Behavioral Description

- La **descrizione comportamentale** (*behavioral*) permette una descrizione *algoritmica o procedurale* (simile ad un programma software) del comportamento di un modulo;

- Questo livello permette *descrizioni a livello più astratto*, perché
  - non contiene riferimenti a come il sistema sarà implementato;
  - non si esplicitano dettagli architetturali o circuitali;

- Questo tipo di descrizione è *utile* in molte situazioni:
  - per scopi di documentazione,
  - per esprimere in forma univoca le specifiche di comportamento di un sistema,
  - per modellare componenti (complessi) di una libreria,
  - per descrivere elementi che servono solo in fase di simulazione (testbench),
  - per evitare eccessivi tempi di simulazione.

Structural Description (1)

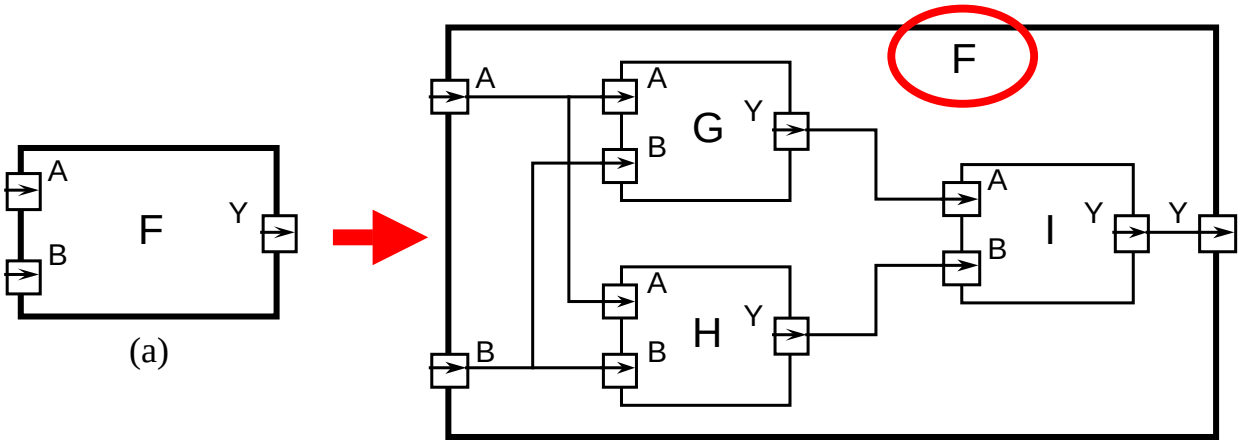
- La **descrizione strutturale** (*structural*) di un modulo specifica:
  - i componenti che costituiscono un modulo;
  - le interconnessioni tra i componenti.
- In altre parole, un modulo è descritto strutturalmente in termini dei moduli che lo compongono e di come i sotto-moduli sono collegati fra loro, con dei segnali (*signals*) che collegano i porti;

- In generale, ciascun modulo componente:
  - è una instanza di qualche entità precedentemente definita;
  - è caratterizzato dalla funzione logica svolta e dal suo timing (tempi di propagazione, di set-up, di hold, ...);
- Un modulo descritto strutturalmente ha una sua funzione logica e un suo timing, che sono funzioni delle caratteristiche corrispondenti dei moduli che lo compongono e del modo in cui sono interconnessi;

- A volte si fa riferimento ad una rappresentazione strutturale in forma grafica (e non testuale come in VHDL) con il termine di schematico (*schematic*).

Structural Description (2)

- Ad esempio, in figura (a) è riportata *l interfaccia del modulo F*;
  - Il modulo F ha 2 input A, B ed un output Y;
- In figura (b) c'è una *descrizione strutturale dell entità F*;
  - l'entità F è composta da instanze delle entità G, H ed I;
  - le entità G, H ed I in questo caso sono delle *black box* (possono avere una descrizione strutturale, comportamentale, … oppure possono essere componenti di libreria).
- In figura (b) sono distinguibili gli ingressi, le uscite e i nodi di interconnessione tra i moduli che compongono il modulo F.



Structural Description (3)

