Outline

• Introduction and motivation
• Co-design advantages
• Application areas
• Digital system design
• HW/SW co-design design flow
• Summary

» Note: Part of this material was derived from the lectures of Prof. Giovanni De Micheli at Stanford University.
Introduction

• Electronic systems consist of:
  – HW platform
  – SW application layers
  – Interfaces
  – Analog components
  – Sensors and transducers

• Main trends:
  – Migration from analog to digital processing
  – Broader system-level integration to support System-On-a-Chip (SOC) approach

Motivation

• To meet system-design objectives by exploiting the synergism of the HW and SW parts through their concurrent design

• Computer-aided co-design is a new opportunity:
  – New open problems for researchers
  – New markets (potentially large)
Objectives in system-design

• Maximize system performances
  – High-speed, low-power,…
• Reduce costs
  – Reduce number of parts to manufacture and their size
  – Extend life time of HW components by supporting re-programmability
  – Ease system-level debugging and testing
• Good trade-off solutions can be found by balancing the HW and SW parts

Co-design advantages

• Explore different design alternatives in the architectural design space
• Tune HW to SW and vice-versa
• Reduce the system design time
• Support coherent design specification at the system-level
• Facilitate the re-use of HW and SW parts
• Provide integrated environment for the synthesis and validation of HW and SW components
Digital system classification

Co-design application areas

- Large system design:
  - Telecom systems
  - Defense systems
- General-purpose computing systems:
  - Personal computers
  - Workstations
  - Mainframes
- Embedded (dedicated) computing and control systems:
  - Automotive and airplanes applications
  - Home appliance control
  - Industrial plant control
  - Robots
Co-design of general-purpose computers

- Architectural support for operating systems
- Pipeline control design:
  - HW pipeline control unit
  - Compiler design
- Cache sizing and control
  - Choice of parameters and tuning
- Multiprocessor design support

---

Co-design of general-purpose computers

- Concurrent HW and SW development
  - Compiler design must begin early

- SW is needed to validate HW performances
  - Co-simulation of both HW and SW parts
Co-design of embedded systems

- Design of dedicated computing and control systems
- Embedded controllers
  - On-line control of manufacturing process
  - Robots guidance and control
  - Aircraft, automobile and ship control
- Data processing and communication systems
  - Telecom
  - Radio-navigation

Co-design of embedded systems

- Design of dedicated HW parts
  - Different design styles:
    - Co-processors, embedded cores, ASIPs, ...
  - Widely varying design scale
- Design of dedicated SW parts
  - Special-purpose operating systems
  - Drivers of peripheral devices
Programming digital systems

- General-purpose computing systems:
  - End-user or operator programs the system

- Embedded computing and control systems:
  - Manufacturer programs the system
  - User has limited access to programmability

Programming digital systems

- Application-level programming
  - Limited user control

- Instruction-level programming
  - Software programs
  - Instruction Set Architectures (ISAs)

- Hardware-level re-configuration
  - Exploit field-programmable technology
Implementation issues

- Technology:
  - CMOS, BiCMOS, GaAs, …

- Design style:
  - Full-custom
  - Semi-custom

- Operating mode:
  - Synchronous digital circuits
  - Asynchronous digital circuits
  - Analog or mixed analog/digital circuits

Selection criteria of design style

- Performance
  - Speed, area, power, …

- Time-to-market
  - Design and simulation time

- Product growth
  - Ease of extending capabilities

- Volume
- Cost
Semi-custom design

- Cell-Based
  - Standard-Cells
  - Macro-Cells
    - Memory Gen.
    - PLA Gen.
    - Gate Matrix Gen.
- Array-Based
  - Pre-Diffused
    - Gate Arrays
    - Sea-of-Gates
    - Compacted Arrays
  - Pre-Wired
    - Anti-Fuse Based
    - Memory Based

Array-based design

- Pre-Diffused Array or Mask Programmable Gate Array (MPGA)

