:

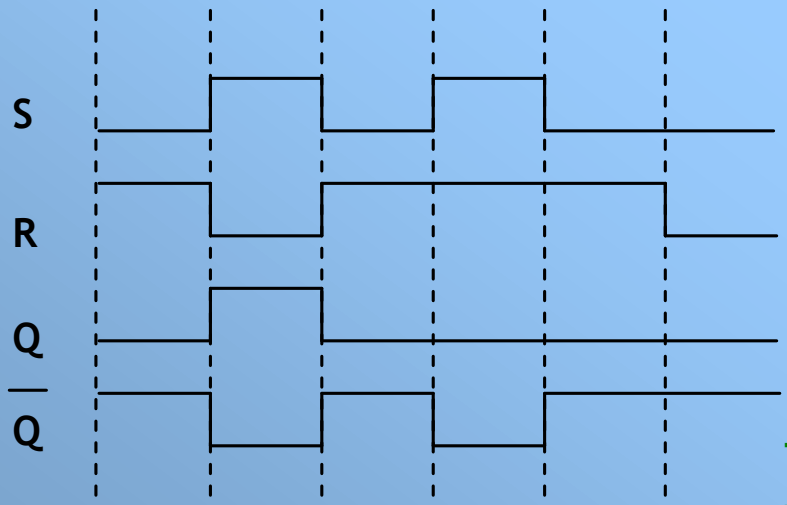


S	R	Q_{t+1}	$\overline{Q}_{t+1}$
0	0	Q_t	$\overline{Q}_t$
0	1	0	1
1	0	1	0
1	1	0	0

- Según su tabla de verdad, se le puede poner a la salida como ‘0’ o ‘1’ y este estado se almacena en el circuito siempre que S = R = ‘0’.

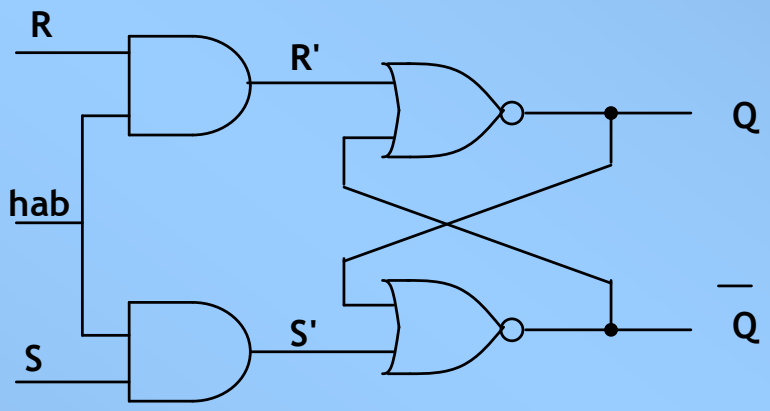
- Latch SR






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

entity latch_sr is
 port (s, r: in std_logic;
    q, qn: buffer std_logic);
end latch_sr;

architecture bhv of latch_sr is
begin
 process (s,r)
 begin
  if s = '1' and r = '0' then
   q <= '1'; qn <= '0';
  elsif s = '0' and r = '1' then
   q <= '0'; qn <= '1';
  elsif s = '1' and r = '1' then
   q <= '0'; qn <= '0';
  end if; -- No se indica lo que
-- sucedería si s = '0' y r = '0'
-- ==> 'q' y 'qn' mantienen su valor
 end process;
end bhv;
```

- **Latch SR con habilitador**

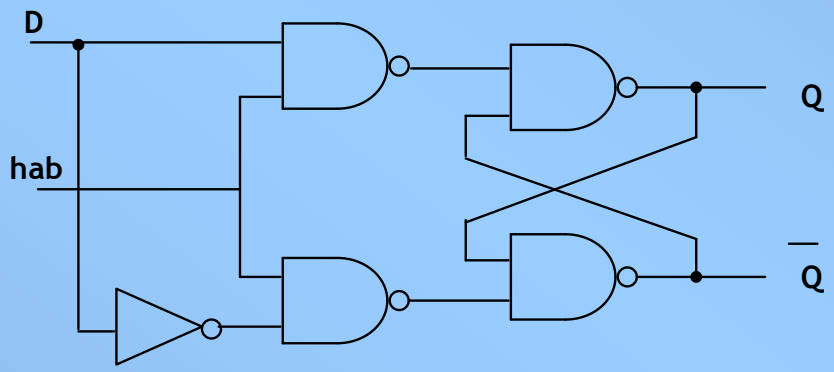


hab	S	R	Q_{t+1}	$\overline{Q}_{t+1}$
0	x	x	Q_t	$\overline{Q}_t$
1	0	0	Q_t	$\overline{Q}_t$
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

- Notar que si hab = ‘0’
  ➔ se retiene la salida

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

entity latch_sr_h is
 port (s, r, hab: in std_logic;
       q, qn: buffer std_logic);
end latch_sr_h;

architecture bhv of latch_sr_h is
begin
 process (s,r)
 begin
  if hab = '1' then
   if s = '1' and r = '0' then
    q <= '1'; qn <= '0';
   elsif s = '0' and r = '1' then
    q <= '0'; qn <= '1';
   elsif s = '1' and r = '1' then
    q <= '0'; qn <= '0';
   end if;
  end if;
 end process;
end bhv;
```

