Introduzione al Digital Logic Design usando il linguaggio VHDL

Pietro Lorefice
Linux & Firmware Engineer, Develer

Pietro Lorefice pietro@develer.com

OBIETTIVI

- Prendere familiarità con la logica digitale

- Acquisire le basi per il design di logica digitale complessa

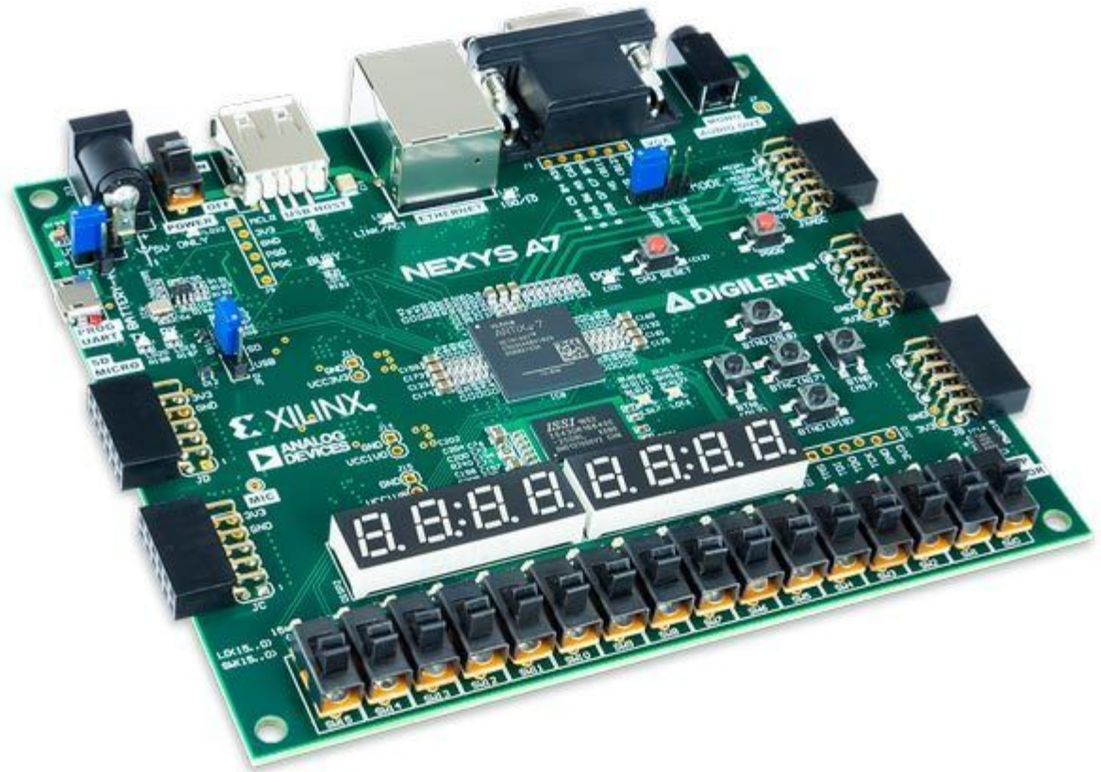
- Apprendere il linguaggio VHDL per la descrizione di hardware digitale

Pietro Lorefice pietro@develer.com

HARDWARE

Digilent Nexys A7

- Xilinx Artix-7 XC7A100T-1CSG324C
- 15,850 Logic Slices
- 450 MHz+ internal clocks
- 128MiB DDR2



develer

Pietro Lorefice pietro@develer.com

SOFTWARE

GHDL

- Simulatore open-source per il linguaggio VHDL

GTKWave

- Visualizzatore di forme d’onda open-source

Pietro Lorefice pietro@develer.com

LAB SETUP

- Scaricare e installare GHDL: **http://ghdl.free.fr/**
- Scaricare e installare GTKWave: **http://gtkwave.sourceforge.net/**
- Eseguire il clone del repository Git all'URL:

https://github.com/plorefice/vhdl-techlab-2019

Pietro Lorefice pietro@develer.com

COS'È LA LOGICA DIGITALE

> *Insieme di regole di inferenza che operano su dei valori digitali (**bit**)*

- Un bit può trovarsi in uno solo di due possibili stati: "**1**" o "**0**"
  - Il valore "**1**" del bit viene detto anche "*stato logico alto*" o "*vero*"
  - Il valore "**0**" del bit viene detto anche "*stato logico basso*" o "*falso*"
- La logica digitale definisce le operazioni effettuabili sui bit, e le regole che le governano

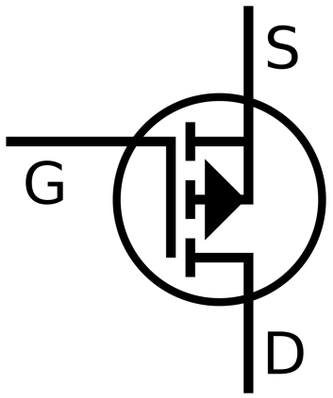
Pietro Lorefice pietro@develer.com

LOGICA DIGITALE “CMOS”

