



Laboratorio di metodologie di progetto HW

Linguaggi di Specifica

Modello e Linguaggio
Classificazione dei linguaggi formali
Linguaggi Grafici
Linguaggi Testuali

Versione del 20/05/04



Obiettivo

- ❑ Il Progetto di un sistema digitale inizia dalla raccolta dei requisiti e dal loro successivo affinamento
- ❑ Il passo successivo è la costruzione del modello:
formalizzazione
 - Processo di trasformazione di una descrizione, tipicamente in linguaggio naturale (semi-formale), in un'altra che utilizza un linguaggio di descrizione formale
 - Questo passo è quello che porta alla realizzazione di un modello simbolico
 - Nota: i modelli a cui ci riferiamo sono modelli simbolici
 - rappresentazione formale delle conoscenze relative ad un fenomeno



Sommario

- Modello e Linguaggio
- Classificazione dei linguaggi formali
- Linguaggi Grafici
 - Schematic Entry
 - Timing Diagrams
- Linguaggi Testuali
 - VHDL
 - SystemC



Modello e Linguaggio

- Un modello (simbolico/matematico) è la rappresentazione formale di idee e conoscenze relative ad un fenomeno finalizzate al raggiungimento di un obiettivo.
- La rappresentazione di un modello simbolico è espressa tramite un linguaggio formale.
- Un linguaggio è formale quando non ammette una interpretazione arbitraria.
 - Un linguaggio di programmazione è un linguaggio formale
 - Il linguaggio naturale è semi-formale
 - Mescolare l'impasto con il cucchiaino
 - "con il cucchiaino" si riferisce a "mescolare"
 - Mescolare l'impasto con la panna
 - "con la panna" si riferisce a "impasto"



Classificazione dei linguaggi formali

□ Grafico

- Ingegneria del software
 - Statecharts, UML, ...
- Ingegneria dell'hardware
 - Timing Diagrams, SE (Schematic Entry), HHL (Heterogeneous Hardware Logic), ...
- Ingegneria di sistema
 - LSC (Live Sequence Charts)

□ Testuale

- Linguaggi basati su linguaggi di programmazione
 - SystemC, HardwareC, Esterel, Java, ...
- Linguaggi orientati alla descrizione dell'hardware
 - VHDL, VERILOG



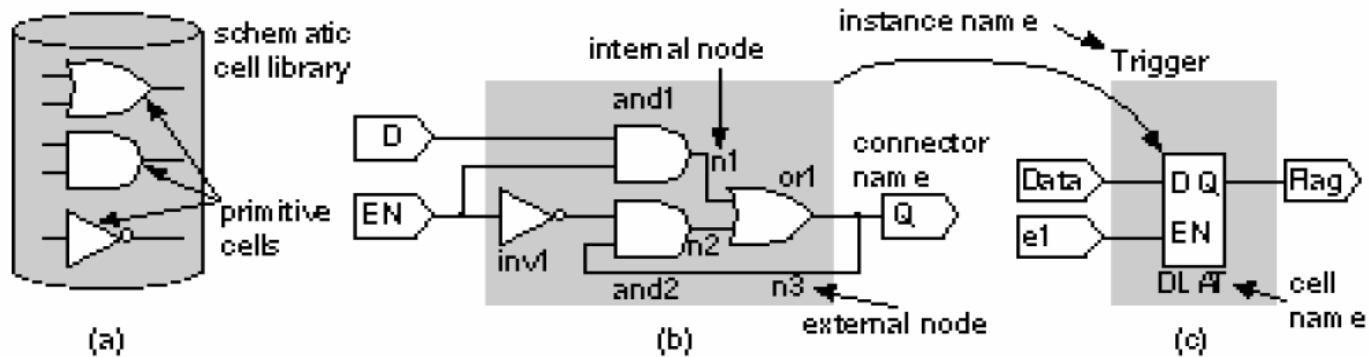
Schematic Entry

- Consente di specificare un progetto graficamente
 - Istanziando componenti di libreria
 - Collegando gli elementi mediante BUS o fili (wire)
 - Consentendo anche il passaggio da BUS a filo e vice versa
 - Gestendo della gerarchia
 - Specificando i connettori tra i livelli
 - Inclusi i connettori (speciali) di I/O la cui istanza richiede, nella maggior parte dei casi un successivo inserimento di buffer
 - Navigando nella gerarchia

