Introduzione ai linguaggi per la progettazione dell'hardware

William Fornaciari
Docente di Reti Logiche A c/o la Facoltà di Ingegneria dell'Univ. Degli Studi di Parma
Affiliazione: Politecnico di Milano, DEI, P.zza L. Da Vinci, 32. 20133 Milano, Italy
fornacia@elet.polimi.it

Sommario

La crescita della complessità dei sistemi digitali degli ultimi 20-30 anni è stata possibile grazie alla messa a punto di metodologie di progetto in grado di automatizzare molte fasi realizzative. Un ruolo fondamentale è stato ricoperto dai linguaggi di descrizione dell'hardware, di cui il presente articolo traccia lo stato presente e le possibili evoluzioni, in relazione anche ai mutati scenari tecnologici. Ormai non esiste più una netta distinzione fra la progettazione Hw e Sw: siamo nell'era della progettazione concorrente a livello sistema.

Realizzazione di un sistema digitale

La crescita esponenziale della "densità" dei circuiti digitali, rappresentata dal numero dei transistori su un *chip* (ben prefigurata dalla cosiddetta "legge di Moore") ha portato alla necessità di definire fasi e modelli del processo di progettazione e realizzazione che consentissero di dominarne la complessità. Tale strutturazione, esemplificata ad alto livello in figura 1, ha diversi vantaggi. Innanzitutto consente di *specializzare* le competenze dei progettisti solo su alcuni passi dell'intero processo; grazie ad una *formalizzazione* dei modelli consente poi di impiegare rappresentazioni utili all'*automatizzazione* del processo di ottimizzazione; infine, ma non meno importante, consente di simulare il sistema con livelli di astrazione e di accuratezza variabili, in modo da essere certi che il comportamento della realizzazione fisica finale sia effettivamente coerente con il progetto originale. A partire da una specifica del sistema, che verrà a un certo punto espressa con un opportuno linguaggio di descrizione o modello rappresentativo, si seguiranno trasformazioni che porteranno la descrizione a divenire sempre meno astratta e sempre più vicina ad una formulazione finale sufficientemente dettagliata per potere guidare il processo di fabbricazione del silicio. Il percorso verso i livelli di astrazione più bassi non è una semplice traduzione, ma un processo di *design* complesso che prevede molte ottimizzazioni, (costo, velocità, potenza dissipata e tempo di progetto per citarne alcune) e, ad ogni passaggio, coinvolge attività di simulazione e verifica per garantire che il circuito continui a rispettare gli obiettivi e vincoli di progetto iniziali.

Per un sistema digitale, con buona approssimazione possiamo affermare che il processo di sintesi della macrofase detta *front-end* si arresta quando vengono identificati gli elementi digitali che compongono il sistema (es. porte logiche, memorie, registri) e le loro connessioni. A partire da questa descrizione inizia l'attività di back-end, che ottimizzerà ulteriormente il sistema cercando di disporre tali elementi in modo da usare il minore spazio possibile (floorplan) e arriverà a definire le caratteristiche geometriche e di drogaggio delle varie aree di silicio, informazioni che porteranno alla creazione di

opportune "maschere" che saranno usate per la fase di produzione vera e propria dell'integrato. Oggi, una linea di produzione per un comune processo tecnologico digitale richiede in genere non meno di 4-5 settimane per la fase di back-end e di realizzazione di un insieme di prototipi su silicio, con un costo per la realizzazione delle maschere difficilmente inferiore al milione di euro. È quindi ovvio che l'identificazione e la messa a punto di eventuali errori dopo la fase di produzione, in molti casi può significare il fallimento di un progetto; di conseguenza, molto sforzo è stato indirizzato verso lo sviluppo di strumenti di progettazione automatica (Electronic Design Automation – EDA), per consentire al progettista di automatizzare molte fasi di sintesi e verifica e di lavorare il più possibile in modo top-down, pur mantenendo la possibilità di confrontare in tempi ragionevoli varie alternative di progetto.
Un progetto, come visto in figura 1, può quindi essere visto a diversi livelli di astrazione, sia sotto il profilo *funzionale*, sia per quanto concerne la *struttura* e l'organizzazione dei blocchi che ne andranno a comporre l'architettura, sia per quanto attiene più da vicino le informazioni *geometriche* usate nel processo di produzione del silicio.

