



Politecnico di Milano – Polo di Como

Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

VHDL

Politecnico di Milano
Polo di Como

Sommario



- Generalità
- Interfaccia
- Funzionalità
 - Livelli di descrizione
- Segnali
- Reti Combinatorie
- Macchine a stati
 - Process, Variabili e segnali, Istruzioni sequenziali
- Gerarchia
- Strutture parametriche
 - Costrutti generic e generate
- VHDL e sintesi

Generalità



- VHDL è un linguaggio concepito per scrivere modelli di sistemi digitali
 - **V**ery (High Speed Integrated Circuit) **H**ardware **D**escription Language
- Ragioni per realizzare un modello
 - Specifica dei requisiti
 - Documentazione
 - Verifica mediante simulazione
 - **Sintesi**
 - Concetto di base: raggiungere la massima affidabilità del processo di progetto riducendo sia il costo sia il tempo che gli errori di progetto

Generalità



- VHDL non è un linguaggio eseguibile
 - Non descrive quali operazioni devono essere eseguite da un esecutore generico
- VHDL è un linguaggio che consente di produrre una specifica che può essere simulata
 - Due aspetti:
 - Specificare un progetto
 - Verificarne il comportamento che la specifica descrive

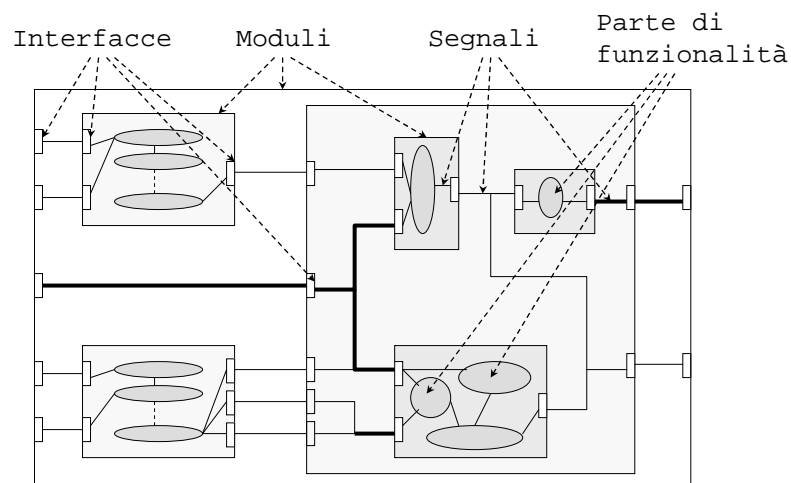
Generalità



➤ Esigenze di progetto

- Struttura di un progetto (specifica)
 - In generale, un sistema hardware è composto da una gerarchia di sottosistemi attivi in parallelo
 - Ogni sottosistema è caratterizzato da
 - Interfaccia
 - » Specifica quello che entra ed esce dal sottosistema
 - Funzionalità
 - » Specifica la relazione tra i dati in ingresso ed i dati in uscita al sottosistema
- Verifica del progetto
 - Un sistema descritto deve poter essere verificato
 - Sia funzionalmente sia temporalmente

Struttura di un progetto



Elementi Base



- Modulo
 - Interfaccia
 - Funzionalità
 - 3 Livelli di astrazione possibili
 - Strutturale o Gate-Level
 - RTL o Data-Flow
 - Algoritmico o Behavioural
- Gerarchia

Interfaccia



- Dichiarazione della entità (*Entity Declaration*)
 - Descrive l'interfaccia di ingresso/uscita di un modulo

```

    Nome del modulo
    Nome delle porte
    entity Full_Adder is
      port (a, b, cin:
            sum, carry:
      end;
    in std_logic;
    out std_logic);
    Modo della porta (direzione)  Tipo della porta
```

Funzionalità

- Dichiarazione della architettura (*Architecture Declaration*)
 - Descrive come il opera modulo
 - Sono possibili più architetture per ogni entità

Nome delle architettura Nome del modulo

```
architecture arch1 of full_adder is
begin
    sum    <= a xor b xor cin;
    carry <= (a and b) or (cin and (a or b));
end;
```

Istruzioni VHDL Concorrenti

Funzionalità: Livello Strutturale

- Livello di astrazione più basso
- La funzionalità è idealmente espressa mediante un grafo
 - Nodi: rappresentano degli elementi logici
 - Funzionalità semplici o complesse
 - Archi: le connessioni tra gli elementi
- La descrizione è attraverso una *netlist*

Nome delle modulo

```
U0: Full_adder port map (a0, b0, Cin, s0, c0);
U1: Full_adder port map (a1, b1, c0, s1, c1);
...
```

Nome delle istanza Connessione tra segnali e porte

Funzionalità: Livello RTL



- Livello di astrazione intermedio
 - RTL: Register Transfer Level
- La funzionalità è idealmente espressa da un flusso di dati che, propagandosi nel circuito da un registro ad un altro, subisce delle trasformazioni
 - Il trasferimento tra i registri ne giustifica il nome
 - Un sotto circuito può anche non contenere registri

```
sum    <= a xor b xor cin;  
carry <= (a and b) or (cin and (a or b));
```

Funzionalità: Livello Algoritmico



- Livello di astrazione più elevato
- La funzionalità è espressa mediante uno o più algoritmi
 - Non deve necessariamente riflettere gli aspetti implementativi del progetto
- Le descrizioni a livello algoritmico sono supportate dalla istruzione **process**

```
count: process (x)  
    variable cnt: integer := -1;  
begin  
    cnt := cnt + 1;  
end process;
```

Funzionalità: Livelli Misti



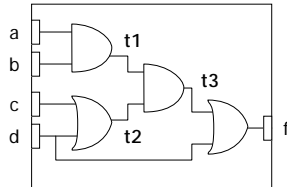
- Una specifica può essere realizzata utilizzando contemporaneamente tutti i livelli di astrazione
 - Tipicamente si utilizzano i livelli RTL e Strutturale
 - Benché non considerato nel corso, una specifica a livello algoritmico può essere combinata con altri livelli d'astrazione
- Direttive di progetto:
 - I moduli elementari (soluzioni effettive) sono descritti a livello RTL e/o Algoritmico
 - I moduli che definiscono la gerarchia sono descritti a livello strutturale
 - Nota: il modulo a livello gerarchico più elevato è denominato *top-entity*.

