DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sistemi Digitali

Circuiti Sequenziali in VHDL: Latch e flip-flop

Marta Bagatin, marta.bagatin@unipd.it

Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

Scopo della lezione

- Descrivere e simulare in VHDL gli **elementi sequenziali di base**
  - Latch SR con ingresso di controllo
  - Latch D
  - Flip-flip di tipo D con reset asincrono
  - Flip-flip di tipo D con reset sincrono

«process»: richiami

- All'interno di un processo, gli statement **vengono eseguiti in modo sequenziale**, cioè uno dopo l'altro, nell'ordine in cui sono scritti e in un tempo infinitesimo (in assenza di appositi comandi espliciti, e.g. wait)
- **Tutti i processi all'interno di un'architettura vengono eseguiti simultaneamente**
- Il valore dei segnali assegnato all'interno di un processo viene aggiornato soltanto quando il processo viene sospeso (o perché sono stati eseguiti tutti gli statement all'interno del processo o perché c'è un'istruzione wait)
- All'inizio di un processo può essere indicata una **lista di sensibilità**, contenente i segnali a cui il processo è sensibile: il processo viene **attivato** quando c'è un **cambiamento** in almeno uno dei segnali della sensitivity list
  - Per descrivere un circuito combinatorio la lista di sensibilità deve contenere tutti gli ingressi del circuito
  - Nei testbench usiamo tipicamente processi senza lista di sensibilità che vengono attivati immediatamente all'inizio della simulazione

«process»: richiami

Sintassi per il costrutto process:

```
[nomeProcesso:] process [( <lista_di_sensibilità> )]
begin
        <istruzioni_sequenziali>
end process;
```

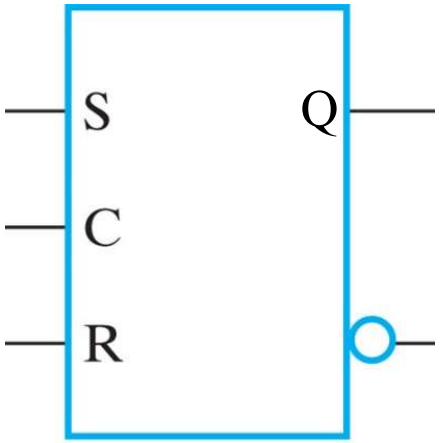
- La **lista di sensibilità** contiene i segnali al cui cambiamento il processo è sensibile. Se è vuota (testbench), il processo viene eseguito di continuo
- Si può dare un **nome** al processo. Va inserito prima della parola chiave `process` e deve essere seguito da «:»

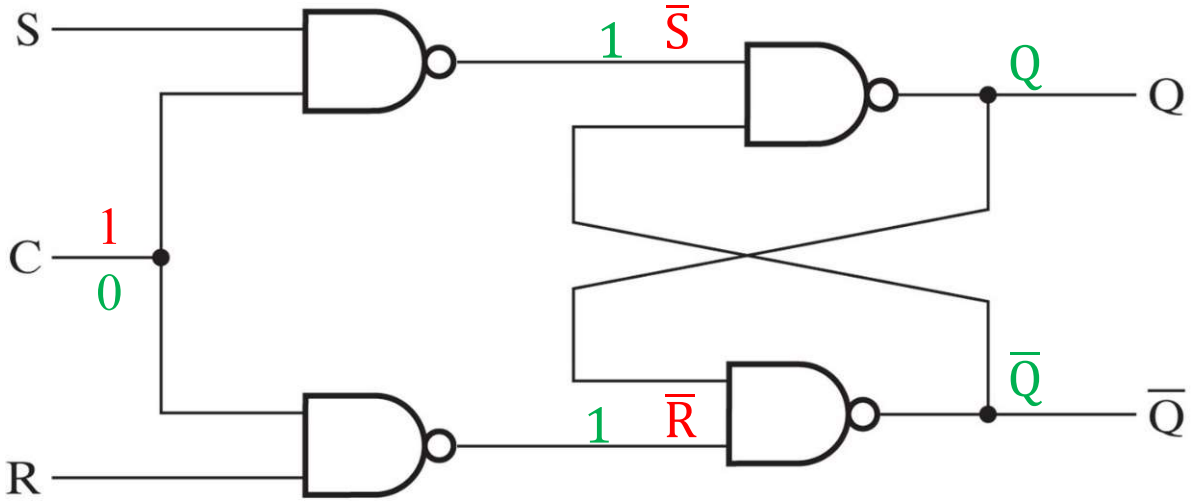
Latch

- Latch SR con ingresso di controllo
- Latch D

Latch SR con ingresso di controllo: richiami

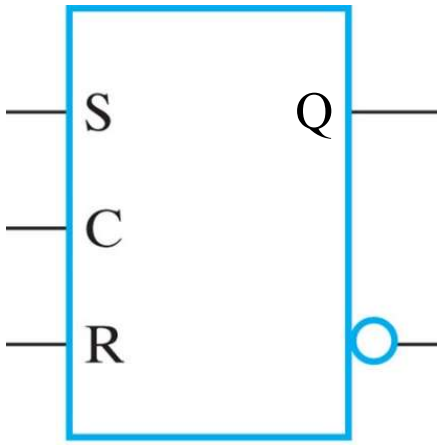
- L'**ingresso di controllo C** funge da segnale di **enable**
  - C = 0: latch non abilitato, mantiene lo stato precedente
  - C = 1: latch abilitato, funziona come un latch di tipo SR
    - S = 1, R = 0: stato di set: Q = 1, $\overline{Q}$ = 0
    - S = 0, R = 1: stato di reset: Q = 0, $\overline{Q}$ = 1
    - S = 0, R = 0: stato di memoria
    - S = 1, R = 1: stato proibito: Q = 1, $\overline{Q}$ = 1

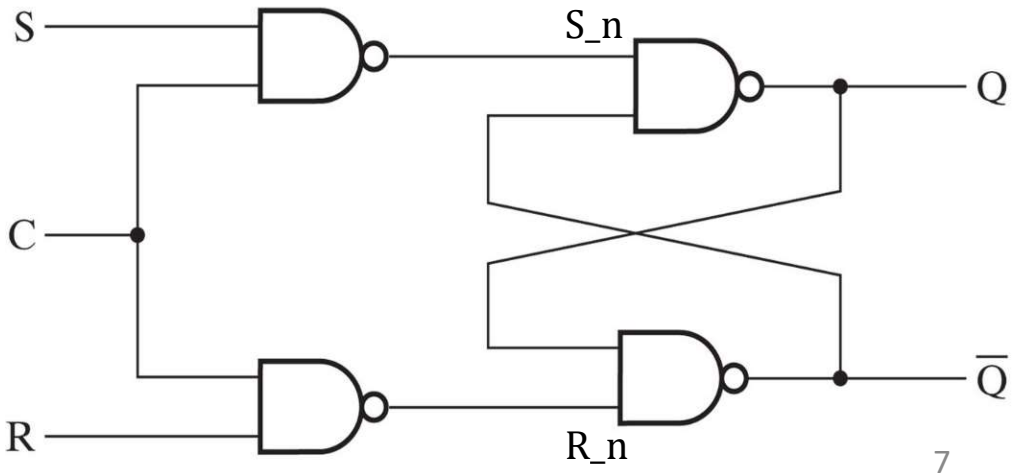




Latch SR con enable: VHDL (dataflow)

