Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Laurea Magistrale in Ingegneria e Scienze Informatiche

PSE

Progettazione di Sistemi Embedded
riassunto dei principali argomenti

Autore:
Danzi Matteo
Matricola VR424987

Anno Accademico 2017/2018

Indice

1 Sistemi Embedded - Introduzione

Definizione: Sistema di Elaborazione specializzato, integrato in un dispositivo fisico in modo tale da controllarne le funzioni tramite un apposito programma SW dedicato.

1.1 Storia

- **Computer ('60 - '80)**: sistemi general purpose.
- **Sistemi di controllo digitale ('80 - '90)**: dedicati al controllo e automazione.
- **Sistemi distribuiti ('90 - '00)**: sistemi general purpose e/o dedicati che cooperano attraverso la rete.
- **Sistemi Embedded ('00 -)**: sistemi distribuiti integrati in oggetti non computazionali e nell'ambiente fisico.
- **Sistemi Ciber-fisici ('10 -)**: sistemi embedded integrati con processi fisici.

1.2 Smart System

Definizione: Sistema Embedded, integrato in una componente nel silicio, che controlla il mondo fisico. Uno smart system ha le seguenti caratteristiche:

- Miniaturizzato - Autosufficiente (autonomo dal p.d.v. energetico)
- Incorpora funzioni di sensore, attuatore, controllore
- Descrive, analizza situazioni, prende decisioni in base ai dati raccolti.
- Ha un comportamento predittivo e attuativo.

Pretective Maintenance: consiste nel riempire la catena di produzione con sensori che controllano e monitorano, cercando di prevenire e predirre guasti in base ai dati rilevati.

2 Come progettare

Per progettare un S.E. non si seguono le stesse direttive e gli stessi principi con cui si progettano sistemi distribuiti general purpose. Ad esempio per quest'ultimi si tende a costruire CPU sempre più veloci, mentre nei sistemi embedded la CPU esiste come mezzo di implementazione di algoritmi di controllo che comunicano con sensori e attuatori.

2.1 Vincoli di progettazione

Durante la progettazione di S.E. ci sono i seguenti vincoli:

- Dimensioni: hand-held electronics (alla portata di mano).
- Peso, potenza: utilizzo di batterie anziché corrente.
- Budget.
- Resistenza del sistema alle condizioni avverse in cui deve operare.

Nella progettazione sono da unire le proprietà di **Calcolo, Controllo, Comunicazione**.

3 Modellazione di Sistemi Embedded

I sistemi elettronici consistono di:

- Piattaforma HW: board di sviluppo;
- Strati di applicazioni SW: in nessun contesto posso permettermi di progettare software da zero, devo riusare parti di software progettate da altri possibilmente opensource;
- Interfacce;
- Componenti analogiche: hano fatto la differenza tra la potenza di calcolo di smartphone della vecchia generazione e quelli di adesso che sono equipaggiati con componenti come accelerometro, riconoscimento impronte, giroscopio ecc.;
- Sensori e trasduttori.

Principalmente si tende a spostare tutta l'elaborazione da analogica a digitale, inoltre si vuole una più ampia integrazione a livello di sistema per supportare l'approccio System-On-a-Chip (SOC). Questo approccio consiste nell'integrare in un chip tutto ciò che normalmente si vede in una board di un componente hardware. Il downside di questo approccio è rappresentato dal fatto che una volta effettuata l'integrazione sul chip non si può più cambiare il componente.