Segnali



- I segnali rappresentano i valori dei dati su linee fisiche di un circuito
- I segnali possono essere
 - Segnali di interfaccia
 - utilizzati dai moduli per comunicare il loro contenuto con l'esterno
 - Segnali interni
 - Utilizzati nel modulo per realizzare la funzionalità desiderata

Segnali



- I segnali utilizzati in un modulo sono definiti a livello di architettura e non sono visibili al di fuori del modulo

```
architecture uno of esempio is  
    signal t1, t2, t3: std_logic;  
begin
```

Segnali



- Il VHDL dispone di un numero elevato di tipi
- Per la sintesi e la simulazione si preferisce utilizzare i tipi definiti nella libreria `std_logic_1164` (IEEE)
 - Deve essere sempre dichiarata

```
library IEEE;  
use IEEE.std_logic_1164.ALL;
```

- I tipi definiti nella libreria `std_logic_1164` utilizzano un sistema logico a 9 valori
 - Consente di contemplare situazioni come, l'alta impedenza e l'indeterminazione.
 - Consente di definire l'interazione tra valori differenti
 - Funzione di risoluzione

Segnali



► Funzione di risoluzione

```

-----
-- Resolution function
-----
CONSTANT resolution_table : stdlogic_table := (
--
-- | U   X   0   1   Z   W   L   H   D   |
--
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', '1' ), -- 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'Z' ), -- Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'W' ), -- W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'L' ), -- L |
( 'U', 'X', '0', '1', 'H', 'W', 'H', 'H', 'H' ), -- H |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'D' ) -- D |
);

```

Non Inizializzato Non Noto (0 o 1) Alta Impedenza Don't care

Segnali



► Scalari

- std_logic
 - signal a,b: std_logic;
- Costanti utilizzano un singolo apice ('0')

► Vettori (BUS)

- std_logic_vector
 - signal a: std_logic_vector(0 to 7);
 - 0 1 2 3 4 5 6 7 (notazione *big endian*)
 - signal b: std_logic_vector(7 downto 0);
 - 7 6 5 4 3 2 1 0 (notazione *little endian*)
 - signal c: std_logic_vector(15 to 22);
 - 15 16 17 18 19 20 21 22
- Costanti utilizzano i doppi apici ("00000000")
 - Utilizzo di _ per aumentare la leggibilità ("0000_0000")

Segnali



► Operazioni sui segnali

- Logiche: `and`, `or`, `not`, `xor`, (`nxor` – VHDL93)
 - Operazioni bit a bit
- Estrazione: `nome(indice1 to/downto indice2)`
 - VHDL87: L'ordine di estrazione è quello imposto dalla dichiarazione del vettore
 - Un errore genera una stringa nulla
 - VHDL93: L'ordine di estrazione è generico
 - Sia a uno `std_logic_vector(0 to 7)`:
 - `a(2 to 3)` seleziona i bit 2 e 3
 - `a(4)` seleziona il bit 4
 - `a(3 downto 0)` non è ammesso
- Concatenamento: `&`

Segnali



► Operazioni sui segnali (continua)

- Assegnamento: `<=`
 - Gli operandi devono essere dello stesso tipo
 - Per assegnare un valore costante non si conosce la dimensione, si può utilizzare come operando sinistro il costrutto `(others => costante)`
 - Prova `<= (others => 'x')`
- Aritmetiche: `+`
 - Operazione di somma
 - Attenzione a considerare che può generare un bit in più
- Relazionali: `<` `<=` `=` `/=` `>=` `>`
 - Nota: l'operatore di confronto `<=` e quello di assegnamento ad un segnale sono identici
 - Il significato dipende dal contesto

► Esempio di estrazione e concatenamento

- signal BUS: std_logic_vector(0 to 13);
- BUS <= "1111_0000_1010_10";
- (BUS(13)&BUS(12))&BUS(0 to 11)) and BUS(0 to 13)
 - BUS(0 to 11)
→ "1111_0000_1010"
 - BUS(13)&BUS(12)
→ "01"
 - (BUS(13)&BUS(12))&BUS(0 to 11))
→ "01_1111_0000_1010"
 - (BUS(13)&BUS(12))&BUS(0 to 11)) and BUS(0 to 13)
→ "01_1100_0000_1010"

► Esempio di somma

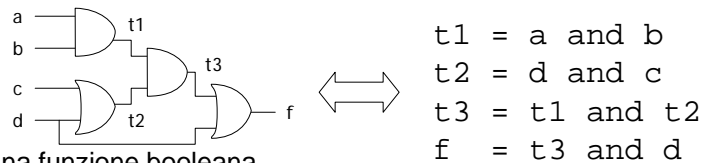
- Siano Somma, A e B dei std_logic_vector(3 downto 0) e Cin e Cout degli std_logic
- Utilizziamo un vettore temporaneo Temp per svolgere l'operazione di somma
 - Sia Temp un std_logic_vector(4 downto 0)
- Allora:
Temp <= (A(3)&A)+(B(3)&B)+("0000"&Cin);
Somma<= Temp(3 downto 0);
Cout <= Temp(4);

Reti Combinatorie



► Una rete combinatoria è descritta da

- Una espressione logica
 - Nota: una rappresentazione circuitale è un modo speciale per rappresentare le espressioni



- Una funzione booleana
 - Tabella della verità o delle implicazioni anche non completamente specificata

Reti Combinatorie



► Espressione logica

- Una espressione logica è tradotta in una corrispondente istruzione VHDL sostituendo:
 - Alle variabili i segnali
 - Alle costanti i valori costanti tra singoli apici (o doppi apici se vettori)
 - Agli operatori logici gli operatori logici VHDL

$$f = a' (b(c+d) + c')$$



– Nota: a, b, c, d e f possono essere anche vettori

```
f <= not(a) and (b and (c or d) or not(c))
```

Reti Combinatorie



► Funzione Booleana

- Una funzione booleana è tradotta nella corrispondente istruzione VHDL utilizzando un assegnamento condizionale concorrente

Quando $a=1$ le uscite valgono $z1=z2=0$ mentre $z3=1$. Se $a=0$, $b=1$ e $c=0$ allora $z3=0$. Quando $a=0$ e $c=1$ allora $z1=z2=z3=1$.



```
with in select          -- in è a&b&c
  out <= "001" when "1--",
        "--0" when "010",
        "111" when "0-1",
        "---" when others;
```

Reti Combinatorie



► Considerazioni sul costrutto **select**

- Anche se sono state elencate tutte le configurazioni di ingresso la clausola **when others** deve essere esplicitata
 - In realtà non si esplicitano mai tutte le possibili configurazioni poiché si utilizza una logica a 9 valori
 - Quindi, considerando per esempio un singolo segnale, indicando cosa fare per 0 e per 1 non si dice come agire per U, X, Z, H, L, W, D.
 - Si utilizzi 'x' o 'D'
- I valori da assegnare possono essere sia costanti sia derivati da altri segnali

```
• with A select
  Z<= B when '0', "01"&c when '1',
      "xxx" when others;
```

