

---

# *SystemC*

---

## Sommario

---

- Introduzione
  - SystemC
    - ▶ Transaction Level Modeling
  - SystemC
    - ▶ Elementi del linguaggio
  - SystemC
    - ▶ Esempio
  - Bibliografia
-

## *Introduzione*

## Introduzione

- Il Progetto di un sistema digitale inizia dalla raccolta dei requisiti e dal loro successivo raffinamento
  - ▶ 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

## Introduzione

- **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 formale
    - Il linguaggio naturale è semi-formale
      - Mescolare l'impasto con il cucchiaino
      - Mescolare l'impasto con la panna

- 5 -

## Introduzione

- **Classificazione dei linguaggi formali**
  - ▶ **Linguaggi Grafici**
    - Ingegneria del software
      - Statecharts, UML, ...
    - Ingegneria dell'hardware
      - Schematic Entry, Timing Diagrams
      - HHL (Heterogeneous Hardware Logic)
    - Ingegneria di sistema
      - LSC (Live Sequence Charts)
  - ▶ **Linguaggi Testuali**
    - Linguaggi basati su linguaggi di programmazione
      - SystemC, VHDL, HardwareC, HandelC
    - Linguaggi orientati alla descrizione dell'hardware
      - VHDL, VERILOG

- 6 -

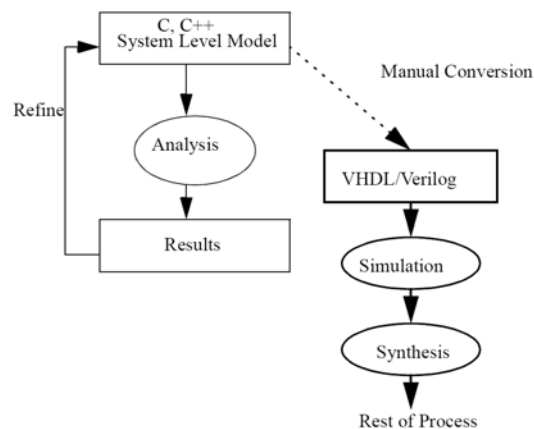
## Introduzione

- I sistemi stanno diventando sempre più complessi e spesso sono composti sia da parti hw che sw
  - ▶ C'è la necessità di alzare il livello di astrazione per
    - Effettuare un'iniziale esplorazione dello spazio delle soluzioni
    - Effettuare la verifica iniziale dell'intero sistema riducendo i rischi
    - Simulare in maniera più veloce
- Occorrono degli strumenti in grado di facilitare il passaggio dalla fase di concettualizzazione a quella di progettazione
  - ▶ Ridurre il "divario produttivo"

- 7 -

## Introduzione

- Classico flusso di sviluppo



- 8 -

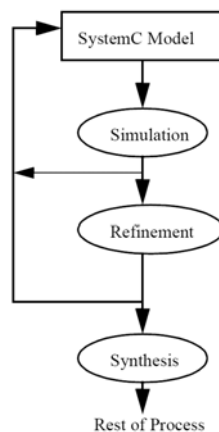
## Introduzione

- Classico flusso di sviluppo
  - ▶ Si definisce un modello C/C++ dell'intero sistema, che viene verificato e validato
  - ▶ Si realizzano poi le parti HW e SW del sistema
    - La realizzazione della parte HW viene effettuata tramite una traduzione manuale
    - Questo processo ha degli svantaggi
      - La conversione manuale genera errori
      - I modelli C/C++ e HDL sono slegati
      - Dopo aver creato il modello HDL le modifiche saranno di solito effettuate esclusivamente su quest'ultimo
    - La realizzazione della parte SW può essere effettuata estraendo il codice dal modello di sistema, adattandolo per l'RTOS da usare

- 9 -

## Introduzione

- Flusso di sviluppo con SystemC



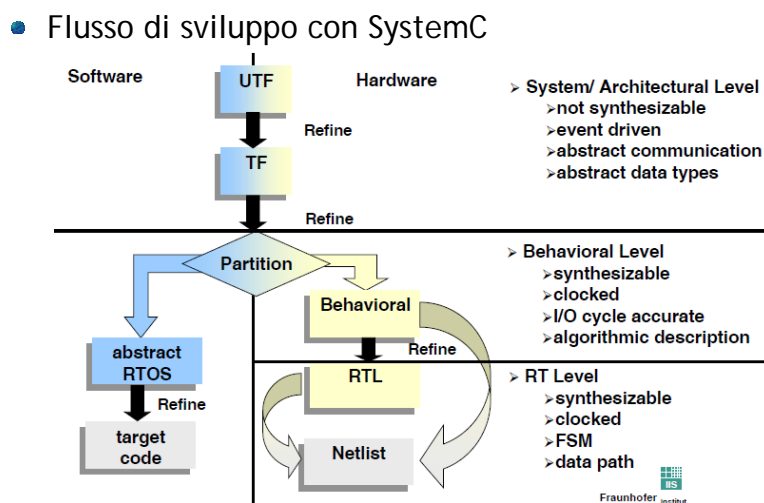
- 10 -

## Introduzione

- Flusso di sviluppo con SystemC
  - ▶ Questo nuovo flusso di progetto prevede la possibilità di modellare il sistema in SystemC e procedere fino alla sintesi hardware.
  - ▶ Questo approccio presenta numerosi vantaggi
    - Non si passa dal modello C a HDL in un singolo passo, ma tramite raffinamenti progressivi
    - Il sistema è scritto in un solo linguaggio che permette più livelli di astrazione
    - È possibile riutilizzare i testbench durante le fasi di raffinamento
    - È possibile simulare il sistema più velocemente
  - ▶ La modellazione dell'RTOS non è ancora possibile, ma dovrebbe essere inserita nelle prossime versioni di SystemC (SystemC 3.0)