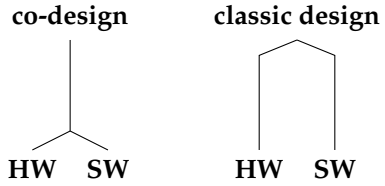
3.1 Sfide nella progettazione di sistemi embedded

- Aumentare la complessità delle applicazioni anche in prodotti standard e di grandi dimensioni
- Aumentare la complessità dei sistemi target

4 Co-design di Hardware/Software

Consiste nella progettazione combinata della parte hardware e della parte software. Gli obiettivi principali del codesign sono:

- Ottimizzazione del processo di progettazione: aumento della produttività
- Ottimizzazione del design: aumento della qualità del prodotto



I compiti del co-design sono:

- co-specification e co-modeling
- co-verification
- co-design process integration e ottimizzazione
- design optimization e co-synthesis

4.1 Vantaggi del co-design

- Permette di esplorare diverse alternative di progettazione nello spazio di progettazione architetturale.
- Permette di adattare l'Hardware al Software e viceversa.
- Riduce il tempo di progettazione del sistema.
- Supporta una coerente specifica del progetto a livello del sistema.
- Facilita il riuso delle parti HW e SW.
- Permette di provvedere di un ambiente integrate per la sintesi e la validazione delle componenti HW e SW.

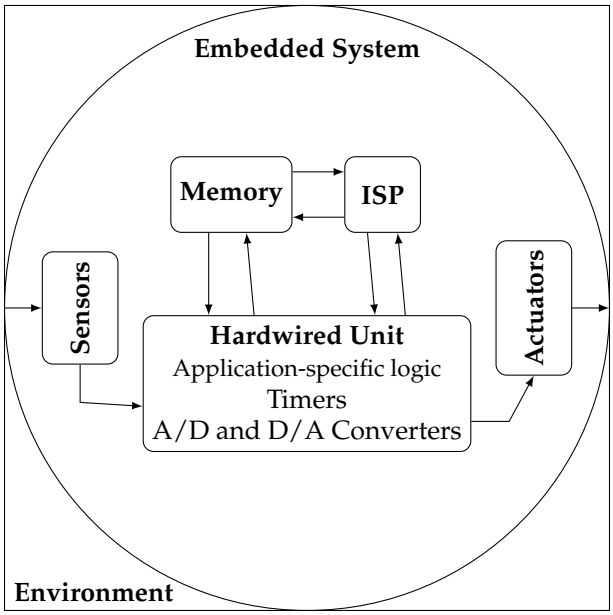
4.2 Co-design di sistemi embedded

Co-progettazione, realizzare un sistema embedded portando avanti il più possibile in contemporanea la sua progettazione hw e sw.
Abbiamo un minor spreco di tempo, è più semplice passare da una descrizione hw a una sw (più facile scegliere cosa diventerà hw e cosa sw), rende più agevolato il riuso di parti sw e hw e rende possibile la co-validazione e la co-simulazione.
Progettazione dedicata di parti HW può comprendere:

- Diversi stili di progettazione: co-processori, embedded core, Application Specific Instruction Processor (ASIP).
- Una scala di progettazione ampiamente variabile.

Progettazione dedicata di parti SW può comprendere:

- Sistemi operativi specializzati (special-purpose).
- Dirver di dispositivi periferici.