- Pre-Wired Array or Field Programmable Gate Array (FPGA)
  - Soft-Programmable (Memory based)
  - Hard-Programmable (Anti-fuse based)
Integration level

- Single chip systems (SOC approach):
  - ASICs with embedded cores and memories
  - Cores (microprocessor, microcontroller, DSP, ...)
- Multiple chip systems:
  - ASICs, FPGAs, ...
  - Memories
  - Programmable components such as processors, DSPs or controllers
    - Off-the-shelf or proprietary components
- Distributed systems

Market trends

- Design of dedicated computing systems
  - Increasing in number
  - Tighter requirements
- Product differentiation through software
  - Design fewer chips
  - Reduce Non-Recurring Engineering (NRE) costs
- Availability of ASICs with embedded cores
  - High-volume parts
  - High intellectual property
- Design and use of ASIPS
ASIPs

- Application-Specific Instruction Processors (ASIPs) are processors dedicated to one or few applications
- Customized instruction sets
- Specific resources
  - Register files, buses, interconnections, ...
- Outperform standard processors on specific applications
- Require specific compilers

ASIPs

- Advantages:
  - Programmability
  - Eases portability
  - Higher performances
  - Lower power dissipation

- Disadvantages:
  - Require development of a dedicated software environment (e.g. compiler)
  - Longer time-to-market than standard processors
### Comparison

<table>
<thead>
<tr>
<th></th>
<th>IP</th>
<th>ASIP</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>+</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>Power</td>
<td>+++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Reuse</td>
<td>+++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>HW design effort</td>
<td>+</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>SW design effort</td>
<td>+</td>
<td>++</td>
<td></td>
</tr>
</tbody>
</table>

#### Embedded systems

![Diagram of Embedded System](image)

- **MEMORY**
- **ISP**
- **HARDWIRED UNIT**
  - Application-specific logic
  - Timers
  - A/D and D/A Converters

- **SENSORS**
- **ACTUATORS**

**ENVIRONMENT**
Embedded system requirements

- **Reactive systems:**
  - The system never stops
  - The system responds to signals produced by the environment

- **Real-time systems:**
  - Timing constraints on task evolution
  - Hard and soft constraints

Objectives in embedded system design

- **Correct implementation of control function**
  - Satisfaction of all hard timing constraints
  - Satisfaction in average of all soft timing constraints

- **Dependability**
  - Reliability, availability and safety
Specific steps in embedded control design

- Architecture selection:
  - Standard microcontroller or microprocessor
  - ASIC
  - ASIC with embedded core or co-processor
- Technology selection for HW resources
- Design dedicated HW, SW and interfaces

Co-design flow of embedded systems

- Modeling, validation and synthesis
  - System-level simulation
- Homogeneous modeling
  - HW/SW partitioning
  - HW/SW or SW/HW migration
- Heterogeneous modeling
  - Direct implementation and re-targeting
- Co-synthesis
  - HW and interface synthesis
  - SW compilation and code generation
- Co-simulation
Modeling

• Functional simulation models:
  – Software programs (C, C++, ...)

• Specification models:
  – HW and SW languages
    • VHDL, Verilog, DFL, Esterel, C, C++, SDL, ...
  – Graphical formalism:
    • Statecharts, Statemate, SpecCharts, Control and Data Flow Diagrams, ...

Main issues in modeling

• Heterogeneous nature of embedded systems
• No global system-level language is available, except for restricted domains
  – The SLDL (System Level Description Language) committee aims at providing an inter-operable language or representation for the specification and design of microelectronics systems at high abstraction level
• Different types of constraints to satisfy
• Support design environments during the co-specification and co-simulation phases
Abstract models

• Useful to interpret system specification
  – Finite State Machines (FSMs)
  – Control and Data Flow Graphs
  – Petri nets
  – Process algebra
• Can be derived from language models by compilation and useful for synthesis and verification

Co-design FSM semantics

• Concurrent FSMs:
  – Synchrony hypothesis
  – All FSMs change state simultaneously
  – Adequate for HW implementation