Reti Combinatorie



➤ Esempio

```
entity mux is
    generic ( NumBit: integer);
    port(s0, s1      : in std_logic;
          i0, i1, i2, i3: in std_logic_vector(0 to NumBit);
          O          : out std_logic_vector(0 to NumBit));
end;

architecture prova of mux is
    signal sel: std_logic_vector(0 to 1);
begin
    sel <= s0&s1;
    with sel select
        O <= i0 when "00",
              i1 when "01",
              i2 when "10",
              i3 when "11",
              (others => 'x') when others;
end;
```

Reti Combinatorie



- Tutte le istruzioni contenute in una architettura sono eseguite parallelamente
 - Paradigma hardware.
- Le istruzioni sono eseguite in “tempo zero”: la simulazione è solo funzionale
- La simulazione segue il modello di esecuzione “event driven”

```
Q <= R nor nQ;
nQ <= S nor Q;
```

Start: R='0', S='0', Q='1', nQ='0'
passo1: R='1', S='0', Q='1', nQ='0' '0' è schedulato per Q
passo2: R='1', S='0', Q='0', nQ='0' '1' è schedulato per nQ
passo3: R='1', S='0', Q='0', nQ='1' nessun evento schedulato
passo2: R='0', S='0', Q='0', nQ='0' nessun evento schedulato

Macchine a stati



- I sistemi sequenziali sono
 - Sincroni
 - Lo stato è aggiornato quando il segnale di clock cambia
 - Asincroni
 - Lo stato è aggiornato non appena quest'ultimo cambia
 - Non è presente il clock
- Nella progettazione digitale ci si riferisce spesso a comportamenti sincroni
 - Si evitano problemi dovuti alla differenza di ritardo nei percorsi combinatori
 - Le alee dinamiche e statiche non hanno effetto
 - Più semplici i problemi di interfacciamento tra moduli
 - Maggiore indipendenza dalla temperatura di funzionamento, dalla tensione e dal processo
 - I tool di sintesi non gestiscono bene l'asincronicità

Macchine a stati



- Nella progettazione digitale ci si riferisce a comportamenti asincroni solo in casi particolari
 - Domini di clock asincroni
 - Problema abbastanza comune
- Progettare sistemi asincroni è più complesso ma produce strutture con prestazioni maggiori e consumi di potenza minori
 - *“solo gli idioti o gli esperti progettano riferendosi alla asincronicità”*

Macchine a stati



➤ Macchina di Moore

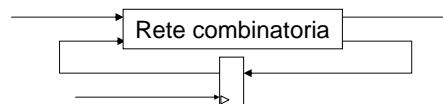
- L'uscita è funzione del solo stato corrente
 - A volte, l'uscita è direttamente lo stato corrente
 - Es: contatori privi di rete di transcodifica

➤ Macchina di Mealy

- L'uscita è funzione dello stato corrente e dell'ingresso

➤ Struttura generale di una macchina a stati

- La rete combinatoria è scomposta in due entità: funzione stato prossimo e funzione d'uscita



Macchine a stati



- Descrivere una macchina a stati significa descrivere l'attività, parallela, delle reti combinatorie e dei registri
- A questo scopo si utilizza l'istruzione `process`

```
Nome del process      Sensitivity list
count : process (x)
  variable cnt: integer := -1;
begin
  cnt := cnt + 1;
end process;
Corpo del process      Dichiarazioni
```


Macchine a stati: process



► Caratteristiche di un process

- Una istruzione `process` può comparire nel corpo di una architettura di un modulo
- Il `process` è eseguito parallelamente a qualunque altra istruzione presente nella architettura di un modulo
- Le istruzioni contenute in un `process` sono eseguite in sequenza (dalla prima all'ultima) e solo in corrispondenza di una variazione (evento) di un segnale specificato nella `sensitivity list`.
 - Il processo si sospende dopo l'esecuzione dell'ultima istruzione.
 - Il processo risvegliato viene eseguito nuovamente dalla prima all'ultima istruzione

Macchine a stati: process



► Caratteristiche di un process (continua)

- Il `process` può contenere istruzioni sequenziali come quelle tipicamente software
 - Le istruzioni sequenziali sono spesso più capaci di quelle parallele ma, di sovente, non hanno una corrispondente implementazione hardware

Macchine a stati: process



➤ Caratteristiche di un process (continua)

– Segnali

- I `signal` sono dichiarati solo a livello di architettura
 - assegnamento: `<=`
- I segnali dichiarati in un modulo, sia di interfaccia sia interni, sono visibili all'interno di un `process`
- I segnali devono essere usati per passare i dati da e verso un `process`
- Un segnale aggiornato da un `process` riceve il suo valore solo alla terminazione del `process` stesso

Macchine a stati: process



➤ Caratteristiche di un process (continua)

– Variabili

- Le `variable` sono dichiarate solo a livello di processo
 - assegnamento: `:=`
- Un assegnamento ad una variabile cambia istantaneamente il suo contenuto

Macchine a stati: process



► Caratteristiche di un process (continua)

- Attenzione: Contrariamente agli assegnamenti esterni al process, quelli interni vengono valutati solo in corrispondenza di una variazione nella sensitivity list.

```
                                Sensitivity list
                                ↙
Mio_xor : process (c)
begin
    a <= b xor c;
end process; ↑
```

L'assegnamento viene aggiornato solo quando cambia c; la funzionalità non rispecchia quanto previsto dallo schema di calcolo per a.

