Lógica Secuencial en VHDL (I)

UNIVERSIDAD NEBRIJA

Introducción

- Las salidas de la lógica secuencial depende de la entrada actual y las anteriores. Tiene ***memoria***.
- Algunas definiciones:
  - **Estado:** toda la información del circuito necesaria para explicar su comportamiento futuro
  - **Latches y flip-flops:** elementos que almacenan un bit de estado
  - **Circuitos secuenciales síncronos:** lógica combinacional seguida por un banco de flip-flops

Circuitos secuenciales

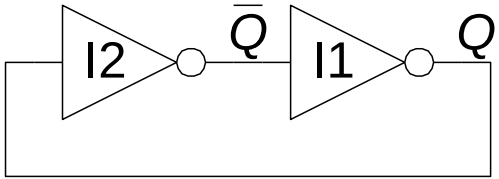
- Dan secuencia a los eventos
- Tienen memoria (a corto plazo)
- Usan retroalimentación desde la salida a las entradas para almacenar la información

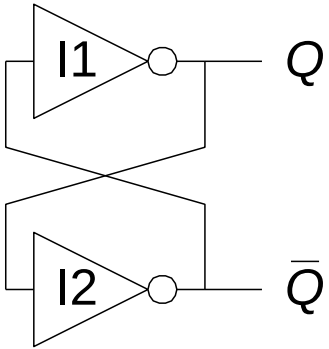
Elementos de Estado

- El estado de un circuito influye en su comportamiento futuro
- Entre los elementos de estado que almacenan información sobre el mismo están:
  - Circuitos biestables
  - Latch SR
  - Latch D
  - Flip-flop D

Circuito biestable

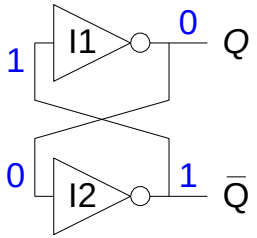
- Bloque fundamental base de los otros elementos de estado
- Dos salidas: Q, Q_{bar}
- Sin entradas



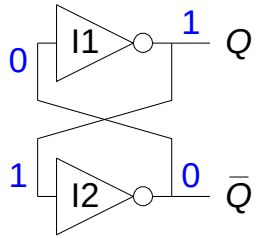


Análisis del circuito biestable

- Dos casos posibles:
  - **Q = 0:**
  - entonces Q = 1, Q_{bar} = 0 (coherente)



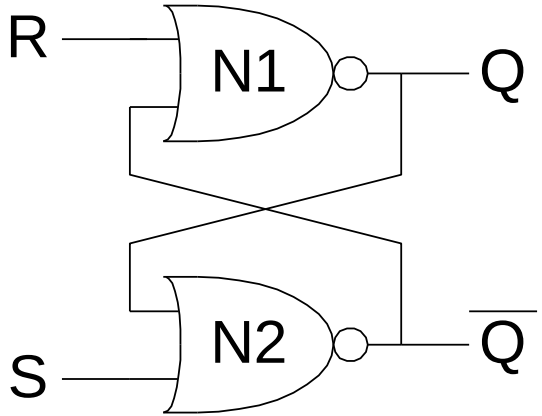
  - **Q = 1:**
  - entonces Q = 0, Q_{bar} = 1 (coherente)



- Almacena un bit de estado en la variable de estado, Q (o Q_{bar})
- Pero **no hay entradas para controlar el estado.**

Latch SR (Set/Reset)

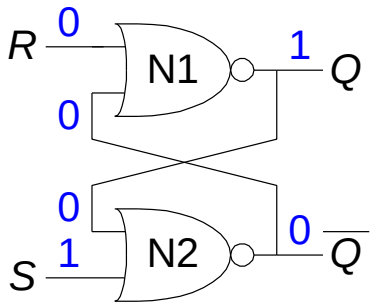
- Latch SR



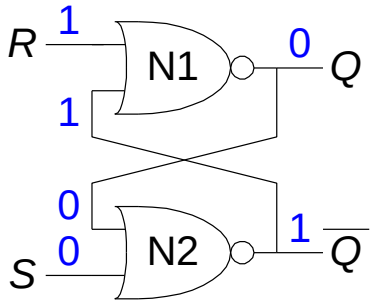
- Considera los cuatro casos posibles:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