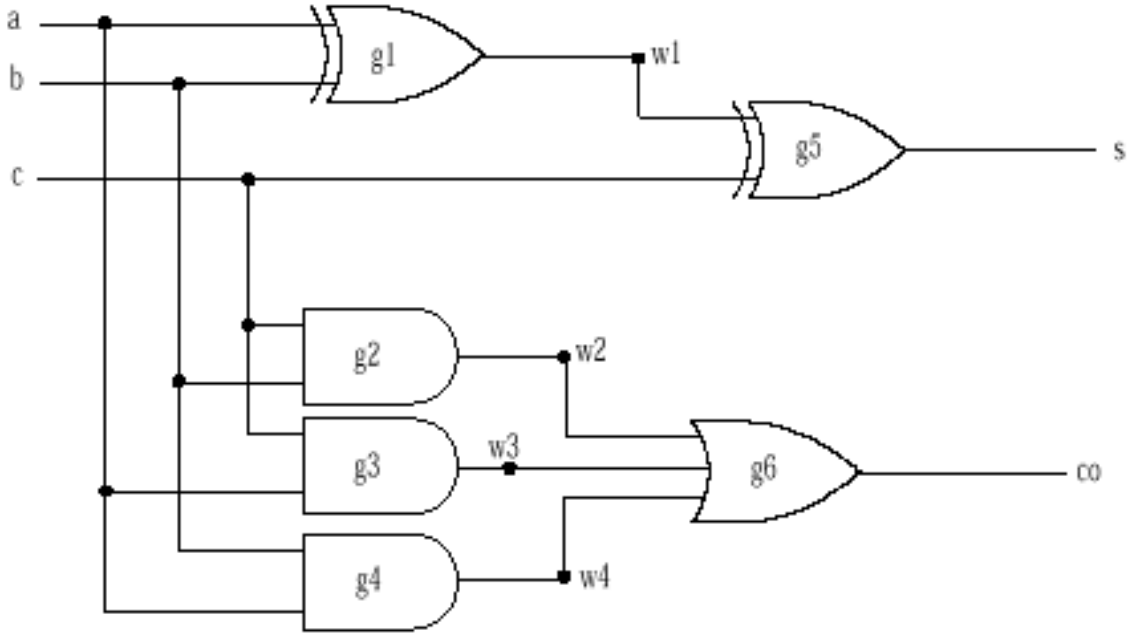
- Ogni modulo è una instanza di una entità (in un certo senso l'insieme entità-architettura è il "tipo" del modulo instanziato) e viene collegato facendo corrispondere i port di ciascuna instanza con dei segnali.

- In generale è difficile estrarre gli aspetti funzionali dalla lettura del codice di una descrizione strutturale;

- La simulazione è più onerosa in termini di tempo e memoria perché c'è la propagazione di molti eventi;

Data-flow Description (1)

- La descrizione di tipo **data-flow** rappresenta il sistema specificando come i segnali si propagano dai pin di ingresso a quelli di uscita;
- Una esempio di descrizione data-flow è la descrizione di una rete combinatoria in termini delle funzioni booleane delle sue uscite.
- Esempio:




```
s <= c NOR (a NOR b);
co <= (a AND b) OR (c AND a) OR (b AND a);
```

Data-flow Description (2)

- **Osservazione**: il circuito precedente potrebbe anche essere descritto a livello strutturale, usando come componenti elementari AND, OR, NOR.
- Ci sono importanti differenze fra una descrizione strutturale e una data-flow;
- La *descrizione strutturale*:
  - descrive il circuito come una netlist di porte fissate e disponibili in una libreria;
  - avendo disponibili una implementazione delle porte necessarie (and, or, nor) si potrebbe realizzare il circuito esattamente secondo lo schematico;
- La *descrizione data-flow*:
  - descrive il legame ingresso-uscita (la tabella di verità) senza far riferimento ad una sua implementazione concreta (e quindi non viene fatto alcun riferimento ad una libreria di porte);
  - si potrebbe realizzare con and-or-not, oppure con sole nand, o sole nor.
- In generale è necessario un passo di sintesi per passare da una descrizione data-flow ad una descrizione strutturale con componenti di libreria (descrizione gate-level) che possa essere implementata.

Gate Level Description

- Una descrizione VHDL gate level contiene:
  - una lista di componenti (gates) che sono usati in un design,
  - una lista delle effettive instanziazioni dei componenti e delle loro interconnessioni.

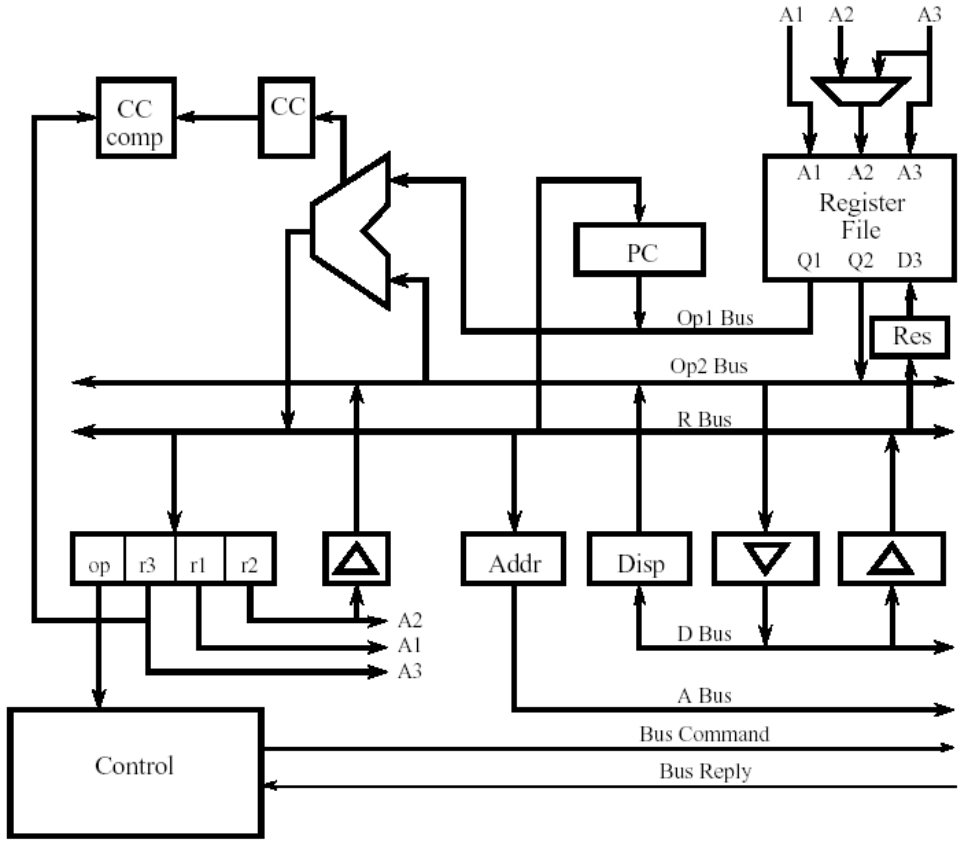
- Tutte le porte usate fanno riferimento ad una tecnologia *target* in cui sono accuratamente descritte ulteriori informazioni sulle porte (area, propagation delay, capacità di ingresso, capacità di pilotaggio, ...).