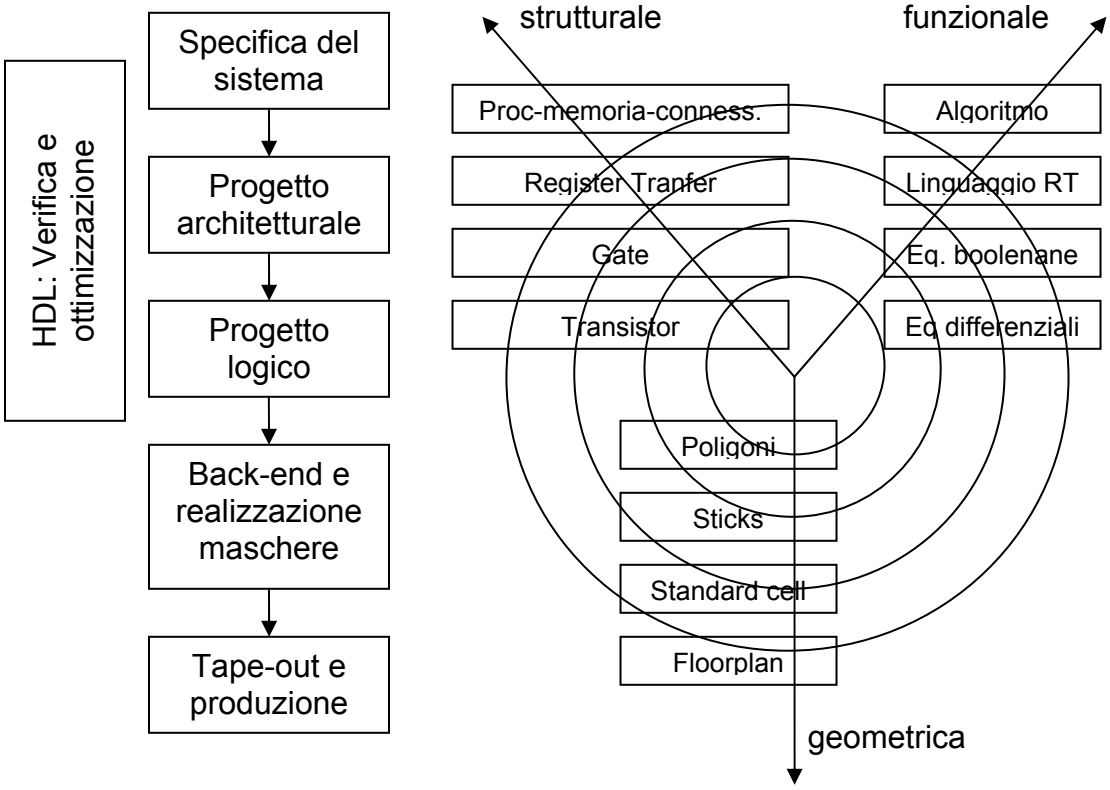


Figura 1. Passi del processo di progettazione di un sistema digitale (sinistra), domini e livelli della modellazione e ottimizzazione (destra)[1].

I linguaggi di descrizione dell'hardware (HDL), come il Verilog e il VHDL, sono sicuramente stati una delle innovazioni più significative degli ultimi 20-30 anni, perché hanno favorito la nascita di strumenti di *progetto automatico*, ridotto drasticamente i *tempi di progetto* e consentito la creazione di *figure professionali* con competenze sempre

[1] Il cosiddetto "diagramma a Y" è stao introdtto da Gajski e Kuhn all'inizio degli anni '80.

meno verticali, passaggio pressoché inevitabile visti i tassi di crescita della complessità dei sistemi e delle loro tecnologie di realizzazione. Attualmente su un unico chip si possono integrare a basso costo dalle decine a centinaia di milioni di transistor, in architetture che possono comprendere diversi microprocessori in aggiunta ad unità di elaborazione dedicate e a memorie, con geometrie del singolo transistor dell'ordine dei 45nm: è persino difficile immaginare che un solo gruppo di progettisti possa gestire l'intera filiera di progetto e realizzazione.