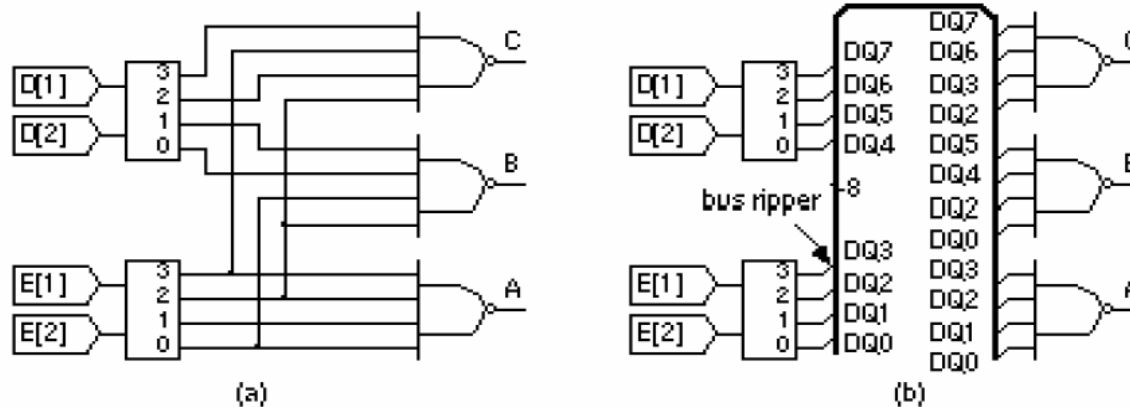


Schematic Entry

- Istanza di componenti di libreria



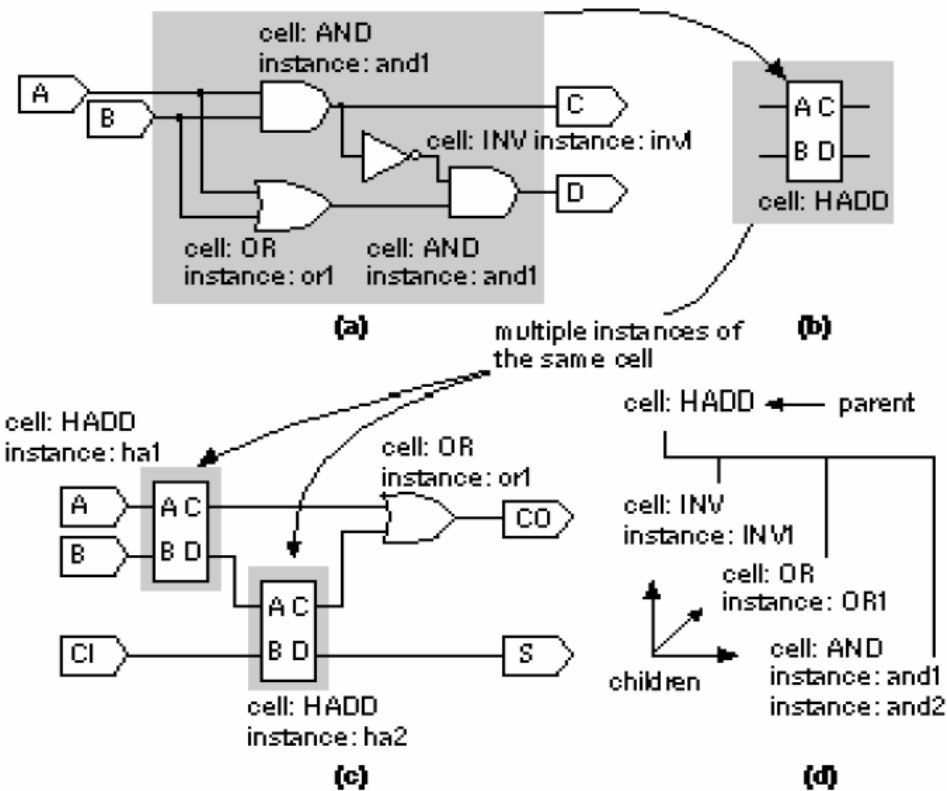
- Collegamento *wire* e BUS





Schematic Entry

□ Esempio (Gerarchia)





Schematic Entry

□ Strumenti commerciali

- Moltissimi strumenti commerciali EDA includono uno *Schematic Entry* o *Editor*
- CADENCE
- Synopsys
- Mentor
- Xilinx
- Altera
- ...



Timing Diagrams

- Descrivono la relazione tra segnali il tempo
 - Tipicamente indicati per lo sviluppo di interfacce
- Gli oggetti generalmente trattati sono:
 - Forma d'onda
 - Segnale, BUS, Clock
 - Fronte
 - Cambiamento di stato di una forma d'onda
 - Dipendenza
 - Descrive come un fronte ne condiziona un'altro
 - Annotazione



Timing Diagrams

- Forma d'onda
 - Segnale
 - Attributi
 - Valore iniziale: 0, 1, X (indefinito), Z (tri state), H e L
 - BUS
 - Attributi
 - Valore iniziale: Z (tri state), V (Valido), X (non valido)
 - Direzione: Ingresso, Uscita, Ingresso/uscita
 - Dimensione: Numero di bit
 - Clock
 - Attributi
 - Frequenza, periodo, offset, duty-cycle
 - Valore iniziale (0 o 1)