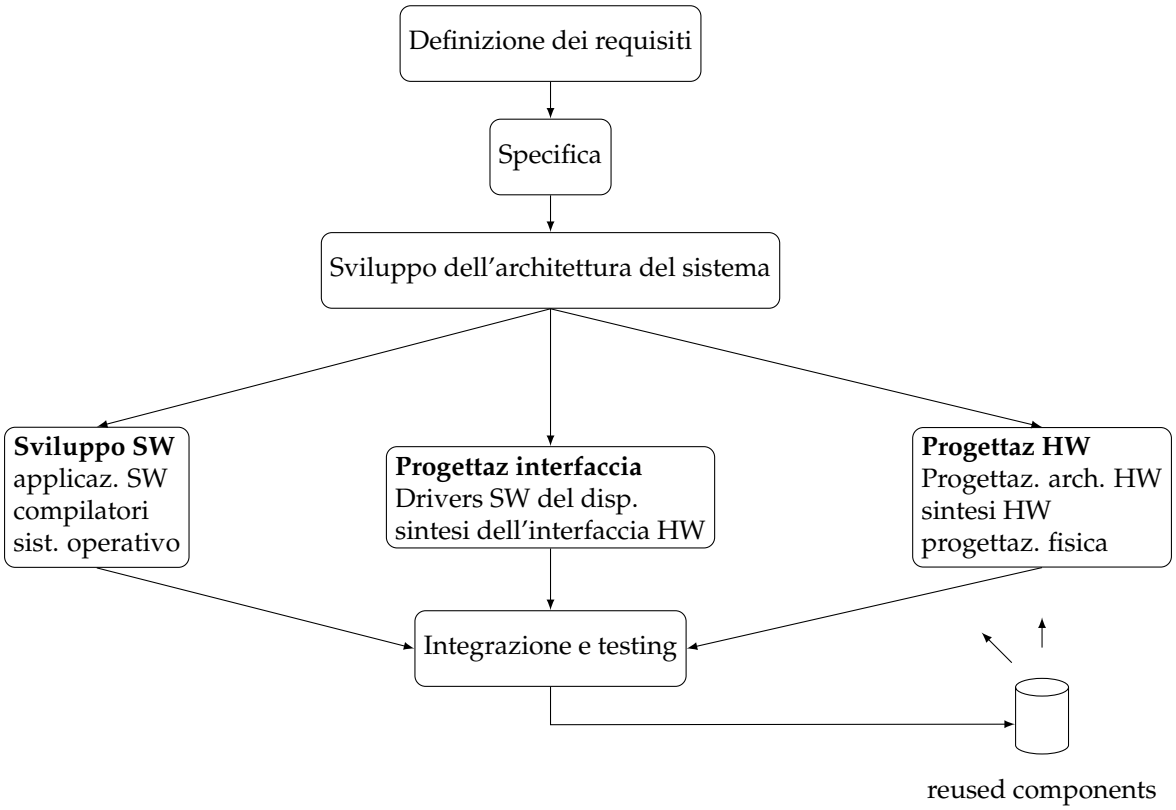
4.3 Array-based design (design basto su array)

ci sono due tipi di dispositivi che si possono progettare con questa tecnica di progettazione:

- **MPGA**: Pre-Diffused Array o più comunemente Mask Programmable Gate Array.
- **FPGA**: Pre-wired Array o più comunemente Field Programmable Gate Array. Sono dispositivi HW (circuito integrato digitale) generici che possono essere programmati con SW dedicato/specifico.

4.4 Flusso di co-progettazione

1. Modellazione, validazione e sintesi: simulazione a livello di sistema.
2. Modellazione omogenea:
   (a) Partizionamento HW/SW
   (b) Spostamento HW/SW o SW/HW
3. Modellazione eterogenea: implementazione diretta e reindirizzamento
4. Co-sintesi
   (c) Sintesi HW e dell'interfaccia
   (d) Compilazione del SW e generazione del codice
5. Co-simulazione



5 Requisiti per un sistema embedded

Un sistema embedded deve essere:

- **REATTIVO**: Non fermarsi mai ed essere sempre pronto a rispondere a segnali provenienti dall'ambiente esterno.
- **REAL TIME**: Rispettare vincoli temporali (HRT, SRT, FRT).

6 Time to market

Tempo che intercorre da quando un prodotto viene ideato a quando viene commercializzato, messo sul mercato per la prima volta.

7 Platform-based design (Progettazione basata sulla piattaforma)

La filosofia platform-based consiste nel progettare architetture HW (piattaforme) stabili, basate su microprocessore, che possano essere facilmente espanse a livello di componenti HW e che sono configurabili a livello SW; quindi sono piattaforme che nonostante abbiano componenti hw ben definite rimangono "a scopo generico" per la loro facile espansibilità, sia HW che SW.

