Hardware-software codesign: basics

Design methodologies

- Process for creating a system.
- Many systems are complex:
  - large specifications;
  - multiple designers;
  - interface to manufacturing.
- Proper processes improve:
  - quality;
  - cost of design and manufacture.
Design flow

- **Design flow**: sequence of steps in a design methodology.
- May be partially or fully automated.
  - Use tools to transform, verify design.
- Design flow is one component of methodology. Methodology also includes management organization, etc.

Waterfall model

- Early model for software development:

  - requirements
  - architecture
  - coding
  - testing
  - maintenance

  Requirements -> Architecture -> Coding -> Testing -> Maintenance
### Waterfall model steps

- Requirements: determine basic characteristics.
- Architecture: decompose into basic modules.
- Coding: implement and integrate.
- Testing: exercise and uncover bugs.
- Maintenance: deploy, fix bugs, upgrade.

### Waterfall model critique

- Only local feedback---may need iterations between coding and requirements, for example.
- Doesn’t integrate top-down and bottom-up design.
- Assumes hardware is given.
Spiral model critique

- Successive refinement of system.
  - Start with mock-ups, move through simple systems to full-scale systems.
- Provides bottom-up feedback from previous stages.
- Working through stages may take too much time.
Successive refinement model

Initial system
- Specify
- Architect
- Design
- Build
- Test

Refined system
- Specify
- Architect
- Design
- Build
- Test

Hardware/software design flow

Requirements and specification
- Architecture
  - Hardware design
  - Software design
- Integration
- Testing
Co-design methodology

- Must architect hardware and software together:
  - provide sufficient resources;
  - avoid software bottlenecks.
- Can build pieces somewhat independently, but integration is major step.
- Also requires bottom-up feedback.

Hierarchical design flow

- Embedded systems must be designed across multiple levels of abstraction:
  - system architecture;
  - hardware and software systems;
  - hardware and software components.
- Often need design flows within design flows.
Hierarchical HW/SW flow

Codesign in time
Codesign in time II

Co-Design Process

Tasks

System design

Shared Design

ASIC design

Fabrication

Test

Time

SW test

PCB test

System-Level Partitioning

System-Level Partitioning

HW/SW co-design process

co-specification

co-design & synthesis

architecture design, HW/SW partitioning and interfacing

HW design

SW design

evaluation (co-simulation)

high-level transformation

process impl. & transformations

system architecture

results

HW developer

SW developer

reused functions and processes

reused and manually optimized HW and SW components

HW architecture and components

system architect

customer/marketing systems architect
Goals of computer-aided hardware/software co-design

- Explore different design alternatives
  - Search for the best solution
- Reduce system design time
  - Reduce product time to market
- Support coherent design specification
  - Facilitate hardware and software reuse
- *Provide integrated environment for synthesis and validation of hardware and software components*

---

**Synthesis**

**HW**: HDL (Behavioral, DataFlow, Structural), Schematic

**Specification**

**Co-Synthesis**

**Synthesis**

**Detailed Representation of Implementation**

**SW**: Algorithm, Textual/Graphical representation

Executable or Compilable code:
The program(s),
OS routines
Co-synthesis steps

- System-level *modeling* and *partitioning*
- *Hardware synthesis* of dedicated units
  - Research/commercial standard synthesis tools
- *Software synthesis* for processor (core)
  - Specialized compiling techniques
- *Interface* synthesis
  - Hardware-software interface and synchronization
  - Drivers of peripheral devices

System-Level Partitioning

- One extreme: Full HW solution
  - High performance due to *Parallelism*
  - High cost and long time of ASIC fabrication
- Other extreme: Full SW solution
  - High-performance, low-cost processors
  - Operation serialization
  - Lack of support for specific tasks
- Best solution is a mix of HW and SW
HW/SW partitioning goals

- *Speed-up* software execution
  - By migrating software functions dedicated to hardware
- *Reduce cost* of hardware implementation
  - By migrating hardware functions to software
- Achieve system *prototypes*
  - By migrating functions to the prototype medium

How it is done?