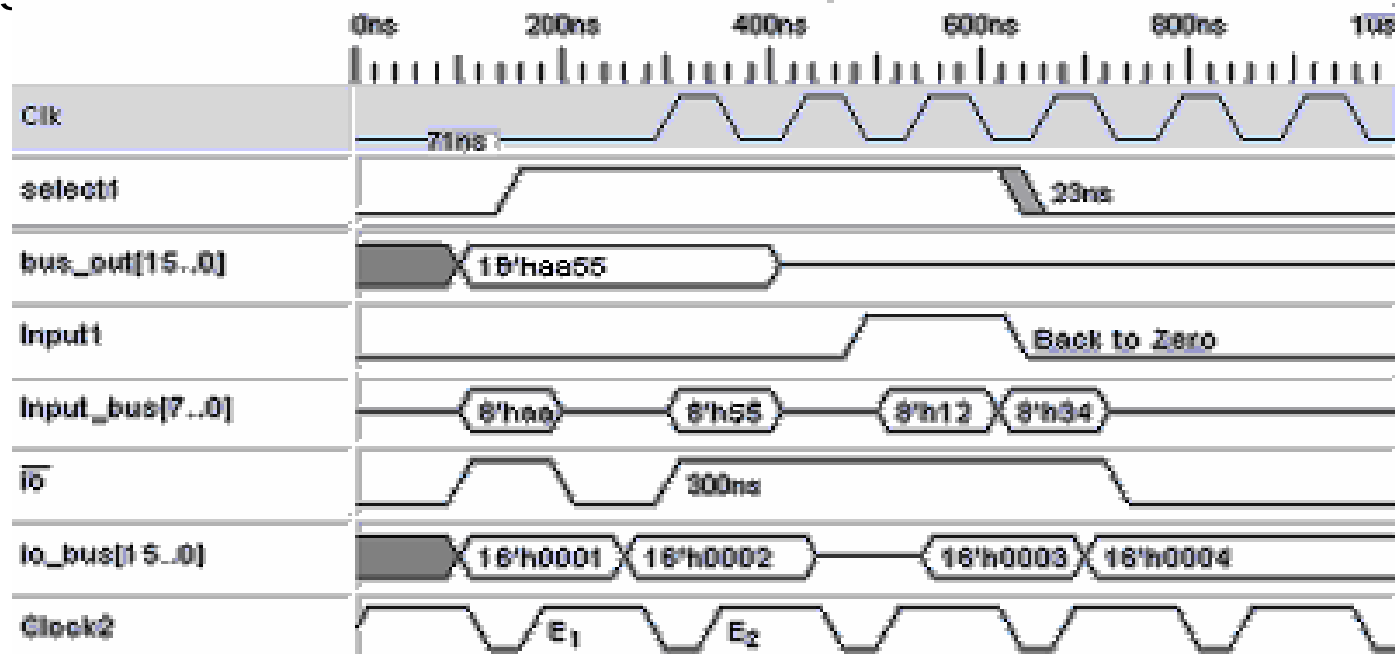
Timing Diagrams

- Fronte
 - Proprietà nel tempo
 - Definizione del tempo di accadimento per gli eventi certi
 - Definizione dell'intervallo di tempo di validità per gli eventi incerti
 - Definizione delle proprietà "lock"
 - tempo di accadimento inalterabile
 - Proprietà nello stato
 - Per i segnali: 0, 1, X (indefinito), Z (tri state), H e L
 - Per i BUS: Z (tri state), V (valido), X (non valido)



Timing Diagrams

□ Esempio





Timing Diagrams

□ Relazioni

- Ritardo (Delay)

- Specifica il tempo (max o min) tra due fronti
- Quando un fronte è spostato, il fronte vincolato si sposta a sua volta per mantenere il ritardo specificato
- I ritardi indicano solo le dipendenze che influenzano il tempo del fronte a valle

- Set-up

- Specifica di quanto (max o min) un fronte deve precedere un altro

- Hold

- Specifica di quanto (max o min) un fronte deve seguire un altro



Timing Diagrams

□ Relazioni

- Garantita

- Specifica che il tempo tra due fronti sarà sempre compreso tra un minimo ed un massimo.

- Vincolata

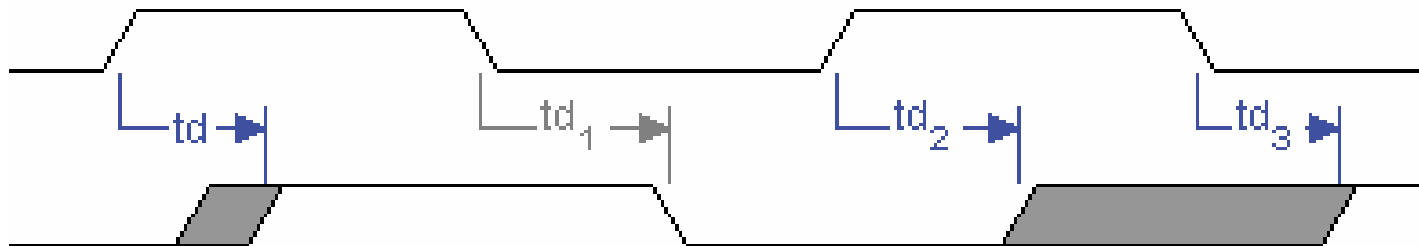
- Specifica che il tempo tra i due fronti è fissato (min o max).



Timing Diagrams

□ Esempio (Ritardo)

- Td: $\min=20$ $\max=40$; condiziona i due fronti, sia quello a t_e (il più in anticipo) sia t_l (il più in ritardo) del segnale controllato.
 - È disegnato tra i due fronti del segnale target
- Td1; non ha parametri (non condizionante).
- Td2: $\min=50$; impone solo il valore minimo e condiziona solo t_e del fronte controllato
 - È disegnato sul fronte più in anticipo del segnale controllato
- Td3: $\max=40$; impone solo il valore massimo e condiziona solo t_l del fronte controllato
 - È disegnato su t_l .