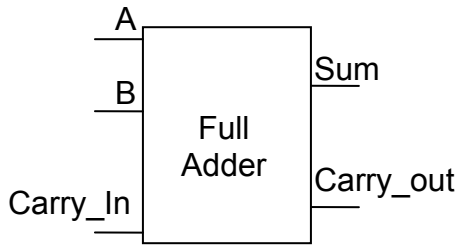
Linguaggi per la modellazione hardware: il VHDL

Per esemplificare l'organizzazione e l'utilizzo di un linguaggio per la descrizione dell'hardware, faremo riferimento al VHDL che è il più diffuso in Europa. Il VHDL in realtà nasce negli USA nell'ambito del progetto VHSIC (Very High Speed Integrated Circuits) sponsorizzato da Dipartimento delle Difesa (DoD) americano. La prima versione risale al 1984, la prima standardizzazione IEEE avviene nel 1987 e i successivi aggiornamenti sono stati compiuti negli anni 1992, 1997 e 2002. Gli obiettivi iniziali del progetto dovevano consentire al DoD di standardizzare le procedure di progettazione e documentazione degli apparati digitali e di svincolarsi da eventuali dipendenze da un particolare fornitore dei sistemi elettronici. La struttura e l'utilizzo del linguaggio hanno forti similarità con un altro ben noto standard, il linguaggio ADA, e si accompagna ad una versatilità che gli consente di essere utilizzato sia per fare progettazione e verifica funzionale, sia come riferimento per l'intero processo di sintesi che porta alla realizzazione del sistema finale.
Nonostante l'apparente similarità con un linguaggio di programmazione, il VHDL ha particolarità che derivano dall'essere concepito per rappresentare sistemi hardware, dove non esiste un unico esecutore sequenziale delle operazioni e dove la sincronizzazione degli accessi alle varie unità non può essere implicitamente considerata come un dato di fatto. Le caratteristiche salienti di un HDL dotato di una buona potenza rappresentativa, quale il VHDL, sono le seguenti:

- *Concorrenza*. Deve essere possibile rappresentare senza difficoltà l'intrinseco parallelismo dei sistemi hardware, così come, se necessario, forzare un ordinamento nell'esecuzione di gruppi di operazioni.
- *Astrazione*. Si vuole lavorare con diversi livelli di astrazione sia nella fase di specifica sia durante la simulazione.
- *Viste*. Si vuole operare con viste differenti dello stesso sistema, che possono per esempio avere le medesime interfacce ma avere rappresentazioni comportamentali piuttosto che strutturali.
- *Strutturazione*. Deve essere possibile rappresentare, per i diversi livelli di astrazione a cui si considera il sistema, i collegamenti fra i vari blocchi componenti. Si possono pertanto identificare gerarchie del sistema digitale; diviene possibile la progettazione modulare di sistemi complessi, grazie anche al supporto al riuso di unità di progetto o di librerie sviluppate anche da terze parti.
- *Tempo.* È necessario specificare precisi parametri temporali per rappresentare le caratteristiche dei sistemi reali che si otterranno dopo la sintesi, così come la sincronizzazione fra diversi sottosistemi, inclusa la gestione dei clock.

Nel caso del VHDL, c'è una separazione netta fra la specifica delle interfacce di un componente e il suo "corpo". Questa caratteristica, ereditata dal linguaggio ADA,

consente sia di associare *ad una sola specifica* differenti viste, come anche progetti alternativi da utilizzarsi in base alla fase del progetto o a specifiche esigenze realizzative.
In VHDL ogni entità da modellare si chiama *design entity*, e si compone di una *entity declaration* e di una o più *architecture*. Per esemplificare l'uso del VHDL consideriamo un semplice sommatore completo (full adder in figura 2) che riceve in ingresso due bit (a e b) e un eventuale riporto (carry_in) e produce il bit di somma (sum) e l'eventuale riporto in uscita (carry_out). La figura 2 mette in evidenza le interfacce del full adder che si traducono nella *entity declaration* riportata a destra.




```
entity full_adder is
port (a,b,carry_in: in bit;
    sum, carry_out: out bit);
end full_adder;
```

Figura 2. descrizione in VHDL delle interfacce di un sommatore completo (*entity declaration*), dove si identificano le "porte" di ingresso e uscita.