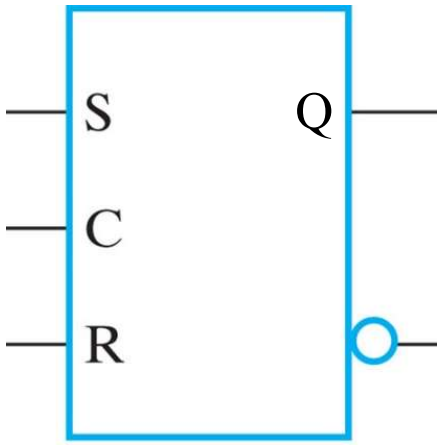
```
library IEEE;
use IEEE.std_logic_1164.all;

entity  SRlatch is
    port( S, R, C : in std_logic;
          Q, Q_n : out std_logic);
end SRlatch;

-- architettura dataflow
architecture dataflow of SRlatch is
    signal S_n, R_n : std_logic;
begin
    S_n <= S nand C;
    R_n <= R nand C;
    Q   <= S_n nand Q_n;
    Q_n <= R_n nand Q;
end dataflow;
```





Latch SR con enable: VHDL (behavioral)

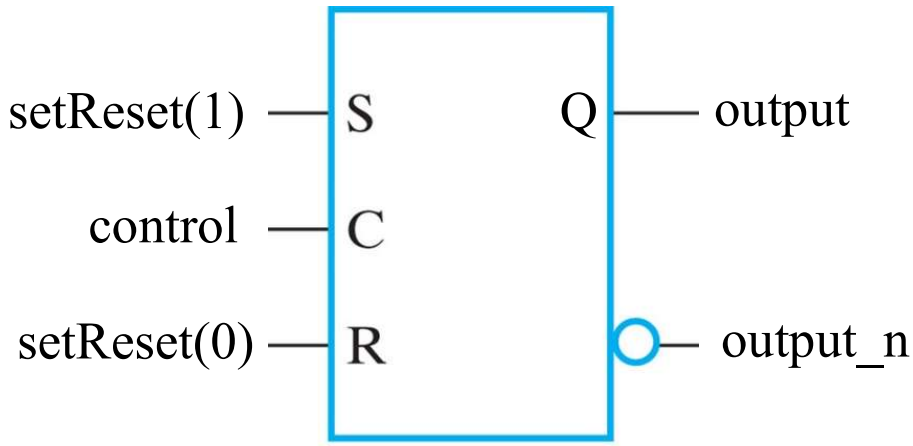
```
architecture behavioral of SRlatch is     -- architettura behavioral
signal SR : std_logic_vector (1 downto 0);
begin
    SR <= S & R;
    process (SR, C) begin
        if ( C = '1') then
            case (SR) is
                when "00" =>              -- S=R=0: MEMORY
                    null;

                when "01" =>              -- S=0, R=1: RESET
                    Q <= '0';
                    Q_n <= '1';

                when "10" =>              -- S=1, R=0: SET
                    Q <= '1';
                    Q_n <= '0';

                when "11" =>              -- S=R=1: FORBIDDEN
                    Q <= '1';
                    Q_n <= '1';

                when others =>
                    Q <= 'X';
                    Q_n <= 'X';
            end case;
        end if;
    end process;
end behavioral;
```



Il costrutto if-else senza tutti i casi specificati (C = '0') genera memoria, quindi logica sequenziale!

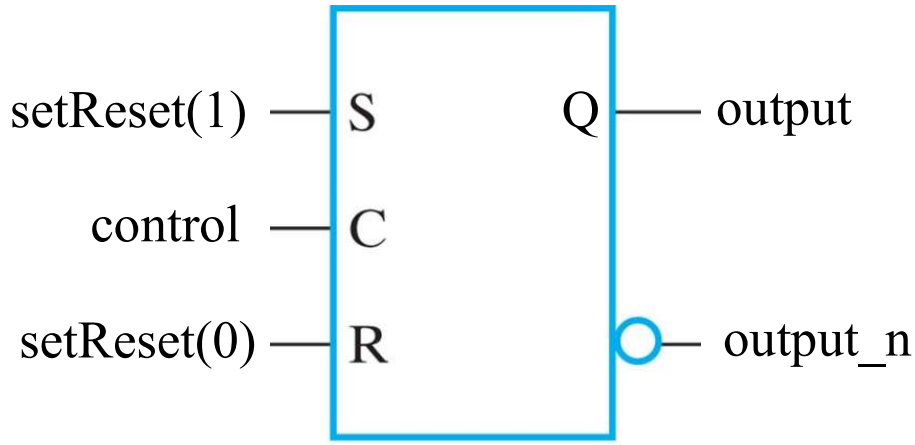
Latch SR con enable: VHDL testbench (1/2)

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

entity testbench is
end testbench;

architecture test of testbench is
    signal setReset: std_logic_vector(1 downto 0);
    signal control, output, output_n: std_logic;
begin
    DUT: entity work.SRlatch(behavioral) port map( setReset(1),
setReset(0), control, output, output_n);
        process begin
            setReset <= "10";
            control <= '1';
            wait for 2 ns;
            setReset <= "00";
            wait for 5 ns;
            setReset <= "01";
            wait for 6 ns;
            setReset <= "00";
            wait for 4 ns;
            setReset <= "10";
            wait for 6 ns;
            setReset <= "01";
            wait for 5 ns;
            setReset <= "00";
```

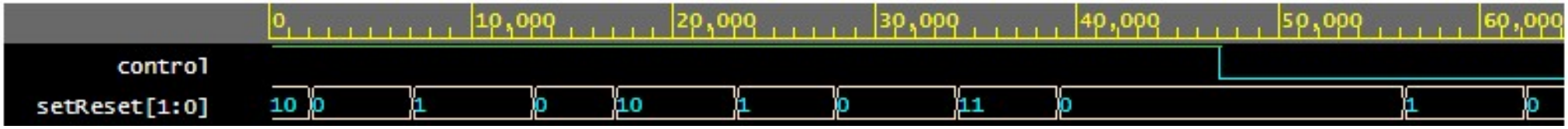


Latch SR con enable: VHDL testbench (2/2)

