Hardware/Software Co-Design/Co-Verification

Sungho Kang
Yonsei University
Outline

- Introduction
- Co-design Methodology
- Partitioning
- Scheduling
- Co-Simulation Systems
- Timed Co-simulation
- Multimedia Examples
- Conclusion
Mixed Implementation

Introduction

Behavioral specification plus constraints

Analog interface

Performance

Mixed implementation

Memory

Hardware

Software

Constrains

Cost

Interface

μP

Program

A mixed implementation

ASIC

Software

Hardware
Co-Design

• Integrated design of systems implemented using both hardware and software components

• Why
  ▪ Advances in enabling technologies
    ✓ System level specification / simulation
    ✓ High level synthesis and CAD frameworks
  ▪ Advanced design methods required due to the increased diversity and complexity
  ▪ Cost and performance of HW/SW systems should be optimized for market competitiveness
  ▪ Produce-to-market time is vital
  ▪ Exploiting concurrency among design threads and tools will result in significant gains

• Difficulties
  ▪ Target moves (e.g. in size and complexity)
  ▪ Tolerance level for error decreases
Co-Design

• Integration of HW and SW design techniques
  ▪ The HW and SW components are interdependent
  ▪ HW and SW are typically described and design using different methodologies, languages and tools

• Advantages
  ▪ Acceleration of the design process
  ▪ Lengthy system integration and test phase can be avoided
  ▪ Dynamic HW/SW trade-offs in the design process

• Co-design methodologies differ widely
  ▪ Widely differing assumptions
  ▪ Interface/communication techniques
  ▪ Design goals

• Example types
  ▪ A microprocessor and its associated glue logic
  ▪ A microprocessor and special-purpose computing engine
Co-Design

Introduction

HW/SW interface abstractions

- application software
- operating system
- device driver
- system bus

software

send, receive, wait
register reads/writes
interrupts
bus transactions

hardware

application specific hardware
bus interface
system bus

Hardware-software co-design activities

- hardware-software co-design
- hardware-software co-synthesis
- hardware-software co-simulation
- hardware-software partitioning

CS&RSOC

YONSEI UNIVERSITY
Co-Simulation

Introduction

• Simulation of heterogeneous systems whose HW and SW components are interfacing
• Roles of co-simulation
  ▪ Verification of system specification before system synthesis
  ▪ Verification of mixed system after system synthesis and integration
  ▪ System performance estimation for system partitioning
• Issues of HW-SW co-simulation
  ▪ Timing accuracy
  ▪ Processor model
  ▪ Performance
  ▪ Interface transparency
  ▪ Transition to co-synthesis
  ▪ Integrated user interface and internal representation
Reason for Co-simulation

- Processor transition to embedded core
  - No existing hardware

- Increased software content
  - Software controls more functions
  - Development and debug times increased

- More complex hardware interfaces
  - Processor interactions with DSP
  - Processor interactions with increasingly complex hardware
Reason for Co-Simulation

- Design problem found
  - Something wrong due to logic error
  - Problem is visible, but not apparent in hardware
  - Found while running software

- Brings SW and HW together
Co-Design Problems

- Instruction set processors
  - Codesign to design well-balanced long-lasting processors
  - Instruction set selection
  - Cache design
  - Pipeline control
    - HW mechanism: flush the pipelines
    - SW solution: reorder instructions or insert no operation
- ASIP
  - SW: retargetable code generation for ASIP data path
  - HW: library binding
  - High performance
  - Desirable programmability
  - Low unit cost than ASIC
- Embedded systems and controllers
  - Real time systems with peripheral devices (sensors, actuators)
Co-Design Methodologies

- Automation of conventional codesign
  - clear early binding
  - easy design decisions

Diagram:
- Constraints/requirements analysis
- System specification
- HW/SW partitioning
- HW synthesis
  - SW generation
  - interface synthesis
- HW/SW cosimulation
- Integrated system evaluation/verification
Co-Design Methodologies