- Una di molte famiglie di logiche digitali (TTL, ECL, …)

- Lo stato logico di un bit è rappresentato da un livello di tensione (es. 0V ~ 5V)
  - Tipicamente 0V rappresenta lo “0”, mentre 5V corrisponde a ”1”

- L’elemento fondamentale di questa famiglia logica è il **MOSFET**

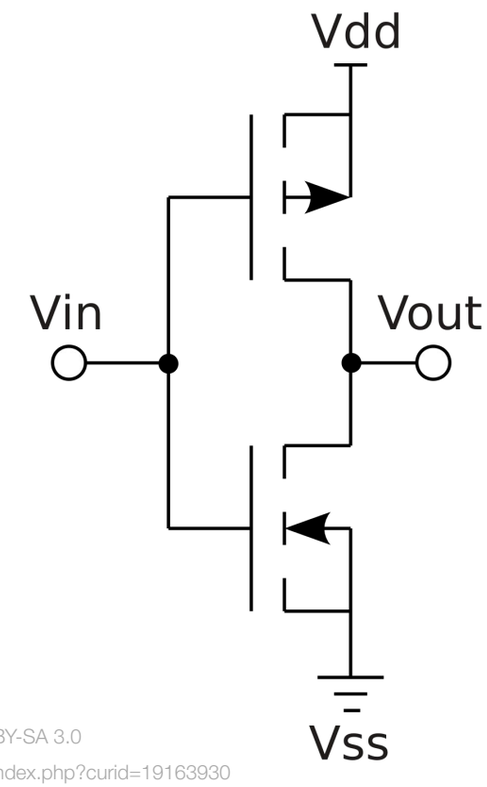


By jjbeard - From Scratch in Inkcape 0.43, Public Domain
https://commons.wikimedia.org/w/index.php?curid=829707

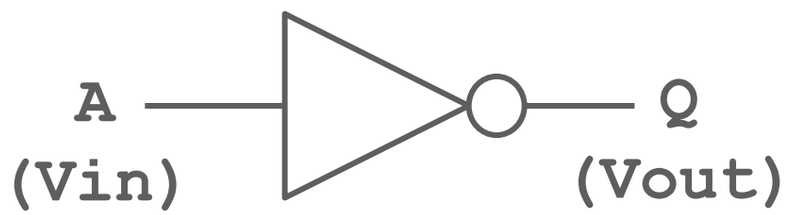
Pietro Lorefice pietro@develer.com

PORTA “NOT” CMOS

- L’elemento fondamentale di ogni famiglia logica è la porta **NOT**
  - In logica CMOS, una porta NOT è costituita da un p-MOS e un n-MOS così disposti



By Abaddon1337 - Own work, CC BY-SA 3.0
https://commons.wikimedia.org/w/index.php?curid=19163930