- Determine hardware and software components from an overall system model
- Behavioral system model
  - *Tasks and dependencies*
  - *Flow-graph representation*
- Constrained partitioning of flow-graph
- *Partition determines the macroscopic system implementation parameters*
Example of abstract model-flow graph

```
Source
  ↓
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
  |      |      |      |      |      |      |      |      |      |      |      |      |      |
Example of partitioning approaches

- **Software speed-up** by using dedicated hardware
- Model system as software program
  - C-like language with performance constraints
- Determine performance bottlenecks
- Migrate critical portions to ASIC
**System level partitioning - Cosyma**

- ES $\rightarrow$ C
- C program
- C compiler
- object code
- ES flow graph
- partitioning
- cost estimation
- runtime analysis
- HardwareC
- HardwareC model
- Olympus
- simulator
- C$^*$ model
- C$^*$ compiler

---

**Partitioning approaches - Vulcan**

- *Reduce hardware costs* by migrating non-critical functions to software running on processor (core)
- Model system in HDL
  - Hardware-C
    - C-like syntax and hardware semantics
  - Extract feasible software threads
  - Synthesize dedicated-software code
System level partitioning - *Vulcan*

**Hardware-C description**

- System Graph Model
  - HERCULES (Behavioral Synthesis)

**VULCAN-II**
- a. System Partitioning
- b. Multi-thread Program Generation
- c. Interface Generation

- C Program
  - compiler
- Interface Graph Model
  - HEBE + CERES (Structural Synthesis)

- Assembly Code
- ASIC and Interface Netlist

Mixed System Implementation

**POSEIDON** (Simulation)

- System Inputs
- System Outputs

---

**System architecture**

- CPU
  - Software
  - accelerator
  - memory
  - I/O

---

**Hardware**

- CPU
  - Software
Issues in system partitioning

- Object granularity in partitioning
  - Operation versus functions
- Estimation of performance and cost from graph model
  - Satisfaction of design constraints
- Incorporate designer’s experience in biasing a partition

Co synthesis after partitioning

- Software functions identified by program threads
- A single processor requires thread serialization or interleave
- Scheduling of threads and instructions
  - Satisfying performance constraints
- System-level run-time scheduler to synchronize software and hardware functions
Hardware synthesis

- After partitioning, HW behavior is still described at high level of abstraction
- HW synthesis alternatives
  1. Manual translation to RTL
  2. Use Behavioral synthesis

SW synthesis

- SW synthesis ≅ Compilation
- Maybe preceded by automatic SW generation steps
- Compilation specialties in embedded systems
  - Code is compiled once => compilation time not important, maximum optimization is desired
  - RT constraints=>code must be tight and fast => assembly
Interface synthesis

- Processor interacts with ASIC and peripheral devices via serial or parallel ports
- Device drivers may be in software or in hardware
- Allocate ports to devices
- Schedule processor communication
- Buffer insertion

Interface synthesis issues

- Specific problems arise from:
  - Communication and synchronization patterns
    - Memory mapped, DMA, etc.
  - Interfacing processors to peripherals (sensors, actuators)
    - Automatic (processor port allocation, deciding to implement device drivers in HW or SW)
  - Scheduling the processor communication
    - Complicated under RT constraints
    - Complicated under data-dependant delays
An alternative approach

- Application-specific hardware is strongly decoupled from the processor
  - It can be viewed as a *hardware accelerator*
- Communication is expensive
- What if I create an application-specific processor (ASIP)?

Application specific instruction processors
ASIP vs. Accelerator

- Much tighter interaction between software and application-specific hardware
  - Communicate through registers, on a cycle-by-cycle basis (low overhead)
  - Instruction fetch bottleneck (loss of parallelism)
- Fine granularity of operations
  - Cannot migrate arbitrarily complex functions in application specific instructions
  - Opportunity for fine tuning other microarchitectural parameters

Co-synthesis flow

- Instruction-mix analysis
- Instruction-set selection
- Compiler development
- Compilation
- Performance analysis
- Source program
- Test data
**System architecture**

- ASIP
- Memory
- I/O

**Software synthesis problems**

- Target architecture is an ASIP
  - Develop a specific compiler
- Developing a new compiler for each new ASIP is unreasonable => retargetable compilers
  - Classification according to retargeting time
    - Portable compiler
    - Compiler-compiler
    - Machine-independent compiler
Retargetable compilers

- **Portable compilers**
  - Compiler needs significant rewrite for porting
- **Compiler compilers**
  - Generates compilers from architectural template
- **Machine-independent compilers**
  - Applicable to different architectures

Retargetable compilers

- Compiler technology adaptable to different architectural back-ends
  - ASIPs have specific instruction sets, memory and interconnection resources
- **Code quality** (i.e. execution speed) is important whereas **compilation time** is less critical
- Assembly code programming is still common practice
Retargetable compiler architecture

A third alternative: mapping to a reconfigurable SoC

- Software mapping
  - Onto an embedded core on the SoC
- Hardware mapping
  - Using FPGA or DSP-FPGA fabric
- Same partitioning decisions!
- Difference: resource bounds are fixed by the chosen reconfigurable architecture
  - Reduces degrees of freedom
Today’s Reconfigurable System

Xilinx Virtex II Pro

- Programmable high-speed links
- Four power PC processors
- Embedded memories
- DSP-oriented macros

Challenges

- Cost: reconfigurability is expensive w.r.t. application-specific hardware
  - Low-volume targets
- Power: reconfigurable fabrics are power hungry
  - Unsuitable for energy-constrained applications
- BUT... scaling is helping for both!!
Summary

- The basics of codesign
- Cosynthesis: first encounter
  - Partitioning
  - Allocation+Scheduling
  - Optimization
- Different targets
  - Accelerated systems
  - ASIPS
  - Reconfigurable SoCs