L'architettura di una entity si compone di un *header* e di un *body* che può essere di tipo strutturale o comportamentale (behavioral). Quest'ultimo fornisce informazioni sufficienti per il calcolo dei valori dei segnali di uscita a partire da quelli in ingresso ed eventuali informazioni di temporizzazione del circuito (utili essenzialmente in fase di simulazione). Indicando con <= l'assegnamento di segnali, una possibile descrizione comportamentale del full adder è riportata in figura 3.

```
architecture behavior of full_adder is
begin
sum <= (a xor b)xor carry_in after 10 ns;
carry_out <= (a and b) or (a and carry_in) or
         (b and carry_in) after 10 ns;
end behavior;
```

Figura 3. Descrizione behavior del sommatore di Figura 2.

Le informazioni di temporizzazione come quelle specificate con la parola chiave *after* sono utili in fase di simulazione per generare forme d'onda in risposta alla variazione degli stimoli di ingresso, tenendo conto della presenza di ritardi reali e finiti (nei circuiti reali) per il calcolo delle uscite. Nel caso di figura 3, le uscite *sum* e *carry_out* saranno stabilizzate sul loro valore finale dopo 10 ns rispetto alla variazione degli ingressi.
Una possibile rappresentazione del full adder di tipo *strutturale*, invece, focalizzerebbe l'attenzione sull'architettura di una possibile realizzazione dello stesso circuito e non sulla sua funzionalità: si vedano le figure 4 e 5, dove il full adder è realizzato connnettendo due semisommatori (half adder i1 e i2) e una porta AND (i3) che sommano semplicemente due bit generando un riporto in uscita.
Tali componenti debbono essere dichiarati all'interno della *component declaration*; tale dichiarazione fornisce le informazioni salienti circa il componente anche se la sua

descrizione completa non è ancora presente nel database del progetto, come tipicamente avviene quando si adotta un approccio top-down alla progettazione. La corrispondenza fra le porte della entity e quelle dei componenti interni è descritta tramite il cosiddetto *port map*.

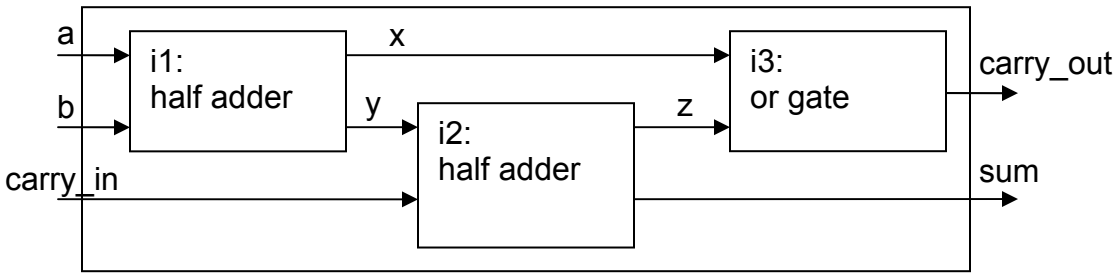


Figura 4. vista strutturale del full adder.

```
architecture structure of full adder is
 component half_adder
  port (in1,in2: in bit; carry: out bit; sum: out bit);
 end component;
 component or_gate
  port (in1,in2: in bit; o: out bit);
 end component;

 signal x,y,z: bit;   -- segnali locali

 begin   -- connessione delle porte
  i1: half_adder port map (a,b,x,y);
  i2: half_adder port map (y, carry_in,z,sum);
  i3: or_gate port map (x,y,carry_out);
 end structure;
```

Figura 5. descrizione strutturale del full adder di figura 4.

Così come col costrutto *after* si modella il ritardo nella generazione delle uscite, VHDL consente di considerare i *livelli* (che sono un'astrazione dei valori di tensione) e la *forza* dei segnali (che invece modellano l'impedenza delle sorgenti elettriche) oltre all'eventuale indeterminatezza nei valori assunti da un segnale. In VHDL ogni uscita è associata ad un *driver*: qualora più uscite siano connesse fra loro, una funzione di *risoluzione* calcola il valore considerando i vari contributi con il loro valore e la loro forza.[2] Non meno importante, VHDL offre costrutti analoghi a quelli dei linguaggi di programmazione, come la selezione, *select/when*, la tipizzazione, i sottoprogrammi, ecc. oltre a una raffinata gestione delle librerie di componenti.