Análisis del Latch SR

- **$S = 1, R = 0$:**
- entonces $Q = 1$ y $Q_{bar} = 0$



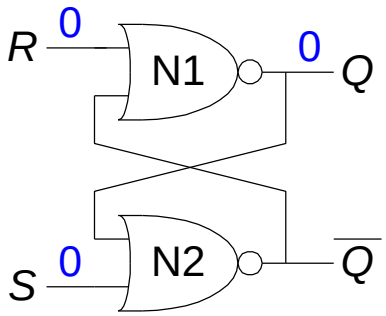
- **$S = 0, R = 1$:**
- entonces $Q = 1$ y $Q_{bar} = 0$



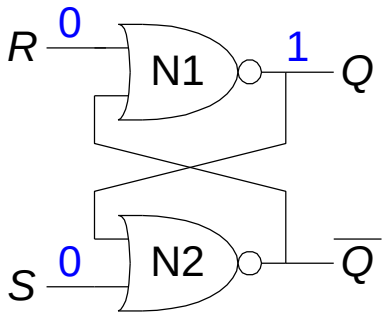
Análisis del Latch SR

- $S = 0, R = 0$:
- entonces $Q = Q_{prev}$
- **¡Memoria!**

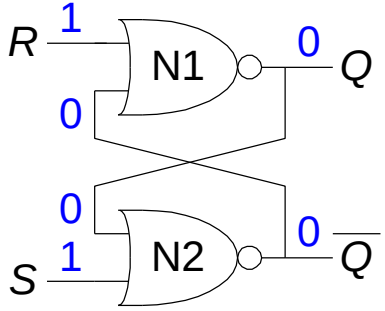
$Q_{prev} = 0$



$Q_{prev} = 1$



- $S = 1, R = 1$:
- entonces $Q = 0, Q_{bar} = 0$
- **Estado inválido**
- $Q \neq \text{NOT } Q_{bar}$

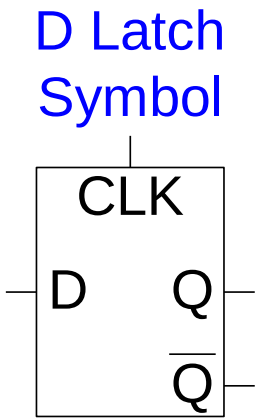


Implementación VHDL

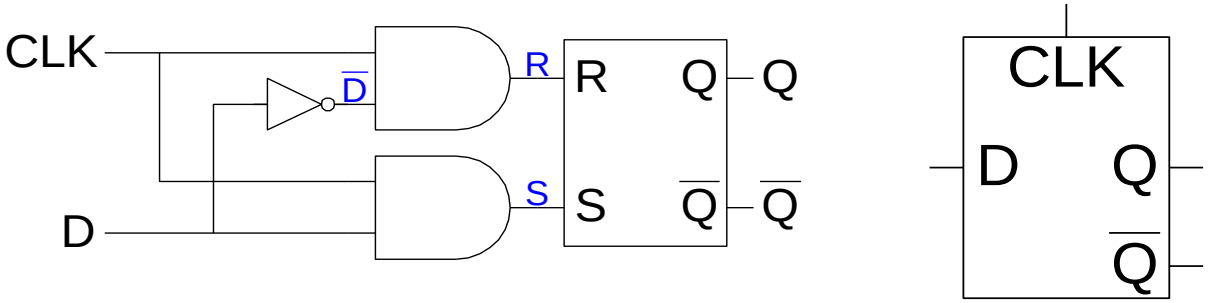
```
entity sr_latch is
        port ( s, r: in std_logic;
                  q, qbar: out std_logic);
end sr_latch;

architecture sr_latch_arq of sr_latch is
    signal q_int, qbar_int : std_logic;
begin
    q_int <= s nand qbar_int;
    qbar_int <= r nand q_int;

    q <= q_int;
    qbar <= qbar_int;
end sr_latch_arq;
```