- Una descrizione gate level è la descrizione con il minimo livello di astrazione in VHDL, perché fornisce indicazioni estremamente dettagliate sul circuito (massima frequenza di clock, area occupata, potenza dissipata, …);
- Essendo una descrizione di tipo strutturale:
  - E' in generale difficile estrarre gli aspetti funzionali dalla lettura del codice;
  - La simulazione è più onerosa in termini di tempo e memoria perché c'è la propagazione di molti eventi;

RT Level Description

- **RTL = Register Transfer Level**
- A questo livello di astrazione il design è diviso in 2 parti:
  - una parte di *logica combinatoria* (che calcola il prossimo stato e le uscite a partire dallo stato corrente e dagli input)
  - *elementi di memoria* (gli elementi di memoria sono controllati dai clock di sistemi).

- E' una descrizione del sistema in termini di memorizzazione, di elaborazione e di spostamenti/ trasformazione dei dati fra i registri.
- In questo tipo di descrizione è contenuta l'informazione sulla struttura del sistema perché sono distinti i componenti che memorizzano i dati da quelli che li elaborano.
- Rispettando un insieme di regole è una descrizione sintetizzabile.



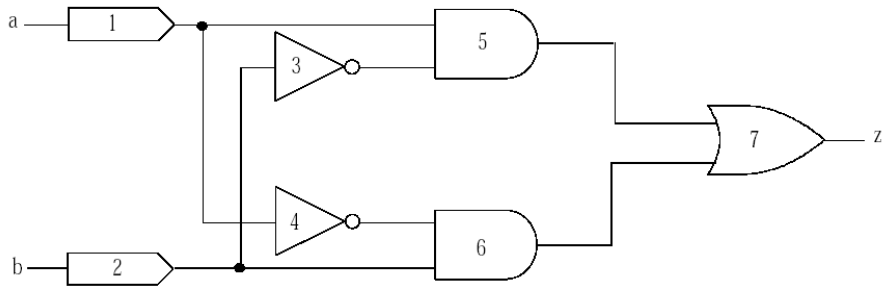
Tecniche di simulazione dell Hw

- La simulazione dell'hardware è imprescindibile nel ciclo di sviluppo dei sistemi digitali;
- Per circuiti complessi è fondamentale che sia efficiente;

- **Simulazione a tempo quantizzato**:
  - Il tempo di simulazione procede per incrementi di ampiezza prefissata (passo di quantizzazione);
  - E' incapace di adattarsi alla dinamica del sistema;
- **Simulazione ad eventi**:
  - Il tempo di simulazione procede per incrementi di ampiezza variabile (solo quando ci sono variazioni in un nodo si calcola la propagazione di queste variazioni);
  - Permette di seguire la dinamica del sistema;
- **Varie tecniche di ottimizzazione**:
  - Ad es. determinazione dello scope di una variazione (per entrambe le tecniche);
  - Variazione del passo di quantizzazione per le tecniche a tempo quantizzato (ad es. Spice);

Simulazione a tempo quantizzato (1)

- Esempio nel caso di *0-delay* (non si tiene conto della tempificazione del circuito ma solo del suo comportamento) :