Istruzioni Sequenziali



► Costrutto if-then-else

- Attenzione: l'ordine di valutazione delle condizioni costituisce una direttiva di implementazione
 - Introduce priorità e possibili sbilanciamenti nei ritardi di propagazione

```
if condizione_1 then
    istruzioni_1
[elsif condizione_2
    istruzioni_2]
...
[else
    istruzioni_n]
end if;
```

Istruzioni Sequenziali



► Costrutto case

- Confronto di un segnale con valori costanti

```
case segnale is
  when costante_1 =>
    istruzioni_1
  ...
  when costante_n =>
    istruzioni_n
  when others =>
    istruzioni_altro
end case;
```

Istruzioni Sequenziali



► Costrutto for

```
for identificatore in intervallo loop
  corpo del ciclo
end loop;
```

Intervallo:
primo to ultimo
ultimo downto primo
array'range

- Identificatore è una variabile di controllo
- a'range, se a è un vettore std_logic_vector(1 to 8), restituisce l'intervallo 1 to 8
 - Consente di accedere a tutti gli elementi di un array senza conoscerne le dimensioni

Istruzioni Sequenziali



- Esempio costruito for
 - Inversione dei bit nei byte (8 byte)

```
process(clock, reset)
begin
    for i in 7 downto 0 loop
        temp(56+i) <= data(63-i);
        temp(48+i) <= data(55-i);
        temp(40+i) <= data(47-i);
        temp(32+i) <= data(39-i);
        temp(24+i) <= data(31-i);
        temp(16+i) <= data(23-i);
        temp(8+i)  <= data(15-i);
        temp(0+i)  <= data(7-i);
    end loop;
end process;
```

Macchine a stati: esempi



- Esempio FF tipo D che commuta sul fronte di salita

```
entity D_FF is
    port( D : in std_logic;
          Clk: in std_logic;
          Q  : out std_logic);
end;
architecture FF_salita of DD_F is
begin
    ff: process(clk)
    begin
        if (clk'event and clk=1) then
            Q <= D;
        end if;
    end process;
end;
```

Macchine a stati: esempi

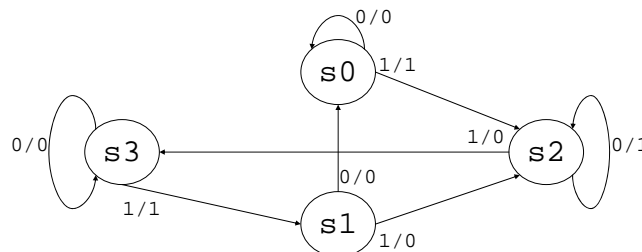
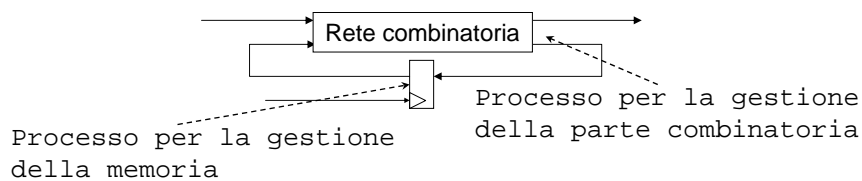
► Esempio di contatore binario naturale

- Istanza un sommatore (non efficiente)

```
entity Counter is
  generic (N : integer := 10)
  port (clk, rst: in std_logic;
        cnt : out std_logic_vector(0 to N));
end;
architecture count of counter is
begin
  ff: process(clk)
  begin
    if (clk'event and clk=1) then
      if (rst = 0)
        cnt <= (others => '0');
      else
        cnt <= cnt + 1;
      end if;
    end process;
  end;
end;
```

Macchine a stati: esempi

► Macchina di Mealy



Macchine a stati: esempi



► Macchina di Mealy

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

entity MEALY is
    port(X, Clock, Reset: in std_logic;
          Z: out std_logic);
end;

architecture BEHAVIOR of MEALY is
    type state_type is (s0, s1, s2, s3);
    signal CURRENT_STATE, NEXT_STATE: state_type;
begin
    -- Process to hold combinational logic
    ...
    -- Process to hold synchronous elements (flip-flops)
    ...
end;
```

Macchine a stati: esempi



```
-- Process to hold combinational logic
combin: process (CURRENT_STATE, X)
begin
    case CURRENT_STATE is
        when s0 =>
            if X = '0' then
                Z <= '0';
                NEXT_STATE <= s0;
            else
                Z <= '1';
                NEXT_STATE <= s2;
            end if;
        when s1 =>
            if X = '0' then
                Z <= '0';
                NEXT_STATE <= s0;
            else
                Z <= '0';
                NEXT_STATE <= s2;
            end if;
        when s2 =>
            if X = '0' then
                Z <= '1';
                NEXT_STATE <= s2;
            else
                Z <= '0';
                NEXT_STATE <= s3;
            end if;
        when s3 =>
            if X = '0' then
                Z <= '0';
                NEXT_STATE <= s3;
            else
                Z <= '1';
                NEXT_STATE <= s1;
            end if;
    end case;
end process;
```

Macchine a stati: esempi



```
-- Process to hold synchronous elements (flip-flops)
memory: process (Clock, Reset)
begin
    if(Clock'EVENT and Clock='1') then
        if(Reset = '0') then -- active low
            CURRENT_STATE <= S0;
        else
            CURRENT_STATE <= NEXT_STATE;
        end if;
    end if;
end process;
```

Macchine a stati: esempi



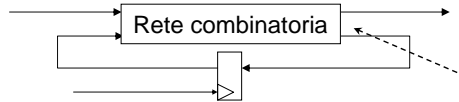
- Osservazione: La codifica dello stato viene fatta dallo strumento di sintesi
 - Codifica binaria a partire dal primo stato della lista
- È possibile forzare una codifica

```
architecture BEHAVIOR of MEALY is
    type state_type is (s0, s1, s2, s3);
    constant s0: std_logic_vector(0 to 3):="0001";
    constant s1: std_logic_vector(0 to 3):="0010";
    constant s2: std_logic_vector(0 to 3):="0100";
    constant s3: std_logic_vector(0 to 3):="1000";
    signal CURRENT_STATE, NEXT_STATE: state_type;
begin
    ...
    ...
end BEHAVIOR;
```

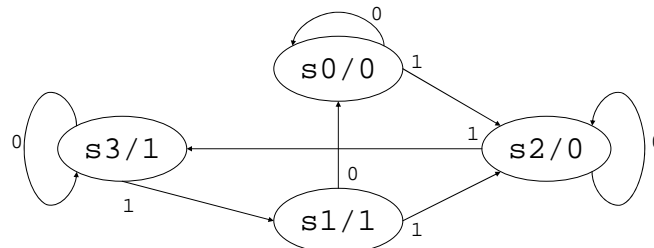

Macchine a stati: esempi



► Macchina di Moore



Rispetto al caso precedente cambia solo il processo per la gestione della parte combinatoria



© Fabio Salice

VHDL

49

Macchine a stati: esempi