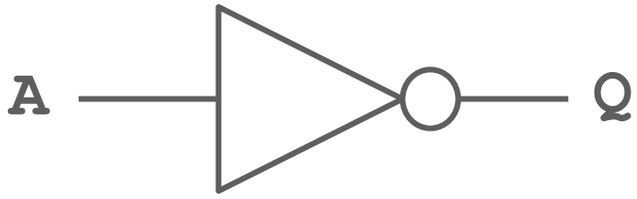
Q = /A

A	Q
0	1
1	0

Pietro Lorefice pietro@develer.com

BLOCCHI FONDAMENTALI

- L’elemento fondamentale della logica digitale è la porta **NOT**



Q = /A

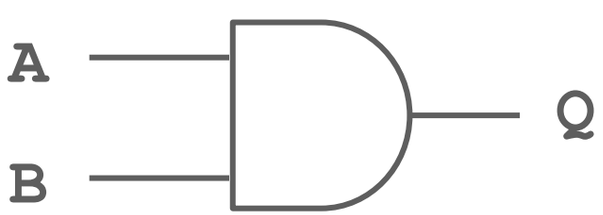
Tabella di verità

A	Q
0	1
1	0

Pietro Lorefice pietro@develer.com

BLOCCHI FONDAMENTALI

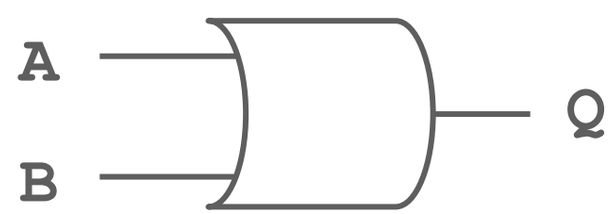
AND



$Q = A \cdot B$

A	B	Q
0	0	0
0	1	0
1	**1**	**1**
1	0	0

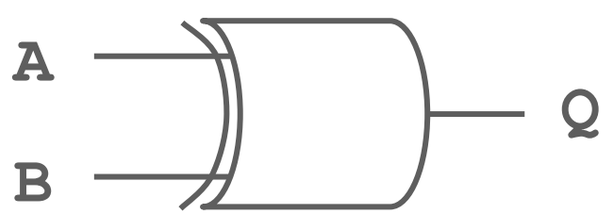
OR



$Q = A + B$

A	B	Q
0	0	0
0	**1**	**1**
1	**1**	**1**
1	**0**	**1**

XOR



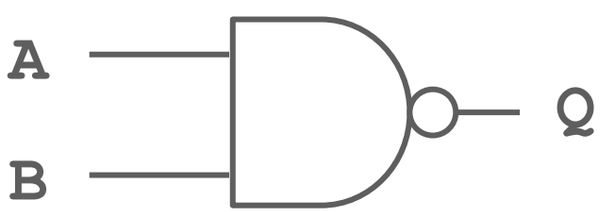
$Q = A \oplus B$

A	B	Q
0	0	0
0	**1**	**1**
1	1	0
1	**0**	**1**

Pietro Lorefice pietro@develer.com

BLOCCHI FONDAMENTALI

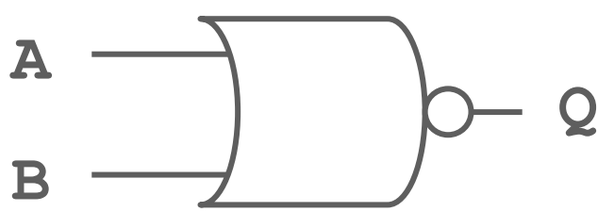
NAND



Q = /(A · B)

A	B	Q
0	**0**	**1**
0	**1**	**1**
1	1	0
1	**0**	**1**

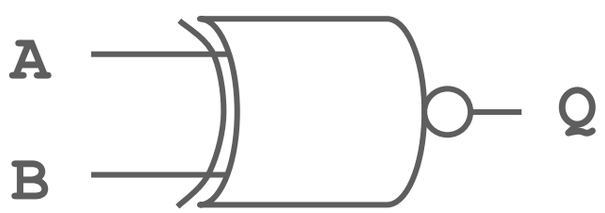
NOR



Q = /(A + B)

A	B	Q
0	**0**	**1**
0	1	0
1	1	0
1	0	0

XNOR



Q = /(A ⊕ B)

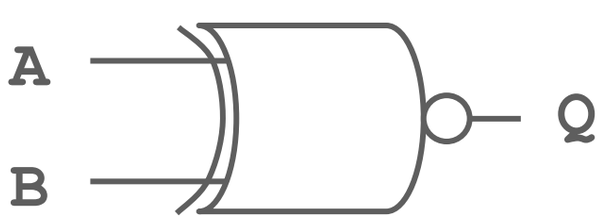
A	B	Q
0	**0**	**1**
0	1	0
1	**1**	**1**
1	0	0

develer

Pietro Lorefice pietro@develer.com

ESEMPIO — EQUALITY TESTER

- Realizzare un componente che dati due bit in ingresso, **A** e **B**, produca in uscita:
  - ‘1’ se **A == B**
  - ‘0’ altrimenti



XNOR !!

A	B	Q
0	**0**	**1**
0	1	0
1	**1**	**1**
1	0	0

Pietro Lorefice pietro@develer.com

VHDL — INTRODUZIONE

- Linguaggio di *descrizione hardware*

- Utilizzato nella descrizione di sistemi digitali e mixed-signals (FPGA, ASIC, …)

- Sintatticamente basato su *Ada*

Pietro Lorefice pietro@develer.com

VHDL — SCOPE

- Linguaggio orientato sia alla **sintesi** che alla **simulazione** di sistemi digitali

  - **Sintesi**: processo di traduzione di un design ad un livello d'astrazione inferiore