- Latch D con habilitador

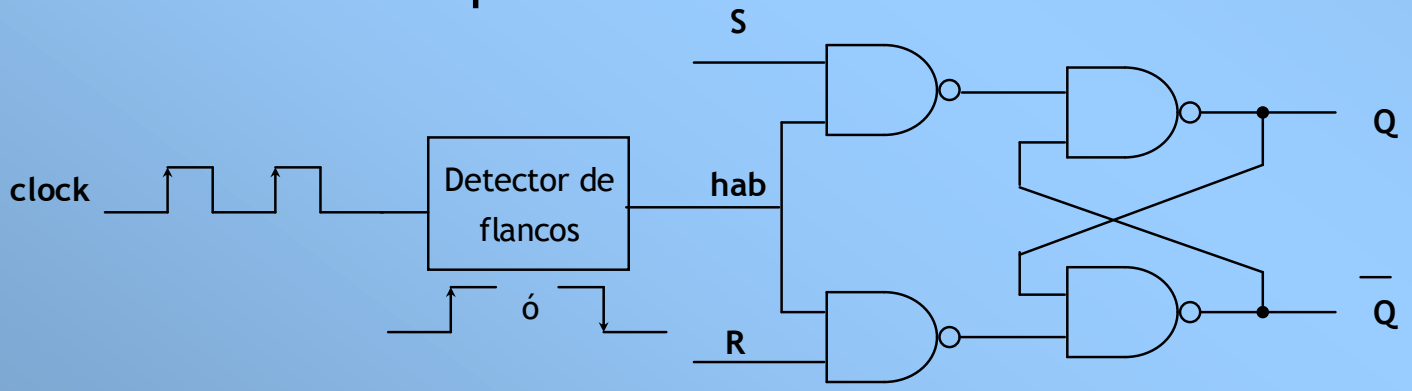


hab	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

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

entity latch_d_h is
 port (d, hab: in std_logic;
       q, qn: buffer std_logic);
end latch_d_h;