- 11 -

## Introduzione



- 12 -

## Introduzione

- SystemC è un linguaggio per il sistem level design, basato su C++
  - ▶ C++ arricchito da una libreria di classi
- C++ è stato scelto come linguaggio su cui basare SystemC poiché
  - ▶ C/C++ sono linguaggi conosciuti e usati dai progettisti hw, progettisti sw e di sistema
  - ▶ Esistono una serie di tool ben consolidati (compilatori, debugger...)
  - ▶ L'integrazione di modelli SystemC con codice C/C++ esistente è semplice

- 13 -

## Introduzione

- Il linguaggio C++ ha delle limitazioni per la descrizione di sistemi complessi in quanto non supporta:
  - ▶ Nozione di tempo
  - ▶ Concorrenza tra processi
  - ▶ Tipi di dato hw (ad esempio valore Z)
- SystemC permette di superare queste limitazioni poiché permette
  - ▶ Comunicazioni hardware, modello del tempo
  - ▶ Concorrenza, tipi di dati per l'hardware
- La specifica è eseguibile
  - ▶ Kernel di simulazione integrato con la specifica

- 14 -

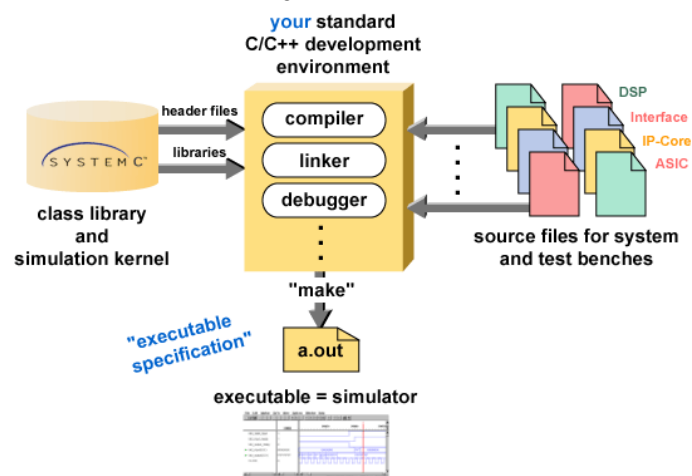
## Introduzione

- La libreria SystemC è disponibile gratuitamente grazie all'OCSI (Open SystemC Initiative), organizzazione formata dalle principali aziende del settore EDA e da università
  - ▶ È possibile scaricare la libreria da [www.systemc.org](http://www.systemc.org)
- Sono disponibili le seguenti versioni della libreria:
  - ▶ SystemC 1.0.2: modellazione RTL
  - ▶ SystemC 2.0, 2.0.1, 2.1: modellazione di sistema
- Inoltre esistono alcune librerie costruite a partire da SystemC
  - ▶ TLM (Transaction Level Modeling) Library, SCV (SystemC Verification Library) Library e Master Slave Library

- 15 -

## Introduzione

- Utilizzo della libreria SystemC

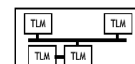
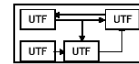


- 16 -



## Introduzione

- **SystemC 1.0**
  - ▶ Modella l'hardware (RTL e Comportamentale)
- **SystemC 2.0**
  - ▶ Estende la modellazione a livello di sistema
    - UTF (UnTimed Functional level)
      - Modello funzionale
        - » Specifica eseguibile senza caratteristiche di tempo
    - TLM (Transaction Level Modeling)
      - Modello transazionale
        - » 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



- 17 -

## Introduzione

- 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 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

- 18 -

# SystemC

## *Transaction Level Model*

# SystemC

- Transaction Level Modeling
  - ▶ Il livello di astrazione Transaction Level aiuta il System Level Design
    - Separando la modellizzazione del comportamento delle “unità funzionali” (moduli) dalla loro comunicazione
    - Fornendo un unico formalismo per descrivere la comunicazione a più livelli di dettaglio
    - Consentendo di modellizzare molti modelli di computazione diversi

## SystemC

- Transaction Level Modeling

- 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)

- 21 -

## SystemC

- Transaction Level Modeling

- Cosa vs. Come

- Concettualmente, la modellizzazione di una transaction avviene rispondendo a due domande
    - Quale dato viene trasferito? (tipo, valore)
      - » Es: il processore legge una parola dalla memoria all'indirizzo "0x00ef00ef"; il valore restituito è "0xaabbaabb"
    - 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!

- 22 -

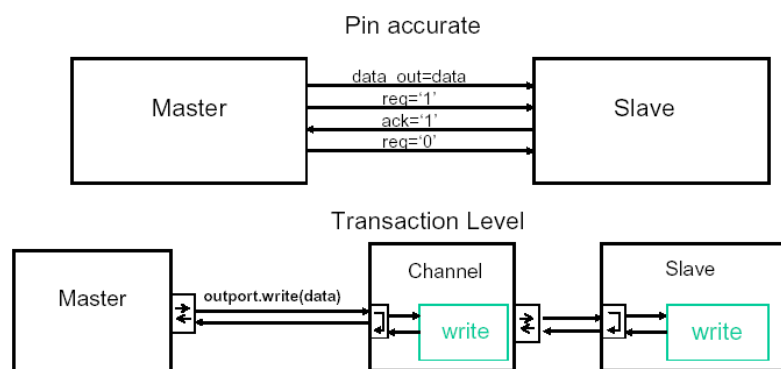
## SystemC

- Transaction Level Modeling
  - ▶ Nel Transaction Level Modelling, la comunicazione può essere descritta a molti livelli di dettaglio temporale
    - *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

- 23 -

## SystemC

- Transaction Level Modeling
  - ▶ Transaction Level vs. Pin Level



- 24 -

## SystemC

- Transaction Level Modeling
  - Confronto tra 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