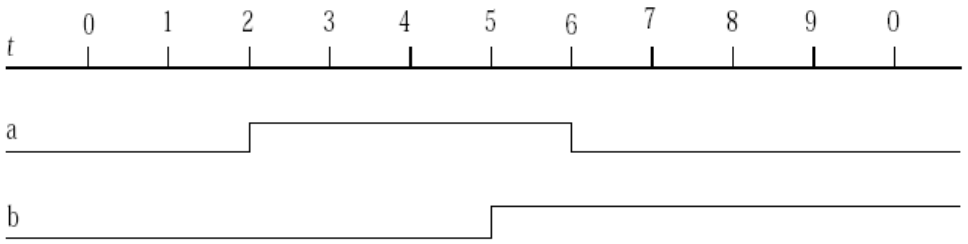


Gate	Function	Input1	Input2	Output Value
1	Buffer	A	-	0
2	Buffer	B	-	0
3	Not	2	-	1
4	Not	1	-	1
5	And	3	1	0
6	And	4	2	0
7	Or	5	6	0

- Lo *stato del circuito* è rappresentato dalla tensione ad ogni nodo del circuito;
- Il *circuito* è rappresentato come una netlist in forma tabulare;
  - Gli ingressi delle gate sono rappresentati dagli identificativi delle porte;
  - Lo stato è rappresentato da un'altra colonna (output value);
  - L'ordine delle righe rappresenta un modo di ispezionare il grafo del circuito in modo da rispettare la dipendenza funzionale delle porte;

Simulazione a tempo quantizzato (2)

- Il simulatore si attiva ad intervalli di tempo fissati T_{SAMPLE} e aggiorna lo stato del circuito (= i valori della tabella);
- Dopo ogni T_{SAMPLE} → aggiornamento dello "stato" del circuito:
  - si campionano gli ingressi;
  - si calcola il valore del potenziale di ogni nodo secondo le gate e le loro interconnessioni, ispezionando la tabella;



Gate	Function	Input1	Input2	T = 0	T = 2	T = 5
1	Buffer	A	-	0	1	1
2	Buffer	B	-	0	0	1
3	Not	2	-	1	1	0
4	Not	1	-	1	0	0
5	And	3	1	0	1	0
6	And	4	2	0	0	0
7	Or	5	6	0	1	0

Simulazione a tempo quantizzato (3)

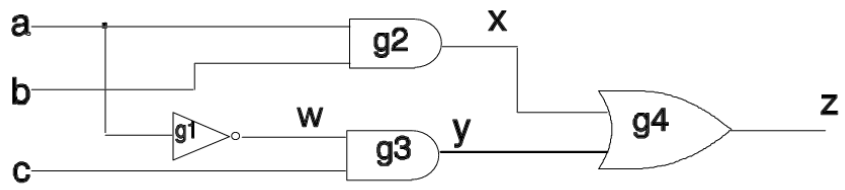
- **Principio di funzionamento:**
  - Il tempo di simulazione procede per incrementi di ampiezza prefissata (*passo di quantizzazione*);
- **Vantaggi:**
  - E' (abbastanza) semplice da implementare nel caso di assenza di ritardi (simulazione puramente funzionale);
  - Può simulare anche variazioni continue (purché discretizzate) dei potenziali (ad es. Spice);
- **Svantaggi:**
  - Questa tecnica è incapace di adattarsi alla dinamica del sistema;
  - E' difficile gestire i tempi di propagazione delle porte;
  - Il passo di quantizzazione dovrebbe essere minore delle costanti di tempo del circuito → cresce l'onere computazionale delle simulazioni;
- **Possibili ottimizzazioni:**
  - Determinazione dello scope di una variazione del potenziale di un nodo;
  - Il passo di quantizzazione può essere variabile e adattarsi all'evoluzione del circuito;

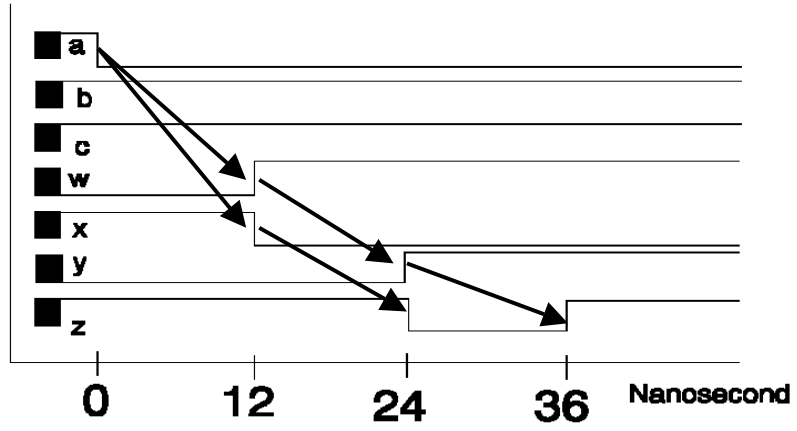
Simulazione ad eventi (1)