architecture bhv of latch_d_h is
begin
 process (d)
 begin
  if hab = '1' then
    q <= d; qn <= not (d);
  end if;
 end process;
end bhv;
```

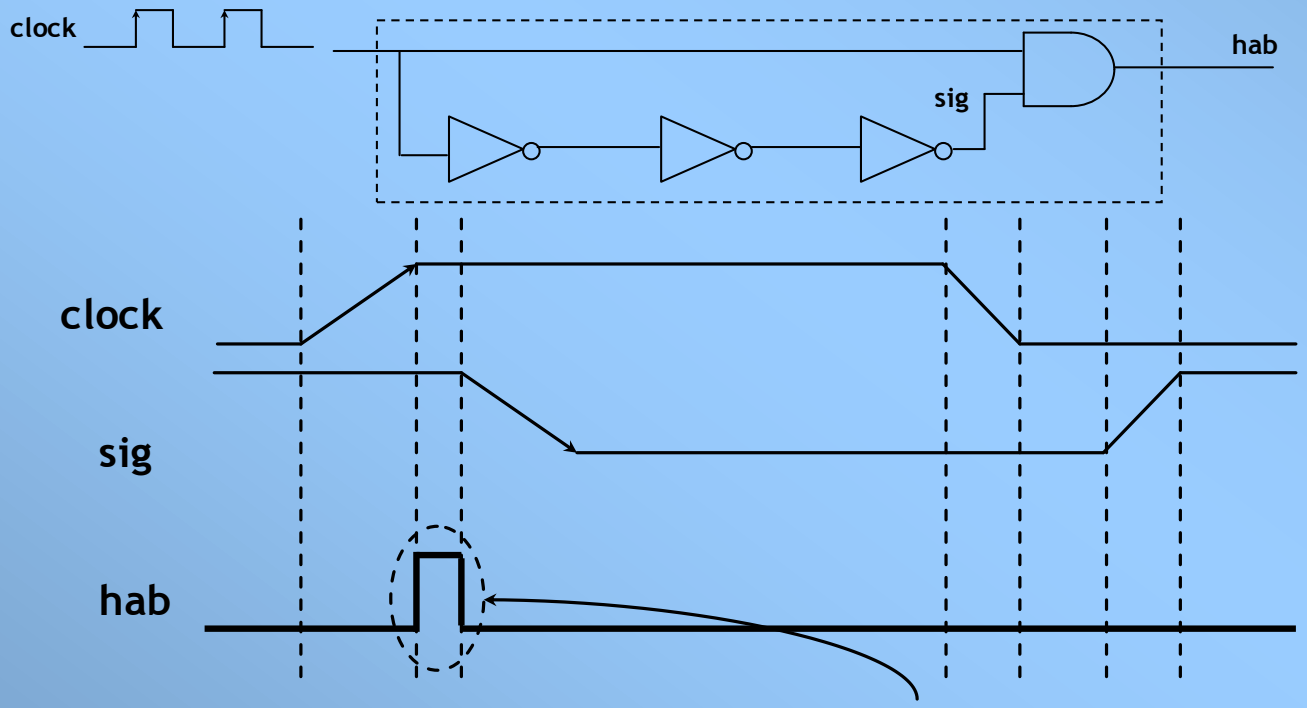
✓ PROCESOS SÍNCRONOS

- **Flip Flops**
- A diferencia del Latch, un flip flop sólo cambia su salida en el flanco (de subida ó bajada) de una señal (llamada señal de reloj). La salida sólo depende de los valores de entrada presentes cuando el flanco de la señal de reloj ocurre. Para esto existe un circuito detector de flancos, el que se activa si ocurre un flanco (de subida o bajada) en la señal de reloj, la que es una onda cuadrada de frecuencia fija, esto para que los cambios en el flip flop estén sincronizados con esta frecuencia.
- Se muestra un flip flop SR, el cual es un Latch SR en donde la señal de habilitación proviene del circuito detector de flancos.



✓ PROCESOS SÍNCRONOS

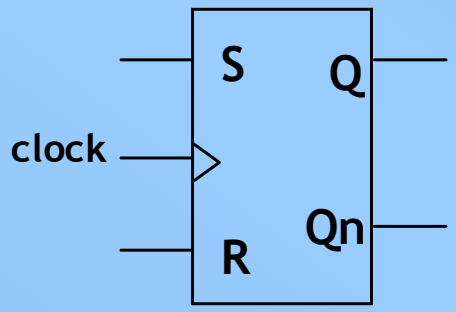
- **Flip Flops:** Recordar que el Latch SR sólo cambia sus salidas si hab = ‘1’. El circuito detector de flancos hace que hab = ‘1’ al detectar un flanco (de subida o de bajada). Se trabajará con circuitos activados por flanco de subida o de bajada, nunca los dos a la vez.
- Se muestra un circuito para detectar el flanco de subida. Las compuertas NOT redundantes causan un retardo tal que se genere un pulso de habilitación cada vez que ocurre un flanco.



Aquí el circuito cambia sus salidas

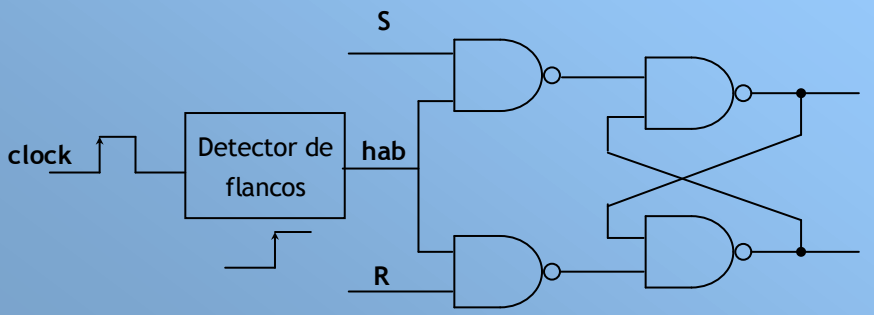
✓ PROCESOS SÍNCRONOS

- **Flip Flop SR**



Detecta el flanco de subida →

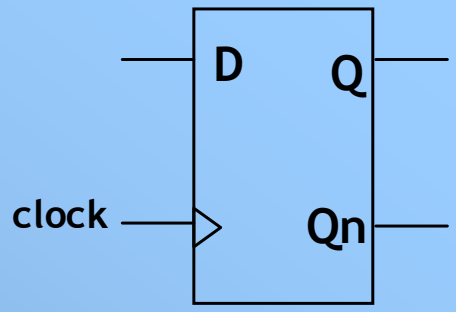
Detecta el flanco de bajada →



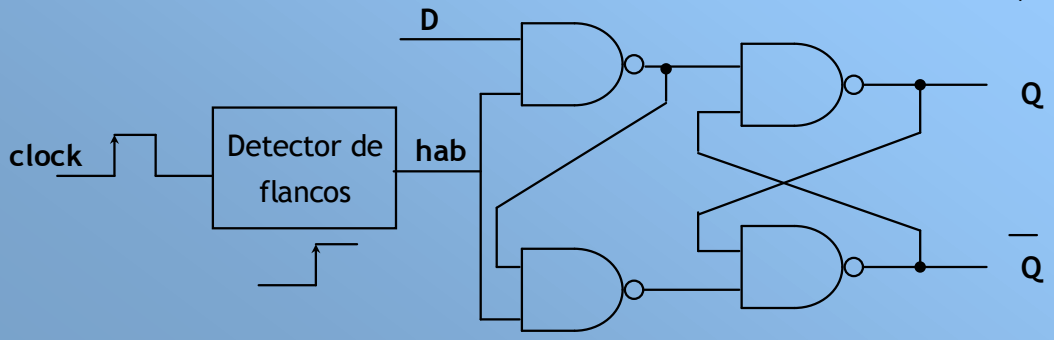