```
            wait for 6 ns;
            setReset <= "11";
            wait for 5 ns;
            setReset <= "01";
            wait for 0.1 ns;
            setReset <= "00";
            wait for 8 ns;
            control <= '0';
            wait for 4 ns;
            setReset <= "00";
            wait for 5 ns;
            setReset <= "01";
            wait for 6 ns;
            setReset <= "00";
            wait for 2 ns;
            wait;
        end process;
end test;
```



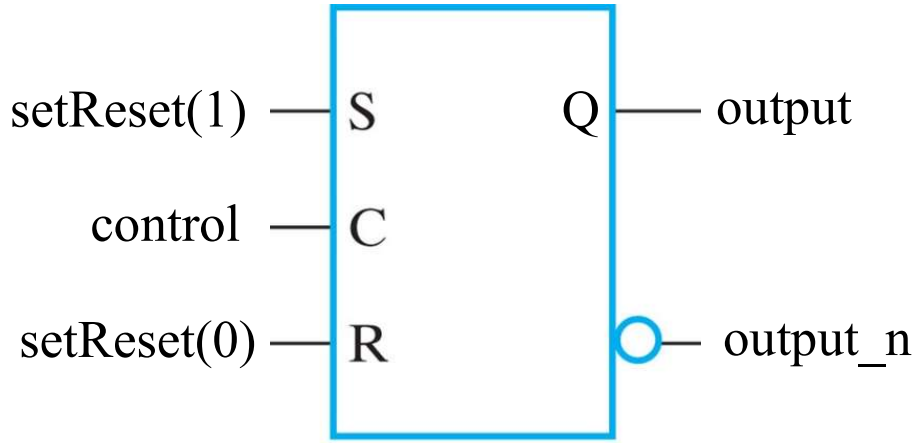
Latch SR con enable: diagramma temporale



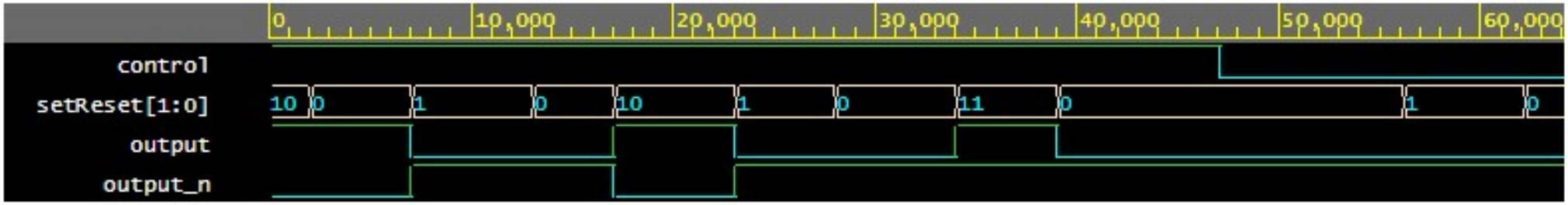
C = 1: latch abilitato

- S = 1, R = 0: set
- S = 0, R = 1: reset
- S = 0, R = 0: memoria
- S = 1, R = 1: proibito

C = 0: latch disabilitato



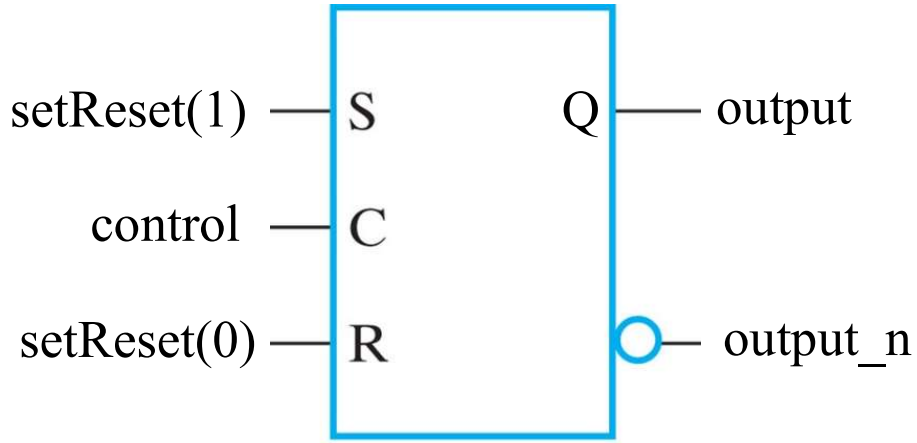
Latch SR con enable: diagramma temporale



C = 1: latch abilitato

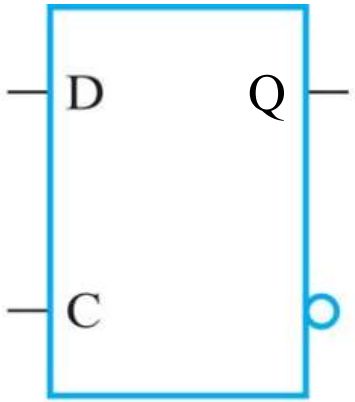
- S = 1, R = 0: set
- S = 0, R = 1: reset
- S = 0, R = 0: memoria
- S = 1, R = 1: proibito

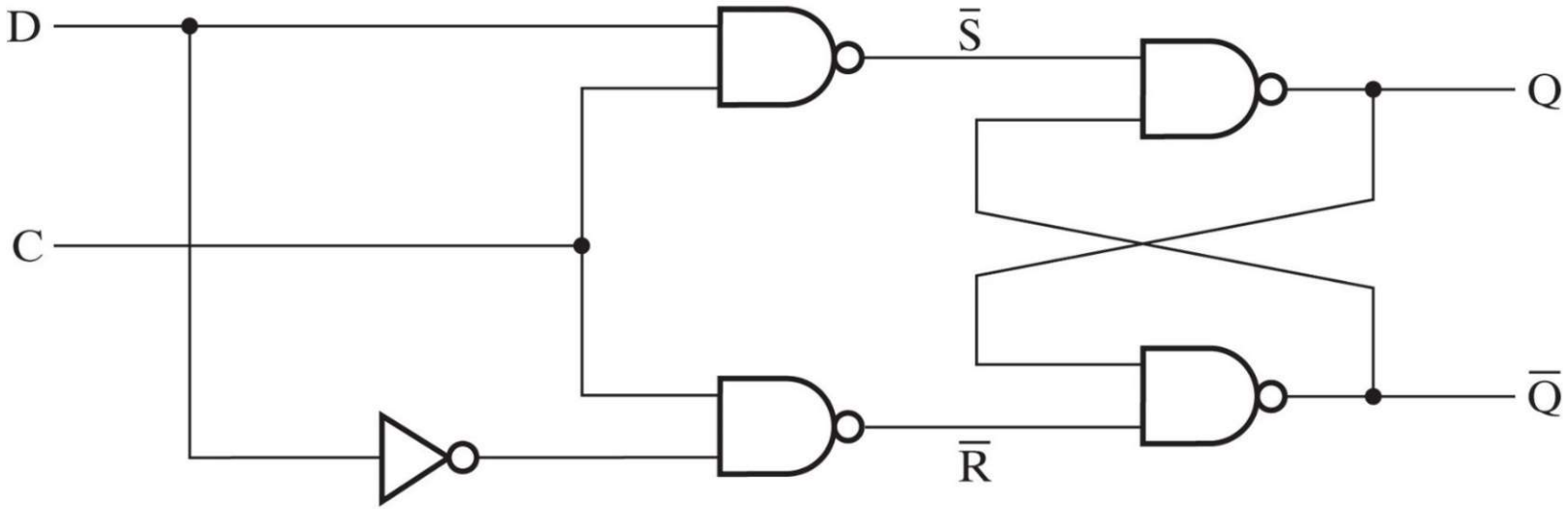
C = 0: latch disabilitato



Latch D: richiami

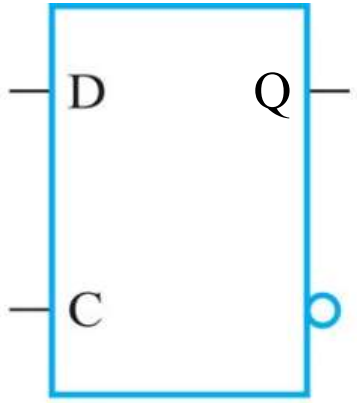
- Il latch D **evita il problema dello stato proibito**, avendo un unico ingresso di dato
  - C = 0: il circuito mantiene lo stato precedente
  - C = 1: D viene trasferito all'uscita Q
    - C = 1, D = 0: stato di reset => Q = 0
    - C = 1, D = 1: stato di set => Q = 1





Latch D: VHDL design

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

entity Dlatch is
       port ( D, C : in std_logic;
              Q, Q_n : out std_logic);
end Dlatch;

-- architettura behavioral
architecture Dlatch_arch of Dlatch is
begin
     process (D, C)
     begin
        if ( C = '1' ) then
           Q  <= D;
           Q_n <= not D;
        end if;
     end process;
end Dlatch_arch;
```



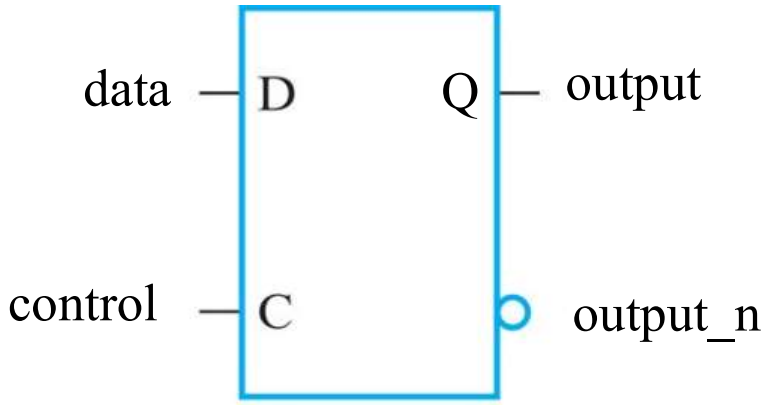
C = 0: mantiene lo stato precedente

C = 1: D viene trasferito all'uscita Q

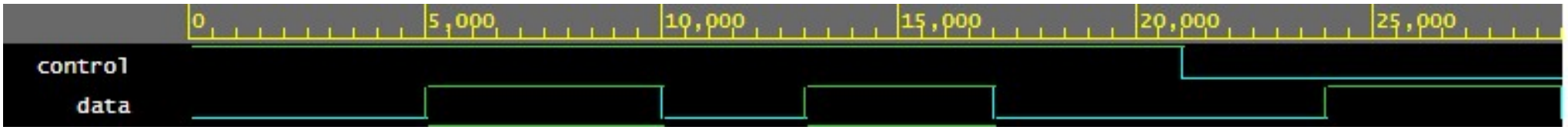
- C = 1, D = 0: stato di reset => Q = 0
- C = 1, D = 1: stato di set => Q = 1