• Co-design FSMs:
  – FSMs emit and react to events
  – Unbounded reaction time
  – Useful to model interleaving in SW
Flow graph semantics

- Vertices: operations or tests
  - Executed in HW or SW
- Edges: dependencies
- Paths: parallel or alternative streams
- Graph executes at given rate
- Hierarchical call to sub-graphs:
  - Model iteration and calls

Simulation and debugging requirements

- Embedded controllers:
  - ASICs plus SW running on a processor
  - VHDL or Verilog plus C programs
  - Weakly heterogeneous systems

- Embedded data processing and communication systems
  - ASICs plus SW running on a processor or ASIP
  - Environmental modeling (e.g. telephone lines)
  - Strongly heterogeneous systems
Co-simulation

- **Desired features:**
  - Level of timing accuracy
  - Speed of simulation runs
  - Visibility of internal states
- **Potential problems:**
  - Meaningful results are obtained with large SW programs
  - Model availability
  - Strong heterogeneity requires specialized environment

Co-simulation paradigms

- **Homogeneous** modeling:
  - HW models in HDL
  - Processor model in HDL
  - SW in assembly code
- Usage of HDL simulator for the whole system including the processor model
- Simple method but quite inefficient
Co-simulation paradigms

- **Weakly heterogeneous** systems
  - a) HDL simulators with processor model
  - b) Compiled SW
  - c) HW emulation

- **Strongly heterogeneous** systems
  - Require specialized simulation environments (e.g. Ptolemy)
  - Communication mechanisms among domains and their corresponding schedulers

---

a) HDL processor modeling

- **Precise timing model**
  - Accurate timing and complete functionality
    - Event-driven simulation

- **Zero-Delay Model (ZDM) for timing**
  - Correct transitions at clock edges
    - Cycle-based simulation

- **Instruction-set simulator**
  - Model emulates processor while insuring correct register and memory values
B) Compiled SW

- Basic assumption:
  - HW/SW communication protocol such that communication delay has no effect on functionality
- SW is compiled and linked to simulator
- HW/SW communication is replaced by handshake
- Simulation speed is limited by HW simulation speed

C) HW emulation

- HW mapped onto programmable HW
  - One order of magnitude loss in speed
- Programmable HW boards connected to workstations
- Limited visibility of internal states
Co-verification

- **Goal**: verify formal properties of specification without simulation
- Need specification with formal properties
- Limited results and tools today (e.g. protocol verification)

HW/SW partitioning

- **Goal**: define the HW- and SW-bound parts from the overall system model
- Behavioral system model
  - Tasks and dependencies
  - Flow-graph representation
- Constrained partitioning of flow-graph
- Partitioning determines the macroscopic system implementation parameters
HW/SW partitioning goals

- Attempt to maximize performance under cost constraints:
  - Speed-up SW execution by migrating SW functions to dedicated HW
- Attempt to minimize cost under performance constraints:
  - Reduce cost of HW implementation by migrating HW functions to SW
- Achieve system prototypes
  - By migrating SW functions to the prototype medium

HW/SW partitioning

- Partition abstract system-level models (e.g. flow-graph models)
- Possible approaches:
  - Maximize performance:
    - Fully SW-oriented initial models (C, C++, ...)
    - Migrate critical SW functions to dedicated HW
    - Example: Cosyma [Ernst, Henkel]
  - Minimize cost:
    - Fully HW-oriented initial models (VHDL, HardwareC, ...)
    - Migrate non-critical HW functions to SW
    - Example: Vulcan [Gupta]
**SW-oriented partitioning**

- Basic idea: Speed-up the SW by using dedicated HW resources
- Model system as SW program
  - C-like language with performance constraints
- Determine performance bottlenecks
- Migrate critical SW portions to dedicated HW resources such as ASICs
- Example: **Cosyma** approach:
  - Modeling in $C^*$ which is a superset of C to include:
    - Performance constraints
    - Assertions to specify loop bounds