- 25 -

## SystemC

*Elementi del linguaggio*

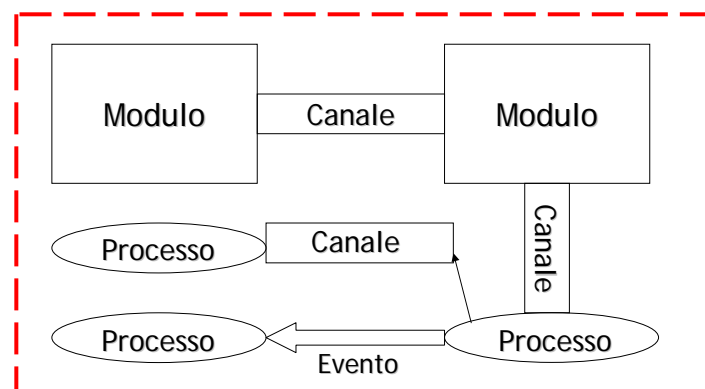
## SystemC

- Elementi del linguaggio
  - ▶ Struttura
  - ▶ Tipi di dati
  - ▶ Interfacce e canali
  - ▶ Moduli
  - ▶ Processi
  - ▶ Gerarchia
  - ▶ Esempi

- 27 -

## SystemC: Sommario

- Struttura



- 28 -

## Struttura

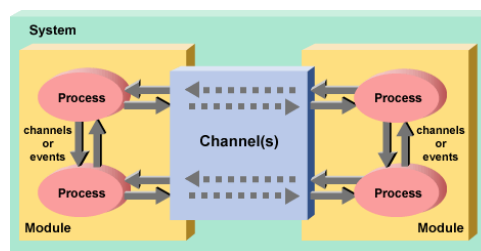
- Struttura della libreria

<b>Methodology-Specific Libraries</b> Master/Slave library, etc	<b>Layered Libraries</b> Verification library TLM library, etc
<b>Primitive Channels</b> Signal, Fifo, Mutex, Semaphore, etc	
<b>Structural Elements</b> Modules Ports Interfaces Channels	<b>Data Types</b> 4-valued logic Bits and Bit Vectors Arbitrary Precision Integers Fixed-point types
<b>Event-driven Simulation</b> Events Processes	
<b>C++ Language Standard</b>	

- 29 -

## Struttura

- Struttura di un sistema in SystemC

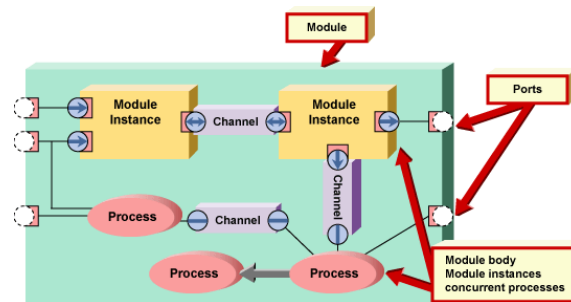


- Il sistema è costituito da **moduli**
  - Il comportamento dei moduli è descritto da **processi** che comunicano attraverso **eventi** o **canali**
  - La comunicazione inter-modulo avviene tramite canali

- 30 -

## Struttura

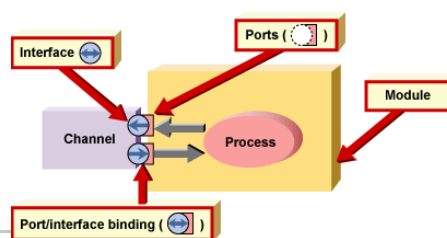
- Istanze di moduli a uno stesso livello sono connesse tramite porte ai canali
  - ▶ I processi sono connessi tramite canali o eventi
- I moduli supportano gerarchia
  - ▶ Le connessioni avvengono attraverso le porte



- 31 -

## Struttura

- Un'interfaccia definisce la signature di un insieme di metodi accessibili
  - ▶ Le interfacce sono implementate dai canali
  - ▶ Le interfacce sono collegate alle porte dei moduli
- I canali possono essere sviluppati indipendentemente dai moduli (devono garantire l'interfaccia, non l'implementazione)



- 32 -



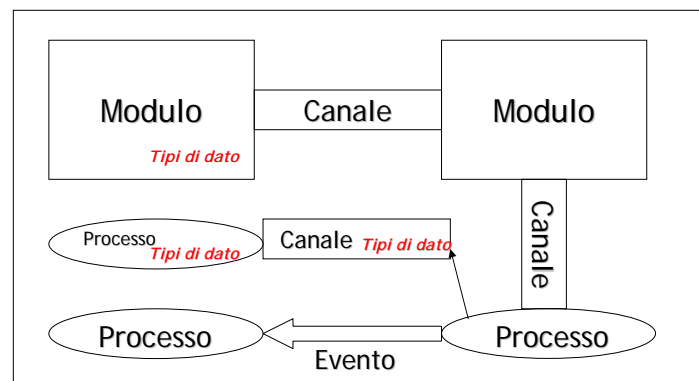
## Struttura

- Gli eventi (sc\_event) sono gli elementi base di sincronizzazione
  - ▶ I processi sono eseguiti e i loro output aggiornati in base agli eventi
    - La libreria SystemC include uno scheduler che gestisce l'esecuzione dei processi
- Un processo è attivato dagli eventi della sua sensitivity list
  - ▶ È possibile creare sensitivity list
    - Statiche: definite prima dell'inizio della simulazione
    - Dinamiche: definite a runtime
    - Un processo può aspettare l'accadimento di un evento
      - wait()

- 33 -

## SystemC: Sommario

- Tipi di dato



- 34 -

## Tipi di dato

- Tipi di dati
  - ▶ SystemC permette di utilizzare sia i tipi nativi del C/C++, sia dei tipi definiti in libreria
  - ▶ A seconda dei tipi di dato utilizzati cambiano le caratteristiche di velocità della simulazione
  - ▶ I tipi numerici possono essere distinti in
    - Precisione fissa
    - Precisione arbitraria
    - Logica a 4 valori
    - Vettori con logica a 4 valori
    - Tipi fixed point

- 35 -

## Tipi di dato

- `sc_uint<>`, `sc_int<>`: tipi interi signed e unsigned
  - ▶ Es: `sc_uint<8> nome_variabale`
  - ▶ Sono rappresentati con un array di lunghezza specificata dall'utente (al massimo 64 bit)
  - ▶ Il bit più a destra è il meno significativo, quello più a sinistra il più significativo
  - ▶ Possono essere usati per operazioni matematiche e logiche
  - ▶ Tutte le operazioni sono effettuate su 64 bit e poi convertite nel formato corretto con un troncamento
  - ▶ Per i numeri signed è usato il complemento a due
  - ▶ L'uso di questi dati è meno efficiente all'uso dei tipi built-in in C++