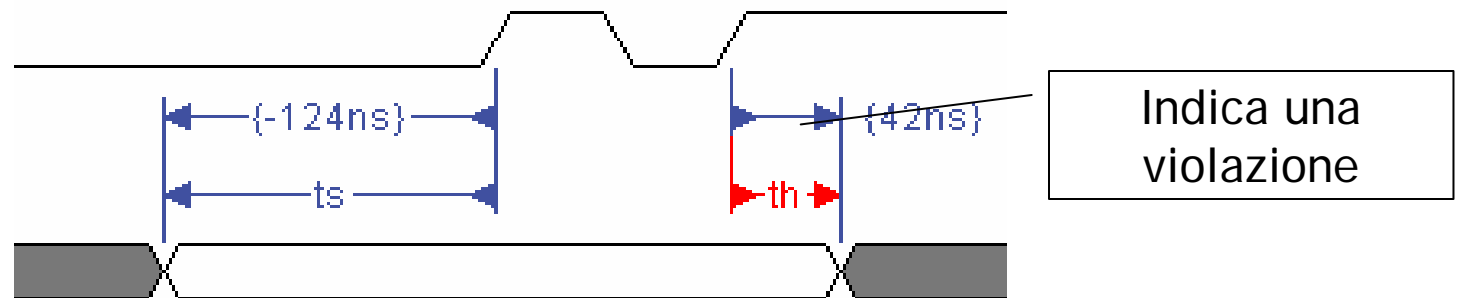




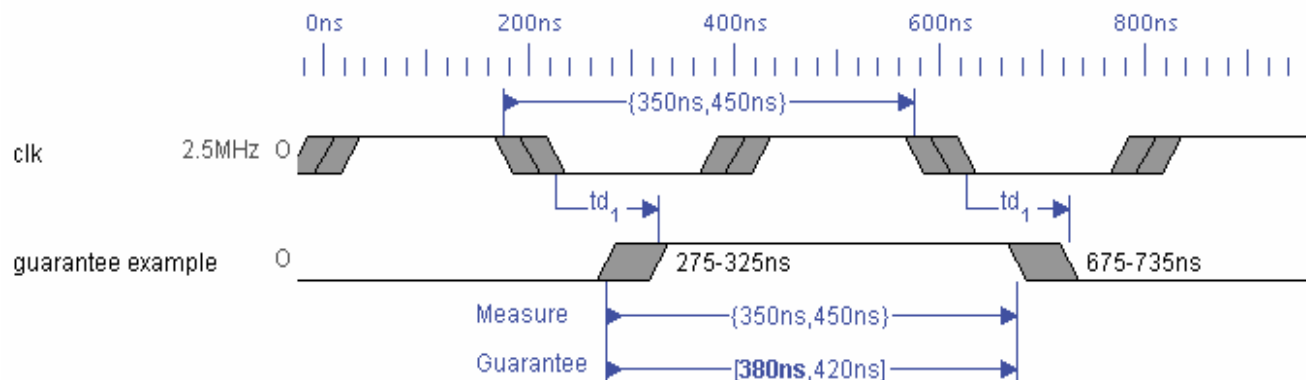
Timing Diagrams

□ Esempio (Set-up e Hold):

- T_s : min=100; T_h : min=50;



□ Esempio (Garantita)





Timing Diagrams

- Dai Timing Diagrams è possibile estrarre
 - Il codice VHDL dell'applicazione
 - Il codice VHDL del test bench
- I Timing Diagrams possono essere manipolati automaticamente per
 - Eliminare le violazioni
 - Ottimizzare i percorsi critici
- Strumenti commerciali
 - TimingTool (TimingTool)
 - TimingDiagrammer Pro (SynaptiCAD)
 - TimingDesigners (FORTE System Design)



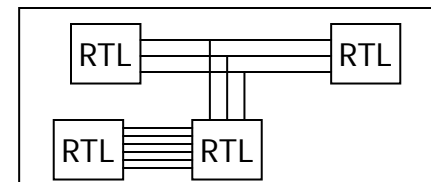
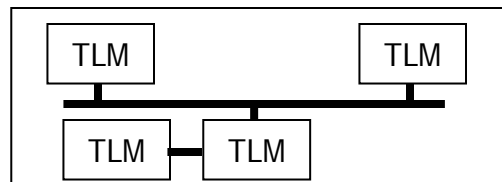
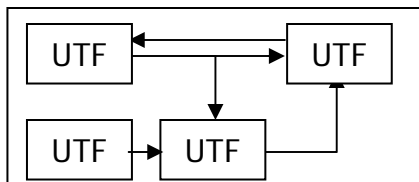
SystemC

- *SystemC* è un linguaggio basato su C++
 - C++ arricchito di una libreria di classi
- L'obiettivo di SystemC è di facilitare il passaggio dalla fase di concettualizzazione a quella di progettazione
 - Ridurre il "divario produttivo"
- SystemC Supporta
 - Comunicazioni hardware
 - Modello del tempo
 - Concorrenza
 - Tipi di dati per l'hardware
- La specifica è eseguibile
 - Kernel di simulazione integrato con la specifica



SystemC

- ❑ *SystemC* 1.0
 - Modella l'hardware (RTL e Comportamentale)
- ❑ *SystemC* 2.0
 - Estende la modellazione a livello di sistema
 - UTF (untimed functional level) - Modello funzionali
 - Specifica eseguibile senza caratteristiche di tempo
 - TLM - Modello transazionale (TLM)
 - Progetto di piattaforme e co-verifica Hw/Sw
 - RTL - Modello architetturale di alto livello (pin-level)
 - Progetto e verifica di sistemi hardware a livello sia RTL sia comportamentale





SystemC: Transaction Level Modelling

- Il livello di astrazione Transaction Level aiuta il System Level Design:
 - 1) separando la modellizzazione del comportamento delle “unità funzionali” (moduli) dalla loro comunicazione
 - 2) fornendo un unico formalismo per descrivere la comunicazione a più livelli di dettaglio
 - 3) consentendo di modellizzare molti modelli di computazione diversi



SystemC: Transaction Level Modelling Cos'è una transaction?