---

**HW-oriented partitioning**

- Basic idea: Reduce the HW cost by migrating non-critical functions to SW running on processor
- Model system in HDL
  - HardwareC (C-like syntax and HW semantics)
- Extract feasible SW threads
- Synthesize dedicated SW code
- Example: **Vulcan** approach:
  - Modeling in HardwareC
  - Enforce performance constraints to include data rates and latency
  - Target: HW gate netlist plus SW code for selected processors
Main issues in HW/SW partitioning

- Define coarse or fine object granularity during partitioning from high-level specification
  - Operations versus functions
- Evaluate cost/performance trade-off points from high-level model
  - Satisfaction of design constraints
- Incorporate designer's experience in biasing a partition
- Typical approaches are based on an iterative improvement from an initial fully HW or fully SW partition of the system

HW/SW re-targeting

- Based on heterogeneous modeling
- Map HW component models to SW or vice-versa
- Heterogeneous modeling with re-targeting is an alternative to partitioning
- Example:
  - Ptolemy
    - Mapping to DSP code or to FSMs
  - Siera
    - SW code or HW synthesis tools
  - Co-Ware:
    - DSP code and HW synthesis
System-level co-synthesis

- System-level co-synthesis is still investigated
- We have good HW synthesis tools and good compiler technology
- Problem: How do we migrate to system-level co-synthesis?

Simple target architecture

- A set of dedicated HW units
- A programmable core or co-processor
- Memory
  - Including SW program storage
- Interfaces and interconnections
System-level co-synthesis flow

- Consistent system-level modeling
- Partitioning into dedicated and programmable units
- HW synthesis of dedicated units
  - Based on research or commercial standard synthesis tools
- SW synthesis for programmable units (processors)
  - Based on specialized compiling techniques
- Interface synthesis
  - Definition of HW/SW interface and synchronization
  - Drivers of peripheral devices

SW synthesis problems

- Target architecture is an ASIP
  - Develop a specific compiler
- Non-executable system-level specification of computer-aided partitioning
  - Synthesize high-level or assembly code
- Interface to HW with given protocol
  - Synthesize interfacing routines
Re-targetable compilers

- Compiler technology suitable for 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.

Re-targetable compilers

- Portable compilers
  - Compiler needs significant re-write for porting

- Compiler compilers
  - Generates compiler from architectural templates

- Machine-independent compilers
  - Applicable to different architectures
Re-targetable compilers

- Compile code into intermediate form and optimize
  - Standard optimizing compiler algorithms
- Instruction selection
  - Pattern matching techniques
- Instruction scheduling
  - Satisfaction of real-time constraints
- Register allocation
- Micro-code compaction

Code synthesis after partitioning

- SW functions identified by program threads
- A single processor requires thread serialization or interleaving
- Scheduling of threads and instructions
  - Satisfying performance constraints
- System-level run-time scheduler to synchronize SW and HW functions
Interface synthesis

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

Co-design for re-configurable systems

- Re-configurable systems are based on programmable HW
- FPGAs try to bridge the gap between HW and SW parts
- Electrical re-configuration
- Programmable like SW but via synthesis techniques
- Emulate HW or SW
**SW execution acceleration**

- Processor with programmable HW board
- Migrate critical functions to programmable HW board
- Key tasks:
  - Performance estimation and partitioning
  - Synchronize SW and HW execution
- Applicable to system-level simulation

**System-level prototyping**

- Prototyping complex systems is useful for debugging and validation
- Traditional styles
  - Bread-boarding, SW emulation
- Programmable HW approach
  - Workstation with programmable boards
- HW/SW systems can be prototyped
Evolvable systems

- HW re-configuration concurrent with execution
  - A part of the system is configured
- Applications:
  - Self-adapting systems
  - Fault-tolerant systems
- Example:
  - FPGA rectangular tissue partitioned into cells
    - Configured by feeding cells with a program

Summary

- HW/SW co-design is a wide and rapid evolving area
- Several application domains, architectures and implementation styles
- Need to define design methodologies with the goal of developing CAD tool support
- The impact of CAD on system-level design will be more profound than the impact of CAD on VLSI design