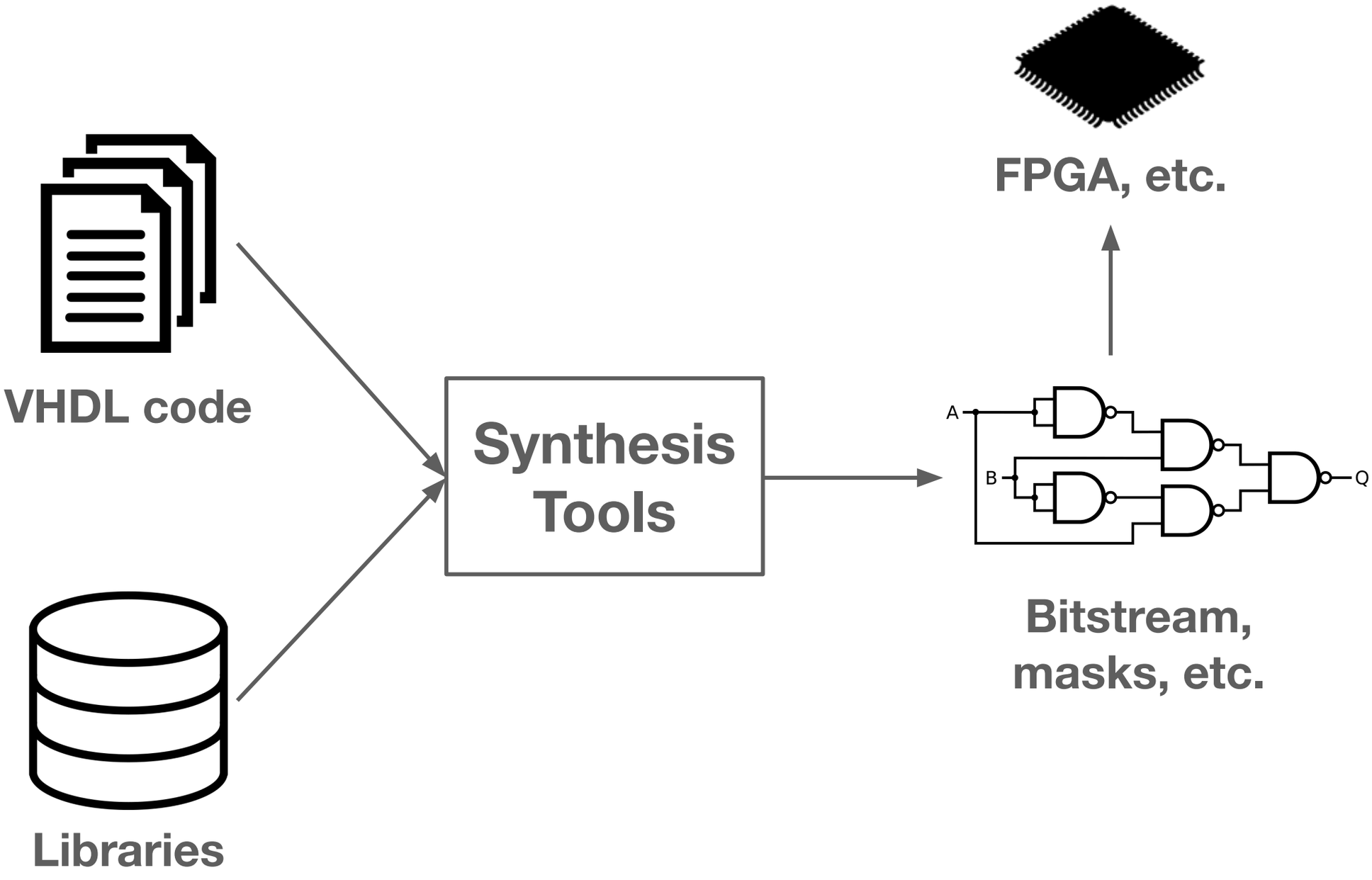
    - Ad es. implementazione di un design in FPGA o realizzazione di ASICs