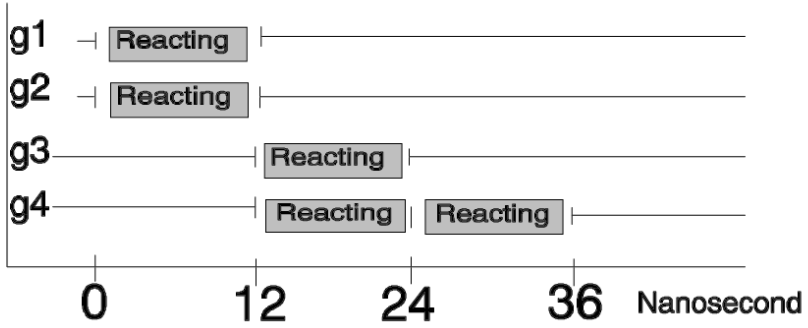
- **Principio di funzionamento:**
  - Valuta il circuito solo quando si verifica un "evento" ovvero un cambiamento del potenziale di un nodo;
  - Se non ci sono eventi, la rete non evolve ma è stabile;
- **Vantaggi:**
  - Permette una simulazione veloce dei circuiti digitali → i simulatori VHDL sono ad eventi (*event driven*);
- **Svantaggi:**
  - È più difficile da implementare;

Simulazione ad eventi (2)

- Esempio del circuito riportato a fianco:
  - Si suppone che tutte le porte abbiano lo stesso ritardo (12ns);
  - Sono mostrate le *waveforms* (forme d'onda) ai nodi del circuito;
- Le variazioni di un nodo si propagano lungo il circuito;
- Ad es. la commutazione sull' ingresso a: 1→0 @ 0ns provoca, tramite la porta NOT g1, una commutazione su w: 0→1 @ 12ns, la quale provoca una commutazione 0→1 @ 24ns sul segnale y…

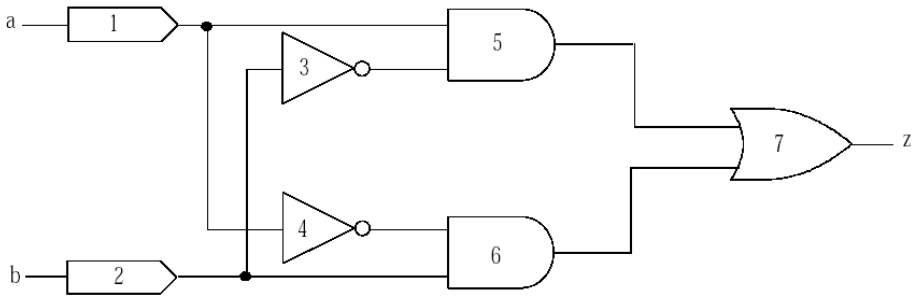






Simulazione ad eventi (3)

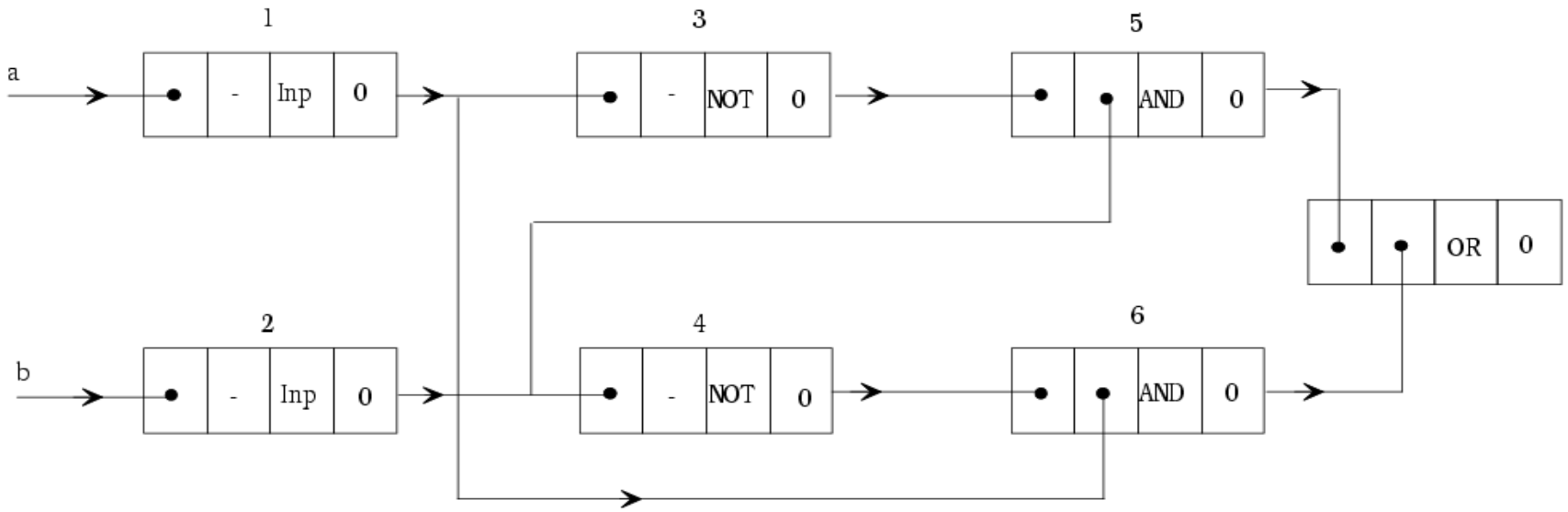
- La rappresentazione del circuito per la simulazione ad eventi si basa su una linked list;