Latch D: VHDL testbench

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

entity testbench is
end testbench;

architecture test of testbench is
   signal data, control, output, output_n : std_logic;
begin
   DUT: entity work.Dlatch port map (data, control, output, output_n);
   process begin
          data <= '0';
          control <= '1';
          wait for 5 ns;
          data <= '1';
          wait for 5 ns;
          data <= '0';
          wait for 3 ns;
          data <= '1';
          wait for 4 ns;
          data <= '0';
          wait for 4 ns;
          control <= '0';
          wait for 3 ns;
          data <= '1';
          wait for 5 ns;
          data <= '0';
          wait;
   end process;
end test;
```



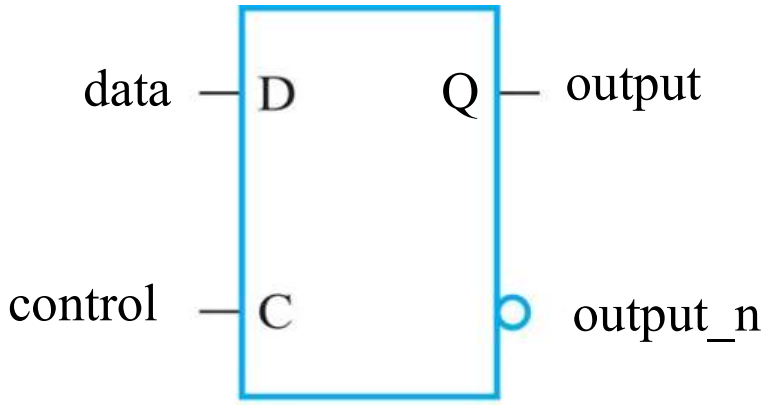
Latch D: simulazione diagramma temporale



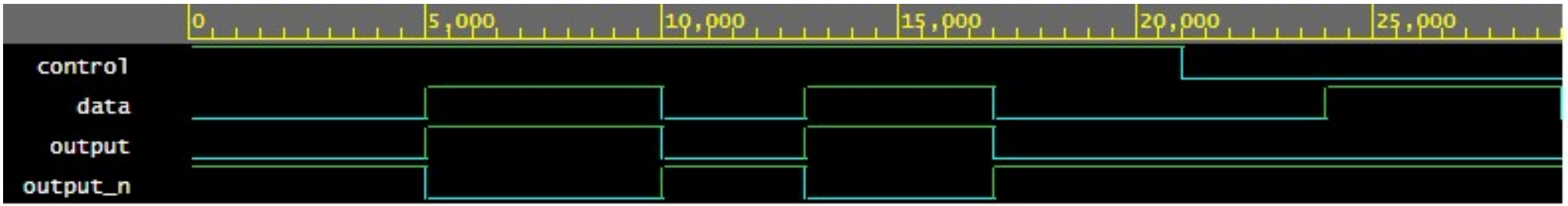
C = 1: D viene trasferito all'uscita Q

- C = 1, D = 0: stato di reset => Q = 0
- C = 1, D = 1: stato di set => Q = 1

C = 0: mantiene lo stato precedente



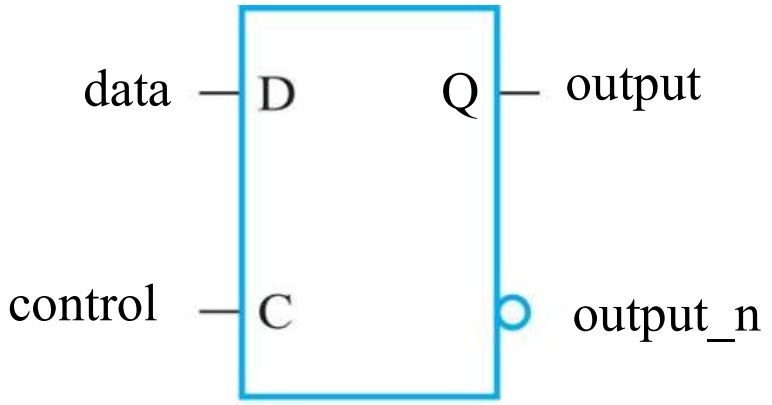
Latch D: simulazione diagramma temporale



C = 1: D viene trasferito all'uscita Q

- C = 1, D = 0: stato di reset => Q = 0
- C = 1, D = 1: stato di set => Q = 1

C = 0: mantiene lo stato precedente

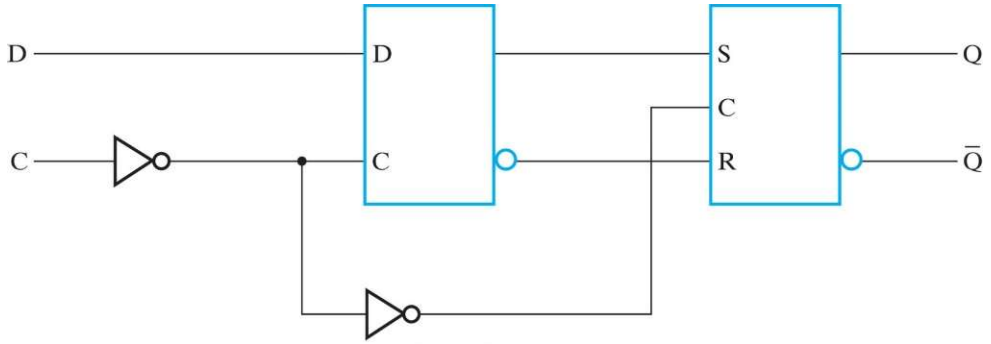


Flip-flop di tipo D

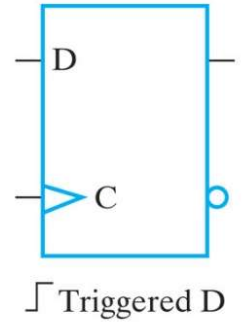
- Con reset asincrono
- Con reset sincrono

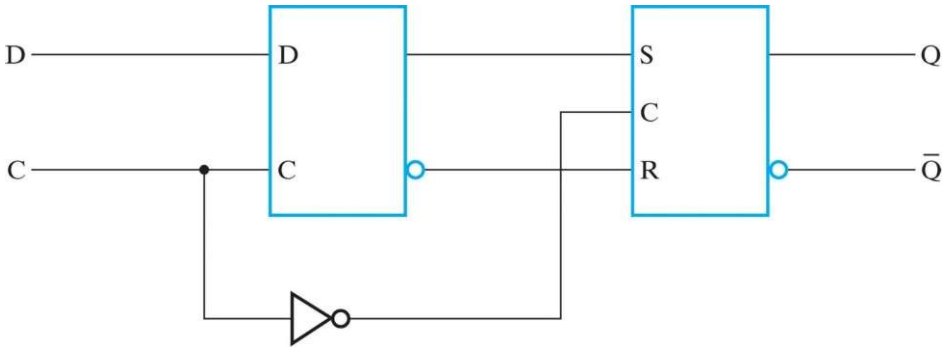
Flip-flop D: richiami

- Un D flip-flop positive-edge-triggered (o negative-edge-triggered) è costituito da due latch in cascata, pilotati da segnali di controllo l'uno complementare dell'altro
  - Ad ogni fronte di salita (o di discesa) del clock, l'ultimo valore di D campionato dal master viene trasferito all'uscita Q del flip-flop
  - In corrispondenza ai fronti di discesa (o di salita) e ai livelli stabili del clock, mantiene lo stato precedente

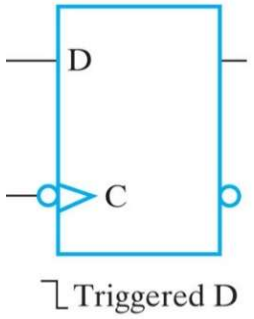


Schema logico
e simbolo
PET-D-Flip-flop



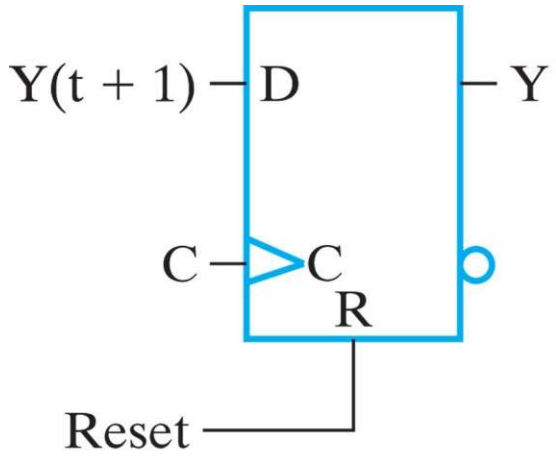


Schema logico
e simbolo
NET-D-Flip-flop

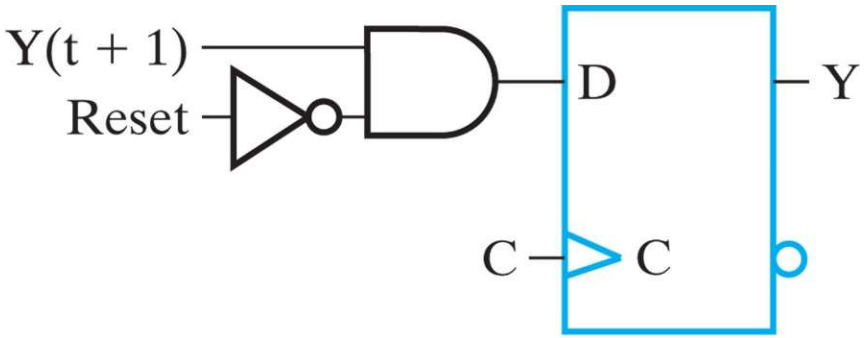


Flip-flop D con segnale di Reset

- **Reset asincrono**: il segnale di reset viene fornito in maniera asincrona rispetto al clock, cioè può arrivare in ogni momento

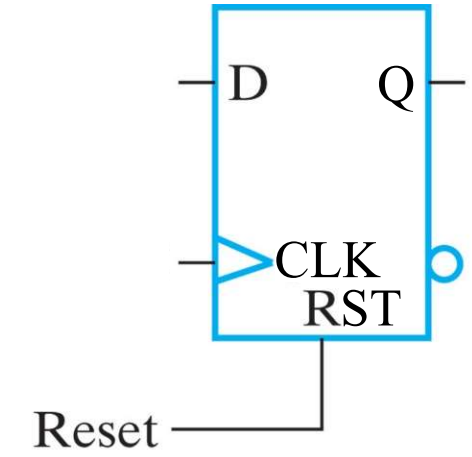


- **Reset sincrono**: il segnale di reset viene fornito in modo sincronizzato con il clock, cioè può arrivare solo ai fronti del clock (positivi o negativi a seconda che si tratti di positive o negative-edge-triggered). Il reset sincrono quindi **richiede un trigger del clock per essere effettivo**



D-flip flop con Reset asincrono: VHDL

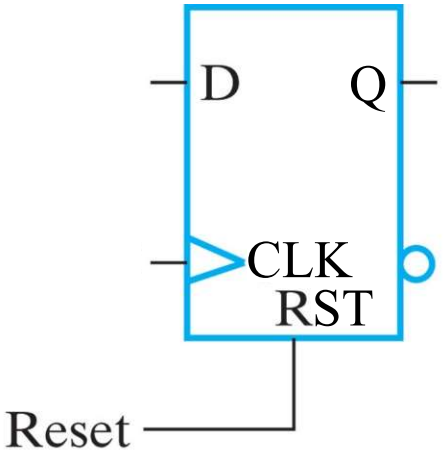
```
library IEEE;
use IEEE.std_logic_1164.all;