  - **Simulazione**: processo di esecuzione di un design in un ambiente software

    - Ad es. scrittura di test cases per la verifica di un design

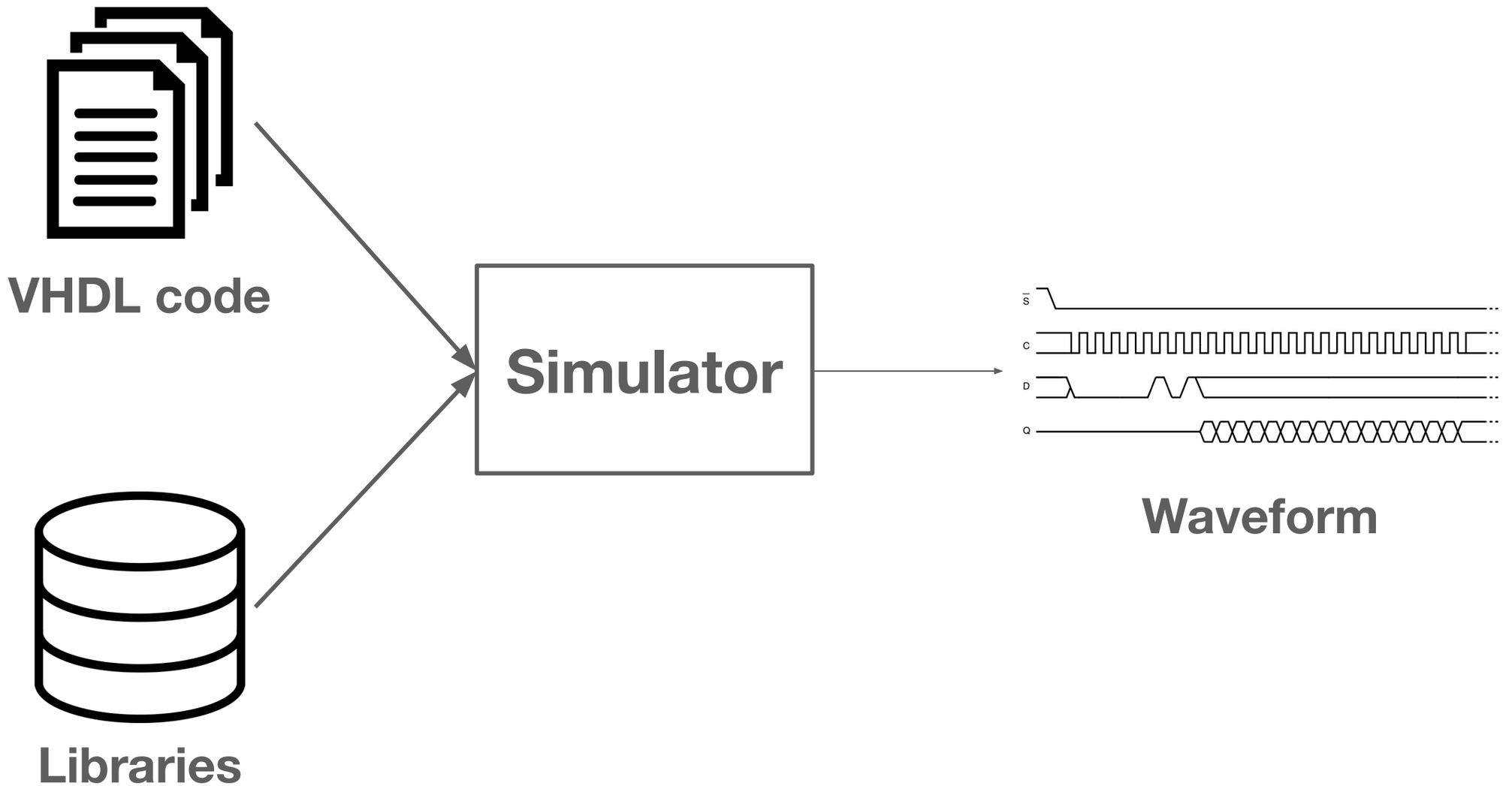
Pietro Lorefice pietro@develer.com

VHDL — SINTESI



Pietro Lorefice pietro@develer.com

VHDL — SIMULAZIONE



Pietro Lorefice pietro@develer.com

VHDL — CONCETTI BASE

- Ciascun design in VHDL contiene almeno un ***entity*** e una relativa ***architecture***

- Una *entity* descrive l'interfaccia del componente verso l'esterno, ie. ingressi e uscite

- Una *architecture* contiene l'implementazione interna del componente
    - Ciascuna *entity* può avere più di una *architecture* associata

- Opzionalmente, un design può includere una o più *librerie*

Pietro Lorefice pietro@develer.com

VHDL — STRUTTURA DI UN FILE

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

entity foo is
    -- ...
end entity;