- 36 -

## Tipi di dato

- `sc_biguint<>`, `sc_bigint<>` : tipi interi signed e unsigned a lunghezza arbitraria
  - ▶ Es: `sc_biguint<80> nome_variabile`
  - ▶ Sono utili per rappresentare valori interi con lunghezza maggiore di 64 bit
- `sc_logic`: sono usati per rappresentare logica a 4 valori ('0', '1', 'Z', 'X')
  - ▶ Es: `sc_logic nome_variabile`
  - ▶ Si usano per rappresentare un solo bit
  - ▶ L'uso di questi tipi risulta meno efficiente rispetto all'uso di logica a 2 valori (`sc_int`, `sc_uint`)

- 37 -

## Tipi di dato

- `sc_lv<>`: array di bit di tipo `sc_logic`
  - ▶ `sc_lv<8> nome_variabile`
- `sc_fixed<>`, `sc_ufixed<>`: tipi fixed point
  - ▶ Possono essere utili per raffinare modelli con dati iniziali in floating point

- 38 -

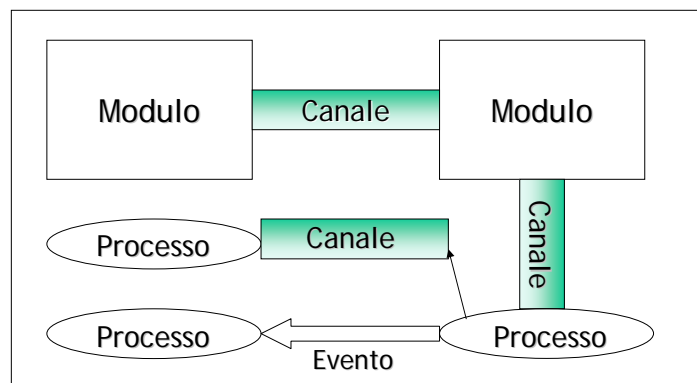
## Tipi di dato

- Occorre prestare attenzione ai tipi che si scelgono per costruire i modelli
  - ▶ I tipi introdotti da SystemC possono modellare in modo preciso i comportamenti desiderati, ma sono dispendiosi in tempo di simulazione
- Per rendere il modello efficiente occorre
  - ▶ Usare il più possibile i tipi nativi del C/C++ (bool, unsigned char, unsigned short char) se ho logica a due valori
  - ▶ Usare sc\_uint, sc\_int se la logica è a 2 valori e si usano meno di 64 bit
  - ▶ Usare sc\_logic, sc\_lv se la logica è a 4 valori

- 39 -

## SystemC: Sommario

- Interfacce e Canali



- 40 -

## Interfacce

- Le interfacce definiscono un insieme di metodi di accesso
  - ▶ L'implementazione delle interfacce è definita dai canali
- Esistono un insieme di interfacce predefinite in SystemC
  - ▶ `sc_signal_in_if`, `sc_signal_inout_if`, `sc_fifo_in_if`, `sc_fifo_out_if`, `sc_mutex_if`, `sc_semaphore_if`
- Ciascuna interfaccia definisce dei metodi virtuali
  - ▶ Ad es. il metodo `write()` sarà definito nelle interfacce `out` e non nelle `read`
- Se un canale implementa un'interfaccia deve implementare tutti i metodi

- 41 -

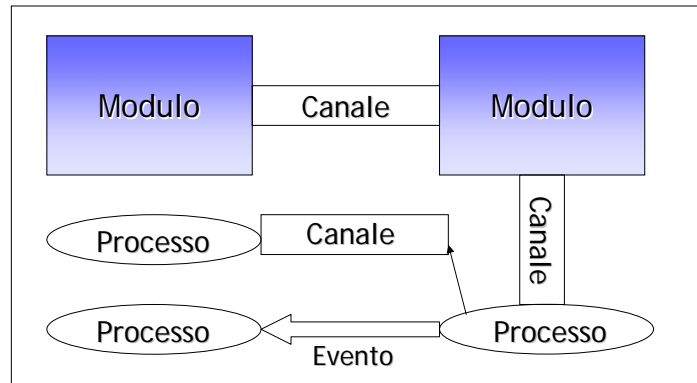
## Canali

- I canali sono distinti in
  - ▶ Primitivi: non hanno una struttura visibile e dei processi, non possono accedere a altri canali primitivi
  - ▶ Gerarchici: sono dei moduli; possono contenere processi e accedere ad altri canali
    - I canali primitivi sono più semplici e perciò aumentano la velocità di simulazione rispetto ai canali gerarchici
- I canali primitivi previsti da SystemC sono
  - ▶ `sc_signal`, `sc_signal_rv`
  - ▶ `sc_fifo`, `sc_mutex`
  - ▶ `sc_semaphore`, `sc_buffer`

- 42 -

## SystemC: Sommario

- Moduli



- 43 -

## Moduli

- Moduli

- ▶ Sono contenitori costituiti da una interfaccia e da una funzionalità
  - Il modulo interfaccia (.h) ed il modulo funzionalità (.cpp) sono separati
- ▶ Un modulo è costituito da:
  - Porte
  - Canali interni
  - Variabili interne
  - Processi (diversi tipi)
  - Altri metodi
  - Costruttore
  - Istanze di altri moduli

- 44 -

## Moduli

- Un modulo è dichiarato tramite la macro SC\_MODULE

```
SC_MODULE ( module_name) {  
    // body of module  
};
```

- ▶ Per accedere al modulo si usano le porte che sono definite nella classe base sc\_port e sono legate a un'interfaccia
  - Le porte possono essere di tre tipi
    - In
    - Out
    - Inout
  - Sintassi: sc\_port<interface\_type, N> port\_name;
    - N è il numero di canali che può essere connesso a una porta

