



CheatSheet Digital Design

Fs 2025 – Prof. Dr. Paul Zbinden

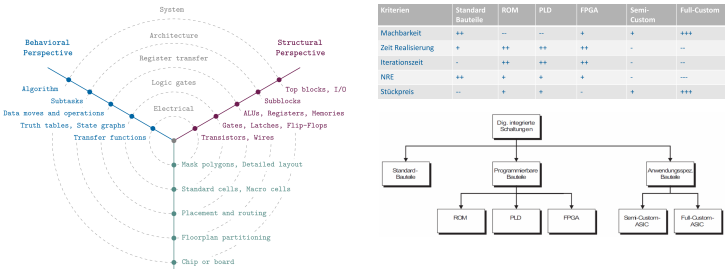
Autoren: Ricca Aaron

<https://github.com/Rin-Ha-n/DigDes>

V 1.0 June 12, 2025

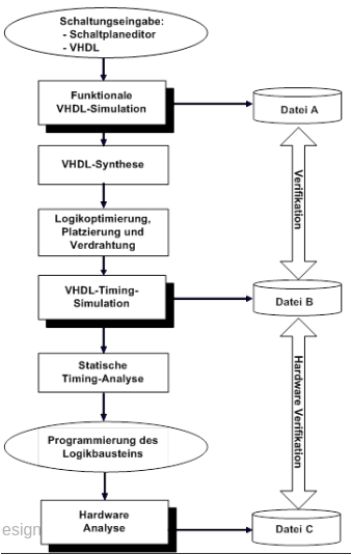
1 Introduzione

1.1 Scelta/caratteristiche dei componenti



1.2 Guida al design

- Design / Entry
- Funktionale Simulation
- Synthese
 - Logikoptimierung
 - Platzierung
 - Verdrahtung
- Timing Simulation
- Statische Timing Analyse
- Herstellungsdaten erzeugen



2 Programmazione VHDL

2.1 Library

Una libreria può contenere componenti e o pacchetti. I componenti sono descrizione di circuiti e realizzazione specifiche, vengono memorizzati nella libreria in modo da poter essere riutilizzati più volte e da più progettisti contemporaneamente. I blocchi di codice di una libreria sono memorizzati in forma compilata, direttamente eseguibile.

Contenuto di una libreria: Components, Packages, Functions, Procedures, Declarations.

```
1 library ieee;
2 use ieee.std_logic_1164.all; -- CPP: using namespace std;
3 use ieee.numeric_std.all; -- Solo per operazioni aritmetiche
   ↳ per vettori
```

2.2 entity

```
1 entity <entity name> is
2     port (
3         {<port_name> : <mode> <type>;}
4     );
5 end entity <entity name>;
```

2.3 architecture

```
1 architecture <architecture_type> of <entity_name> is
2     [type_declaration]
3     [subtype_declaration]
4     [constant_declaration]
5     [signal_declaration]
6     [component_declaration]
7
8     begin
9         -- codice di architettura
10    end <architecture_type>;
```

2.4 component

I componenti sono utilizzati per definire le porte di un'entità, in modo da poterla utilizzare in altre entità.

```
1 component <component_name> is
2     port (
3         {<port_name> : <mode> <type>;}
4     );
5 end component <component_name>;
```

2.5 Processi

I processi sono sezioni di codice che vengono eseguite ogni volta che un **Segnale sensibile** nella lista sensibile cambia di stato.

```
1 process (clk, reset)
2     begin
3         if reset = '1' then
4             -- codice di reset
5         elsif rising_edge(clk) then
6             -- codice di clock
7         end if;
8     end process;
```

2.6 Tipi

• <architecture_type> = Behavioral | Structural | RTL | Dataflow | Tb

• <mode> = in | out | inout

• <type> = bit | bit_vector | std_u logic | std_u logic_vector | integer | boolean

2.6.1 <architecture_type>

Behavioral: si occupa di descrivere il comportamento del circuito, senza preoccuparsi della struttura fisica.

```
1 if rising_edge(clk) then
2     if A = '1' then
3         Y <= B;
4     end if;
5 end if;
```

Structural: si occupa di descrivere la struttura fisica del circuito, utilizzando componenti e connessioni tra di essi.

```
1 U1: and_gate port map (A => A, B => B, Y => Y1);
2 U2: or_gate port map (A => A, B => C, Y => Y);
```