- Dispositivi con gradi di configurabilità (? SW ?)
- Posso usare lo stesso oggetto per più applicazioni
- ASIC
- FPGA

Concetti principali a Livello di Sistema:
Concorrenza, Gerarchia dei moduli, Comunicazione tra sottosistemi, Sincronizzazione.

8 SystemC

SystemC è una libreria che estende il linguaggio di programmazione C++, linguaggio strettamente SW.

Permette di fornire una descrizione del sistema a diversi livelli di astrazione: un livello più vicino all'HW, in cui si definiscono i dettagli implementativi come porte, segnali e interfacce (RTL) e un livello in cui si definisce uno standard per la comunicazione tra i moduli e ci si concentra sulla funzionalità (TLM).

Tali astrazioni rendono possibile fare co-design, cioè una co-progettazione del sistema in cui si cerca di portare avanti il più possibile in parallelo la progettazione hw e quella sw; il co-design diminuisce il tempo di progettazione, rende più facile il passaggio da una componente hw a una sw (quindi la scelta di quale componente implementare come hw e quale come sw) e permette una co-simulazione e una co-validazione.

- Permette di fare co-design.
- Diversi livelli di astrazione (RTL-TLM).
- Linguaggio di definizione dell'hardware (HDL).
- Fornisce una descrizione hw a livello RT (RTL), dove ho un controllo sui segnali, sui registri e sugli operatori utilizzati. Descrivo un sistema con una FSMD.
- Le sue caratteristiche permettono di gestire gli aspetti più importanti per un sistema embedded -> gerarchizzazione dei moduli, concorrenza (processi sync/async), comunicazione (porte/segnali), timing (clock/fasi/tempo), reattività (eventi), tipi di dato hw (oltre ai tipi c++), kernel di simulazione, debug (c++).
- Ha introdotto una standardizzazione del modello transazionale (modo in cui si definiscono le transazioni tra moduli) attraverso il livello di astrazione TLM.
- Riuso di moduli/componenti già pronti messi in comunicazione grazie allo standard per le transazioni definito da SystemC.

9 SystemC RTL

Con SystemC RTL (Register Transfer Level) è possibile descrivere il funzionamento di un circuito digitale in termini di segnali, elementi di memoria dei segnali (registri) e di operazioni logiche tra segnali.

9.1 Moduli

I moduli sono i blocchi di costruzione di base per partizionare un design. Sono classi C++, permettono di partizionare sistemi complessi in componenti più piccole. Nascondono la rappresentazione dei dati interni, utilizzano interfacce. Contengono porte, segnali, dati locali, altri moduli, processi, costruttori e distruttori. Segue un esempio:

Listing 1: module

```
sc_module(module_name)
{
    // dichiarazione di porte
    // dichiarazione di segnali
    // costruttore del modulo: SC_CTOR
    //      SC_METHOD
    // creazione di sottomoduli e sensitivity list
    // inizializzazione dei segnali
}
```

Il costruttore del modulo:

- Inizializza e dichiara tutti i processi contenuti nel modulo e le regole per attivarli
- Inizializza le variabili ai valori di default oppure a valori definiti dall'utente.

Listing 2: full adder constructor

```
SC_CTOR( FullAdder )
{
    SC_METHOD( doIt );
        sensitive << A;
        sensitive << B;
}
```

9.2 Processi

I processi sono funzioni he sono identificate dal kernel di SystemC:

- I processi sono molto simili a funzioni C++ o metodi
- I processi implementano le funzionalità dei moduli
- Sono chiamate se un segnale della sensitivity list cambia il suo valore.

I processi possono essere **Metodi**, **Threads** o **CThreads (deprecated)**.