```
library ieee;
use ieee.std_logic_1164.all;

entity ff_sr is
 port (s, r, clock: in std_logic;
       q, qn: buffer std_logic);
end ff_sr;

architecture bhv of ff_sr is
begin
 process (s,r,clock)
 begin
  if (clock'event and clock = '1') then
  --if (clock'event and clock = '0') then
    if s = '1' and r = '0' then
     q <= '1'; qn <= '0';
    elsif s = '0' and r = '1' then
     q <= '0'; qn <= '1';
    elsif s = '1' and r = '1' then
     q <= '1'; qn <= '1';
    end if;
  end if;
 end process;
end bhv;
```

✓ PROCESOS SÍNCRONOS

- **Flip Flop D**

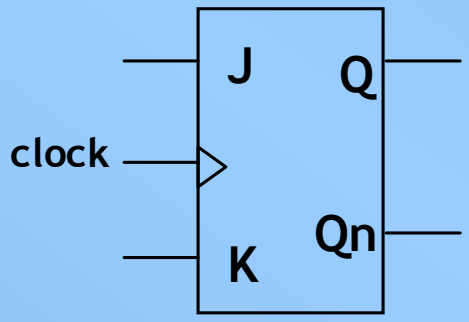



```
library ieee;
use ieee.std_logic_1164.all;

entity ff_d is
 port (d, clock: in std_logic;
       q, qn: buffer std_logic);
end ff_d;

architecture bhv of ff_d is
begin
 process (d,clock)
 begin
  if (clock'event and clock = '1') then
    q <= d; qn <= not(d);
  end if;
 end process;
end bhv;
```



✓ PROCESOS SÍNCRONOS

- **Flip Flop JK**

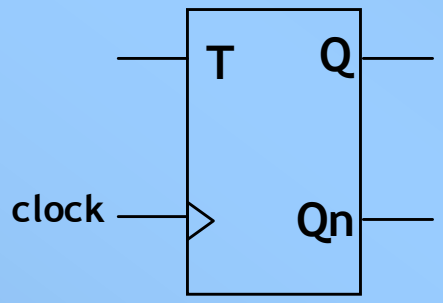


clock	J	K	Q(t+1)
↑	0	0	Q(t)
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q(t)}$

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

entity ff_jk is
 port (j,k, clock: in std_logic;
       q, qn: buffer std_logic);
end ff_jk;

architecture bhv of ff_jk is
begin
 process (j,k,clock)
 begin
  if (clock'event and clock = '1') then
   if j = '1' and k = '1' then
    q <= not (q);
   elsif j = '1' and k = '0' then
    q <= '0';
   elsif j = '0' and k = '1' then
    q <= '1'
   end if;
  end if;
 end process;
 q <= not(q);
end bhv;
```

✓ PROCESOS SÍNCRONOS

- **Flip Flop T**

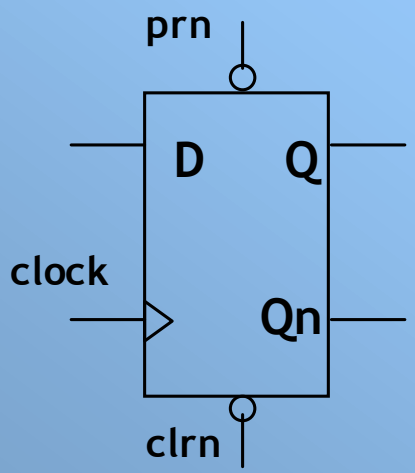


clock	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

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

entity ff_t is
 port (t, clock: in std_logic;
       q, qn: buffer std_logic);
end ff_t;