```
-- Process to hold combinational logic
combin: process (CURRENT_STATE, X)
begin
  case CURRENT_STATE is
    when s0 =>
      Z <= '0';
      if X = '0' then
        NEXT_STATE <= s0;
      else
        NEXT_STATE <= s2;
      end if;
    when s1 =>
      Z <= '1';
      if X = '0' then
        NEXT_STATE <= s0;
      else
        NEXT_STATE <= s2;
      end if;
    when s2 =>
      Z <= '0';
      if X = '0' then
        NEXT_STATE <= s2;
      else
        NEXT_STATE <= s3;
      end if;
    when s3 =>
      Z <= '0';
      if X = '0' then
        NEXT_STATE <= s3;
      else
        NEXT_STATE <= s1;
      end if;
  end case;
end process;
```

© Fabio Salice

VHDL

50

Macchine a stati



► Consigli per una buona macchina a stati

- Reset sincrono
- Non avere stati non raggiungibili
 - Evitare che si creino dei *dead-loop*
 - vengono raggiunti per guasti transitori
- Ingressi registrati
 - Per evitare che si presentino dei cicli combinatori difficili da individuare
- Usare nomi simbolici per gli stati
 - Ottima politica per strumenti di sintesi che consentono di applicare, mediante comando, delle politiche di codifica dello stato

Macchine a stati



► Consigli per una buona codifica dello stato

- Minimizzare il numero dei flip-flop può non essere una buona politica
 - Un numero ridotto di FF può avere un impatto negativo sulla funzione d'uscita
- La codifica one-hot permette di ottenere una funzione stato-prossimo veloce e semplice
 - Utilizzare la codifica one-hot nelle FPGA poiché ne esistono molti e, se non utilizzati, sono persi.
 - Conduce ad un utilizzo efficiente delle CLB e aumenta le prestazioni dell'intero sistema.
- La codifica one-hot garantisce distanza di hamming due tra gli stati
- Per ridurre la potenza si potrebbe utilizzare una codifica gray tra gli stati adiacenti
 - Di difficile applicazione se non per i contatori

Gerarchia



- La gerarchia viene descritta mediante una o più rappresentazioni funzionali a livello strutturale
 - Alcuni/tutti dei moduli realizzati sono raggruppati in un unico modulo; quest'ultimo individua il modulo ad essi gerarchicamente superiore
 - I moduli utilizzati sono chiamati componenti
- Tre aspetti relativi ai componenti utilizzati
 - Dichiarazione
 - Definisce i componenti che dovranno essere collegati fra loro
 - Configurazione
 - Associa alla interfaccia l'architettura a cui si intende riferirsi
 - Solo nel caso che un modulo abbia più di una architettura
 - Istanziamento
 - Creazione di una copia del modulo

Gerarchia



- Dichiarazione
 - Locata nella parte dichiarativa della architettura

```

                                Nome del modulo
                                ↘
component Full_Adder is
    port (a, b, cin :
          sum, carry:
end component;
                                Nome delle porte
                                ↘
                                in std_logic;
                                out std_logic);
                                ↗
                                Tipo della porta
                                ↗
                                Modo della porta (direzione)

In blue le differenze rispetto a entity
```

Gerarchia



➤ Istanziamento

- Locata nel corpo della architettura

➤ Port map posizionale

Nome del modulo (entità)
↓
U0: Full_Adder **port map** (a0, b0, Cin, s0, c0);
U1: Full_Adder **port map** (a1, b1, c0, s1, c1);
...
Nome della istanza Connessione tra segnali e porte

➤ Port map nominale

U0: Full_Adder **port map**
 (a=>a0,b=>b0,cin=>Cin,sum=>s0,carry=>c0);

Gerarchia



➤ Istanziamento diretta (VHDL '93)

- Dichiarazione e istanziamento sono fatti contemporaneamente
 - Codice più semplice e chiaro ma non compatibile con tutte le versioni degli strumenti di sintesi
 - Preferito per net-list semplici

Nome della istanza Nome della architettura
↓ ↓
U0: **entity** WORK.And2(ex1) **port map** (a, b);
...
Libreria di lavoro corrente Nome del modulo
(direttorio di lavoro corrente)

Gerarchia



- Se non tutti gli ingressi e/o le uscite di un componente sono utilizzate:
 - Associare agli ingressi un valore costante
 - Utilizzare la parola riserva **open**
- Utile per la realizzazione di funzionalità parametriche

Gerarchia



➤ Esempio

```
entity esempio is
  port(  x,y : in std_logic; inv: in std_logic := '0';
        a, o; out std_logic);
end entity esempio

architecture ex di esempio is
begin
  a <= (y and (x xor inv)) or (inv and not y);
  o <= (not x and (y xor inv)) or (x and not inv);
end architecture ex;

...

Uo: entity WORK.esempio(ex) port map (x,y,open,a,open);
```

Strutture Parametriche



- La parametrizzazione consente di realizzare modelli generali
- Due aspetti
 - Generalizzare l'interfaccia del modulo; tale informazione si propaga all'interno del modulo
 - Costrutto `generic`
 - Generalizzare l'architettura del modulo
 - Costrutto `generate`
 - Il costrutto `generate` ha l'obiettivo meno ambizioso di replicare moduli dello stesso tipo; tale proprietà può essere sfruttata vantaggiosamente per la realizzazione di architetture parametriche

Strutture Parametriche



➤ Costrutto `generic`

```
entity Adder is
  generic (N : integer);
  port (a, b: in std_logic_vector(0 to N-1);
        cin : in std_logic;
        sum : out std_logic_vector(0 to N-1)
        cout: out std_logic);
end;
```

Nome del parametro Tipo del parametro

Porte parametriche

Strutture Parametriche



► Esempio

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

entity shifter_left_N_Dimop is
  generic ( N      : integer;
            Dim_sh : integer );
  port ( in_sh  : in std_logic_vector (0 to Dim_sh - 1);
        out_sh : out std_logic_vector(0 to Dim_sh - 1 + N));
end shifter_left_N_Dimop;

architecture rtl of shifter_left_N_Dimop is
  signal zeros: std_logic_vector (0 to Dim_sh);
  signal zero: std_logic;
begin
  zero    <='0';
  zeros   <=(others=>zero);
  out_sh  <= in_sh & zeros(0 to N-1);
end rtl;
```

Strutture Parametriche



- Dichiarazione di un componente con generic
 - Equivalente alla dichiarazione priva di generic
- Istanziamento di un componente con generic
 - Oltre al port map si svolge una generic map che ha lo scopo di associare al parametro del componente un valore costante o un ulteriore parametro derivato dall'architettura in cui è istanziato