- Possiamo definire una “transaction” come un qualsiasi passaggio di dati tra due unità funzionali del nostro modello:
 - il trasferimento di un risultato intermedio tra stadi di una pipeline (livello microarchitetturale);
 - il passaggio di una parola da un processore ad una memoria (livello architetturale);
 - il passaggio di una struttura dati complessa tra due fasi di un'elaborazione dati (livello algoritmico) (es: la trasformata di Fourier di un segnale da un filtro ad un altro);



SystemC: Transaction Level Modelling Cosa vs. Come

- Concettualmente, la modellizzazione di una “transaction” avviene rispondendo a due domande:
 - 1) quale dato viene trasferito? (tipo, valore);

Es: il processore legge una parola dalla memoria all’indirizzo “0x00ef00ef”; il valore restituito è “0xaabbaabb”
 - 2) come avviene il trasferimento? (mezzo di comunicazione, protocollo)

Es: il processore fornisce l’indirizzo, asserisce il comando di lettura, aspetta l’acknowledge etc.
- La risposta alla prima domanda è necessaria per ottenere un modello funzionale, la seconda no!



SystemC: Transaction Level Modelling Come vs. Come

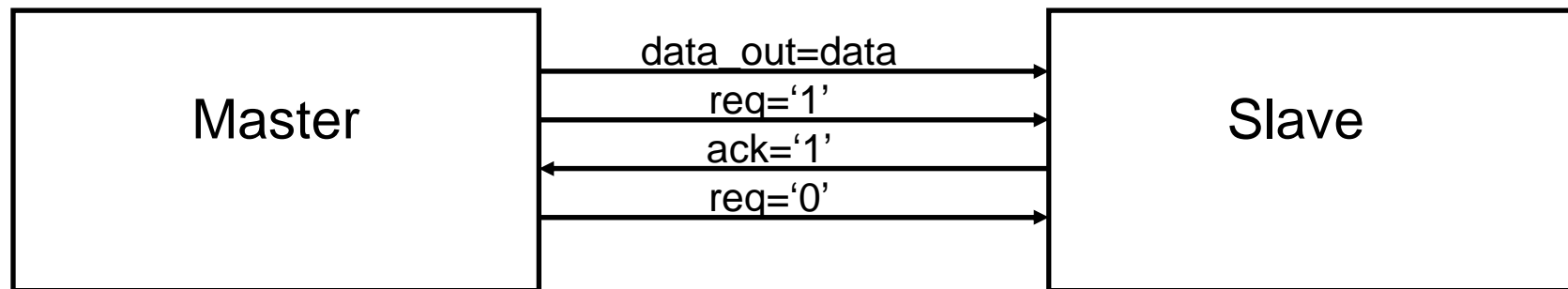
- Nel Transaction Level Modelling, la comunicazione può essere descritta a molti livelli di dettaglio temporale, come ad esempio:
 - untimed functional: viene modellizzato solo il passaggio di dati tra le unità; le transizioni avvengono in tempo nullo;
 - bus cycle accurate: la granularità corrisponde a transizioni (anche burst) di dati su un bus; si possono comunque modellizzare transizioni *bloccanti*;
 - cycle accurate: accuratezza al ciclo di clock;



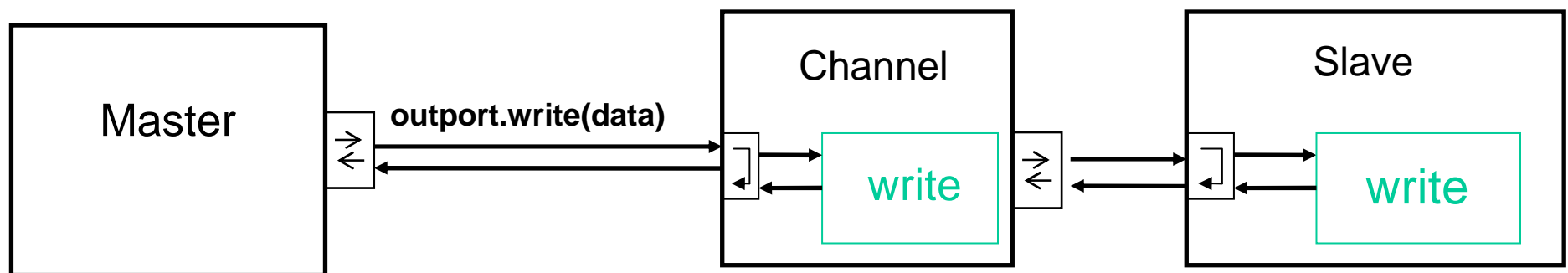
SystemC: Transaction Level Modelling

Transaction Level vs. pin level

Pin accurate



Transaction Level





SystemC: Transaction Level Modelling Confronto fra le tecniche di modellazione

	TLM	RTL	Emulazione hw/sw	Prototipazione
Prestazioni in simulazione	100KHz	10-100Hz	2MHz	Real-time
Complessità descrizione	6.500 righe codice	65.000 righe codice	650.000 righe codice	Non applicabile
Capacità esplorazione	Alta	Bassa	Bassa	Nulla
Costo	Basso	Medio	Altissimo	Alto