architecture bar of foo is
begin
    -- ...
end architecture;
```

develer

Pietro Lorefice pietro@develer.com

VHDL — DATA TYPES

- **`std.standard`**
  - `bit, boolean, integer, real, character, physical`
- **`ieee.std_logic_1164`**
  - `std_logic, std_logic_vector`
- **`ieee.numeric_std`**
  - `signed, unsigned`
- User defined
  - `type, array, record`

Pietro Lorefice pietro@develer.com

VHDL — DATA KINDS

- `signal`
  - Emula il comportamento di un filo
  - Può essere usato sia in codice *parallelo* che *sequenziale*
  - L'assegnamento avviene tramite operatore **<=** ed è affetto da *ritardi*

- `variable`
  - Simile al concetto di variabile nei linguaggi di programmazione classici
  - Può essere dichiarata e usata solo in codice *sequenziale*
  - L'assegnamento avviene tramite operatore `:=` ed è *immediato*
  - **Non** possono essere usate nell'interfaccia di un design (ie. in una *entity*)

Pietro Lorefice pietro@develer.com

VHDL TIME!

Pietro Lorefice pietro@develer.com

VHDL — TESTING

- Aspetto fondamentale del processo di design hardware
  - Per individuare errori prima di andare in fab o spendere ore in una sintesi
  - Quanto più esaustivo e completo possibile

- Simulazioni volte a testare un design vengono anche chiamate ***testbench***

- VHDL scritto per testing (simulazione) è concettualmente diverso dal VHDL per sintesi