- **Metodi**:
  - Quando attivati, vengono eseguiti e ritornano.
  - **SC_METHOD**(process_name).
- **Threads**:
  - Possono essere sospese a riattivate.
  - La funzione `wait()` sospende.
  - Un evento sulla sensitivity list riattiva.
  - **SC_THREAD**(process_name)
- **CThreads (DEPRECATED)**:
  - Sono attivate all'incremento del clock
  - **SC_CTHREAD**(process_name, clock_value)

type	**SC_METHOD**	**SC_THREAD**	SC_CHTREAD
Attivaz. all'exec.	Evento	Evento	Incremento clock
Sosp. dall'exec.	No	Sì	Sì
Loop infinito	No	Sì	Sì
sosp/riattiv da	n.d.	`wait()`	`wait()` o `until()`

Costruttore e sensitivity def	**SC_METHOD**`(call_back);`
	`sensitive(signals);`
	`sensitive_pos(signals);`
	`sensitive_neg(signals);`
		SC_THREAD`(call_back);`
		`sensitive(signals);`
		`sensitive_pos(signals);`
		`sensitive_neg(signals);`
			SC_CTHREAD`(...)`

Tabella 1: Tabella riassuntiva

10 SystemC TLM

SystemC TLM (Transaction Level Modeling) è una libreria del C++ che permette di rappresentare i principali componenti architetturali di piattaforme hardware.

È uno standard per la comunicazione tra i moduli (sia HW che SW) di un sistema, attraverso transazioni con caratteristiche ben definite. L'aver definito uno standard per il modo in cui si interfacciano diverse componenti ha facilitato il riuso dei componenti, cioè la possibilità di integrare facilmente nel proprio sistema moduli provenienti da fonti esterne e di interfacciarli tra loro semplicemente conoscendo le specifiche del protocollo di comunicazione.

Il riuso dei componenti porta a grandi vantaggi anche nel cercare di diminuire il TTM, infatti posso risparmiare del tempo utilizzando componenti già implementate, di cui conosco le prestazioni e sono sicuro del loro funzionamento, piuttosto che ricrearle da zero e cercare di interfacciarle col mio sistema ad un livello più basso di astrazione.

(Con TLM è inoltre possibile fornire velocemente una descrizione SW del comportamento, in modo da poter proporre ad un eventuale cliente un primo prototipo del sistema con tempistiche brevi.)

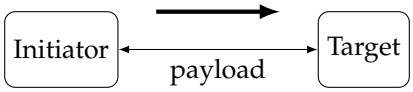
- Permette una esplorazione architetturale e modellazione delle performance.
- Consente l'esecuzione su modelli virtuali di piattaforme hardware.
- È disponibile prima dell'RTL.
- Viene simulato più velocemente dell'RTL.
- Consente di modellare sistemi a livello transazionale.

10.1 Transazione TLM

TLM si basa sul concetto di **transazione** cioè il trasferimento di dati tra moduli che comprende operazioni di scrittura/lettura.
La transazione è rappresentata da un oggetto **payload**: viene scambiato con chiamate primitive, contiene sia dati che informazioni di controllo.
Gli attori della transazione sono:

- **Initiator**: fa partire la transazione tramite socket
  - Crea un oggetto transazione.
  - Chiama o si connette con il metodo target per inviare il payload.
- **Target**: agisce come destinazione finale della transazione. Elabora il payload.

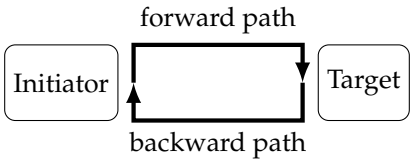
È un tipico comportamento master/slave:



10.2 Cammini di transazione

Il payload può seguire due diversi path:

- **Forward Path**: percorso di chiamata dall'initiator nella direzione del target. Il master chiama lo slave.
- **Backward Path**: percorso di chiamata attraverso cui il target ritorna indietro il payload nella direzione dell'initiator. Lo slave risponde al master.