SystemC

- La specifica è eseguibile
 - Verifica veloce
 - Un modello a livello di transazione consente una simulazione accurata al ciclo di bus
 - Più di 100.000 cicli/sec
- I tre livelli possono convivere in uno stesso modello
 - Consente un raffinamento graduale dei moduli
 - Consente di verificare la funzionalità in ogni fase preliminare dello sviluppo
 - Non è richiesta l'interazione tra simulatori differenti, uno per ogni livello di astrazione considerato

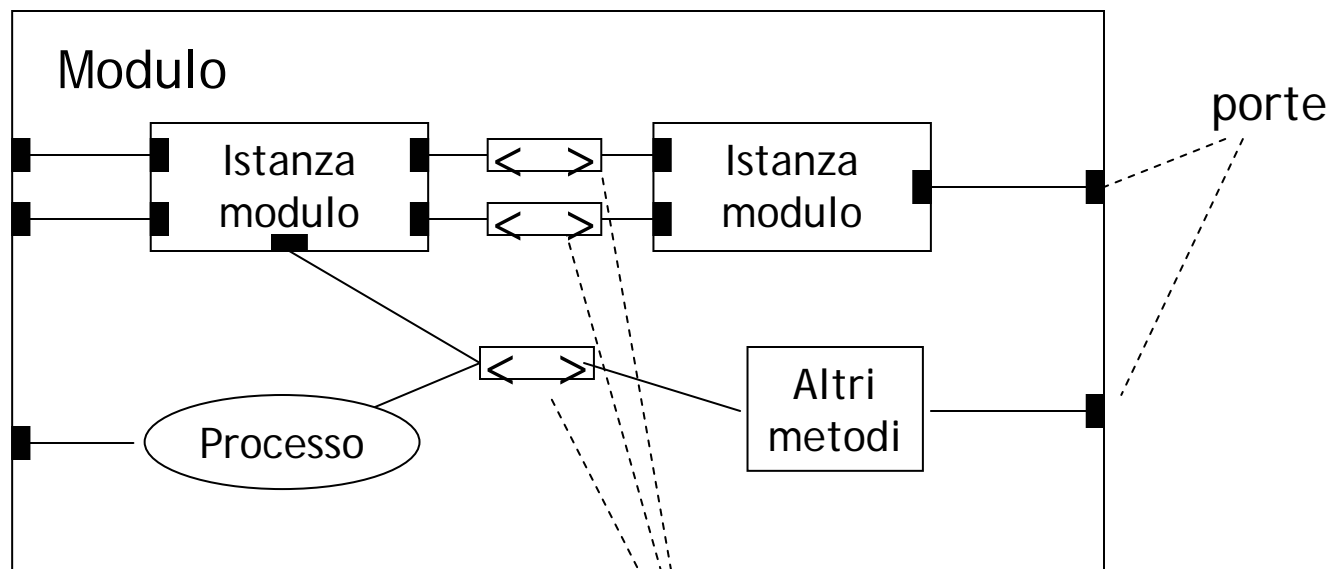


SystemC

- Elementi del linguaggio
 - Tipi di dati
 - Moduli porte e segnali
 - Processi
 - Gerarchia
- Tipi di dati
 - Numerico, Booleano, bit
 - Differenti tipi di dati: da `int` a `float`
 - Da `int` a `float` peggiorano le caratteristiche di velocità della simulazione
 - Es: `sc_int<5> a; // a è un intero a 5 bit`



- Moduli porte e segnali (cont.)
 - Struttura di modellazione



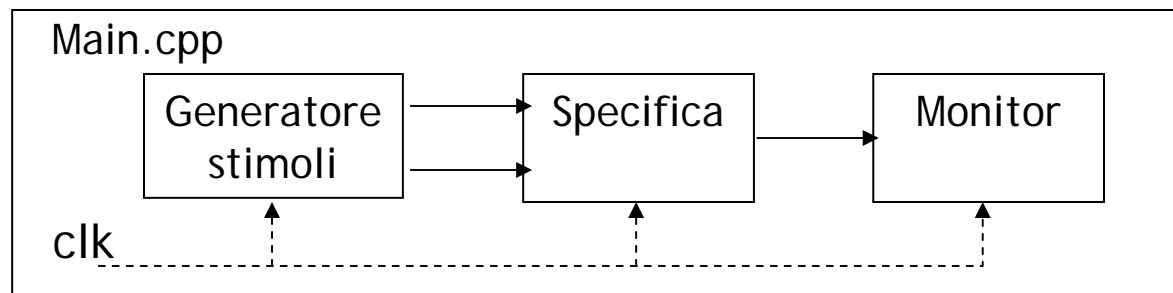
Segnali che collegano gli elementi
Dichiarati nel modulo



□ Moduli porte e segnali

- Modulo

- È un contenitore costituito da una interfaccia e da una funzionalità
- Il modulo interfaccia (.h) ed il modulo funzionalità (.cpp) sono separati
- Un modulo contiene
 - Porte, variabili segnali interne, variabili dati interni, processi (diversi tipi), altri metodi, istanze di altri moduli, costruttore





SystemC

□ Moduli porte e segnali (cont.)