architecture bhv of ff_t is
begin
 process (t,clock)
 begin
  if (clock'event and clock = '1') then
   if t = '1' then
    q <= not (q);
   end if;
  end if;
 end process;
 qn <= not(q);
end bhv;
```

✓ PROCESOS SÍNCRONOS

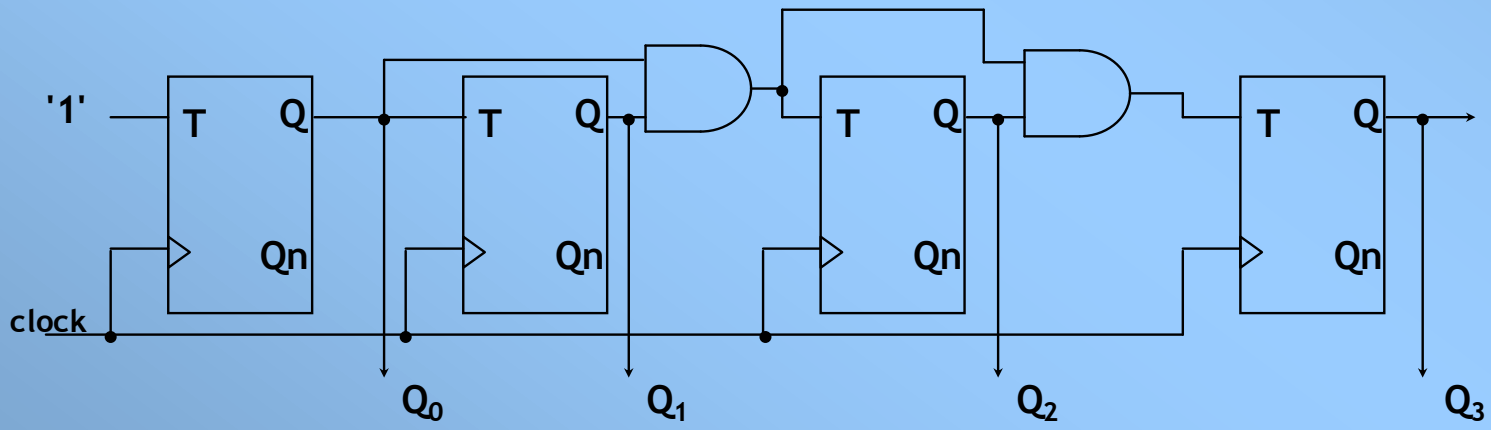
- **Flip Flop D con ‘clrn’ y ‘prn’ asíncronas**
- clrn = ‘0’ ➔ q = ‘0’
  prn = ‘0’ ➔ q = ‘1’
- Estas entradas cambian las salidas en forma inmediata
- Esta característica es útil si se desea inicializar el circuito sin esperar al flanco de reloj.




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

entity ff_d_asin is
 port (t,clrn,prn,clock: in std_logic;
       q, qn: buffer std_logic);
end ff_d_asin;

architecture bhv of ff_d_asin is
begin
 process (d,clrn,prn,clock)
 begin
  if clrn = '0' then
   q <= '0';
  elsif prn = '0' then
   q <= '1';
  elsif (clock'event and clock = '1') then
    q <= d;
  end if;
 end process;
 qn <= not(q);
end bhv;
```

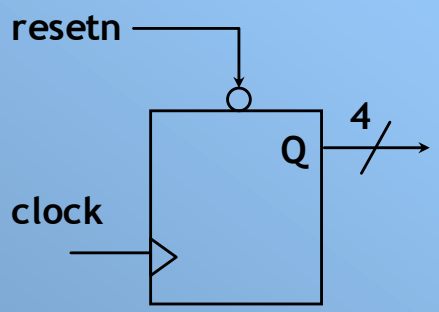
✓ PROCESOS SÍNCRONOS

- **Contadores síncronos**
- Los contadores son usados en los sistemas digitales para múltiples própósitos. Pueden contar el número de ocurrencias de un evento, generar intervalos de tiempo para el control de varias tareas, rastrear el tiempo transcurrido entre eventos específicos, etc.
- Los contadores síncronos cambian su salida con cada flanco de reloj. Los contadores se componen de flip flops y lógica externa, en los contadores síncronos **todos los flip flops comparten la misma señal de reloj,** como en el siguiente contador síncrono de 4 bits (cuenta de 0000 hasta 1111).