10.3 Stili di programmazione TLM

In TLM 2.0 non ci sono livelli di astrazione standardizzati, come avveniva in TLM 1.0:

- Program View (PV)
- Program View with Time (PVT)
- Cycle Accurate (CA)

Questi livelli possono essere usati per esprimere diverse astrazioni dello stesso sistema.
In TLM 2.0 si è deeciso di standardizzare come il tempo e i dati sono collegati:

- **Untimed (UT)**: senza tempo.
  - Interfaccia bloccante.
  - Punti di sincronizzazione predefiniti.
- **Loosely Timed (LT)**:
  - Interfaccia bloccante.
  - Due punti di sincronizzazione (invocazione e return).
  - Temporal Decoupling
- **Approximately Timed (AT4)**: si dettaglia maggiormente il sincronismo tra Initiator e Target
  - Interfaccia transport non bloccante
  - Annotazione del tempo e fasi multiple durante il tempo di vita della transazione.
  - Protocollo 4-handshaking: inizio/fine richiesta, inizio/fine risposta.

UT è il livello più alto, **AT** è il livello più basso.

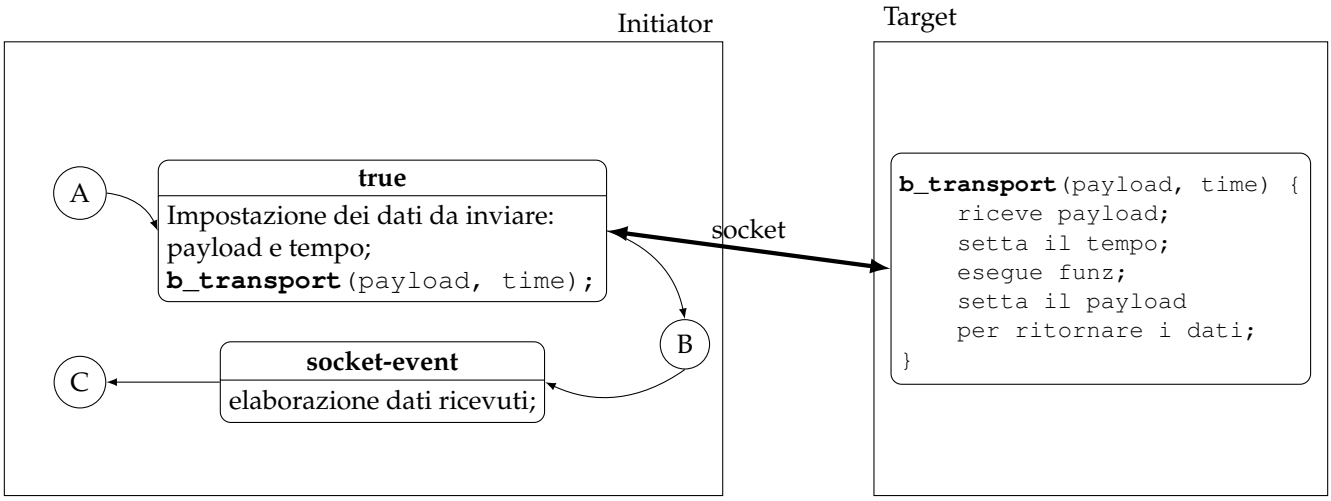
La comunicazione tra master e slave può essere *bloccante* (master rimane in attesa finché lo slave non ha finito i calcoli) o *non bloccante* (il master passa i dati allo slave ma può andare comunque avanti), in questo modo si può decidere il livello di parallelismo del software. Possono esserci componenti di interconnessione tra initiator e target, e servono per modellare bus astratti.
Se aumento i punti di sincronizzazione (arrivando a sincronizzare ogni ciclo di clock) posso arrivare ad avere una sincronizzazione cycle accurate.

Temporal decoupling

Initiator e target quando si sono de-sincronizzati possono andare avanti con tempi diversi e sono poi riallineati nel momento in cui si sincronizzano (comunicazioni non bloccanti), il più veloce aspetterà il più lento e si sincronizza. Il T.D. permette ai due moduli di evolvere insieme senza sincronizzarsi, rendendo più veloce la simulazione.