  - Molti costrutti validi in simulazione **non sono sintetizzabili!**

develer

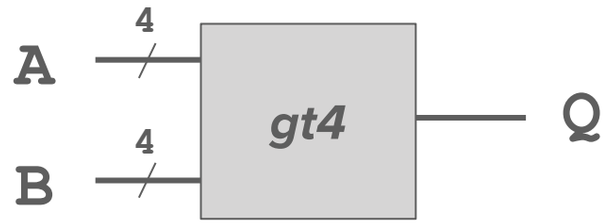
Pietro Lorefice pietro@develer.com

VHDL TIME!

Pietro Lorefice pietro@develer.com

ESEMPIO — 4-BIT COMPARATOR

- Realizzare un componente che dati due segnali a 4-bit in ingresso, produca in uscita:
  - ‘1’ se **A > B**
  - ‘0’ altrimenti



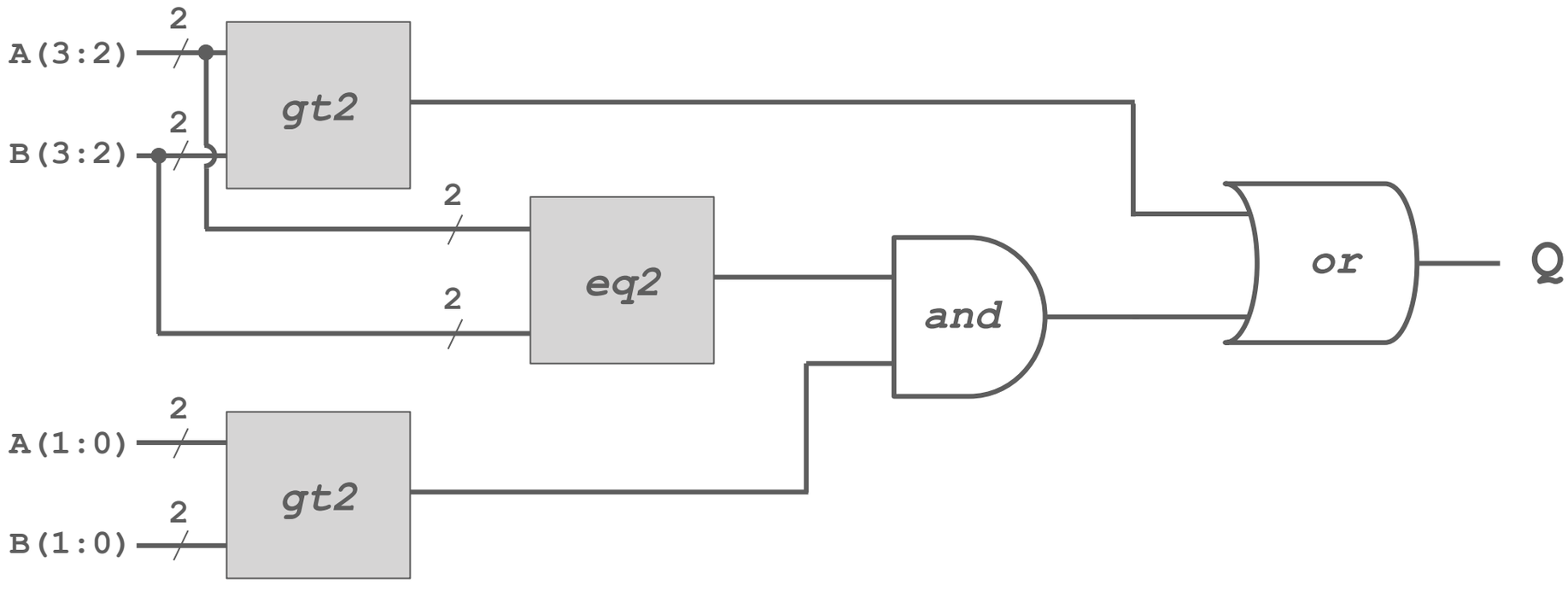
??

A	B	Q
0000	0000	0
0001	**0000**	**1**
0010	**0000**	**1**
…		…
0000	0001	0
0001	0001	0
0010	**0001**	**1**
…		…

Pietro Lorefice pietro@develer.com

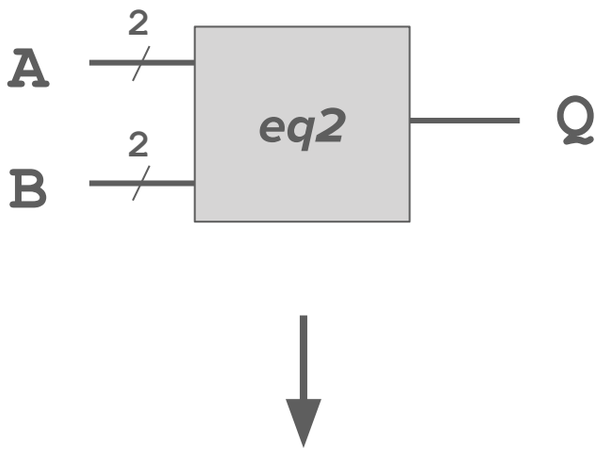
ESEMPIO — 4-BIT COMPARATOR

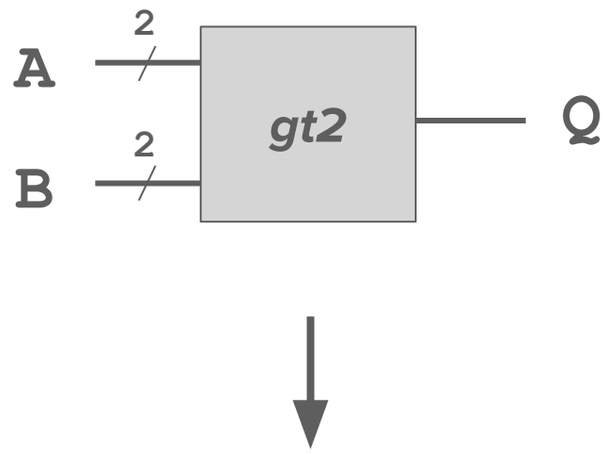
- Posso notare che:
  - `A(3:2)  > B(3:2)` → `Q <= '1'`
  - `A(3:2) == B(3:2) && A(1:0) > B(1:0)` → `Q <= '1'`
  - `otherwise` → `Q <= '0'`

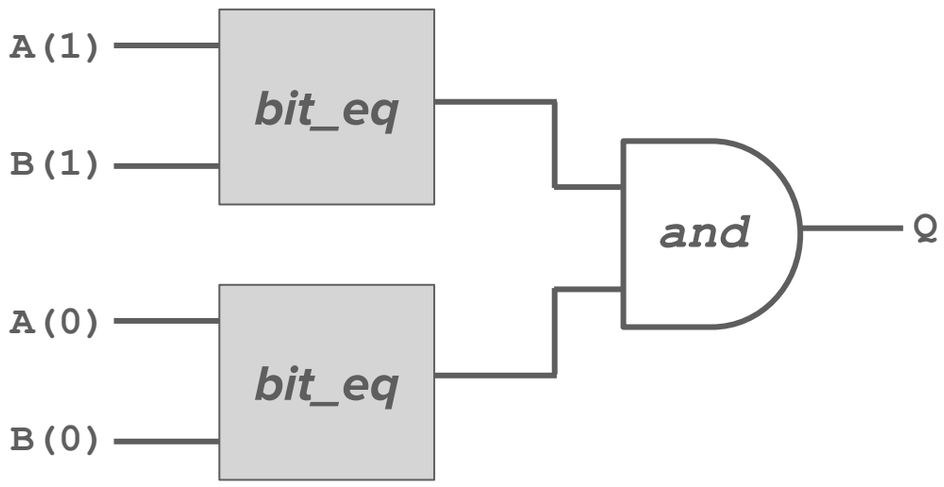


Pietro Lorefice pietro@develer.com

ESEMPIO — 4-BIT COMPARATOR





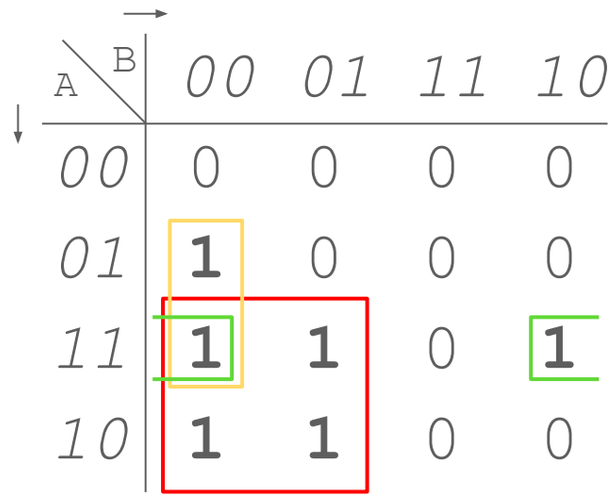


A \ B	00	01	11	10
00	0	0	0	0
01	**1**	0	0	0
11	**1**	**1**	0	**1**
10	**1**	**1**	0	0

Pietro Lorefice pietro@develer.com

MAPPE DI KARNAUGH

- Metodo per la semplificazione di espressioni algebriche booleane
  - Permette di trovare l'espressione minima in termini di *somme di prodotti*



Q = (+ +

Pietro Lorefice pietro@develer.com

VHDL TIME!

Pietro Lorefice pietro@develer.com

VHDL — LOGICA SEQUENZIALE