La semantica del VHDL ha una formalizzazione che consente di realizzare *simulatori* pienamente deterministici, che consentono quindi di procedere a fasi di *verifica del progetto mediante simulazione*. Diverso è il discorso per gli strumenti di sintesi automatica, poiché ogni singolo produttore di strumento EDA, al fine di ottimizzare il progetto, oltre che la complessità e l'efficienza del proprio strumento, potrebbe far si che

[2] Lo standard a cui si fa riferimento più comunemente, che assicura anche una ottima portabilità dei progetti, è chiamato IEEE 1164.

usando i vari strumenti a partire dalla stessa descrizione VHDL l'utente giungesse risultati differenti (anche se funzionalmente equivalenti). Negli anni si è arrivato a definire un insieme di schemi (*template*) di progetto e di regole generali a cui attenersi che sono utili per restingere la descrizione ai soli costrutti che ha senso sintetizzare (eliminando per esempio costrutti per utili sono in fase di simulazione e verifica) e ad ottenere in massimo delle prestazioni dagli strumenti EDA.

Tecnologie emergenti a supporto della sintesi

L'attuale tecnologia del silicio consente di realizzare sistemi *SoC* (System on a Chip) che, anche su di un solo chip, sono in grado di integrare la complessità di un intero apparato, come uno o più processori comunicanti con acceleratori hardware (come accade per esempio per un sistema di navigazione automoblistico, un PDA, un Set-Top-Box multimediale, ecc.). Sempre nell'ultimo decennio, con previsioni per il prossimo di gran lunga superiori a qualunque altra tecnologia, si stanno affermando anche tecnologie per la realizzazione dell'hardware dette *FPGA* (Field Programmable Gate Arrays). Tali sistemi standardizzano l'architettura su cui deve essere sintetizzato il sistema, che in genere ha la topologia di un array, con complessità delle singole celle che lo compongono in grado di raggiungere tipicamente quella di una macchina a stati. La traduzione delle funzionalità in un circuito digitale reale avviene specializzando solo i collegamenti *locali* alla singola cella e *globali* all'intera FPGA, cioè fra celle. Tale attività può essere svolta direttamente dal progettista finale, nel giro di pochi minuti, utilizzando componenti che quindi sono standardizzati e disponibili off-the-shelf a costi abbordabili anche per bassissimi volumi di produzione, senza attendere settimane o rischiare il costo (milionario) dello sviluppo delle maschere.
Siamo quindi da poco entrati in uno scenario dove ormai è possibile (e necessario) progettare a *livello di sistema* (System Level Design), considerando in modo concorrente funzionalità che spaziano dell'hardware al software, con la possibilità anche per piccole ditte di realizzare "in casa" sistemi complessi che fanno uso circuiti integrati con tecnologia FPGA che hanno complessità e velocità come quelle dei circuiti integrati di pochi anni orsono. Tale disponibilità di tecnologie non è stata però accompagnata da una maturazione altrettanto rapida delle metodologie e degli strumenti EDA, in grado di elevare al livello di sistema l'astrazione dell'applicazione a cui lavora il progettista.

Verso nuovi formalismi di descrizione

Molti sforzi sono attualmente legati alla messa a punto di strumenti che supportano il cosiddetto *IP-based design*, ovvero la possibilità di progettare rapidamente sistemi in modo non verticale, ma utilizzando anche blocchi IP (Intellectual Properties) che possono essere eventualmente acquisiti da terze parti o sviluppati internamente con lo scopo di essere riusabili facilmente. Allo stesso modo si sta cercando (con alterne fortune) di elevare il livello di astrazione dei formalismi per la descrizione dei sistemi, in modo da potere coprire sia le componenti hardware sia quelle software di un sistema. Probabilmente il linguaggio **SystemC**, nato nel 1999 e standardizzato da IEEE nel 2005, è il rappresentante di tale filosofia con le maggiori probabilità di sopravvivenza[3]. SystemC, la cui architettura è mostrata in figura 6, definisce una libreria di classi C++ in