10.4 Interfaccia bloccante

Questa interfaccia è supportata da UT e LT. L'initiator completa la transazione con il target in una chiamata a funzione. Ci sono 2 punti di sincronizzazione: l'invocazione e il return. Utilizza solamente il **forward path**.



10.5 Interfaccia non bloccante

Questa interfaccia è supportata da AT4. Permette di dettagliare la sequenza di interazioni tra l'initiator e il target. È formata da due fasi:

1. Inizio/fine richiesta
2. Inizio/fine risposta.

Utilizza sia il forward che il backward path. Utilizza il classico protocollo di handshaking a 4 fasi.

10.6 Transactor TLM: (transattore)

- Modulo systemC che permette di far comunicare modelli a diversi livelli di astrazione (Es. RTL-TLM).
- Utile perché posso raffinare singolarmente prima i componenti e poi farli comunicare col transattore.
- Diversi livelli di astrazione: PV, PVT, AC.
- In PVT nel momento in cui devo temporizzare (non rispettando precisamente i cicli di clock) la comunicazione devo passare attraverso un bus (sia per moduli HW che SW) come AMBA (adottato da ARM).
- In PVT posso già decidere cosa far diventare HW (moduli che impiegano più tempo per essere calcolati) e cosa SW, e analizzare il traffico sul bus per unire per esempio moduli che hanno un alto traffico tra loro.
- In CA ho sempre bisogno di un bus ma che rispetti esattamente i cicli di clock.

10.7 Standard TLM 2.0

- Non ha senso standardizzare i livelli di astrazione. Si è passati da livelli di astrazione diversi a stili di codifica (LT e AT), lasciando al progettista la possibilità di decidere che tipo di flusso di progettazione seguire.
- La comunicazione avviene con un campo (payload) ben definito e standardizzato.
- TLM standardizza la comunicazione e non la funzionalità.
- **Initiator** fa partire la transazione, **Target** riceve la transazione (tramite socket)

- I cammini di transazione sono forward path e backward path
- Coding style NON sono specifici livelli di astrazione.
- Si possono definire coding style propri, ma potrei non riuscire a far comunicare moduli.
- In ogni transazione LT ci sono 2 punti di sincronizzazione (inizio e fine) e non ci sono molte informazioni sugli aspetti temporali.
- I processi hanno assegnati dei quanti di simulazione.
- Si spezza la sincronizzazione in più fasi, avendo più punti di sincronismo
- Per avere un modello UT prendo un modello LT e metto il tempo a 0.
- Per LT e AT4 si definiscono delle interfacce: bloccante (UT e LT) e non bloccante (AT4).
- Un altra interfaccia è la **DMI** usata per accedere ad aree di memoria condivisa tra Initiator e Target e modificarne i dati permette di far comunicare codici SystemC con codici di altri linguaggi (co-simulazione).

11 SystemC AMS

È una estensione del SystemC, permette la modellazione System-lever per sistemi con segnali analogici misti (Analog Mixed-Signal Systems).
Metodologie e casi d'uso per sistemi con segnali mixed:

- Specifiche eseguibili: verificare la correttezza dei requisiti del sistema utilizzando la simulazione.
- Prototipizzazione virtuale: modello ad alto livello (untimed/timed) dell'architettura hardware.
- Esplorazione di architetture: valutazione del mapping tra il comportamento e l'architettura di sistema.
- Validazione euristica integrata: verificare la correttezza delle componenti integrate.

