CheatSheet Digital Design

Fs 2025 – Prof. Dr. Paul Zbinden

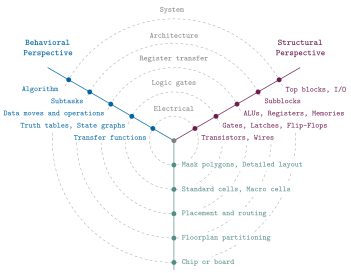
Autoren: Ricca Aaron

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

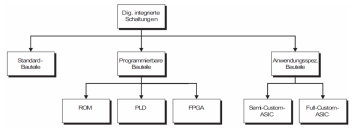
V 1.0June 12, 2025

1 Introduzione

1.1 Scelta/caratteristiche dei componenti

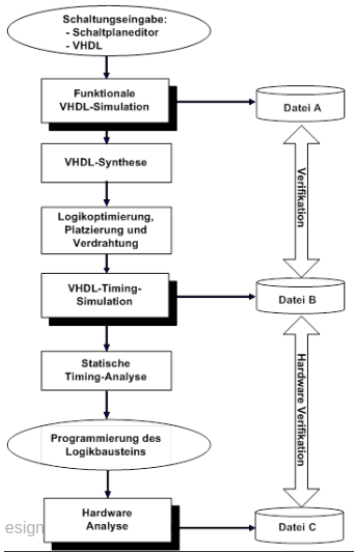


Kriterien	Standard Bauteile	ROM	PLD	FPGA	Semi-Custom	Full-Custom
Machbarkeit	++	--	--	+	+	+++
Zeit Realisierung	+	++	++	++	-	--
Iterationszeit	-	++	++	++	-	--
NRE	++	+	+	+	-	---
Stückpreis	--	+	+	-	+	+++



1.2 Guida al design

1. Design / Entry
2. Funktionale Simulation
3. Synthese
4. Implementierung
   - Logikoptimierung
   - Platzierung
   - Verdrahtung
5. Timing Simulation
6. Statische Timing Analyse
7. Herstellungsdatenerzeugen



2 Programmazione VHDL

2.1 Library

Una libreria puo contenere componenti e o pachetti. I componenti sono descrizione di circuiti e realizzazione specifiche, vengono memorizzati nella libreria in modo da poter essere riutilizati piu volte e da piu 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.

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

L'entità descrive il componente del progetto. In primo luogo l'entità descrive l'interfaccia (schnittstelle) del componente.

2.2 entity

```vhdl
entity <entity name> is
    port (
        {<port_name> : <mode> <type>;}
    );
end entity <entity name>;
```

L'architettura descrive il comportamento del componente, come funziona e come è realizzato.

2.3 architecture

```vhdl
architecture <architecture_type> of <entity_name> is
    [type_declaration]
        [subtype_declaration]
        [constant_declaration]
        [signal_declaration]
        [component_declaration]
begin
    -- codice di architettura
end <architecture_type>;
```

2.4 Port mapping

Il port mapping è utilizzato per collegare le porte dell'entità con i segnali dell'architettura.

```vhdl
U1: entity_name
    port map (
        <port_name> => <signal_name>,
        <port_name> => <signal_name>
    );
```

2.4.1 Esempio

```vhdl
architecture structural of half_adder is
    component xor2
        port (
            in1, in2 : in bit;
            oup      : out bit
        );
    end component;

    component and2
        port (
            in1, in2 : in bit;
            oup      : out bit
        );
    end component;

    begin
    -- instantiation of ocmponents XOR2 and AND2
    -- Mappatura esplicita
    U1 : xor2
        port map (
            in1 => q,
            in2 => p,
            oup  => s
        );
    -- Mappatura implicita
    U2 : and2
        port map (p, q, s) -- L'ordine delle porte segue quello della dichiarazione del componente!
```

2.5 component

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

```vhdl
component <component_name> is
    port (
        {<port_name> : <mode> <type>;}
    );
end component <component_name>;
```

2.6 Processi

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

```vhdl
process (clk, reset)
    begin
        if reset = '1' then
            -- codice di reset
        elsif rising_edge(clk) then
            -- codice di clock
        end if;
    end process;
```

2.7 Tipi

- `<architecture_type>` = `Behavioral` | `Structural` | `RTL` | `Dataflow` | `Tb`
- `<mode>` = `in` | `out` | `inout`
- `<type>` = `bit` | `bit_vector` | `std_ulogic` | `std_ulogic_vector` | `integer` | `boolean`

2.7.1 `<architecture_type>`

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

```vhdl
if rising_edge(clk) then
    if A = '1' then
    Y <= B;
    end if;
end if;
```

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

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

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

```vhdl
if rising_edge(clk) then
    reg1 <= A and B;
    reg2 <= reg1 xor C;
end if;
```

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

```vhdl
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.

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

2.7.2 <type>

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

```
signal A : bit;
```

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

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

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

```
signal C : std_logic;
```

- `std_logic_vector`: rappresenta un vettore di std_logic, con valori `'0'`, `'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).

```
signal E : std_ulogic;
```

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

```
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).

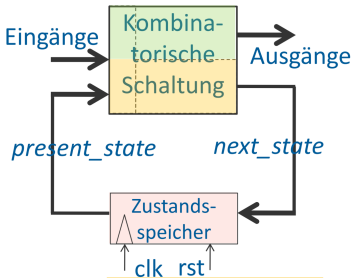
```
signal G : integer range 0 to 255; -- intervallo di
    ↪ utilizzo
```

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

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

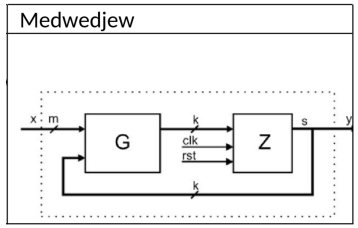
3 State machine

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 ed é regolata da segnali di clock e reset.



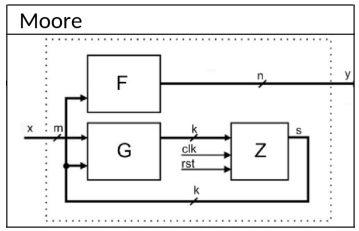
3.1 Medwedjew

Composta da una un blocco di logica combinatoria(G) che risolve la funzione desiderata e da un blocco di memoria(Z) che memorizza gli stati.
=> Gli Input e lo stato precedente della FSM vengono processati da una logica combinatoria(G)
=> Il risultato della logica viene memorizzato nella Zustandspeicher(Z)
=> L'uscita è esattamente la copia di tutti gli stati memorizzati(s).



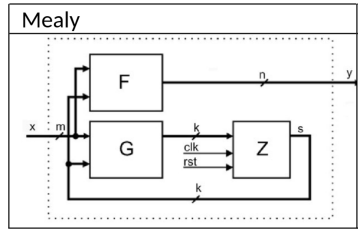
3.2 Moore

Come Medwedjew ma con una logica dedicata sul ramo di output(s).
Tipicamente utile per output più complessi/numerosi rispetto agli stati memorizzati($k \neq n$).
=> logica combinatoria aggiuntiva sul ramo d'uscita (F)
=> Più efficiente di Medwedjew per la memorizzazione degli stati



3.3 Mealy

Si tratta della Versione Moore dove la logica sul ramo d'uscita è dipendente anche a segnali provenienti direttamente dagli input della FSM
=> Necessaria se y dipende asincronamente da delle entrate
=> Più complessa, spesso riducibile a Moore



3.4 Codice scheletro FSM (f,g,z)

3.5 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.