[3] Si veda www.systemc.org per maggiori informazioni circa tale iniziativa.

modo da fornire al progettista un insieme di elementi per effettuare modellazioni e simulazioni con l'accuratezza sino a livello di ciclo (cycle accurate) delle specifiche del sistema. Le librerie SystemC contengono tutti i costrutti per modellare sistemi, incluse temporizzazioni hardware, concorrenza e comportamento reattivo che non sono native del C++. L'obiettivo è quello di consentire anche a chi ha principalmente un background di sviluppatore software, di muovere alcuni passi significativi nella progettazione di sistemi hw/sw complessi.

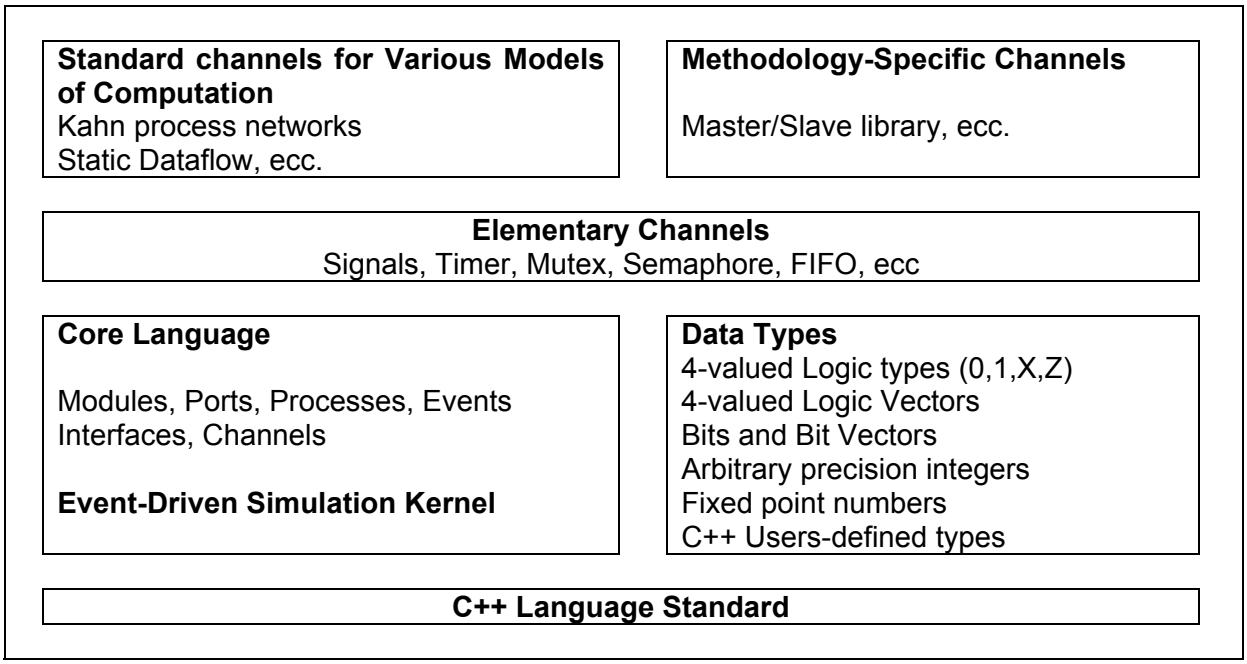


Figura 6. Architettura del linguaggio SystemC.