11.1 Tempo discreto

Nelle descrizioni a tempo discreto:

- La funzione esiste solo in punti ben precisi nel tempo, negli altri non esiste e non posso valutarla è come se scrivessi una procedura. I segnali e le quantità fisiche sono definiti in punti di tempo discreti. Si assumono costanti in mezzo a questi punti.
- Vengono usate per rappresentare approssimazioni di rappresentazioni a tempo continuo, signal processing: rappresentazione di funzioni di elaborazioni di segnali (tipo filtri)

11.2 Tempo continuo

Nelle descrizioni a tempo continuo:

- La grandezza fisica che descrivo esiste in qualsiasi punto nel tempo come una equazione differenziale. I segnali e le quantità fisiche sono descritti nel come funzioni nei reali. Il tempo è considerato come un valore continuo.
- Si comportano come equazioni differenziali algebriche (DAE) o equazioni differenziali ordinarie (ODE). Vengono risolte da un solver lineare o non-lineare.
- Adatto per descrivere comportamenti fisici di sistemi dinamici,

11.3 Descrizioni conservative

Nelle descrizioni conservative:

- Il comportamento
- È tutto implementato in blocchetti di base che noi colleghiamo e che garantiscono in modo automatico che le leggi di conservazione di Kirchhoff vengano rispettate.
- ho vincoli sulla conservazione dell'energia le correnti d'entrata uguali a quelle di uscita
- Ci sono grandi quantità equazioni da risolvere, è molto dispendioso dal p.d.v. computazionale.

11.4 Descrizioni non conservative

Nelle descrizioni conservative:

- Il comportamento è espresso con flussi di segnali di tempo continuo o variabili. Vengono applicate funzioni di processing (filtering e integrazione).
- Può essere descritta la dinamica non lineare
- Non è supportata l'interazione tra componenti AMS.
- Le relazioni tra quantità non rispettano le leggi di conservazione dell'energia di Kirkhhoff.

11.5 Formalismi di modellazione

In SystemC AMS ci sono i seguenti formalismi di modellazione:

- **Time Data Flow (TDF)**: modelli a eventi discreti.
  - Tempo discreto, modellazione non conservativa
  - Scheduler statico, basato su dataflow
- **Linear Signal Flow (LSF)**: modelli a eventi continui.
  - Tempo continuo, modellazione non conservativa
  - Basato su DAE e ODE, rappresentate da connessioni di primitive per segnali reali nel dominio del tempo.
- **Electric Linear Network (ELN)**: per modellazione di reti elettriche.
  - Tempo continuo, modellazione conservativa
  - Primitive per reti lineari (resistenze, condensatori)
  - Relazioni continue tra voltaggio e corrente.

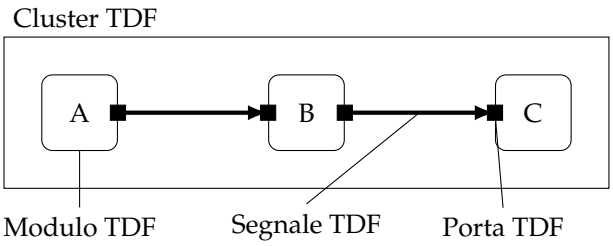
	Discreto (Scheduler Statico)	Continuo (Scheduler Dinamico)
Conservativo	$\emptyset$	**ELN**
Non-Conservativo	**TDF**	**LSF**

11.6 Time Data Flow (TDF)

Formalismo basato su Synchronous Data Flow (SDF), senza tempo, modello a eventi discreti in tempo discreto. Ogni modulo del modello a eventi discreti contiene un metodo C++:

- Elabora una funzione matematica, a seconda degli input. Può dipendere anche dai suoi stati interni.
- Composizione di moduli

Esempio di composizione di funzioni: $f_C(f_B(f_A))$



Data una funzione, questa viene eseguita se e solo se ci sono abbastanza sample disponibili nella porta di input. C'è un numero fissato di sample consumati e prodotti. Ogni sample ha il suo time stamp. L'intervallo fissato è chiamato time step.

- Time step (modulo)
- Time step (porta)
- Rate (porta)
- Delay (porta)
- Time offset (porta specializzata)

Nello scheduling i cicli possono essere fonte di problemi. Ogni loop deve presentare minimo una porta di delay. Le porte di delay possono portare all'inconsistenza, quindi bisogna specificare un valore iniziale.

TDF non supporta la gerarchia