- Porte

- Passano dati da e verso il modulo; sono dichiarate nella sezione di interfaccia
- Tre tipi
 - input, output, inout
- Il tipo di dato è passato come parametro di riferimento alla porta.

```
SC_MODULE(nome) {  
    sc_in<int> a;  
    sc_out<bool> b;  
    ...  
};
```



SystemC

□ Moduli porte e segnali (cont.)

- Segnali

- Scambio di dati parallelo all'interno del modulo; sono dichiarati nella sezione di interfaccia
- Esiste solo un tipo di segnale
- Il tipo di dato è passato come parametro di riferimento al segnale.

```
SC_MODULE(nome) {  
    // ports  
    sc_in<int> a;  
    sc_out<bool> b;  
    // signals  
    sc_signal<sc_int<10> > c;  
    ...  
};
```



SystemC

- Moduli porte e segnali (cont.)
 - Per leggere un valore da una porta di utilizza sia il metodo `.read()` sia l'operatore di assegnamento
 - Per scrivere un valore da una porta di utilizza sia il metodo `.write()` sia l'operatore di assegnamento
 - I due metodi sono utilizzati nel modulo funzionalità e agiscono su di un segnale dichiarato nel modulo di interfaccia
 - Interfaccia: `sc_signal<int> dato;`
`sc_signal<bool> condizione;`
`int a;`
 - Funzionalità: `a=dato.read();`
`condizione.write(a);`



□ Moduli porte e segnali (cont.)

- Esempio - Interfaccia

```
SC_MODULE(encode)
{
    //PORTE
    sc_in_clk clock;      sc_in< bool > reset;
    sc_in< bool > input;  sc_out< sc_bv<3> > output;
    //VARIABILI
    sc_bv<8> trellis;    sc_bv<3> tmp;  sc_bv<8> input1;

    //PROTOTIPO FUNZIONE
    void codeGen();

    //COSTRUTTORE
    SC_CTOR(encode)
    {
        SC_CTHREAD(codeGen, clock.pos());
        watching(reset.delayed() == true);
    }
};
```



□ Moduli porte e segnali (cont.)

- Esempio - Funzionalità

```
#include "encode.h"
void encode::codeGen()
{
    //INIZIALIZZAZIONE
    trellis=0x00;
    wait();

    //PROCEDURA
    while(true) {
        input1[0]=input.read(); //Lettura ingresso
        trellis = ((trellis)<<1)|input1; //Inserimento nuovo valore

        //Calcolo codifica
        tmp[2]=trellis[7]^trellis[4]^trellis[2]^trellis[0];
        output.write(tmp); //Scrittura uscita
        wait();
    }
}
```



SystemC

□ Processi

- I processi utilizzano i segnali per comunicare
- Tre tipi di processi
 - SC_METHOD (funzione asincrona)
 - È sensibile ad un insieme di segnali
 - » Sensitivity list: sensitive(segnale1), sensitive<<s1<<s2<<s3, sensitive_pos<<clk, sensitive_neg<<clk
 - Ogni volta che viene invocato, l'intera funzionalità è eseguita e le istruzioni che la compongono sono eseguite sequenzialmente in tempo infinitesimo (rispetto al tempo di simulazione)



SystemC

□ Processi

- Tre tipi di processi (cont.)
 - SC_THREAD (Thread asincrono)
 - È sensibile ad un insieme di segnali
 - » Sensitivity list: `sensitive(segnale1),`
`sensitive<<s1<<s2<<s3, sensitive_pos<<clk,`
`sensitive_neg<<clk`
 - Ogni volta che viene invocato, la funzionalità è eseguita in tempo infinitesimo fino al primo `wait()`; le istruzioni sono eseguite sequenzialmente.
 - » Alla successiva riattivazione l'esecuzione ricomincerà da dopo il costrutto `wait()` su cui si era fermata.
 - Le variabili interne preservano il valore
 - Utilizzato per modellare sia comportamenti asincroni sia comportamenti sincroni.



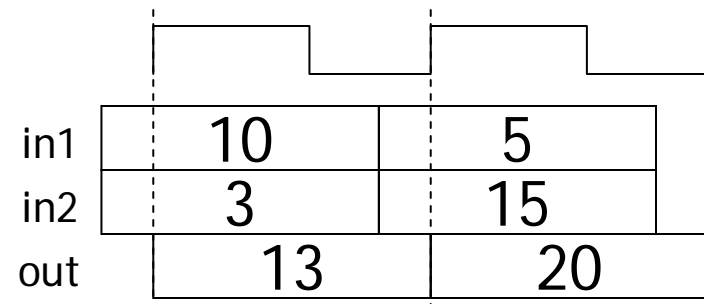
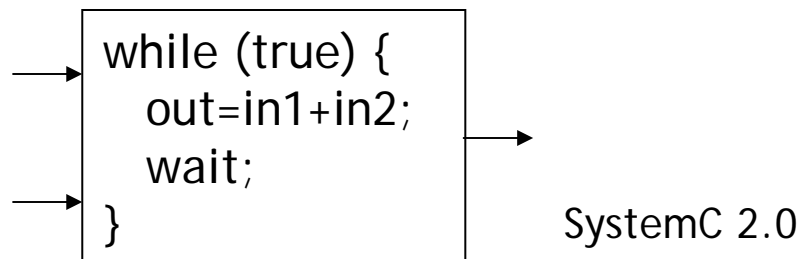
SystemC

□ Processi

- Tre tipi di processi (cont.)