Legend:

In1	In2	Fnc	Out

In1: Input 1; In2: Input2; Fnc: Function; Out: Output Value



Il modello di tempificazione VHDL (1)

- Abbiamo detto che il VHDL permette di rappresentare il fatto che diversi oggetti (moduli, assegnazioni di segnali, process…) evolvano contemporaneamente ("concorrentemente");
- Le assegnazioni e le instanziazioni dei moduli sono concorrenti ed è del tutto indifferente l'ordine con cui vengono specificate.

```
ENTITY example IS
        PORT (a, b, c : IN BIT ; z : OUT BIT);
END example;
--
ARCHITECTURE concurrent OF example IS
        SIGNAL w, x, y : BIT;
BEGIN
        w <= NOT a AFTER 12 NS;
        x <= a AND b AFTER 12 NS;
        y <= c AND w AFTER 12 NS;
        z <= x OR y AFTER 12 NS;
END concurrent;
```

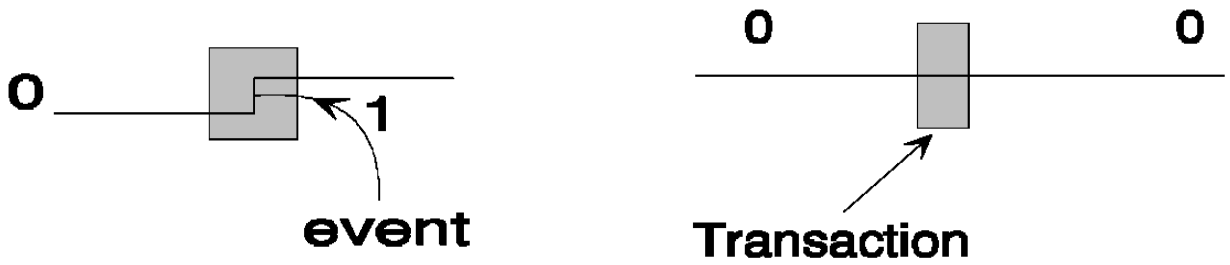
Il modello di tempificazione VHDL (2)

- Una *assegnazione* su un segnale comporta l'aggiunta di una *transazione* (transaction) sul segnale target:

```
w <= NOT a;
```

```
x <= a AND b AFTER 12 NS;
```

- Una transazione ha un valore e una componente di tempo (value, time)
  - *value* rappresenta il valore che verrà assegnato al segnale e viene calcolato nell'istante in cui si genera la transazione;
  - *time* rappresenta dopo quanto tempo il valore value sarà assegnato al segnale.
- Non sempre una assegnazione comporta una variazione del valore di un segnale. Si parla di *evento* quando questo avviene, altrimenti di semplice transazione.



Il modello di tempificazione VHDL (3)

- Ad esempio se a = 1 → 0 @ 0s, l'assegnazione riportata

```
w <= NOT a AFTER 12 NS;
```

genera una transazione di valore (1, 12ns).

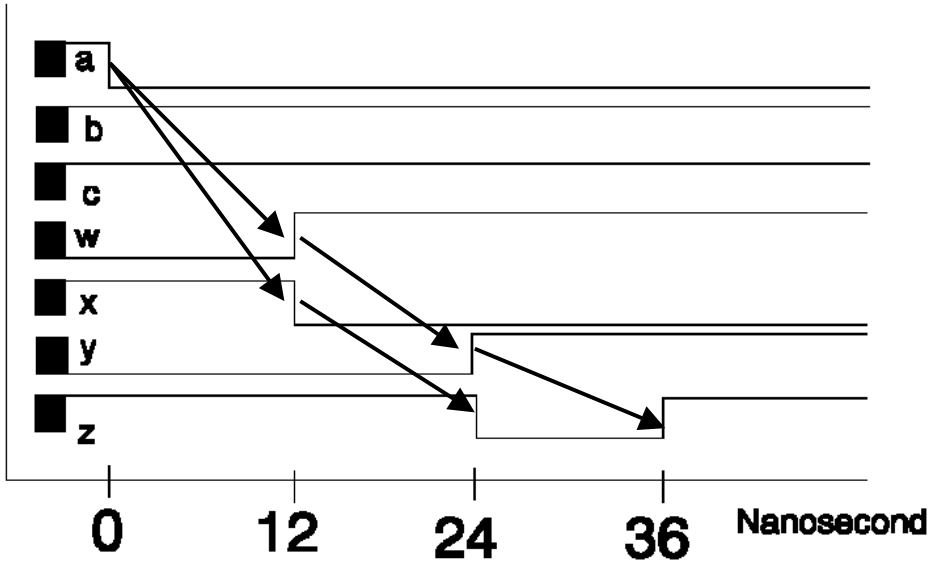
- Dopo 12ns dall'istante 0, l'evento 0 → 1 avverrà sul segnale w.