✓ PROCESOS SÍNCRONOS

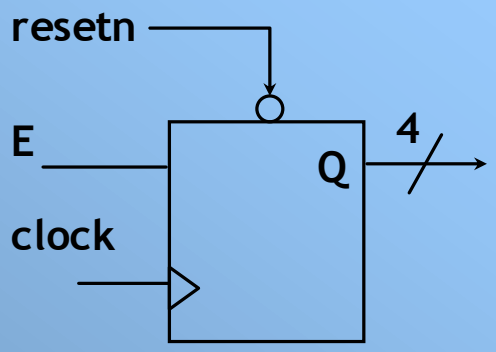
- **Contador síncrono de 4 bits con señal de resetn asíncrona**
- La señal ‘resetn’ hace que la salida sea “0000” sin esperar al flanco




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

entity cont4 is
 port (clock, resetn: in std_logic;
       Q: buffer integer range 0 to 15);
end cont4;

architecture bhv of cont4 is
begin
 process (resetn,clock)
 begin
  if resetn = '0' then
   Q <= 0;
  elsif (clock'event and clock = '1') then
   Q <= Q + 1;
  end if;
 end process;
end bhv;
```

✓ PROCESOS SÍNCRONOS

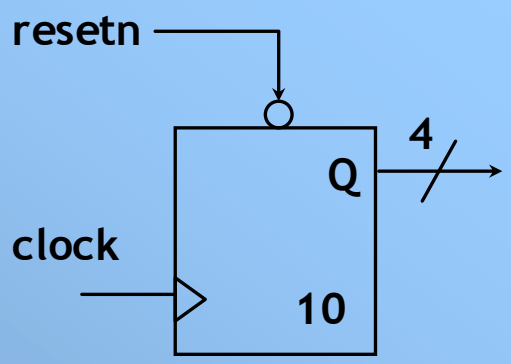
- **Contador síncrono de 4 bits con señal de resetn y habilitador**
- Ojo que la señal 'E' es síncrona, es decir el circuito sólo le hace caso en el flanco




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

entity cont4_e is
 port (clock, resetn, E: in std_logic;
       Q: buffer integer range 0 to 15);
end cont4_e;

architecture bhv of cont4_e is
begin
 process (resetn,E,clock)
 begin
  if resetn = '0' then
   Q <= 0;
  elsif (clock'event and clock = '1') then
   if E = '1' then
     Q <= Q + 1;
   end if;
  end if;
 end process;
end bhv;
```

✓ PROCESOS SÍNCRONOS

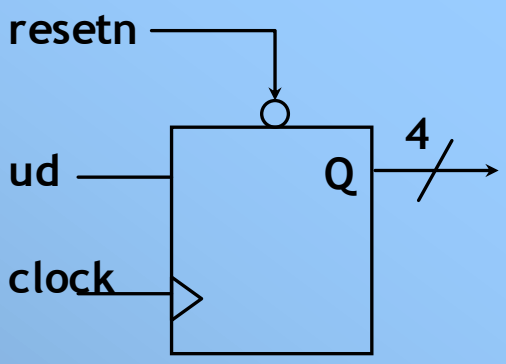
- **Contador síncrono de décadas con señal de resetn asíncrona**




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

entity cont_dec is
 port (clock, resetn: in std_logic;
       Q: buffer integer range 0 to 15);
end cont_dec;

architecture bhv of cont_dec is
begin
 process (resetn,clock)
 begin
  if resetn = '0' then
   Q <= 0;
  elsif (clock'event and clock = '1') then
   if Q = 9 then
     Q <= 0;
   else
     Q <= Q + 1:
   end if;
  end if;
 end process;
end bhv;
```

✓ PROCESOS SÍNCRONOS

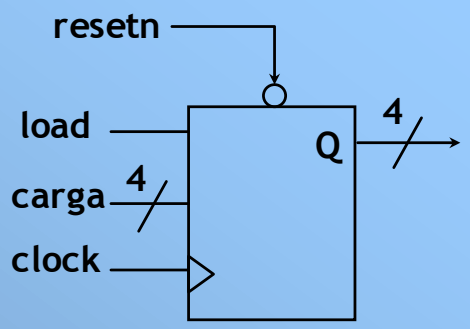
- **Contador síncrono ascendente/descendente de 4 bits**




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

entity cont_ud is
 port (clock, resetn, ud: in std_logic;
       Q: buffer integer range 0 to 15);
end cont_ud;

architecture bhv of cont_ud is
begin
 process (resetn,clock)
 begin
  if resetn = '0' then
   Q <= 0;
  elsif (clock'event and clock = '1') then
   if ud = '0' then
     Q <= Q - 1;
   else
     Q <= Q + 1:
   end if;
  end if;
 end process;
end bhv;
```