entity dff is
    port (CLK, RST, D: in std_logic;
                    Q: out std_logic);
end dff;
-- architettura behavioral
architecture dff_as of dff is
begin
       process (RST, CLK)
    begin
        if (RST = '1') then
              Q <= '0';
        elsif (CLK'event and CLK = '1') then
              Q <= D;
        end if;
       end process;
end dff_as;
```



Reset non sincronizzato con il clock: può arrivare in qualunque momento

D-flip flop con Reset asincrono: VHDL

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

entity dff is
    port (CLK, RST, D: in std_logic;
                    Q: out std_logic);
end dff;
-- architettura behavioral
architecture dff_as of dff is
begin
       process (RST, CLK)
    begin
        if (RST = '1') then
              Q <= '0';
        elsif (CLK'event and CLK = '1') then
              Q <= D;
        end if;
       end process;
end dff_as;
```



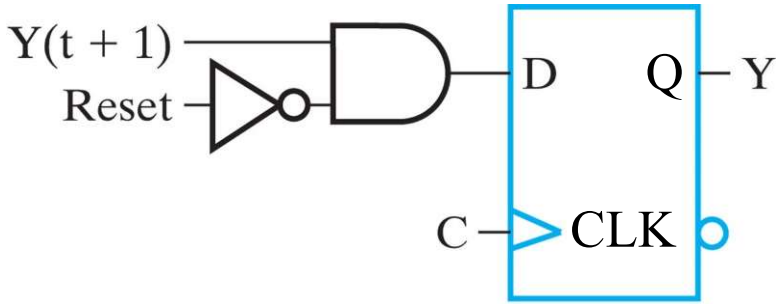
Il processo si attiva al cambiamento di almeno uno tra **CLK** e **RESET**

CLK'event è 1 se si è verificato un cambiamento nel segnale CLK. Questa variabile booleana in AND con CLK = '1' indica un fronte positivo. «event» è un attributo del segnale clock

RESET: uscita diventa 0 (domina il clock)
FRONTE di SALITA del CLK: uscita diventa D

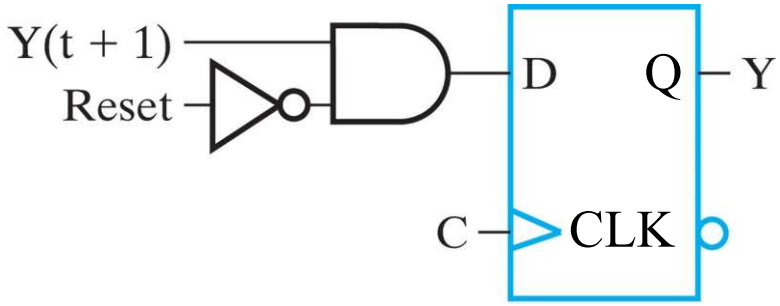
D-flip flop con Reset sincrono: VHDL

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

entity dff is
    port (CLK, RST, D: in std_logic;
                    Q: out std_logic);
end dff;
-- architettura behavioral
architecture dff_sync of dff is
begin
process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (RST = '1') then
                Q <= '0';
            else
                Q <= D;
            end if;
        end if;
end process;
end dff_sync;
```



Reset sincronizzato con il clock: può arrivare solo al fronte attivo del clock (salita in questo caso)

D-flip flop con Reset sincrono: VHDL




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

entity dff is
    port (CLK, RST, D: in std_logic;
                    Q: out std_logic);
end dff;
-- architettura behavioral
architecture dff_sync of dff is
begin
process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (RST = '1') then
                Q <= '0';
            else
                Q <= D;
            end if;
        end if;
end process;
end dff_sync;
```