Il modello di tempificazione VHDL (4)

- Si consideri la descrizione data-flow:

```
w <= NOT a AFTER 12 NS;
x <= a AND b AFTER 12 NS;
y <= c AND w AFTER 12 NS;
z <= x OR y AFTER 12 NS;
```

- La tempificazione dei segnali a causa di una commutazione sul segnale a 1 → 0 @ 0 (b è sempre alto) è riportata in figura:



Delta cycles (1)

- Se si assegna un valore ad un segnale senza specificare il ritardo si assume che il ritardo sia 0:

```
w <= NOT a;
```

è equivalente a:

```
w <= NOT a AFTER 0ns;
```

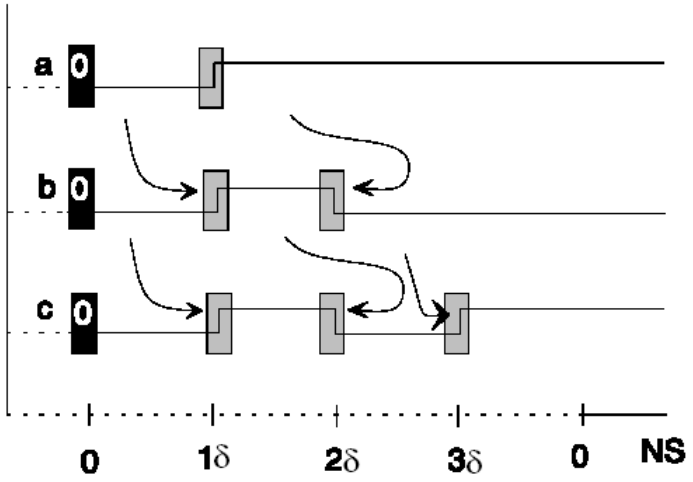
- La transazione è "schedulata" (*scheduled*) per lo stesso istante dell'assegnazione, ma concretamente il simulatore VHDL la eseguirà nel ciclo di simulazione successivo (→ il *delta cycle* successivo);
- Il tempo della simulazione è del tipo: $x_1T+x_2\delta$ dove T è un tempo (ns, ms, s) e delta (δ) è un ciclo di simulazione che non corrisponde a tempo reale;
- In altre parole una transazione con tempo zero scade un "delta" dopo che è stata posta sul segnale;

- **Osservazione:**
  - Delta è usato per lo scheduling: un miliardo di delta non compongono un femtosecondo!

Delta cycles (2)

- Vediamo un esempio di assegnazione senza ritardo:

```
ARCHITECTURE concurrent OF es_delta IS
        SIGNAL a, b, c : BIT := '0';
BEGIN
        a <= '1';
        b <= NOT a;
        c <= NOT b;
END concurrent;
```



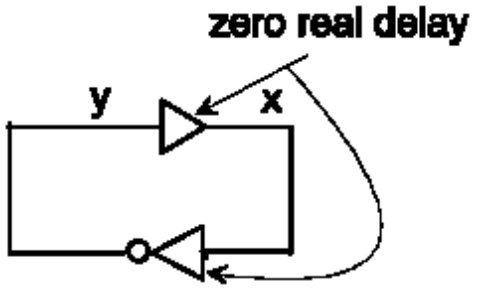
- All'inizio i segnali sono tutti 0 (*inizializzazione*);
- Ma (a,b,c) = (0,0,0) è uno stato "instabile" (incompatibile con il circuito) e la rete comincia ad *andare a regime* → (0,1,0);
- La transizione di a 0→1 @ 0ns+1δ provoca l'evento 0→1 dopo 1 delta e la variazione si propaga nella rete.

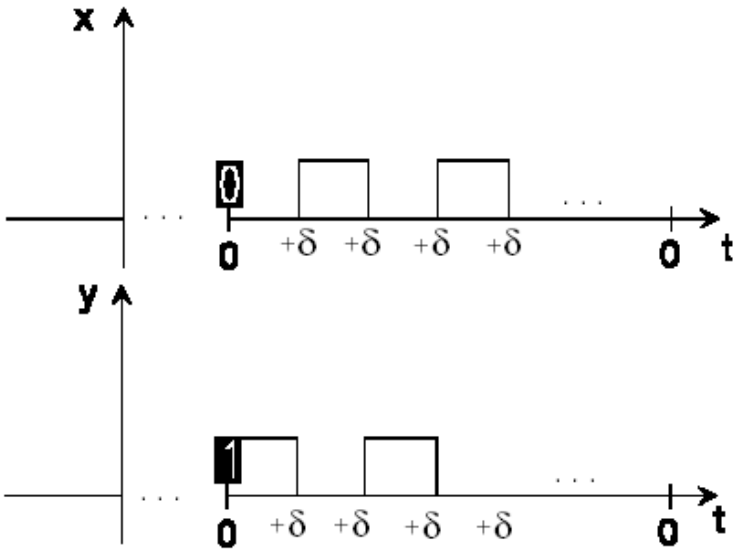
```
                ps delta a b c
=======================
                 0    +0 0 0 0
                 0    +1 1 1 1
                 0    +2 1 0 0
                 0    +3 1 0 1
```