Latch D

- Dos entradas: *CLK*, *D*
  - ***CLK*:** controla *cuándo* cambia la salida
  - ***D*** (entrada de datos): controla *cuáles* son los cambios de la salida
- Función
  - Cuando ***CLK* = 1**,
  - *D* pasa a *Q* (*transparente*)
  - Cuando ***CLK* = 0**,
  - *Q* mantiene su valor anterior (*opaco*)
- Evita el caso inválido en el que $Q \neq \text{NOT } Q$



Circuito interno del Latch D



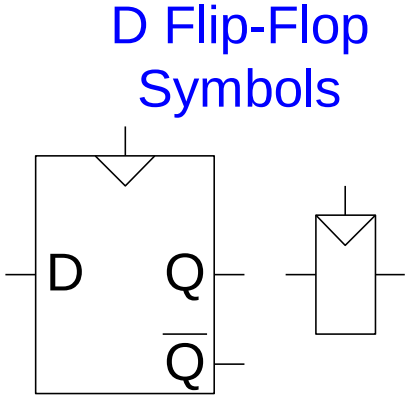
CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Implementación VHDL

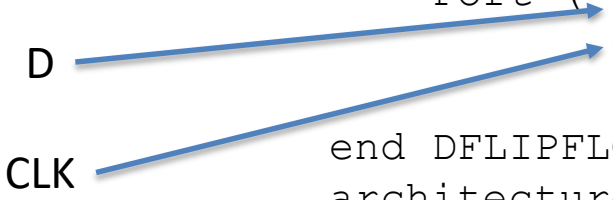
```
entity dlatch is
    port (
            signal d, clk: in std_logic;
            signal q,qn: out std_logic);
end dlatch;

architecture behavior of dlatch is
begin
    process (clk, d)
    begin
        if (clk = '1') then
            q < = d;
            qn <= not d;
        end if;
    end process;
end behavior;
```

Flip-Flop D

- **Entradas:** *CLK*, *D*
- **Función**
  - Muestrea *D* en el flanco de subida del reloj *CLK*
    - Cuando *CLK* sube de 0 a 1, *D* pasa a *Q*
    - En cualquier otro caso, *Q* mantiene su valor anterior
  - *Q* cambia solo en el flanco de subida del reloj *CLK*
- Se le llama *disparado por flanco*
- Activado en el flanco del reloj



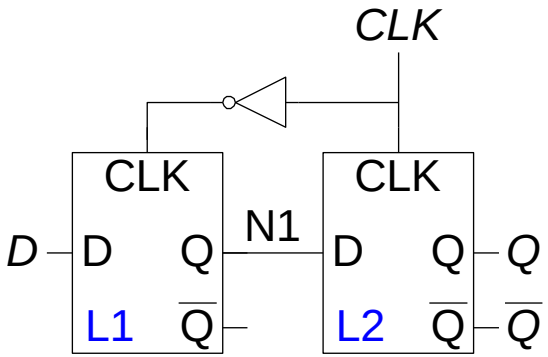
Implementación VHDL



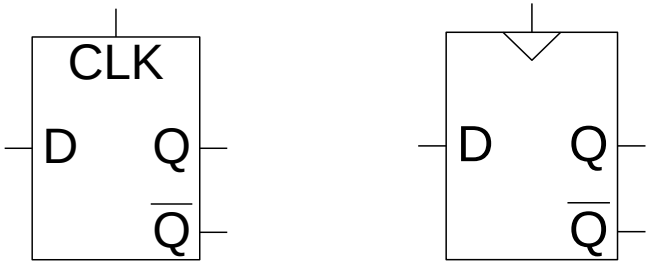

```
entity DFLIPFLOP is
    Port ( DATAIN : in std_logic;
           CLOCK : in std_logic;
           DATAOUT : out std_logic);
end DFLIPFLOP;
architecture Behavioral of DFLIPFLOP is
   begin
            process (DATAIN, CLOCK)
            begin
                if rising_edge(CLOCK) then
                   DATAOUT <= DATAIN;
                end if;
            end process;
end Behavioral;
```