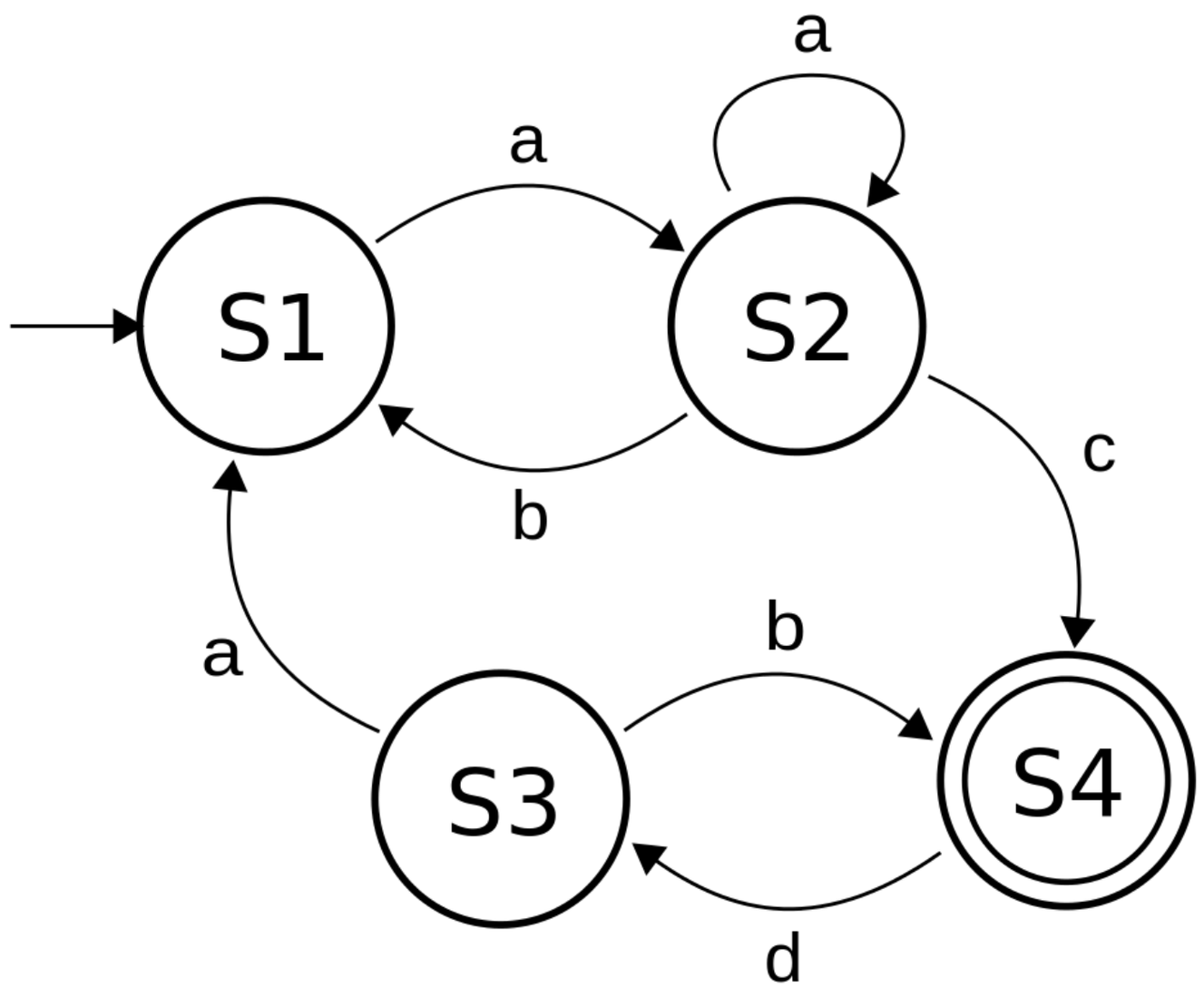
- Fino ad ora abbiamo analizzato esempi di *logica combinatoria*
  - Il segnale in uscita è una *funzione pura* del segnale in ingresso, ie. `f(A) = Q`

- Nella *logica sequenziale* l'uscita è funzione dell'ingresso attuale e degli *ingressi passati*
  - Vale a dire, la logica sequenziale è dotata di **memoria** e quindi di uno **stato**

- La logica sequenziale è tipicamente *sincrona*
  - L'aggiornamento dello stato avviene in corrispondenza di un **segnale di clock**

Pietro Lorefice pietro@develer.com

VHDL — FINITE STATE MACHINE



Pietro Lorefice pietro@develer.com

VHDL — FLIP-FLOP

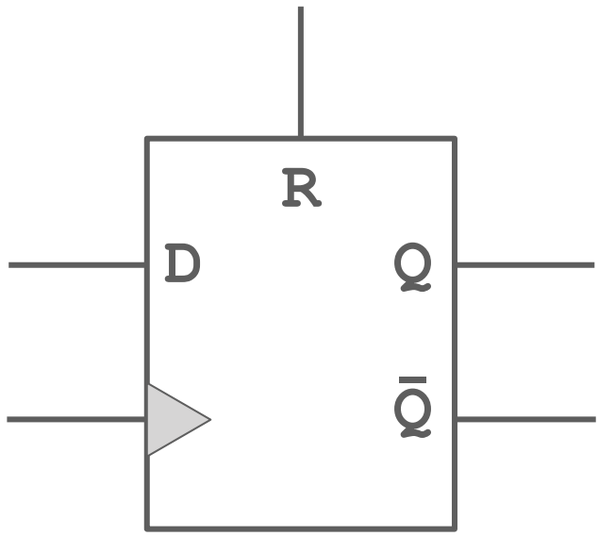
- Blocco fondamentale della logica sequenziale

- Un flip-flop ha due stati stabili e può immagazzinare un bit di informazione
  - Uno dei due stati codifica un ‘1’, l’altro uno ‘0’

- Esistono diverse tipologie di flip-flop, a seconda di input e sincronizzazione
  - La più diffusa è l’***edge-triggered D-flip-flop***

Pietro Lorefice pietro@develer.com

VHDL — D-FLIP-FLOP



R	D	>	Q
1	–	–	0
0	0	⌐	0
0	1	⌐	1
0	–	–	Q

Pietro Lorefice pietro@develer.com

VHDL TIME!

Pietro Lorefice pietro@develer.com

DOMANDE?

Pietro Lorefice pietro@develer.com

CONTATTI

github

github.com/plorefice

e-mail

pietro@develer.com

web

develer.com/pietro-lorefice

Pietro Lorefice pietro@develer.com