## Moduli

- Un modulo può dichiarare dei canali interni
  - ▶ I canali interni sono usati per
    - Connettere moduli allo stesso livello di gerarchia
    - Connettere processi all'interno di un modulo
    - Connettere un processo di un modulo padre con la porta di un modulo figlio
- Un modulo può dichiarare delle variabili interne
- Un modulo può dichiarare funzioni e processi interni
  - ▶ I processi sono delle particolari funzioni registrate nel costruttore del modulo

## Moduli

- Il modulo ha un costruttore usato per creare l'istanza di un modulo e la struttura dati interna
  - ▶ Inizializza variabili e segnali interni
  - ▶ Definisce i processi

```
SC_CTOR(my_module) {  
    SC_METHOD(internal_process); //Method Process  
    sensitive << in_p;  
    internal_variable= 1;  
}
```

## Moduli

- Per i canali `sc_signal`, `sc_signal_rv`, `sc_fifo` esistono porte specializzate per abbreviare le dichiarazioni
  - ▶ `sc_in<T>`, `sc_out<T>`, `sc_inout<T>` si usano per gli `sc_signal`
    - T è un tipo qualunque
  - ▶ `sc_in_rv<T>`, `sc_out_rv<T>`, `sc_inout_rv<T>` si usano per gli `sc_signal_rv`
  - ▶ `sc_fifo_in<int>`, `sc_fifo_out<T>`, `sc_fifo_inout<T>` si usano per i canali `sc_fifo`



## Moduli

- 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 una porta dichiarata nel modulo di interfaccia
    - Interfaccia
      - sc\_signal<int> dato;
      - sc\_signal<bool> condizione;
      - int a;
    - Funzionalità
      - a=dato.read();
      - condizione.write(a);

- 49 -

## Moduli

### • Esempio - Interfaccia

```
SC_MODULE(encode){
    sc_in<bool> clock; //Porte
    sc_in< bool > reset;
    sc_in< bool > input;
    sc_out< sc_bv<3> > output;
    sc_bv<8> trellis; //Variabili
    sc_bv<3> tmp;
    sc_bv<8> input1;
    void codeGen(); //Prototipo funzione
    SC_CTOR(encode) // Costruttore
    {
        SC_CTHREAD(codeGen,clock.pos());
        watching(reset.delayed() == true);}
};
```

- 50 -

## Moduli

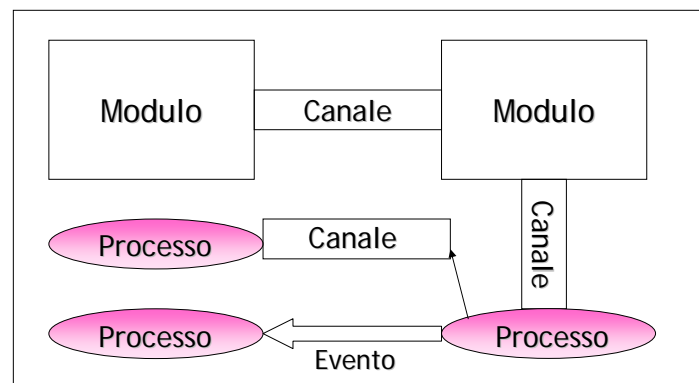
- Esempio - Fuzionalità

```
#include "encode.h"
void encode::codeGen(){
    trell=0x00; //INIZIALIZZAZIONE
    wait();
    //PROCEDURA
    while(true){
        input1[0]=input.read();
        trell=((trell)<<1)|input1;
        tmp[2]=trell[7]^trell[4]^trell[2]^trell[0];
        output.write(tmp); //Scrittura uscita
        wait();}
}
```

- 51 -

## SystemC: Sommario

- Processi



- 52 -

## Processi

- Gli eventi sono un oggetto base per la sincronizzazione tra due processi
- Sintassi: `sc_event event_name`
- Un evento può essere notificato
  - ▶ In maniera immediata: i processi sensibili all'evento diventano ready nella fase di valutazione del delta-cycle corrente
  - ▶ In maniera ritardata: i processi sensibili all'evento diventano ready nella fase di valutazione del prossimo delta-cycle
  - ▶ Dopo un determinato tempo

- 53 -

## Processi

- Le funzionalità sono descritte nei processi
  - ▶ I processi utilizzano gli eventi o i canali per comunicare
    - Devono essere registrati nei costruttori dei moduli
  - ▶ Esistono tre tipi di processi
    - `SC_METHOD`; `SC_THREAD`; `SC_CTHREAD`
      - Alcuni processi (`SC_METHOD`) si comportano come delle funzioni: sono eseguiti dopo una chiamata e alla terminazione ritornano il controllo al chiamante
      - Altri processi (`SC_THREAD` e `SC_CTHREAD`) si comportano come thread: sono chiamati una sola volta e si sospendono e riattivano in maniera indipendente

- 54 -

## Processi

- 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)