U1: Adder

```
generic map (10 => N)
port map (INa, INb, cin, OUT, cout);
```

Strutture Parametriche



► Esempio di dichiarazione

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

entity PROVA is
    generic ( A, B, M : integer);
    port (in_P : in  std_logic_vector (0 to A-1);
          out_P: out std_logic_vector (0 to B-1) );
end PROVA;

architecture rtl of PROVA is
    component COMPONENTE_PARAMETRICO is
        generic ( N : integer;
                  C : integer );
        port (in_CP  : in std_logic_vector (0 to C - 1);
              out_CP : out std_logic_vector(0 to C - 1 + N));
    end component COMPONENTE_PARAMETRICO;
    ...
end rtl;
```

Strutture Parametriche



► Esempio di istanziazione

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

entity PROVA is
    generic ( A, B, M : integer);
    port (in_P : in  std_logic_vector (0 to A-1);
          out_P: out std_logic_vector (0 to B-1) );
end PROVA;

architecture rtl of PROVA is
    signal X: std_logic_vector (0 to A-1+M);
    ...
begin
    SH1: COMPONENTE_PARAMETRICO
        generic map (C => A, N => M )
        port map (in_P,X);
    ...
end rtl;
```


Strutture Parametriche



► Costrutto **generate**

- Utilizzato durante l'istanziamento di moduli per realizzare architetture con una struttura ripetitiva
 - Es: architetture bit-wise
- Generazione condizionale

```
Label: if condizione generate  
      [regione dichiarativa (VHDL'93)]  
      begin  
        {istruzioni concorrenti}  
      end generate [label];
```

Strutture Parametriche



► Costrutto **generate** (cont.)

- Generazione iterativa

```
Label: for indice in intervallo generate  
      [regione dichiarativa (VHDL'93)]  
      begin  
        {istruzioni concorrenti}  
      end generate [label];
```

```
Intervallo:  
  primo to ultimo  
  ultimo downto primo  
  array'range
```

Strutture Parametriche



► Esempio di registro a scorrimento SIPO a 8 bit

```
L : for i in 0 to 7 generate
begin
  R0: if (i=0) generate
begin
  U1:component DFF
    port map(CLK=>ck,D=>din, Q=>dout(i));
end generate;
  RI: if (i/=0) generate
begin
  U1:component DFF
    port map(CLK=>ck,D=>dout(i-1), Q=>dout(i));
end generate;
end generate;
```

Strutture Parametriche



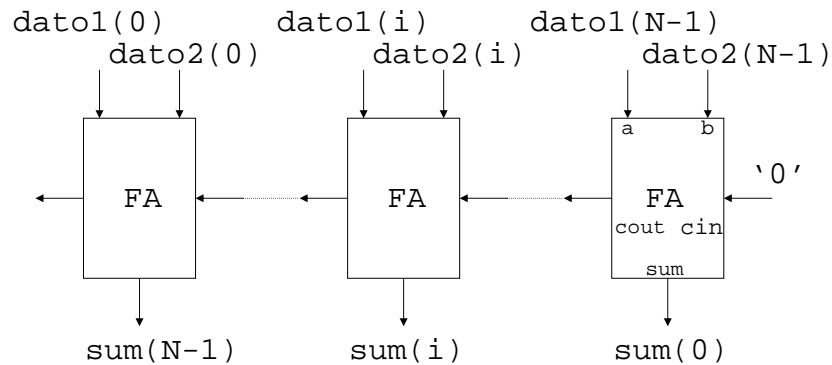
► Funzioni e operatori per il dimensionamento parametrico (e non solo)

- `abs (...)`
 - Valore assoluto – `abs(A-B)`; applicabile a qualunque tipo numerico
- `mod`
 - $A = B * N + (A \bmod B)$; applicabile al tipo integer
- `rem`
 - $A = (A/B) * B + (A \bmod B)$; applicabile al tipo integer
- `*` ; `/`
- `**`
 - Esponenziale - N^{**2} ; l'operando a sinistra deve essere integer o virgola mobile, l'operando a destra (esponente) deve essere integer

Esempio

► Sommatore Parametrico (Ripple Carry)

- Schema di principio (il bit 0 è MSB)



Esempio

► Sommatore Parametrico (Ripple Carry)

- Full-Adder

```
library IEEE;
use IEEE.std_logic_1164.all;
entity FA is
    port (a, b, cin    : in std_logic;
          sum, cout    : out std_logic);
end FA;

-- description of full adder
architecture rtl of FA is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or (cin and a) or (cin and b);
end rtl;
```

Esempio



► Sommatore Parametrico (Ripple Carry)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Adder_ripple is
    generic (N : integer); -- N bit number
    port (dato1 : in std_logic_vector (0 to N-1);
          dato2 : in std_logic_vector (0 to N-1);
          somma : out std_logic_vector (0 to N-1));
end Adder_ripple;

-- description of full adder
architecture rtl of Adder_ripple is
    component FA is
        port (a, b, cin : in std_logic;
              sum, cout : out std_logic);
    end component FA;
    ...
end rtl;
```

Esempio



► Sommatore Parametrico (Ripple Carry)

```
signal carry: std_logic_vector (0 to N);
signal sum: std_logic_vector (0 to N-1);
signal zero: std_logic;

begin
    zero <= '0';
    FA_f: FA port map
        (dato1(N-1), dato2(N-1), zero, sum(N-1), carry(N-1));
    adder_ripple:
        for i in N-2 downto 0 generate
            FA_i: FA port map
                (dato1(i), dato2(i), carry(i+1), sum(i), carry(i));
        end generate;
    somma <= sum;
end rtl;
```

VHDL e sintesi



- I tool di sintesi sono privi di intelligenza e non esistono termini riservati per specificare se un modello è sequenziale o combinatorio
- Il problema fondamentale nella sintesi modelli VHDL è di assicurarsi che quanto prodotto corrisponda alle reali aspettative
- Uno degli errori più probabili è quello relativo alla istanziazione non desiderata di Flip-Flop e Latch
 - In particolare, un FF o un latch è istanziato se una variabile o un segnale non cambia valore per qualche periodo di tempo

VHDL e sintesi



- In linea di principio, i processi possono produrre elementi di memoria indesiderati quando:
 - È attivato da una *sensitivity list* che non considera tutte le variabili o i segnali che ne aggiornano altri e non sono prodotti nel *process*
 - Alcuni percorsi assegnano valori a variabili o segnali mentre altri no
- In pratica, i tool di sintesi riconoscono un insieme abbastanza ridotto di strutture.
 - Ci si riferisce a quest'ultime

Istanza di Latch



```
U0: process (ctrl, A) is
begin
    if (Ctrl = '1') then
        Z <= A;
    end if;
end process u0;
```

- Il processo è attivato su A e Ctrl;
- Il valore di Z
 - È aggiornato quando il segnale Ctrl assume valore 1
 - È inalterato quando Ctrl assume valore 0
- Deve essere inferito un *latch* per conservare il valore di Z

Istanza di Latch



