Metodologie di progettazione

Metodologie di progettazione

- Una procedura per progettare un sistema
- Il flusso di progettazione può essere parzialmente o totalmente automatizzato.
- Un insieme di tool possono essere usati per automatizzare i passi della metodologia:
  - Software engineering tools,
  - Compilers,
  - Computer-Aided Design tools,
  - ecc.
- Un'adeguata metodologia di progettazione aiuta la gestione del processo di design e migliora la qualità, le performance e i costi del progetto.

Design goals

Fornire la funzionalità richiesta nel rispetto dei requisiti del sistema espressi in termini di:

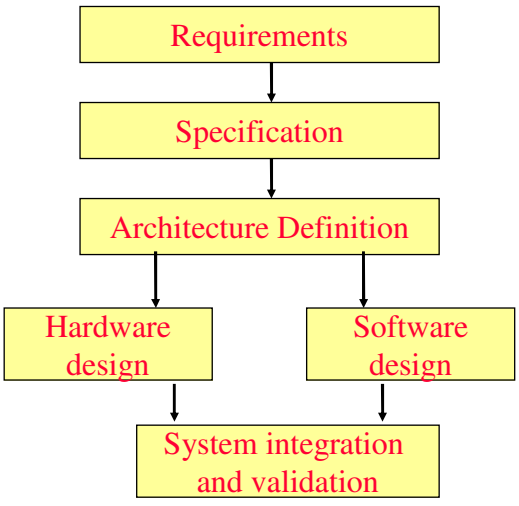
– Performance (latenza e throughput)

– Power

– Costo

– Altri requisiti (dimensione, peso, ecc.)

Un flusso di progettazione semplificato



Top-down vs. bottom-up

- Top-down design:
  - Parte dalla descrizione più astratta;
  - Opera verso una descrizione più dettagliata.
- Bottom-up design:
  - Lavora dai componenti più piccoli verso il grande sistema.
- La progettazione reale usa entrambe le tecniche.

Refinement

- Ad ogni livello di astrazione bisogna:
  - analizzare il progetto per individuare le caratteristiche dello stato corrente del progetto;
  - rifinire il progetto aggiungengo dettagli.

Requirements

- Descrizione nel linguaggio naturale di cosa vuole e cosa pensa di ottenere.
- Può essere ottenuta in diversi modi:
  - Parlando direttamente con il cliente;
  - Parlando con rappresentante di vendita;
  - ecc.

Functional vs. non-functional requirements

- Functional requirements:
  - Uscita come funzione degli ingressi.
- Non-functional requirements:
  - Tempo di computazione;
  - dimensione, peso, ecc.;
  - consumo;
  - etc.

System-Level Specification

- Una descrizione più precisa del comportamento del sistema
  - Non deve ancora definire una architettura di riferimento
  - Fornisce gli ingressi per il processo di architecture design
- Può includere sia *functional* sia *non-functional* requirements
- Può essere eseguibile per realizzare delle simulazioni oppure può essere un modello matematico per una verifica formale.

Architecture Design

- Componenti HW:
  - Processore, Memoria, Periferiche, ecc.
- Componenti SW :
  - Programmi e le loro operazioni
- Alcuni componenti potrebbero essere già disponibili (design reuse), alcuni devono essere modificati da progetti preesistenti, altri devono essere del tutto progettato

System Integration

- Mette assieme i componenti HW e SW del sistema
- I componenti possono essere:
  - Modelli eseguibili
  - Prototipi fisici
- Questa fase è cruciale per validare l'interazione tra i diversi componenti.
- Molti errori a livello di system-level possono essere scoperti solo in questa fase
- E' importante validare l'integrazione del sistema il prima possibile!

HW/SW Co-design Flow

HW/SW Co-design

- HW/SW Co-design is the field that emphasizes a unified view of HW and SW and develops synthesis tools and simulators that enable the co-development of systems by using both hardware and software.
- Main goal: To meet the design goals at the system-level exploiting the synergy of HW and SW parts through their concurrent design.

HW/SW Co-design: Main Advantages

- To explore different design alternatives.
- To evaluate cost-performance trade-offs
- To reduce system design time ⇒ Reduction of product time-to-market and cost
- To improve product quality through design process optimization
- To facilitate the reuse of hardware and software parts.
- To provide an integrated framework for the synthesis and validation of hardware and software components.

Main phases of HW/SW co-design process

- Requirements Analysis
- Functional Specification (System Modelling)
- Task Concurrency Management
- HW/SW Partitioning
- Co-synthesis
- Co-simulation and co-verification

Co-design phases

- Functional Specification (System-Level Modelling): to capture the system functionality in a formal language.
- Task Concurrency Management: To identify and to manage tasks in the system; To define tasks granularity (size of tasks)
- HW/SW Partitioning: To map system functionalities to either hardware or software.

Co-design phases

- Co-synthesis: To automatically derive a system implementation
  - Hardware synthesis of dedicated units
  - Software synthesis for processors (Specialized compiling techniques)
  - Interface synthesis to support HW/SW interface and synchronization
- Co-simulation and co-verification: To validate the design descriptions with respect to system-level requirements.

HW/SW Partitioning

To map system functionalities to either dedicated HW components or SW (processors).

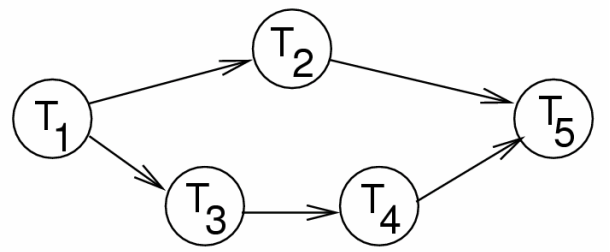
- One extreme: Fully HW solution
  - High performance due to parallelism
  - High cost and long design time
- Other extreme: Fully SW solution
  - High-performance low-cost processors
  - Operations serialization
  - Lack of support for specific tasks
- Best solution: Mix of HW and SW components based on cost-performance trade-offs

HW/SW Partitioning Goals

- To speed-up software execution
  - By migrating critical SW functions to dedicated HW
- To reduce the cost of hardware implementation
  - By migrating non-critical hardware functions to software running on processor core

Open issues in HW/SW partitioning

- Object granularity in partitioning
- Estimation of performance and cost metrics from graph model of the system



Co-synthesis

- To automatically derive a system implementation
  - Hardware synthesis of dedicated units
  - Software synthesis for processors (Specialized compiling techniques)
  - Interface synthesis to support HW/SW interface and synchronization of SW functions identified by program threads
  - A single processor requires threads serialization or interleaving
  - Scheduling of threads and instructions
  - System-level run-time scheduler to synchronize SW and HW functions

HW Synthesis

- After partitioning, HW behavior is still described at high-level of abstraction (behavioral level)
- High-level synthesis alternatives:
  - Manual translation to RTL (Register Transfer Level) description
  - Use behavioral synthesis tools

SW Synthesis

- SW synthesis for processors is based on specialized compiling techniques
- Compilation characteristics in embedded systems
  - Code is compiled once ⇒ Compilation time is not important, maximum optimization is desired
  - Performance constraints ⇒ Code must be tight and fast ⇒ Assembly code
  - Energy-aware compilation for low-power portable embedded systems

Interface Synthesis

- Interfacing processors to ASICs and peripheral devices
- Device drivers may be in SW or in HW
- Define models for communication mechanisms
- Scheduling the processor communication