Il processo si attiva solo al cambiamento di **CLK**

Il **RESET** non domina più il clock, può avvenire solo in corrispondenza al fronte di salita

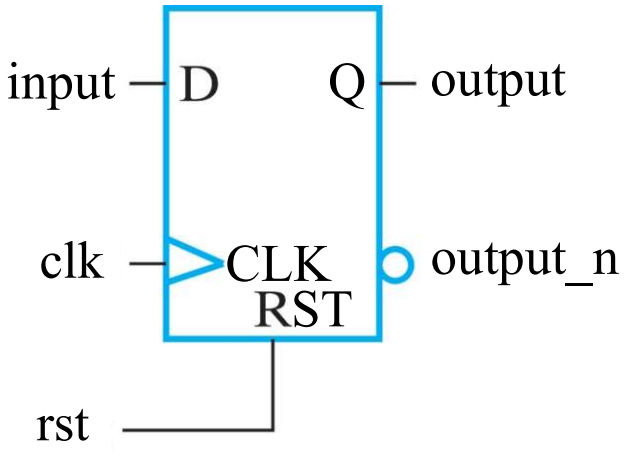
Simulazione di logica sequenziale sincrona

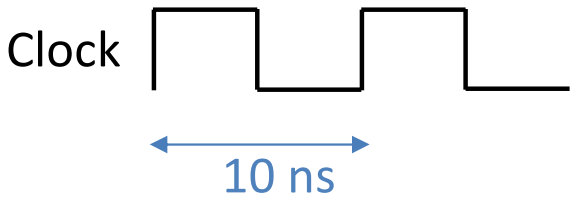
- E' necessario **cambiare gli input lontano dal fronte di clock attivo** (per es. durante il fronte di discesa del clock nell'ipotesi di D-FF PET). In questo modo, la logica combinatoria ha tempo di calcolare uscita e stato futuro fino al successivo fronte attivo del clock
- Realizzeremo il testbench con **due processi**, che funzionano in parallelo:

1) il primo **genera il clock**. Questo processo non viene mai fermato esplicitamente e riprende da capo ogni volta che ha esaurito le istruzioni contenute al suo interno, garantendo che il clock continui ad oscillare
2) il secondo **pilota i valori dei segnali in ingresso**, incluso il reset, del sistema (sincronizzando le transizioni opportunamente, per es. ai fronti di discesa del clock). Questo processo viene fermato con il comando "`std.env.stop`", che termina la simulazione

D flip-flop con Reset asincrono: testbench (1/2)

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

entity test_dff is
end test_dff;

architecture test of test_dff is
signal clk, rst, input, output: std_logic;
begin
DUT: entity work.dff(dff_as) port map(clk, rst, input, output);
    generate_clock: process
    begin
        clk <= '1';
        wait for 5 ns;
        clk <= '0';
        wait for 5 ns;
    end process;
```



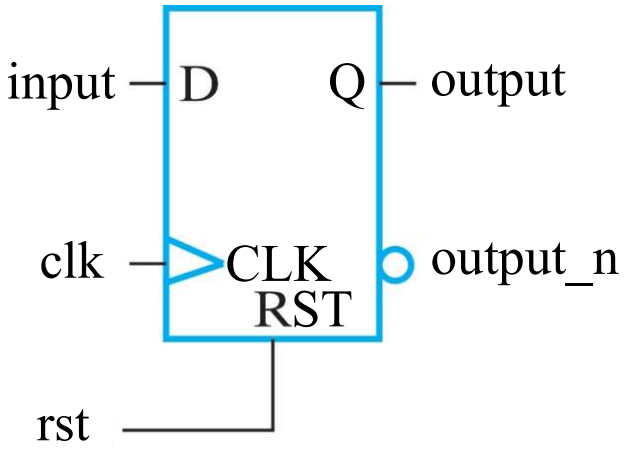


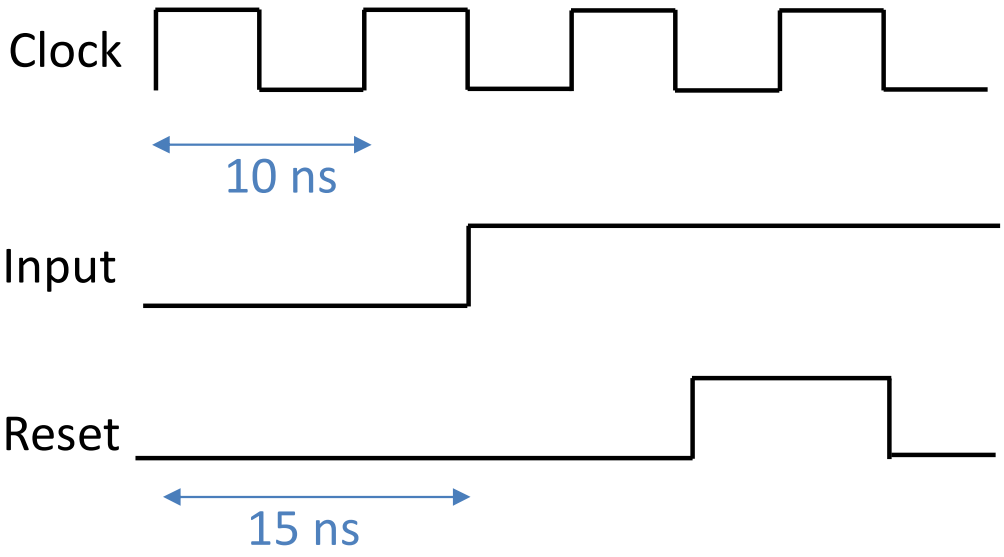
Processo che genera il **clock** (onda quadra con periodo 10 ns)

D flip-flop con Reset asincrono: testbench (2/2)

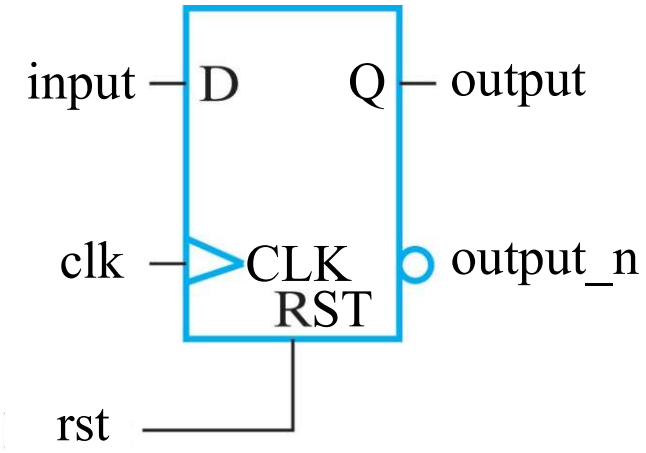
```
apply_inputs: process
    begin
        rst <= '0';
        input <= '0';
        wait for 15 ns;
        input <= '1';
        wait for 10 ns;
        rst <= '1';
        wait for 20 ns;
        rst <= '0';
        wait for 10 ns;
        input <= '0';
        wait for 10 ns;
        input <= '1';
        std.env.stop;
    end process;