```
U0: process (Ctrl, A) is
Begin
    if (Ctrl = '1') then
        W <= A;
    else
        W <= W;
    end if;
end process u0;
Z <= W; -- Z è out
```

- Il processo è attivato su A e Ctrl;
- Il valore di w
 - È aggiornato con A quando il segnale Ctrl assume valore 1
 - È inalterato quando Ctrl assume valore 0
- Deve essere inferito un *latch* per conservare il valore di w

Istanza di Latch



```
process (ctrl, a, b) is
begin
  case ctrl is
    when "00" => z <= a;
    when "01" => z <= b;
    when others => null;
  end case;
end process;
```

- Il processo è attivato su a, b e ctrl;
- Il valore di z
 - È aggiornato con quando il segnale Ctrl assume valore "00" oppure "01"
 - È inalterato quando Ctrl assume le altre configurazioni
- Deve essere inferito un *latch* per conservare il valore di z

Istanza di FF



```
process (clk) is
begin
  if (clk='1' and clk'event) then
    z <= a;
  end if;
end process;
```

- Il processo è attivato su clk;
- Il valore di z
 - È aggiornato con a quando il segnale clk assume valore '1' ed è rilevato un fronte (fronte di salita)
 - È inalterato quando non si rileva un fronte di salita su clk
- Deve essere inferito un *FF* per conservare il valore di z

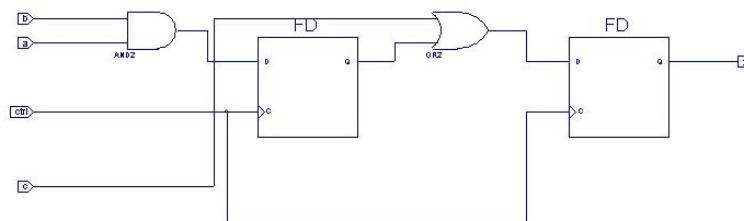
Istanza di FF



```
process (clk) is
begin
  if (clk='1' and clk'event) then
    w <= a and b;
    z <= w or c;
  end if;
end process;
```

- Il processo è attivato su `clk`
- Sul fronte di salita di `clk`, i valori di `w` e `z` vengono riaggiornati con `a and b` e `wold or c`
 - Si ricordi che i segnali vengono aggiornati alla fine del *process*
- Devono essere inferiti due *FF* (pipeline)

Istanza di FF



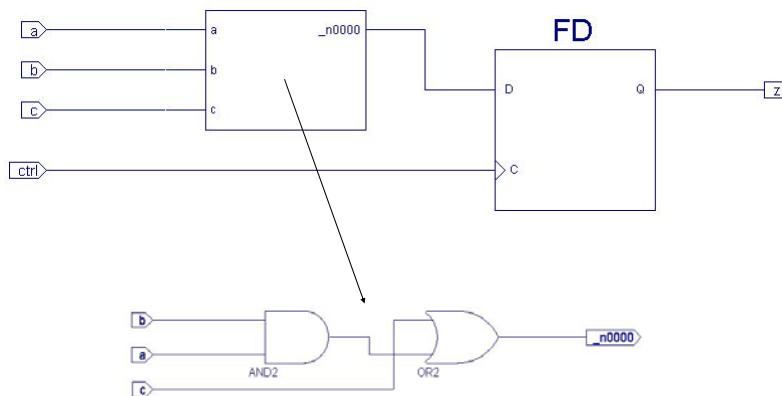
Istanza di FF



```
process (clk) is
  variable w : std_logic;
begin
  if (clk='1' and clk'event) then
    w := a and b;
    z <= w or c;
  end if;
end process;
```

- Il processo è attivato su `clk`
- Sul fronte di salita di `clk`, il valore di `z` viene riaggiornato con w_{new} or `c` dove $w_{new} = a \text{ and } b$
 - Si ricordi che **solo** i segnali vengono aggiornati alla fine del *process*
- Deve essere inferito un *FF*

Istanza di FF



Reti combinatorie



```
u0: process (a) is
begin
    z := a or b;
end process u0;
```

- Il processo è sensibile ad a ma non a b
- Il risultato dipende dallo strumento
 - O viene istanziata una coppia di FF: uno sensibile al fronte di salita ed uno sensibile al fronte di discesa di a.
 - Oltre ad una complessa rete combinatoria che genera z
 - Oppure viene sia segnalato un warning in fase di analisi HDL del modulo sia corretto aggiungendo implicitamente b nella sensitivity list.

Reti combinatorie

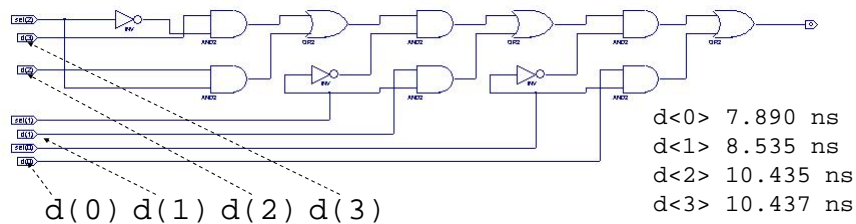


- Costrutti IF e CASE
 - Il costrutto IF crea una logica con priorità sui segnali coinvolti
 - L'IF può contenere un insieme di differenti predicati
 - Il costrutto CASE implementa una logica parallela
 - Il CASE è agisce sulla base di una espressione comune
 - Nel caso l'IF agisca su una base di valutazione riconoscibile come comune, il risultato della sintesi può coincidere con quello prodotto dal CASE

Reti combinatorie



```
process (sel, d)
begin
    if (sel(0) = '1') then o <= d(0);
    elsif (sel(1) = '1') then o <= d(1);
    elsif (sel(2) = '1') then o <= d(2);
    else o <= d(3);
    end if;
end process;
```



© Fabio Salice

VHDL

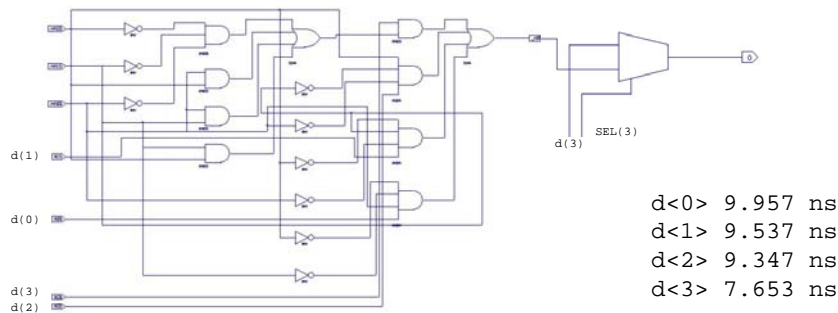
85

Reti combinatorie