Delta cycles (3)

- Un "paradosso":

```
ARCHITECTURE concurrent OF timing_demo IS
        SIGNAL x, y : BIT := '0';
BEGIN
    x <= y;
    y <= NOT x;
END concurrent;
```

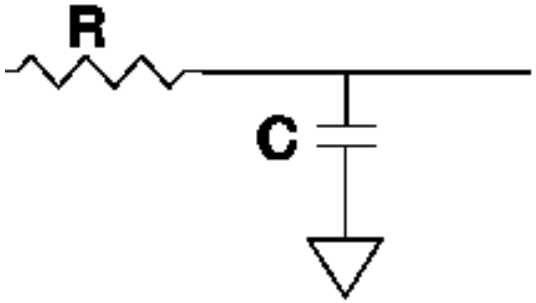




- I segnali x ed y oscillano indefinitamente, senza che il tempo reale della simulazione avanzi

Inertial e transport delays (1)

- Spesso, a causa dei parametri parassiti, le interconnessioni dei circuiti reali possono essere schematizzate come delle reti RC;
- La presenza di queste reti filtra gli impulsi troppo brevi rispetto alla costante di tempo RC della rete;

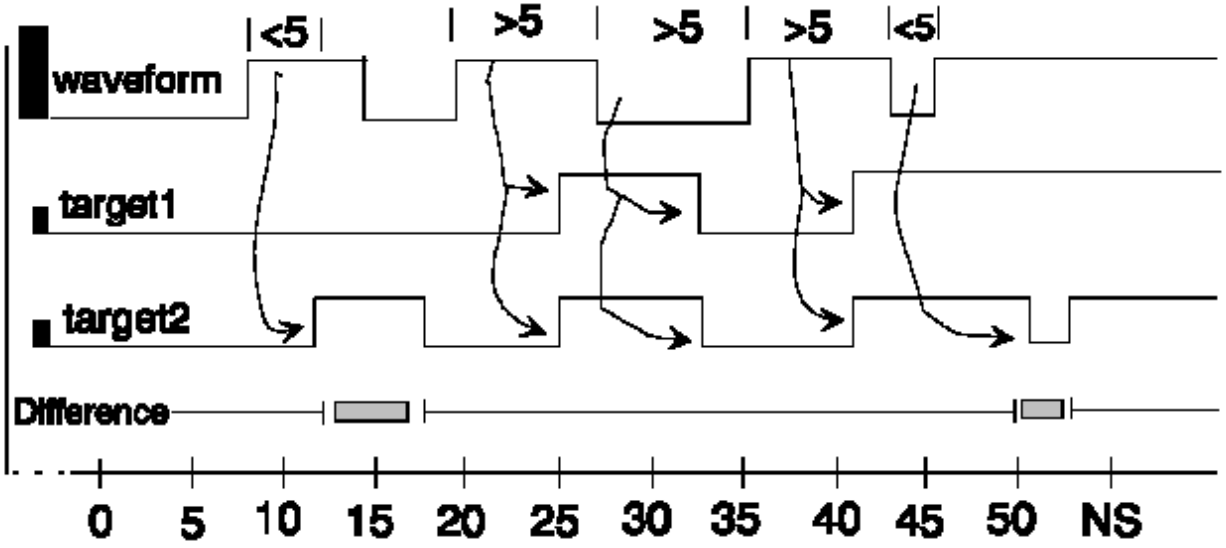


- Questo fenomeno può essere modellato in VHDL con dei ritardi "inerziali" (*inertial delays*):
  - I ritardi **inerziali** rappresentano un fenomeno di ritardo che però non propaga tutti gli impulsi (0→1→0, 1→0→1), ma solo quelli che durano più di un certa soglia;
  - I ritardi **transport** invece propagano sempre le transazioni indipendentemente dalla distanza temporale fra due transazioni successive;
- Tipicamente i ritardi inertial sono poco usati in VHDL;
- Un ritardo senza la parola chiave INERTIAL è di tipo TRANSPORT.

Inertial e transport delays (2)

- Esempio di ritardo inertial e transport:

```
ARCHITECTURE delay OF example IS
   SIGNAL target1, target2, waveform : BIT;
BEGIN
   waveform <=                                    -- forma d onda in figura
   target1 <= INERTIAL waveform AFTER 5 NS;  -- inertial delay
   target2 <= TRANSPORT waveform AFTER 5 NS; -- transport  delay
END delay;
```



Interazioni fra transazioni

- In generale possono essere applicati, in successione, al medesimo segnale diverse transazioni;
- In questo caso la regola di risoluzione è la seguente:

	TRANSPORT	INERTIAL
New Transaction is BEFORE Already Existing	Overwrite existing transaction.	Overwrite existing transaction.
New Transaction is AFTER Already Existing	Append the new transaction to the driver.	Overwrite existing if different values, otherwise keep both.

- Sono fenomeni che si incontrano raramente nel VHDL;
- Navabi fa una trattazione estesa di questi casi di interazione fra transazioni.