end test;
```

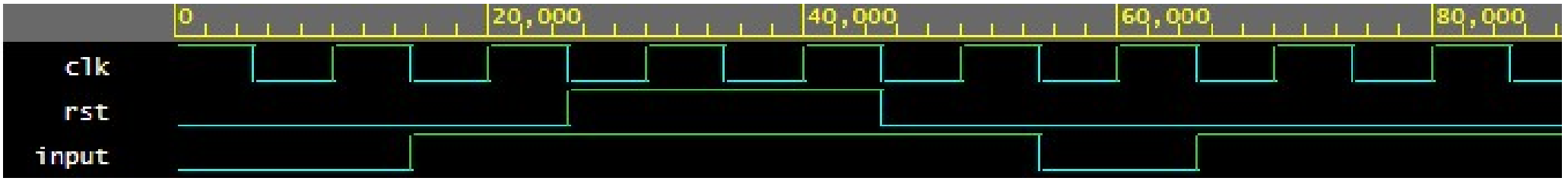
Processo che **cambia gli ingressi al fronte di discesa del clock**



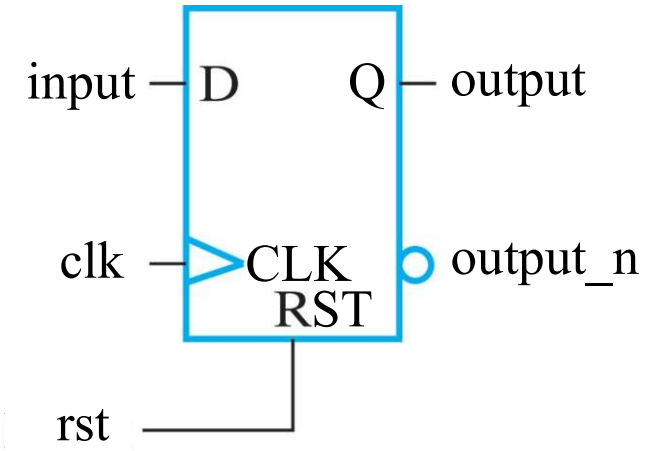


D flip-flop con Reset asincrono: forma d'onda

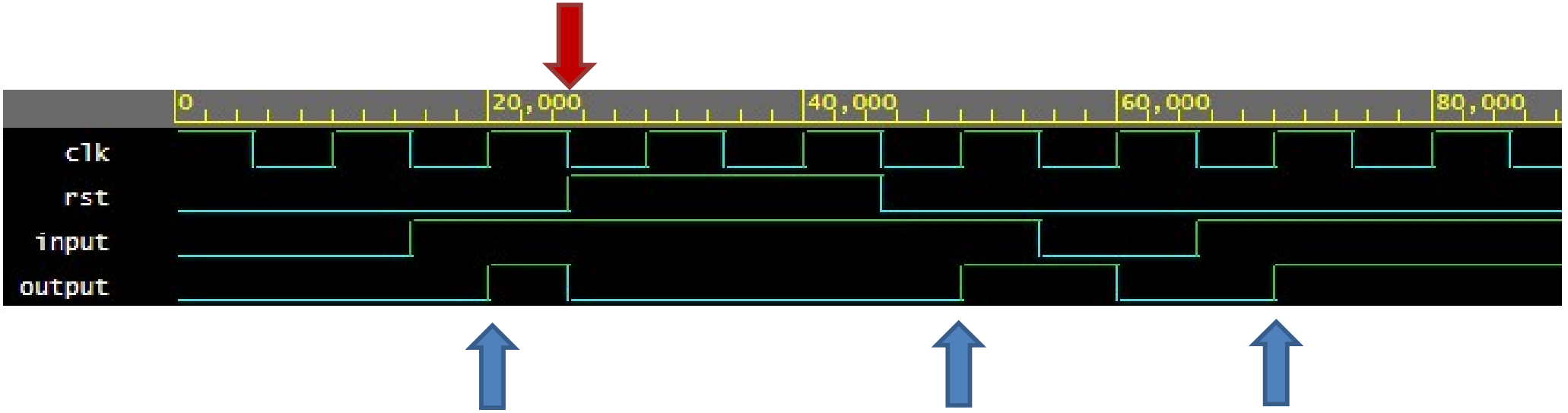




D flip-flop con Reset asincrono: forma d'onda



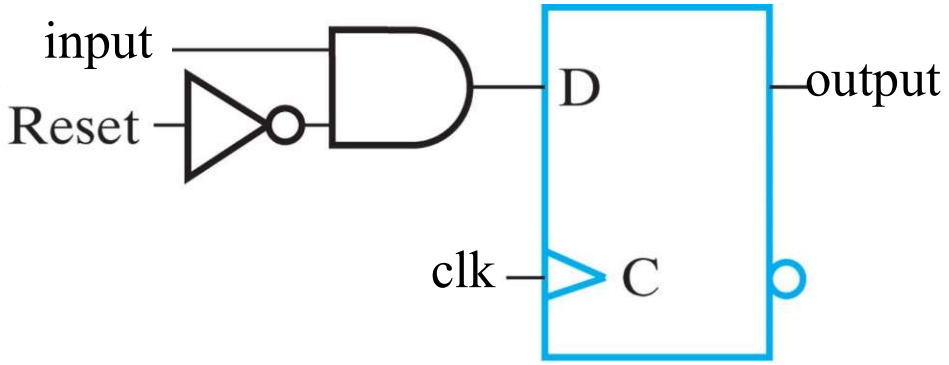
Alla salita di reset (in qualunque istante, anche non in corrispondenza del fronte del clock) l'uscita diventa 0

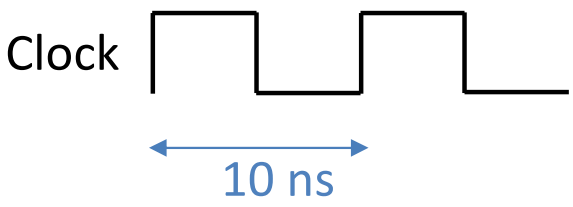


Alla salita del clock l'uscita segue l'ingresso (se reset è zero)

D flip-flop con Reset sincrono: testbench (1/2)

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

entity test_dff is
end test_dff;

architecture test of test_dff is
signal clk, rst, input, output: std_logic;
begin
DUT: entity work.dff(dff_sync) port map(clk, rst, input,
output);
    generate_clock: process
    begin
        clk <= '1';
        wait for 5 ns;
        clk <= '0';
        wait for 5 ns;
    end process;
```