• SC_CTHREAD (Thread sincrono)

- È sensibile solo ad un fronte di un solo clock
- Ogni volta che viene invocato, la funzionalità è eseguita in tempo infinitesimo fino al primo `wait()`; le istruzioni sono eseguite sequenzialmente.
 - » Alla successiva riattivazione, l'esecuzione ricomincerà da dopo il costrutto `wait()` su cui si era fermata.
- Le variabili interne preservano il valore
- Utilizzato per modellare comportamenti sincroni.





SystemC

□ Processi

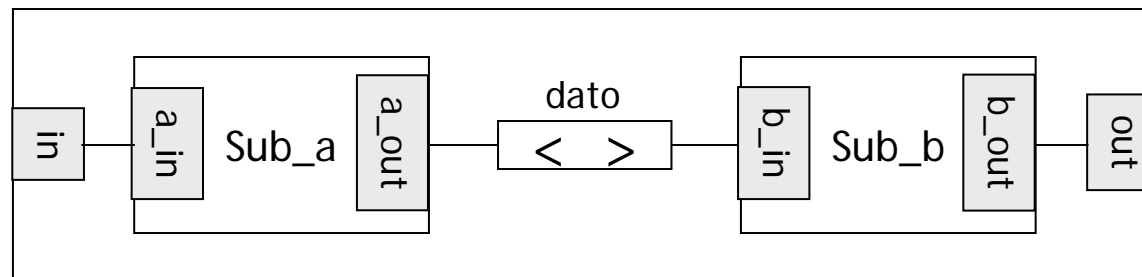
- Funzione WAIT()

- Usata solo in SC_THREAD e SC_CTHREAD
- Sospende l'esecuzione del processo fino a quando il processo non viene invocato nuovamente
- SystemC 1.0
 - wait()
 - wait(<var_int>) - attesa per un numero definito di cicli
- SystemC 2.0
 - wait(event)
 - wait(e1 | e2 | e3) - attesa del primo degli eventi
 - wait(e1 & e2 & e3) - attesa di tutti e tre
 - wait(100, SC_NS, e1 | e2) - attesa con time-out



□ Gerarchia

- I moduli possono contenere istanze di altri moduli
 - Nell'interfaccia
 - Si definiscono le variabili di connessione tra i moduli
 - Si definiscono variabili che sono puntatori alle classi dei moduli per tutte le istanze
 - Nel costruttore si creano le istanze con `new`
 - Nel costruttore si leggono i segnali e si realizza il mappaggio





□ Gerarchia

- Esempio

```
SC_MODULE(esempio)
{
  sc_in<int> in; sc_out<int> out;
  sc_signal<int> dato;
```

```
  Sub_a *Istanza1; Sub_b *Istanza2;
```

```
  SC_CTOR(esempio)
```

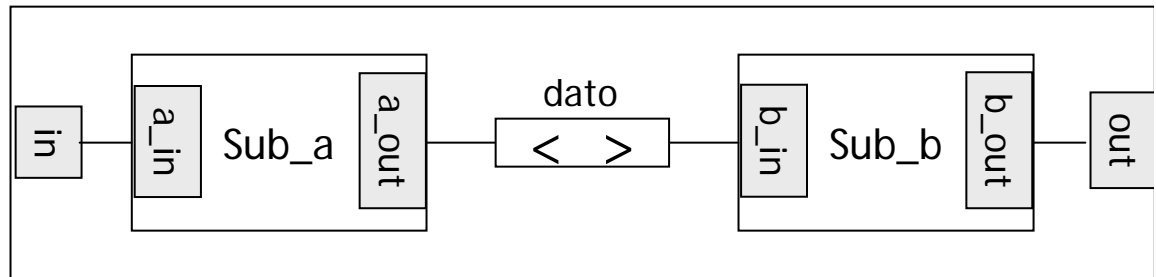
```
  {
    Istanza1= new Sub_a("istanza1_arbitrario");
    Istanza2= new Sub_b("istanza2_arbitrario");

    (*Istanza1).a_in(in); (*Istanza1).a_out(dato);
    (*Istanza2).b_in(dato); (*Istanza2).b_out(out);
```

```
    SC_CTHREAD(codeGen,clock.pos());
    watching(reset.delayed() == true);
```

```
  }
```

```
};
```





Bibliografia

- Timing Tool, "TimingTool", manuale applicativo, versione 1.3, Novembre 2003
- B. Niemann, "An introduction to SystemC 1.0.x", Appunti del corso, Fraunhofer Institute for Integrated Circuit, 2001
- AA.VV., "SystemC 2.0 User's guide update for SystemC 2.0.1", Synopsys, 2002