- Model-based codesign
  - late partitioning/binding after refining
  - easy to handle design changes
  - component reuse
  - modular hierarchical models

![Diagram of Co-Design Methodologies]

1. Constraints/requirements analysis
2. System specification
3. System modeling
4. Model base
5. Validation/Simulation
6. Refinement (decompose into submodels)
7. Verified model (desired granularity)
8. Technology assignment (into HW/SW/interface components)
Models and Languages

- **Term-rewriting technique**
  - (e.g.) $f(g(1,y))$ reduces to $f(f(1))$ by applying the rewrite rule $g(x,y) \rightarrow f(x)$
  - Bundgen and Kuchlin, Univ. Tubinger, Germany

- **Funmath (Functional mathematics)**
  - Boute, Univ. nijmegen, Netherlands

- **Interaction refinement calculus**
  - Broy, Tech. Univ. Munchen, Germany

- **Graphical model (data flow diagram)**
  - Evans and Morris, Univ. of Manchester, U.K.

- **Synchronized Transitions**
  - Model a computation as a fine-grain parallel computation.
  - Model interface with state variables.
  - Staunstrup. Tech. Univ. of Denmark

- **Hierarchical Petri Nets**
  - Dittrich, Univ. Dortmund, Germany

- **Solar (FSMs and state tables)**
  - Jerraya and O’Brien, TIMA/INPG, France

- **C, C++, VHDL**
Modeling for Co-Simulation

• Simulation of a mixed system
  ▪ SW: functional model, instruction set level model, detailed architectural model
  ▪ HW: functional model, netlist level model

• Processor modeling at different levels of abstraction, depending on the availability and the desired simulation accuracy
  ▪ Detailed processor model
    ✈ Discrete-event model of their internal HW architecture
  ▪ Bus model
    ✈ Simulate the activity on the periphery of a processor
    ✈ Useful for verifying low-level interactions
    ✈ Difficult to model the activity accurately
  ▪ Instruction set architecture model: efficient (C program)
  ▪ Compiled simulation (binary-to-binary translation): very fast
  ▪ HW models (physical hardware)
Embedded System Design

Co-design Example

- Importance factors
  - Design time
  - Manufacturing cost: consumer appliances
  - Modifiability: prototype communication system
  - Reliability: anti-lock brake system

- Tasks
  - Partitioning into smaller interacting components
  - Allocating the partitions to microprocessors or other hardware units
  - Scheduling
  - Mapping each functional description into an implementation
Embedded Microprocessor System

- SW on a dedicated microprocessor
- The HW is modeled/designed at a much lower level of abstraction than the SW.

![Diagram of embedded microprocessor system]

An embedded microprocessor system
Heterogeneous Multi-Processing System

Co-design Example

- Distributed heterogeneous embedded processor system
  - Choose the number and type of PEs.
  - Map SW tasks onto PEs.
  - Meet performance objective while minimizing HW cost.

![Diagram of distributed heterogeneous multi-processing system]

- software
- hardware

appl. code

\[ \mu \text{ proc. type 1} \]

interconnection network
Application Specific Instruction Set

- Application-Specific:
  - Optimize for the given application SW
- HW-SW boundary can be moved by adding/deleting instructions
- Modifiability should be considered in finding the best partition

An application-specific instruction set processor
Application Specific Co-Processors

- An Instruction set processor + custom co-processors
- Custom co-processor systems afford many degrees of freedom (DSP, SISD, SIMD, MIMD)
- Different HW-SW partitioning strategies
  - Move the performance-critical regions of code into HW.
  - Reduce HW cost without decreasing performance.
  - Minimize the communication between HW & SW components and maximize the concurrency.
  - Trade-off performance and cost.