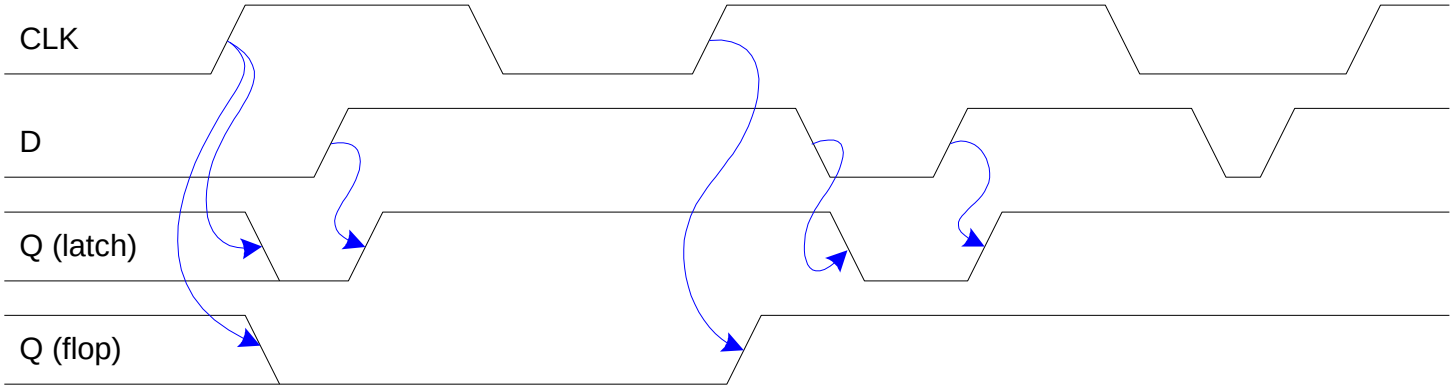
Circuito interno del Flip-Flop D

- Dos latches (L1 y L2) seguidos controlados por relojes de valores opuestos
- Cuando ***CLK* = 0**
  - L1 es transparente
  - L2 es opaco
  - *D* pasa a N1
- Cuando ***CLK* = 1**
  - L2 es transparente
  - L1 es opaco
  - N1 pasa a *Q*
- Así, en el flanco de reloj (cuando ***CLK* sube de 0 a 1**)
  - *D* pasa a *Q*