✓ PROCESOS SÍNCRONOS

- **Contador con carga paralela de 4 bits y señal de 'resetn' asíncrona**




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

entity cont_cp is
 port (clock,resetn,load: in std_logic;
  carga: in std_logic_vector (3 downto 0);
  Q: buffer std_logic_vector (3 downto 0));
end cont_cp;

architecture bhv of cont_cp is
begin
 process (resetn,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if load = '0' then
     Q <= carga;
   else
     Q <= Q + "0001";
   end if;
  end if;
 end process;
end bhv;
```

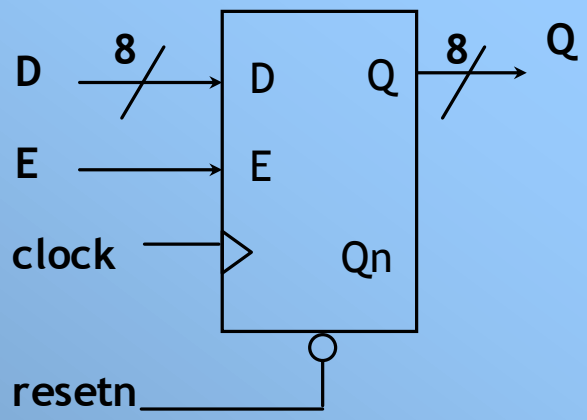
✓ PROCESOS SÍNCRONOS

- **Registros**
- Son circuitos secuenciales que almacenan los valores de las señales. Existen múltiples tipos de registros para varias aplicaciones. Por ejemplo: registros para manejar interrupciones en la PC, registros del microprocesador, etc.
- Se usarán flip flops para implementar los registros, debido a que los flip flops son la base de los sistemas digitales síncronos.
- Tipos de Registros:
  - Registro de carga paralela y salida paralela
  - Registro de carga paralela y salida serial
  - Registros de Desplazamiento:

    Registro de carga serial y salida paralela
    Registro de carga serial y salida serial

✓ PROCESOS SÍNCRONOS

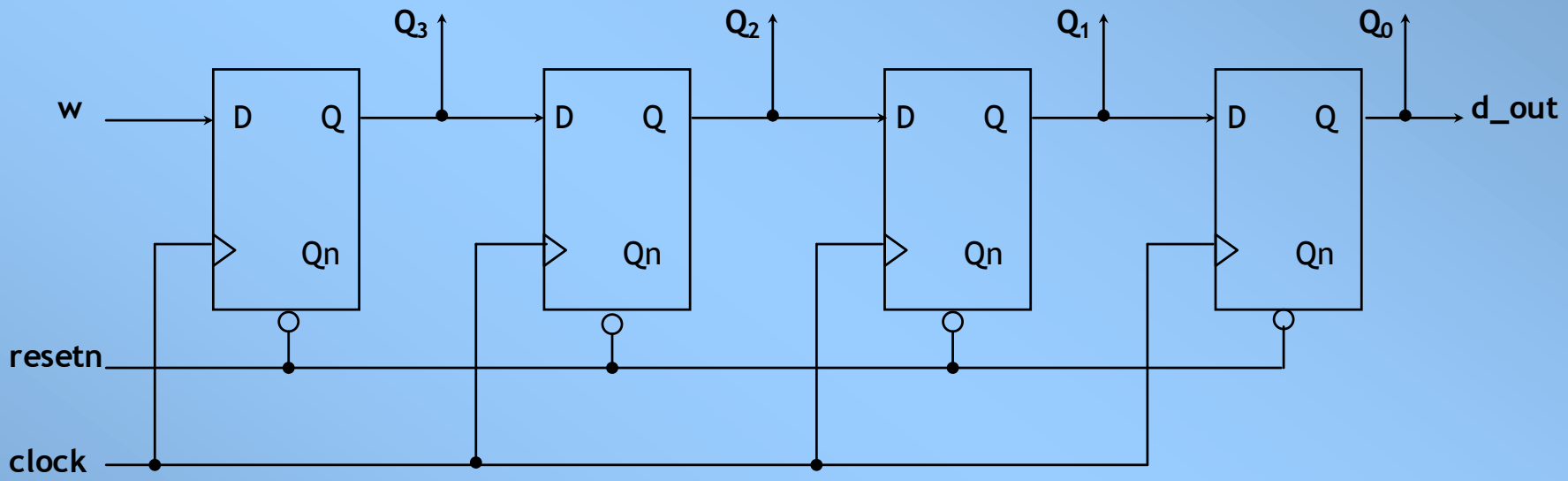
- **Registro de 8 bits: carga paralela, salida paralela, con habilitador y resetn asíncrono.**




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

entity reg8 is
 port (clock, resetn, E: in std_logic;
  D: in std_logic_vector (7 downto 0);
  Q: out std_logic_vector (7 downto 0));
end reg8;

architecture bhv of reg8 is
begin
 process (resetn,E,clock)
 begin
  if resetn = '0' then
   Q <= (others => '0');
  elsif (clock'event and clock = '1') then
   if E = '1' then
     Q <= D;
   end if;
  end if;
 end process;
end bhv;
```