RTL: si occupa di descrivere il circuito a livello di registro e logica combinatoria, utilizzando registri e porte logiche.

```
1 if rising_edge(clk) then
2     reg1 <= A and B;
3     reg2 <= reg1 xor C;
4 end if;
```

Dataflow: si occupa di descrivere il circuito a livello di flusso di dati, utilizzando porte logiche e segnali.

```
Y <= (A and B) or (not C);
```

Tb: si occupa di descrivere il circuito a livello di testbench, utilizzando segnali di test e componenti di test.

```
1 A <= '0'; wait for 10 ns;
2 A <= '1'; wait for 10 ns;
3 assert (Y = expected_value) report "Test failed" severity
   ↳ error;
```

2.6.2 <type>

• bit: rappresenta un singolo bit, con valori '0' e '1'.

```
1 signal A : bit;
```

• bit_vector: rappresenta un vettore di bit, con valori '0' e '1'.

```
1 signal B : bit_vector(7 downto 0); -- vettore di 8 bit
```

• std_logic: rappresenta un singolo bit con valori '0', '1'.

```
1 signal C : std_logic;
```

• std_logic_vector: rappresenta un vettore di std_logic, con valori '0', '1'.

```
1 signal D : std_logic_vector(7 downto 0);
```

• std_ulogic: rappresenta un singolo bit con valori '0', '1', 'Z' (alta impedenza) e 'X' (indeterminato).

```
1 signal E : std_ulogic;
```

• std_ulogic_vector: rappresenta un vettore di std_ulogic, con valori '0', '1', 'Z' e 'X'.

```
1 signal F : std_ulogic_vector(7 downto 0);
```

• integer: rappresenta un numero intero, con valori compresi tra -2^{31} e $2^{31}-1$ (è necessario definire l'intervallo di utilizzo).

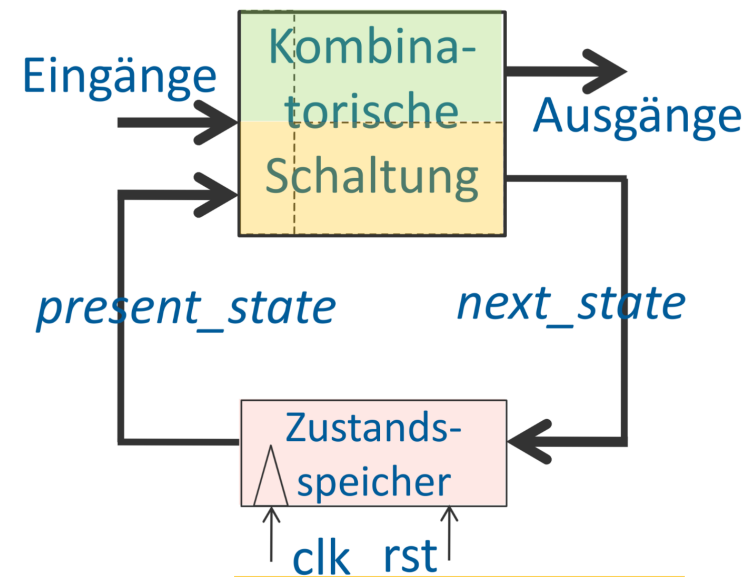
```
1 signal G : integer range 0 to 255; -- intervallo di
   ↳ utilizzo
```

• boolean: rappresenta un valore booleano, con valori true e false.

```
1 signal H : boolean; -- true or false
```

3 State machine

Le **Finite State Machine** (FSM) sono circuiti sequenziali che possono essere in uno stato tra un insieme finito di stati. La transizione tra gli stati avviene in base a segnali di ingresso e può essere condizionata da segnali di clock e reset.



3.1 Codice scheletro FSM (f,g,z)

3.2 Codifica degli stati

Gli stati di una FSM possono essere codificati in diversi modi, tra cui:

- **Codifica binaria:** ogni stato è rappresentato da un codice binario unico.
- **Codifica Gray:** simile alla codifica binaria, ma le transizioni tra stati adiacenti cambiano solo un bit alla volta.
- **Codifica one-hot:** ogni stato è rappresentato da un bit attivo, con tutti gli altri bit a zero.
- **Codifica one-cold:** simile alla codifica one-hot, ma solo un bit è a zero e tutti gli altri sono attivi.