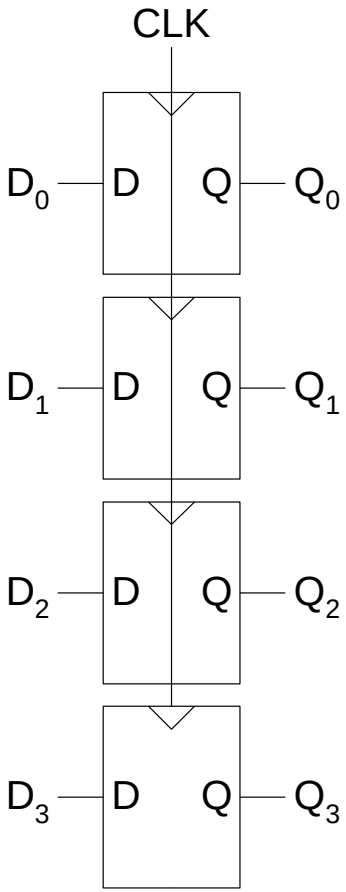


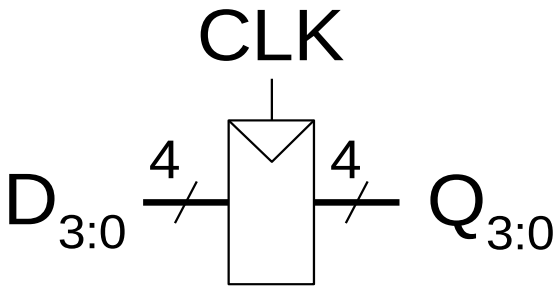
Latch D vs. Flip-Flop D





Registros





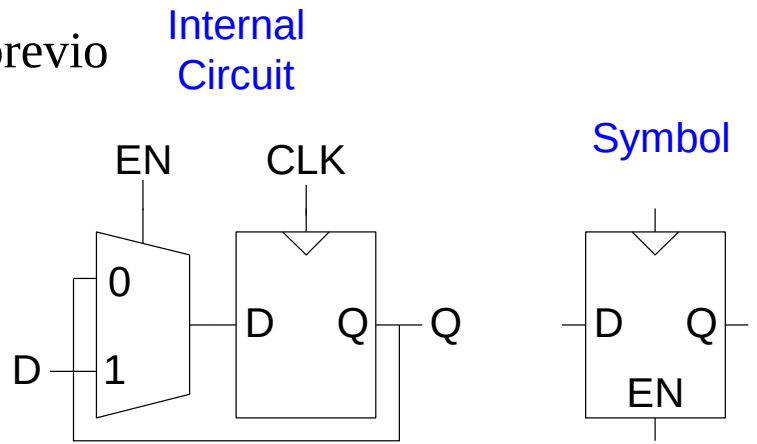
Implementación VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flop is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC_VECTOR(3 downto 0) ;
       q:   out STD_LOGIC_VECTOR(3 downto 0)) ;
end flop;

architecture synth of flop is
begin
  process(clk) begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
end synth;
```

Flip-Flops con Enable

- **Entradas:** *CLK*, *D*, *EN*
  - La entrada enable (*EN*) controla cuándo se almacena el nuevo dato (*D*)

  Función
  - ***EN* = 1:** *D* pasa a *Q* en el flanco de reloj
  - ***EN* = 0:** el flip-flop mantiene su estado previo



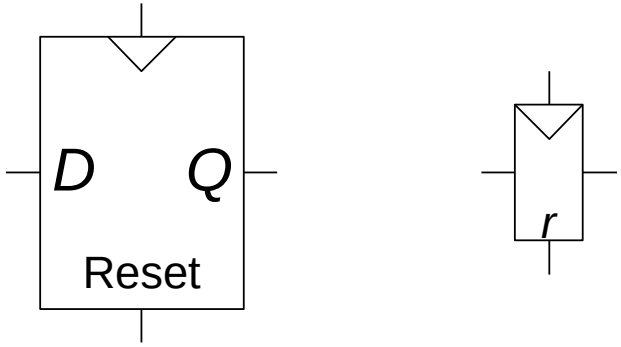
Implementación VHDL

```
entity DFLIPFLOP3 is
    Port ( DATAIN : in std_logic;
           CLOCK : in std_logic;
           ENABLE : in std_logic;
           DATAOUT : out std_logic);
end DFLIPFLOP3;
architecture Behavioral of DFLIPFLOP3 is
begin
            process (DATAIN, CLOCK)
            begin
               if ENABLE='1' then
                          if rising_edge(CLOCK) then
                               DATAOUT <= DATAIN;
                          end if;
               end if;
            end process;
end Behavioral;
```

Flip-Flops con Reset

- **Entradas:** *CLK*, *D*, *Reset*
- **Función:**
  - ***Reset* = 1:** *Q* se establece a 0
  - ***Reset* = 0:** el flip-flop se comporta como un flip-flop D normal

Symbols



Flip-Flops con Reset

- Dos tipos:
  - **Síncronos:** reset solo en el flanco de reloj
  - **Asíncronos:** reset inmediato cuando *Reset* = 1

Implementación VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopenr is
  port(clk,reset, en: in  STD_LOGIC;
       d:  in  STD_LOGIC_VECTOR(3 downto 0);
       q:  out STD_LOGIC_VECTOR(3 downto 0));
end flopenr;

architecture asynchronous of flopenr is
begin
  process(clk, reset) begin
    if reset='1' then
      q <= "0000";
    elsif rising_edge(clk) then
      if en then
        q <= d;
      end if;
    end if;
  end process;
end asynchronous;
```

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopr is
  port(clk   : in  STD_LOGIC;
       reset: in  STD_LOGIC;
       d:     in  STD_LOGIC_VECTOR(3 downto 0);
       q:     out STD_LOGIC_VECTOR(3 downto 0));
end flopr;

architecture synchronous of flopr is
begin
  process(clk) begin
    if rising_edge(clk) then
       if reset='1' then
          q <= "0000";
       else
          q <= d;
       end if;
    end if;
  end process;
end synchronous;
```

Flip-Flops con Set

- **Entradas:** *CLK*, *D*, *Set*
- **Función:**
  - ***Set* = 1:** *Q* se pone a1
  - ***Set* = 0:** el flip-flop se comporta como un flip-flop D normal

Symbols