- 55 -

## Processi

- Processi: 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

- 56 -

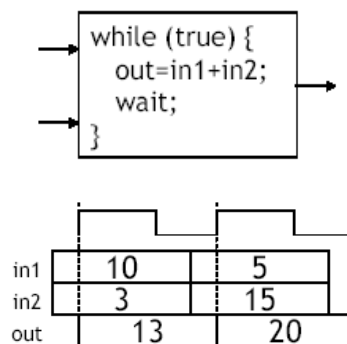
## Processi

- Processi: 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

- 57 -

## Processi

- Processi: SC\_CTHREAD (thread sincrono)
  - ▶ Esempio SystemC 2.0



- 58 -

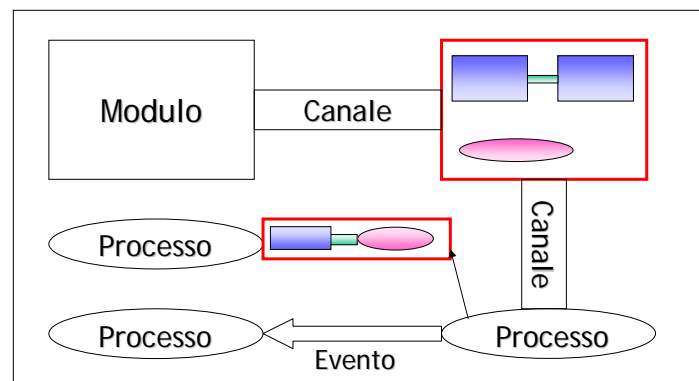
## SystemC

- 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

- 59 -

## SystemC: Sommario

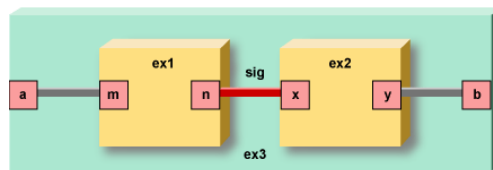
- Gerarchia



- 60 -

## Gerarchia

- Attraverso i moduli è possibile costruire una struttura gerarchica
  - ▶ Come in VHDL è possibile collegare le porte del modulo padre direttamente alle porte del modulo figlio, occorrono invece dei canali per collegare due moduli allo stesso livello



- 61 -

## Gerarchia

```
SC_MODULE(ex3){
    sc_port<sc_fifo_in_if<int> >a;
    sc_port<sc_fifo_out_if<int> > b;
    sc_fifo<int> sig1;
    // Instances of ex1 and ex2
    ex1 ex1_instance;
    ex2 ex2_instance;
    // Module Constructor
    SC_CTOR(ex3):
    ex1_instance("ex1_instance"),
    ex2_instance("ex2_instance")
    {
        // Named connection for ex1
        ex1_instance.m(a);
        ex1_instance.n(sig1);
        // Positional connection for ex2
        ex2_instance(sig1, b);
    }
};
```

Per creare una struttura gerarchica occorre:

- Creare l'istanza del modulo
- Inizializzare le istanze dei moduli
- Effettuare il bind delle porte

Il bind può essere effettuato:

- Per nome
- Per posizione

- 62 -

## Gerarchia

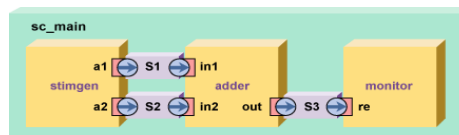
- Il top level è costituito da una speciale funzione `sc_main()`, che viene chiamata dal kernel di simulazione all'inizio dell'esecuzione
  - ▶ La procedura per istanziare dei moduli nel main è simile a quella per istanziare dei moduli in un altro modulo
- Alla fine della funzione `sc_main()` si usa la funzione `sc_start()` che permette di iniziare la simulazione
  - ▶ `sc_start()`: esegue per sempre
  - ▶ `sc_start(arg)`: esegue per `arg` unità di tempo

- 63 -

## Gerarchia

```
#include "systemc.h"
#include "adder.h"
#include "stimgen.h"
#include "monitor.h"
int sc_main(int argc, char *argv[ ])
// Create fifos with a depth of 10
sc_fifo<int> s1(10), s2(10), s3(10);

// Module instantiations
stimgen stim("stim");
stim(s1, s2);
// Adder
adder add("add");
add(s1, s2, s3);
// Response Monitor
monitor mon ("mon");
mon.re(s3);
...
sc_start();
}
```



- 64 -



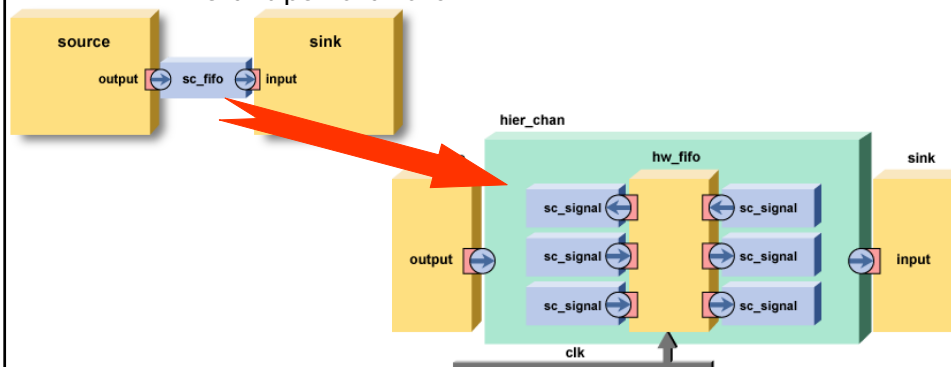
## Canali gerarchici: esempio

- Oltre ai canali primitivi definiti da SystemC è possibile usare dei canali gerarchici
  - ▶ I canali gerarchici sono moduli che possono contenere processi, altri moduli, etc.
    - I canali gerarchici sono più complessi, quindi riducono la velocità di simulazione
    - Devono essere usati solo quando è necessario

- 65 -

## Canali gerarchici: esempio

- Quando è necessario?
  - ▶ Es.: sostituire un canale FIFO con un canale gerarchico che funge da wrapper per una descrizione FIFO di tipo hardware



- 66 -

## Confronto VHDL/SystemC

	VHDL	SystemC
<b>Hierarchy</b>	Entity	Module
<b>Communication</b>	Signal	Signal, Channel
<b>Functionality</b>	Process	Process
<b>TestBench</b>		Object orientation
<b>System Level</b>		Channel, interface, event, abstract data types
<b>I/O</b>	Simple file I/O	C++ I/O capabilities

- 67 -

## Confronto VHDL/SystemC

### ● Flip flop D

```

library ieee;
use ieee.std_logic_1164.all;
entity dffa is
  port( clock : in std_logic;
        reset : in std_logic;
        din   : in std_logic;
        dout  : out std_logic);
end dffa;

architecture rtl of dffa is
begin
  process(reset, clock)
  begin
    if reset = '1' then
      dout <= '0';
    elsif clock'event and clock = '1' then
      dout <= din;
    end if;
  end process;
end rtl;

```

```

// dffa.h
#include "systemc.h"

SC_MODULE(dffa)
{
  sc_in<bool> clock;
  sc_in<bool> reset;
  sc_in<bool> din;
  sc_out<bool> dout;

  void do_ffa()
  {
    if (reset) {
      dout = false;
    } else if (clock.event()) {
      dout = din;
    }
  };

  SC_CTOR(dffa)
  {
    SC_METHOD(do_ffa);
    sensitive(reset);
    sensitive_pos(clock);
  }
};