```
process (sel, d)
begin
    if (sel="0001") then o <= d(0);
    elsif (sel="0010") then o <= d(1);
    elsif (sel="0100") then o <= d(2);
    else o <= d(3);
    end if;
end process;
```

```
process (sel, d)
begin
    case sel is
        when "0001" => o <= d(0);
        when "0010" => o <= d(1);
        when "0100" => o <= d(2);
        when others => o <= d(3);
    end case;
end process;
```



© Fabio Salice

VHDL

86

Regole di sintesi



- Reti combinatorie
 - Sensitivity list: tutti i segnali primari nella parte destra degli assegnamenti e quelli utilizzati nei predicati (IF e CASE)
 - Branch: devono essere completi
- Latch
 - Sensitivity list: tutti i segnali primari nella parte destra degli assegnamenti e quelli utilizzati nei predicati (IF e CASE)
 - Branch: non completi
- Flip-flop
 - Sensitivity list: Clock ed i segnali asincroni di set e reset
 - Si utilizzano costrutti per rilevare il fronte
 - Branch: non completi

Note (VHDL 93)



- Operatore `rol`
 - Rotazione sinistra (es: `Sreg <= Sreg rol 2;`)
- Operatore `ror`
 - Rotazione destra (es: `Sreg <= Sreg ror 2;`)
- Operatore `sla`
 - Traslazione aritmetica a sinistra (es: `D <= D sla 8;`)
- Operatore `sll`
 - Traslazione logica a sinistra (es: `D <= D sll 8;`)
- Operatore `sra`
 - Traslazione aritmetica a destra (es: `D <= D sra 8;`)
- Operatore: `srl`
 - Traslazione logica a destra (es: `D <= D srl 8;`)

Block Ram (Xilinx) 1/3



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memory is
  port (
    addra: IN std_logic_vector(4 downto 0);
    addrb: IN std_logic_vector(4 downto 0);
    clka:  IN std_logic;
    clkb:  IN std_logic;
    reset: IN std_logic;
    dinb:  IN std_logic_vector(7 downto 0);
    douta: OUT std_logic_vector(7 downto 0);
    web:   IN std_logic);
end memory;
```

Block Ram (Xilinx) 2/3



```
architecture Behavioral of memory is
  subtype RAM_WORD is std_logic_vector (7 downto 0);
  type RAM_TYPE is array (31 downto 0) of RAM_WORD;
  signal ram_block : RAM_TYPE;
begin
  ...
  ...
end Behavioral;
```

Block Ram (Xilinx) 3/3



```
architecture Behavioral of memory is
...
begin
    -- write permitted only if reset is low
    write : process (clkb)
    begin
        if (clkb'event and clkb = '1') then
            if (web = '1') then
                ram_block(conv_integer(addrb)) <=
                    RAM_WORD'(dinb);
            end if;
        end if;
    end process write;
...
end Behavioral;
```

Block Ram (Xilinx)



```
architecture Behavioral of memory is
...
begin
    -- write permitted only if reset is low
    ...
    -- if reset is high, the first word is available on
    output
    read : process (clka, reset)
    begin
        if (clka'event and clka = '1') then
            if (reset = '1') then
                douta <= "00000000";
            else
                douta <= ram_block(conv_integer(addrb));
            end if;
        end if;
    end process read;
end Behavioral;
```

Open-Collector



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

entity OpenCol is
port(
    data: inout std_logic;
    ctrl,clk: in std_logic );
end;
architecture rtl of OpenCol is
    signal Q :std_logic;
    ...
end rtl;
```

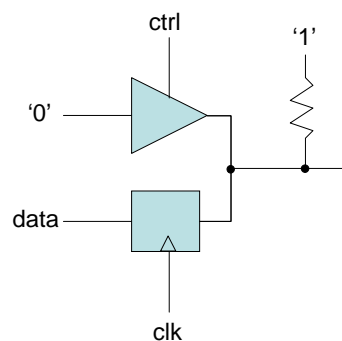
Open-Collector



```
architecture rtl of OpenCol is
    signal Q :std_logic;

begin
    process (clk)
    begin
        if clk'event AND clock='1' then
            Q <= data;
        end if;
    end process;

    process (Q,ctrl)
    begin
        if (Q='0' and ctrl='1') then
            data<='0';
        else
            data<='Z';
        end if;
    end process;
end rtl;
```



Bibliografia



➤ Materiale Generico

- www.amontec.com/fix/vhdl_memo/
- www.electronicweekly.com/ → Tool Kit
 - The VHDL Tool Kit
 - The Jitter Timing Analysis Toolkit → Jitter Issues in Digital Design
 - Interessante per argomenti che riguardano la metastabilità, clock skew, ecc.

➤ VHDL: Tutorial e mini-reference

- www.eng.auburn.edu/departement/ee/mgc
 - VHDL tutorial
- www.eng.auburn.edu/departement/ee/mgc/vhdl.html
 - VHDL mini-reference
- www.engineering.uiowa.edu/~digital/s2004/digital.htm
- www.seas.upenn.edu/~ee201/vhdl/vhdl_primer.html
 - VHDL tutorial

Bibliografia



- V. Štuikys, G. Ziberkas, “Domain specific reuse with VHDL”, 1999, www.soften.ktu.lt/~stuik/knyga/
- <http://www.acc-eda.com/vhdlref/index.html>
- <http://www.eda.org/rassp/vhdl/guidelines/1164qrc.pdf>
 - Quick reference card VHDL'87

➤ Sintassi VHDL

- www.fys.uio.no/studier/kurs/fys329/doc/notes/VHDL87-93.pdf
- http://opensource.ethz.ch/emacs/vhdl87_syntax.html
- http://opensource.ethz.ch/emacs/vhdl93_syntax.html

Bibliografia



➤ Libri e Appunti

- M. Zwolinski, “Digital System Design with VHDL”, second edition, Pearson Education, 2004
ISBN 0 130 39985 X
- C. Brandolese, “Introduzione al linguaggio VHDL”,
web.cefriel.it/~brandole/courses/co-rla/course.htm
- J.Bhasker, “VHDL primer”, Third Ed., Prentice-Hall
- A. Rushton, “VHDL for Logic Synthesis”