A multi-threaded custom co-processor
Partitioning Problems

- Two-way vs multi-way partitioning
- Performance-driven partitioning (system performance)
  - Speed
  - Power
- Layout-driven partitioning
- Partitioning with replication (FPGA pins)
- Hardware-software partitioning (trade-off)
  - Co-synthesis rather than partitioning
  - Some factors are not easily quantifiable
Partitioning in Co-Design

- Partitioning
  - HW/SW level
  - Architecture partitioning
  - Early binding: easy design decisions
  - Late binding: easy to handle change requirements
    \(\text{\# better solution}\)
- Partitioning on a high level
  - Hardware sharing among exclusive operations
  - Concurrent scheduling to each partition
- Parameterized modules from library for specific design needs
  - Interface: signal exchange, interrupt driven, center scheduler
  - co-simulation: parallelism vs synchronization
HW-SW Partitioning

- Performance requirements
  - Some functions may need to be implemented in HW
  - The overhead of synchronization and data transfer should be considered
- Implementation costs
  - HW can be shared
- Modifiability
  - SW can be easily changed
- Nature of computation
  - Some function may have an affinity for either HW or SW
  - Degree of data parallelism
HW-SW Partitioning

- Software-preferred
  - Task which calls OS often
- Hardware-preferred
  - Arithmetic operation
  - High degree of data parallelism
  - Multiple threads of control
  - Customized memory architecture
Software-Oriented Partitioning

- As many operations as possible in software
  - High memory density of standard microprocessors
  - Optimally adapted compilers
  - Carefully verified standard cores
  - Simpler software debugging
  - Smaller hardware
  - Flexibility in case of modification

- External hardware
  - When timing constraints are violated
  - Basic and inexpensive I/O functions
Hardware-Oriented Partitioning

- Move hardware functions to software checking timing constraints and synchronism
  - Min/max delay constraints
  - Execution rate constraints

- Challenges
  - Interface synchronization
  - Hierarchical memory schemes
  - Multiple processors
  - Pipelining
Performance Estimation

• At a low abstraction level
  ▪ Easy and accurate
  ▪ Long iteration time

• At a higher level of abstraction
  ▪ Necessary to reduce the exploring time
  ▪ Currently quick and dirty
  ▪ Goal: quick and accurate

• Performance and cost estimation is important for HW-SW partitioning and for HW/SW synthesis/optimization
Scheduling for Real-Time Systems

- Intricate timing requirements at different levels
  - Processors must generate a sequence of control signals and read/write with appropriate time intervals
  - Higher level timing constraints
- System-level scheduler
  - accurate run-time estimation by scheduling after resource binding
  - Under sequencing, rate, and response time constraints
- Low-level scheduler
  - Each atomic sequence at system level is scheduled
  - Hardware and software device interactions are considered
Parallel Scheduling

- The task of scheduling
  - Assign actors to processors
  - Order execution of the actors on each processor
  - Determine when each actor fires such that all data precedence constraints are met

- Non-preemptive schedule
  - An actor can not be interrupted in the middle
  - Preemption entails a significant implementation overhead

- Static/dynamic schedule
  - Static scheduling for hard real-time constraints
  - Dynamic scheduling at run time may lead to a more flexible implementation
  - But difficult to achieve real-time performance guarantees

- Performance metric
  - The (average) iteration period $T$
  - The throughput $1/T$
Parallel Scheduling

- Optimal multiprocessor scheduling of an acyclic graph is known to be NP-hard.
- List scheduling is popular.
  - Assign priorities to tasks.
  - The ready task with the highest priority is scheduled as soon as a processor is available.
- Overlapped schedules are superior to blocked schedules with unfolding and retiming.

![Diagram](image)

Fully static schedule
Hardware Reusability

- Key to cost effective design
- Reusability
  - Possibility of several operators to a HW module
  - External controllability
  - Dependent on the characteristics of the circuit
  - The ability of the CAD tool is also important
  - The functionality of HW modules can be extended or adapted by means of automatic transformations (library, structure, behavior)
Design Space Exploration

- **Design parameters**
  - The number of HW functional units
  - The max number of buses
  - The max number of variable references in a step.