```

- 68 -

## Confronto VHDL/SystemC

---

- A livello RTL esistono poche differenze tra VHDL e SystemC
  - ▶ La semantica rimane invariata
- SystemC permette molti vantaggi nella descrizione a livello di sistema
  - ▶ Unica descrizione eseguibile
  - ▶ Diversi livelli di astrazione
  - ▶ Velocità di simulazione
  - ▶ ...
- Esistono anche degli svantaggi
  - ▶ La sintesi risulta meno soddisfacente rispetto ai linguaggi HDL

---

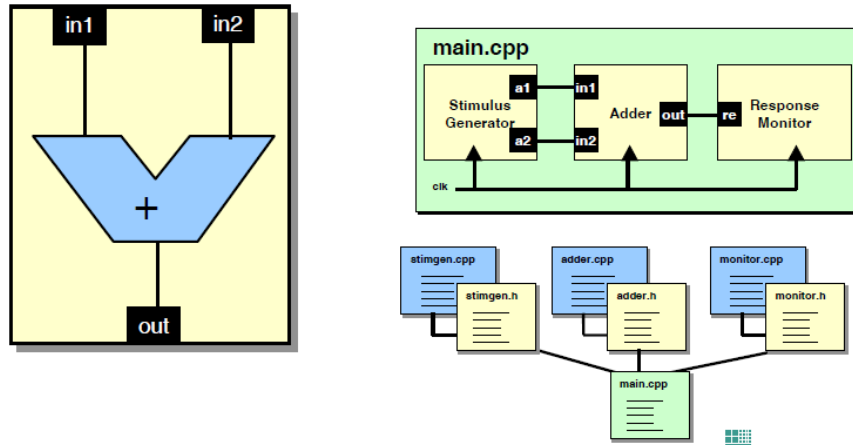
- 69 -

---

*Esempio*

---

## Overview



- 71 -

## Simulation model

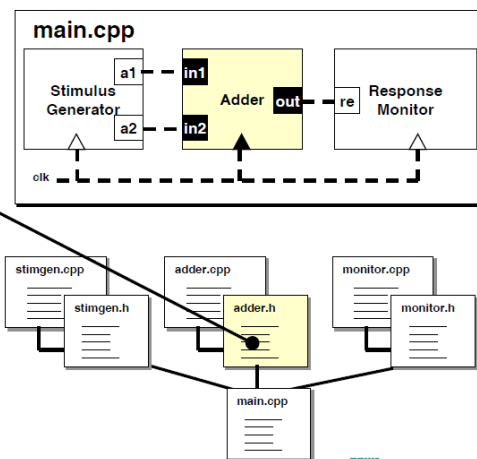
```
// header file adder.h
typedef int T_ADD;

SC_MODULE(adder) {
    // Input ports
    sc_port<sc_fifo_in_if<T_ADD>> in1;
    sc_port<sc_fifo_in_if<T_ADD>> in2;

    // Output ports
    sc_port<sc_fifo_out_if<T_ADD>> out;

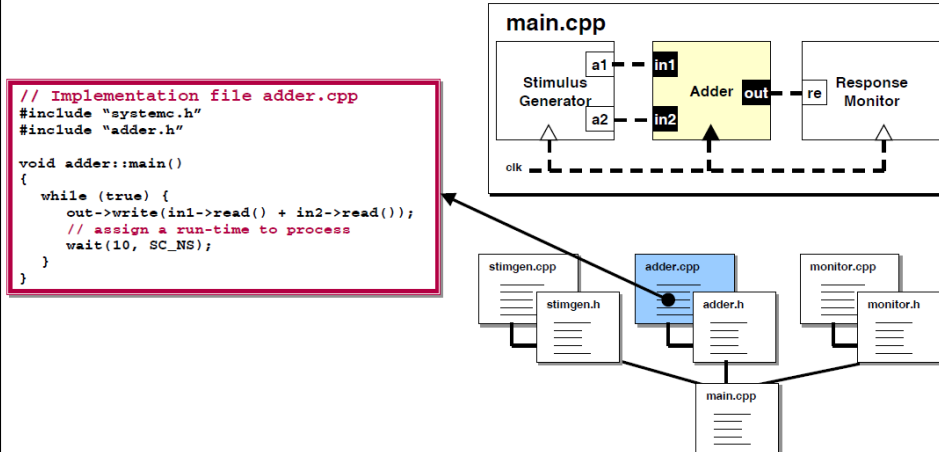
    // Constructor
    SC_CTOR(adder){
        SC_THREAD(main);
    }

    // Functionality of the process
    void main();
};
```