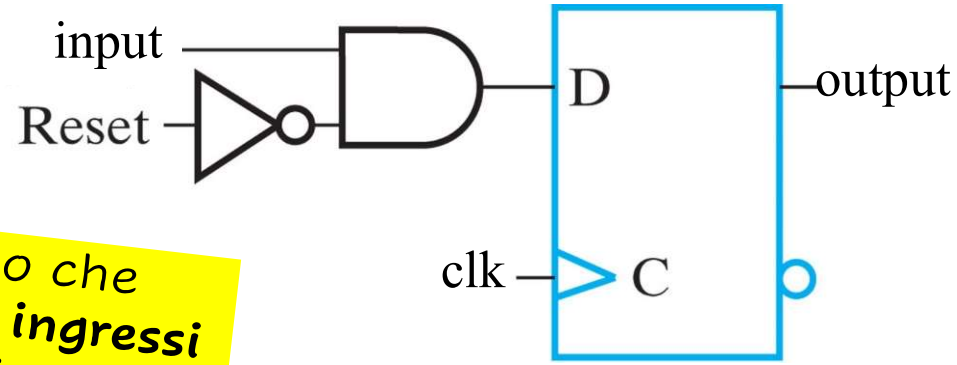




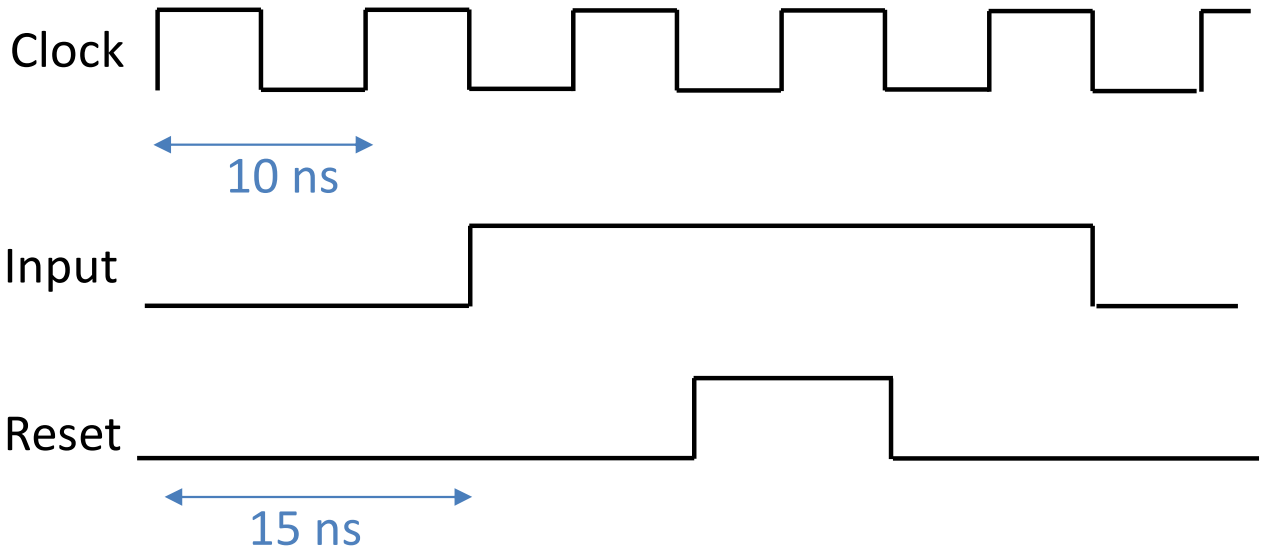
Processo che genera un **clock** con periodo 10 ns

D flip-flop con Reset sincrono: testbench (2/2)

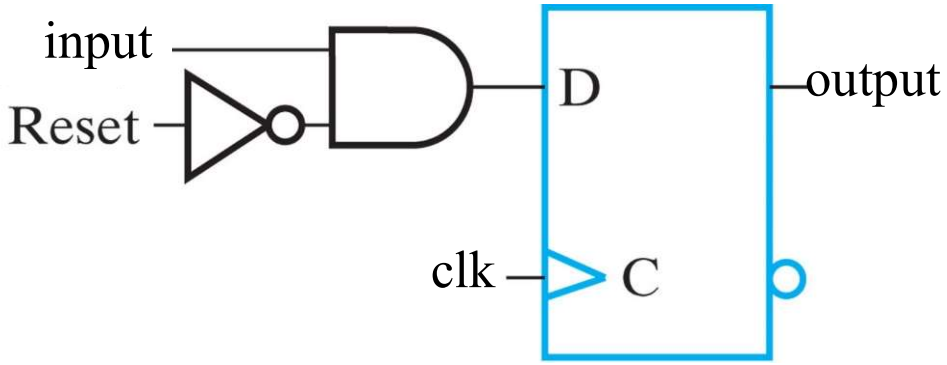
```
 apply_inputs: process
   begin
        rst <= '0';
        input <= '0';
        wait for 15 ns;
        input <= '1';
        wait for 10 ns;
        rst <= '1';
        wait for 10 ns;
        rst <= '0';
        wait for 10 ns;
        input <= '0';
        std.env.stop;
   end process;
end test;
```

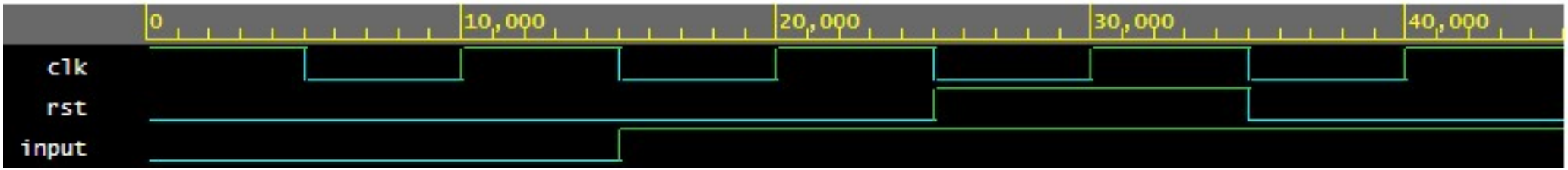


Processo che **cambia gli ingressi e reset al fronte di discesa del clock**

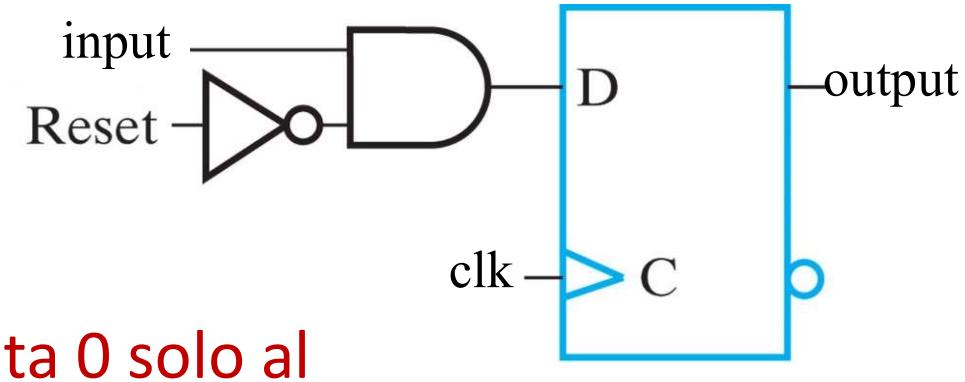


D flip-flop con Reset sincrono: forma d’onda

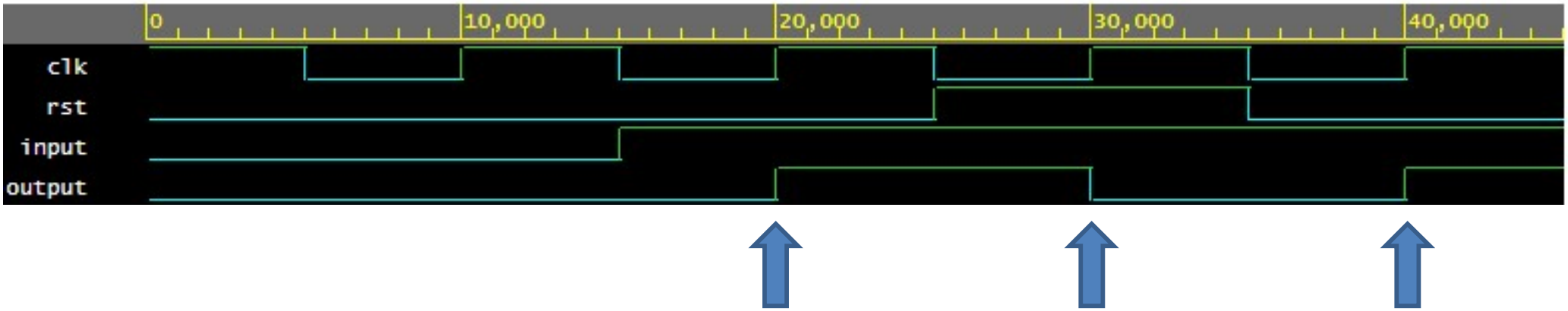




D flip-flop con Reset sincrono: forma d'onda



Reset sale e l'uscita diventa 0 solo al successivo fronte di salita del clock



Alla salita del clock l'uscita segue l'ingresso (se reset è zero)

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin

© 2016 Pearson Education, Ltd