La metodologia di progetto basata su SystemC fornisce la possibilità di utilizzare un unico formalismo per Hw e Sw, rimanendo al livello funzionale per la verifica del comportamento, e muovendo verso livelli incrementali di dettaglio sino a giungere al punto in cui esiste uno strumento di sintesi automatica. In questo modo si può usare un unico formalismo per la simulazione e sintesi dell'hw, del sw e per lo sviluppo dei testbench. Purtroppo, almeno per ora, la macanza di strumenti di sintesi con la maturità ed efficienza di quelli basati su VHDL porta ancora i progettisti a ricorrere ai canonici flussi e strumenti commerciali per la sintesi di hardware reale. Ciononostante, la possibilità di creare modelli (dalla versione 2.0) con un livello di astrazione detto *transazionale* (TLM, Transactional Level Model), in virtù della sua efficienza in fase di simulazione e potenza rappresentativa ne sta favorendo l'accettazione da parte dei progettisti. Uno stile di descrizione TLM porta a rappresentare i componenti di un sistema in modo affine all'invocazione di metodi remoti: vi sono moduli (SC_MODULE) che si interfacciano in modo più astratto rispetto al concetto di segnale, mediante invocazioni dirette (sc_port) di metodi realizzati da un modulo ed esportati (sc_export) tramite canali (sc_channel). Ovviamente, come accadeva per il VHDL, i costrutti utilizzabili per la sintesi delle parti hardware sono una restrizione rispetto a quelli messi a disposizione dal linguaggio. Per tale motivo la fase di sintesi prevede una serie di passaggi per raffinare la specifica in modo da renderla implementabile, per esempio

convertendo i tipi del linguaggio come il float o il double nella precisione effettivamente necessaria, eliminando le chiamate di sistema operativo, utilizzando solo costrutti per cui esiste semantica hardware ben definita (escludendo quindi allocazione dinamica della memoria, la ricorsione o l'uso del goto).
Un altro linguaggio, la cui diffusione negli USA è superiore al VHDL e il **Verilog**, standardizzato IEEE per la prima volta nel 1995, che ha raggiunto la versione 2.0 nel 2001. Molte caratteristiche del Verilog sono affini a quelle del VHDL ed il loro ruolo nella progettazione può essere considerato analogo. I progetti Verilog sono composte da interconnessione di design entità che possono essere descritte a livello comportamentale, così come sono usati i processi per rappresentare la concorrenza dell'hardware. Le versioni a partire dalla 3.0 del 2003 sono note come **SystemVerilog** e contengono estensioni che ne ampliano potenzialità e livello di astrazione, consentono per esempio di chiamare funzioni C/C++, creare processi dinamicamente, standardizzare la comunicazione e sincronizzazione tra processi (inclusi i semafori) e una interfaccia standard per la verifica formale. La capacità di utilizzare anche il C/C++ che dovrebbe rendere possibile l'interfacciamento verso modelli SystemC, un sistema di simulazione migliorato e l'apertura verso la verifica formale sono i pilastri principali dei fautori di tale linguaggio.
A conclusione della panoramica è necessario citare la presenza di strumenti di modellazione e simulazione matematica tipicamente utilizzati in ambiti sistemistici e telecom, come **MATLAB/Simulink**. Tali tool stanno rapidamente evolvendo in modo da fornire non solo la fase di modellazione della realtà fisica ma anche capacità implementative per esempio delle funzioni di controllo o di elaborazioni numerica, tramite opportune estensioni in grado di generare modelli C e/o VHDL da usare come base per la compilazione/sintesi vera e propria della realizzazione finale.

Il laboratorio (facoltativo) del corso

L'obiettivo dell'attività di laboratorio è di presentare, nell'arco di circa tre incontri, come si possa realizzare un semplice progetto digitale, mappandolo e simulandolo su una logica programmabile. In questi anni si è fatto uso di una versione disponibile per studenti di ISPlever prodotto da Lattice. L'entry del progetto non fa uso del VHDL ma si appoggia su un editor grafico (schematic) che consente di "vedere" i sistemi digitali al livello di astrazione e con le modalità di rappresentazione cui si è abituati a lezione.
Gli studenti davvero volenterosi possono sviluppare un semplice progetto sotto la supervisione del responsabile di laboratorio, al fine di "arrotondare" la valutazione dell'esame finale del corso.

Bibliografia

P.Ashendan, *VHDL cookbook*, disponibile liberamente sulla rete.
T.Grotker, S.Liao, G.Martin, S.Swan, *System Design with SystemC*, Kluwer Academic Publisher, 2002. ISBN: 1-4020-7072-1.
Siti di fornitori di logiche programmabili: www.latticesemi.com, www.xilinx.com, www.altera.com.
Siti dei principali CAD vendor: www.mentorgraphics.com, www.synopsys.com.