- **Estimation of resources, time steps, buses, variable accesses**
  - Scheduling can be computationally intensive.
  - A lower bound can be heuristically estimated.
    - Complete scheduling gives an upper bound
  - Window concept by the threshold $W$
    - $W=1 \Rightarrow$ complete schedule
    - $W>1 \Rightarrow$ partial schedule
      - (degree of freedom $\leq W$)
    - Partition DAGs, until $\#\ t\_steps(DAG) \leq W$
    - Depth first search branch and bound, to reduce memory requirement.
Design Space Exploration

- Optimization problem
  - Dominating design
    - Each criterion is no worse than those of other designs
      - Find the set of designs which are not dominated by other designs
  - When resource estimation and partial scheduling return a design point
    - Discard it if it is dominated by others
    - Incorporate it in the design space, otherwise
  - The number of time steps for each DAG is initially set to its critical length, and is relaxed if a constraint is violated
Becker, Singh, and Tell (1992)

- Co-simulation of Network Interface Unit (NIU) using PLI of Verilog-XL simulator, C++, and Unix Socket
- Synchronized handshake + cycle accurate processor interface model

System under Development

Co-simulation Environment

NIU Firmware in C++
- Interface functions
- Co-simulation support library
- Unix

NIU HW description in Verilog
- SPARC model
- Port model
- Other device model
- IPC tasks
- Built-in tasks
- Verilog-XL
- Unix

NIU Monitor program in C++
- Interface functions
- Co-simulation support library
- Unix

Network
Co-simulation using PLI of Verilog-XL simulator and Unix Socket

Synchronized handshake with no processor model
Poseidon

- Gupta, Coelho, and Ed Micheli, 1992
- Pin-level(cycle accurate) model of processors
- Software in assembly code of the target processor(DLX)
**Timed Co-Simulation**

- **Untimed co-simulation**
  - Hardware and software time scales are not synchronized
  - Good for verifying functional completeness
  - Correct results for handshaking protocol

- **Timed co-simulation**
  - Hardware and software time scale are synchronized at each data exchange
  - Useful for overall performance assessment
  - Can be used for both handshaking and non-handshaking protocol
How to implement

- Nano-second accurate co-simulation
  - Need expensive processor model
  - Very slow
- Cycle-based co-simulation
  - Need cycle-based processor model
  - Slow
- Synchronization through software timing estimation
  - No need of processor simulation model
  - Fast
Software Timing Estimation

Modify the intermediate C program for profiling
Synchronization

- Lockstep
  - Synchronize at every step
  - Large IPC overhead
- Optimistic
  - Simulate to optimistic next synchronization point
  - In the event of inconsistency, rollback
Non-IPC-based implementation

- SW instructions in the event list

Timed Co-Simulation

SW clock period(*n)

HW events & elements

SW instruction

bend_loopstart, LT

lacl var
Timed Co-Simulation

- Implementation with VHDL description

```vhdl
procedure sw_part (  
    data : inout integer;  
    next : out TIME;  
)

attribute foreign of sw_part : procedure is "C";

-- in the architecture body
SW_processor : process  
variable next_time : TIME;  
begin  
    sw_part(data=>hw_data, next=>next_time);  
    wait for next_time - now;  
end process;
```
Conclusion

- Suggestion for Co-design
  - Design systems/prototypes by using co-design models
    - This facilitates postponing partitioning decision
    - This constitutes reusable modules
  - Develop successful partitioning methods
  - Develop validation techniques
    - Co-simulation (C+VHDL)
    - HW/SW emulation
    - Formal verification
- Co-simulation Open Issues
  - Well-defined abstract models for HW-SW co-simulation
  - Simulation-based performance estimation
  - Time-accurate and fast co-simulation
  - Integration with other co-design tools