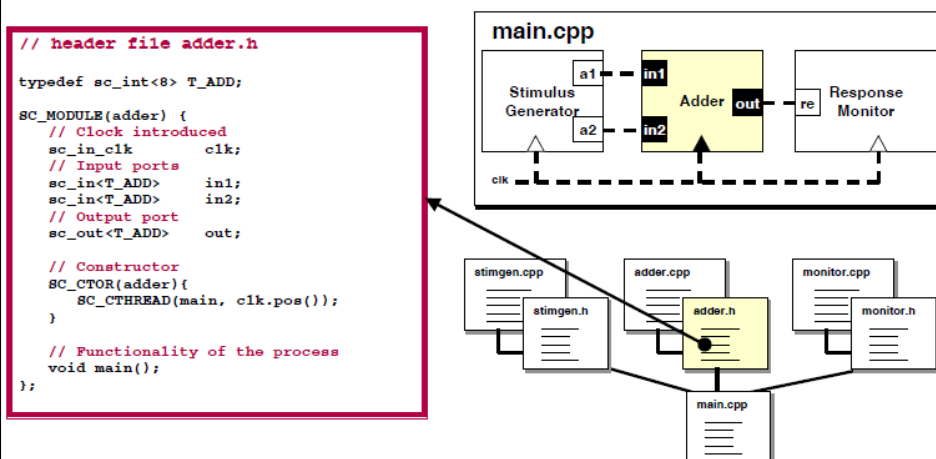
- 72 -

## Simulation model



- 73 -

## Behavioral model



- 74 -

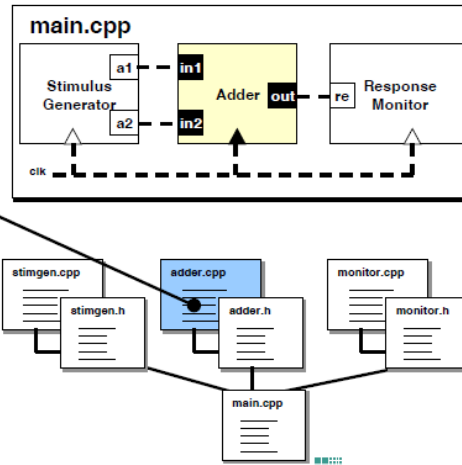
## Behavioral model

```
// Implementation file adder.cc
#include "systemc.h"
#include "adder.h"

void adder::main()
{
    // initialization
    T_ADD __in1 = 0;
    T_ADD __in2 = 0;
    T_ADD __out = 0;

    out.write(__out);
    wait();

    // infinite loop
    while(1) {
        __in1 = in1.read();
        __in2 = in2.read();
        __out = __in1 + __in2;
        out.write(__out);
        wait();
    }
}
```



- 75 -

## RTL model

```
// header file adder.h

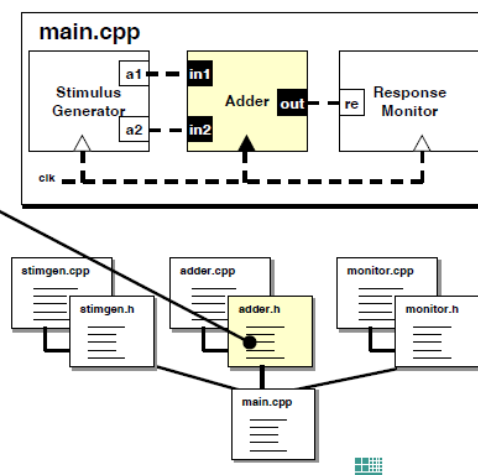
typedef sc_int<8> T_ADD;

SC_MODULE(adder) {
    // Clock introduced
    sc_in_clk clk;
    // Input ports
    sc_in<T_ADD> in1;
    sc_in<T_ADD> in2;
    // Output port
    sc_out<T_ADD> out;

    // internal signal
    sc_signal<T_ADD> sum;

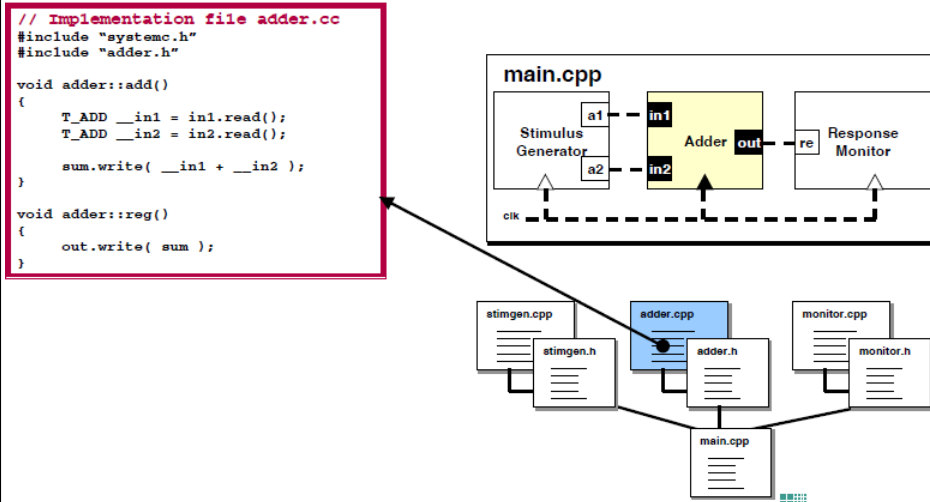
    // Constructor
    SC_CTOR(adder){
        SC_METHOD(add);
        sensitive << in1 << in2;
        SC_METHOD(reg);
        sensitive_pos << clk;
    }

    // Functionality of the process
    void add();
    void reg();
};
```



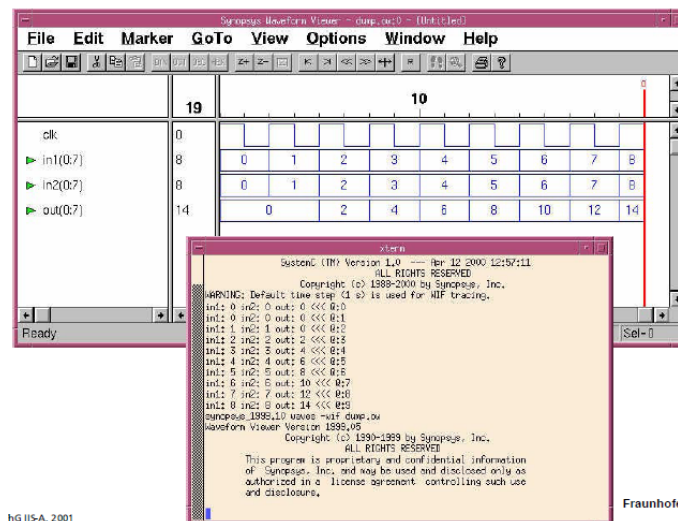
- 76 -

## RTL model



- 77 -

## Screenshot



- 78 -

## Bibliografia

---

- [www.forteds.com](http://www.forteds.com)
- AA.VV., "SystemC 2.0 User's guide update for SystemC 2.0.1", Synopsys, 2002
- AA. VV. "Standard SystemC Language Reference Manual 2.1"
- B. Niemann, "An introduction to SystemC 1.0.x", Appunti del corso, Fraunhofer Institute for Integrated Circuit, 2001
- [www.systemc.org](http://www.systemc.org)