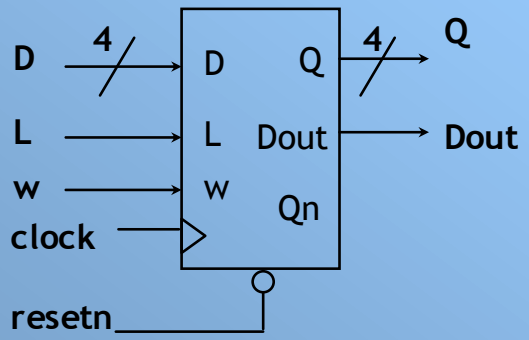
✓ PROCESOS SÍNCRONOS

- **Estructura de un registro de 4 bits, con carga serial, salida paralela/serial. Desplazamiento a la derecha**



✓ PROCESOS SÍNCRONOS

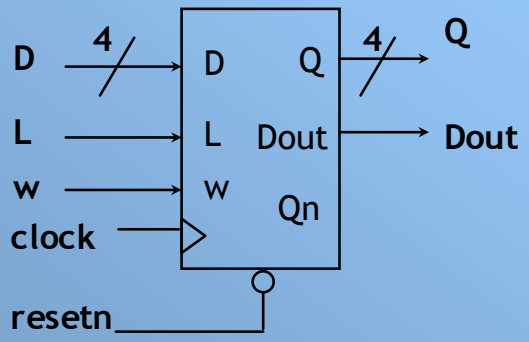
- **Registro de 4 bits: Carga paralela/ serial, salida paralela/ serial, Desplazamiento a la derecha**
- La señal ‘L’ decide si la carga es paralela o serial
- ‘Dout’ es la salida serial
- ‘Q’ es la salida paralela




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

entity reg_t is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_t;

architecture bhv of reg_t is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     Q(0) <= Q(1); Q(1) <= Q(2);
     Q(2) <= Q(3); Q(3) <= w;
   end if;
  end if;
 end process;
Dout <= Q(0);
end bhv;
```

✓ PROCESOS SÍNCRONOS

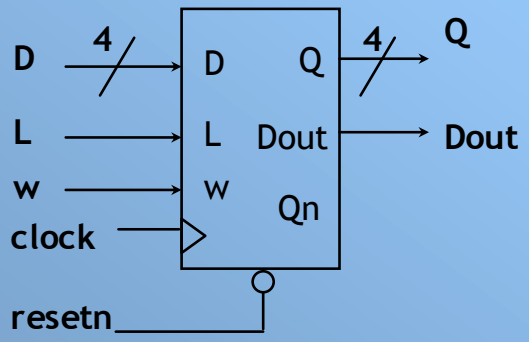
- **Registro de 4 bits: Carga paralela/ serial, salida paralela/ serial, Desplazamiento a la derecha**
- La señal ‘L’ decide si la carga es paralela o serial
- ‘Dout’ es la salida serial
- ‘Q’ es la salida paralela




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

entity reg_t is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_t;

architecture bhv of reg_t is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     gg: for i in 0 to 2 loop
           Q(i) <= Q(i+1);
         end loop;
         Q(3) <= w;
   end if;
  end if;
 end process;
Dout <= Q(0);
end bhv;
```

✓ PROCESOS SÍNCRONOS

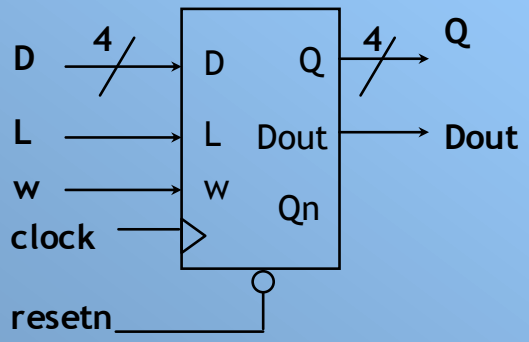
- **Registro de 4 bits: Carga paralela/ serial, salida paralela/ serial, Desplazamiento a la izquierda**
- La señal ‘L’ decide si la carga es paralela o serial
- ‘Dout’ es la salida serial
- ‘Q’ es la salida paralela




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

entity reg_i is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_i;

architecture bhv of reg_i is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     Q(3) <= Q(2); Q(2) <= Q(1);
     Q(1) <= Q(0); Q(0) <= w;
   end if;
  end if;
 end process;
Dout <= Q(3);
end bhv;
```

✓ PROCESOS SÍNCRONOS

- **Registro de 4 bits: Carga paralela/ serial, salida paralela/ serial, Desplazamiento a la izquierda**
- La señal ‘L’ decide si la carga es paralela o serial
- ‘Dout’ es la salida serial
- ‘Q’ es la salida paralela




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

entity reg_i is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_i;

architecture bhv of reg_i is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
      gg: for i in 1 to 3 loop
           Q(i) <= Q(i-1);
        end loop;
          Q(0) <= w;
   end if;
  end if;
 end process;
Dout <= Q(3);
end bhv;
```