Hardware/Software Codesign
EMT 483 | System on Chip
Hello!

I am Siti Zarina Md Naziri

Topics from Week 8 - 14

You can find/contact me @ Block 5, Level 1 (Next to MMIC Lab) | sitizarina@unimap.edu.my | 012-322-5646
Contents

- Hardware/Software Codesign
- System Specification
- Hardware/Software Partitioning
- Cosynthesis
- Architecture Generation (System Integration)
1. Hardware/Software Codesign

Definition | Motivation | Domain | Problem
Definition

- Concurrent design of hardware and software components of complex electronic systems with the goal to optimize and/or satisfy design constraints such as performance, cost, power and speed of the final product.
Key-Concept / Types

- **Concurrent**
  - Hardware & software developed at the same time in parallel paths

- **Integrated**
  - Interaction between hardware & software developments to produce designs that meet performance criteria and functional specification
Motivation

- Driven by the increasing complexity of the design
- Aims to reduce the time-to-market frame considerably
- The need for early prototypes needed for customer with feedback during the design process
- The sale and popularity of new products will not be driven so much anymore by just technological progress, but more and more by tools to design better quality and more reliable system with a given technology than any competitor
Motivation (Contd.)

- Improves overall system performance, reliability, and cost effectiveness because defects found in hardware can be corrected before tape-out.
- Benefits the design of embedded systems and SoCs, which need HW/SW tailored for a particular application.
  - Faster integration: reduced design time and cost
  - Better integration: lower cost and better performance
  - Verified integration: lesser errors and re-spins
- Embedded Systems
  - Consumer electronics
  - Telecommunications
  - Manufacturing control
  - Vehicles
- Instruction Set Architectures
  - Application-specific instruction set processors.
    - Instructions of the CPU are also the target of design.
  - Dynamic compiler provides the trade-off of HW/SW.
- Reconfigurable Systems
○ Hardware and software can be acquired separately and independently, with successful and easy integration of the two later
○ Hardware problems can be fixed with simple software modifications
○ Once operational, software rarely needs modification or maintenance
○ Valid and complete software requirements are easy to state and implement in code
Classic Design vs Codesign

classic design

co-design

HW

SW

HW

SW
The specification is divided into hardware and software portions early in the design process.
Often leads to difficulties when integrating the entire system at the end of the process.
Impact the product time-to-market delaying its deployment and restricts the ability to explore hardware and software trade-offs.
Integrates the hardware and software paths by envisioning a common platform, and by increasing the possibility of interaction between the hardware and software development.

The interaction focuses on keeping the software and hardware spaces unified at all times as much as possible to avoid a sudden complete integration in the last phase.
Typical Codesign Process

1. System Description (Functional)
2. HW/SW Partitioning
3. SW
   - Software Synthesis
4. HW
   - Hardware Synthesis
5. System Integration
6. Concurrent processes
   - Programming languages
7. Unified representation
   - (Data/control flow)
8. Instruction set level
   - HW/SW evaluation

FSM-directed graphs
Another HW/SW partition
Codesign Concept (Graphical)
Codesign Process

- System specification
- HW/SW partitioning
  - Architectural assumptions:
    - Type of processor, interface style.
  - Partitioning objectives
    - Speedup, latency requirement, silicon size, cost.
  - Partitioning strategies
    - High-level partitioning by hand, computer-aided
    - Partitioning technique.
- HW/SW synthesis
  - Operation scheduling in hardware
  - Instruction scheduling in compiler
  - Process scheduling in operating systems
Requirements for the Ideal Codesign Environment

- Unified, unbiased hardware/software representation
  - Supports uniform design and analysis techniques for hardware and software
  - Permits system evaluation in an integrated design environment
  - Allows easy migration of system tasks to either hardware or software

- Iterative partitioning techniques
  - Allow several different designs (HW/SW partitions) to be evaluated
  - Aid in determining best implementation for a system
  - Partitioning applied to modules to best meet design criteria (functionality and performance goals)
Requirements for the Ideal Codesign Environment (cont.)

○ Integrated modelling substrate
  ◦ Supports evaluation at several stages of the design process
  ◦ Supports step-wise development and integration of hardware and software

○ Validation methodology
  ◦ Insures that system implemented meets initial system requirements
Fast growth in both VLSI design and software engineering has raised awareness of similarities between the two

- Hardware synthesis
- Programmable logic
- Description languages

Explicit attempts have been made to “transfer technology” between the domains
Conventional Codesign Methodology

Analysis of Constraints and Requirements
  - System Specs..
    - HW/SW Partitioning
      - Hardware Descript.
      - Software Descript.
        - Interface Synthesis
          - HW Synth. and Configuration
            - Configuration Modules
            - Hardware Components
          - HW/SW Interfaces
            - Software Gen. & Parameterization
            - Software Modules
        - HW/SW Integration and Cosimulation
          - Integrated System
            - System Evaluation
            - Design Verification

© IEEE 1994 [Rozenblit94]
Key Issues during Codesign Process

- Unified representation
- HW/SW partition
  - Partition Algorithm
  - HW/SW Estimation Methods
- HW/SW co-synthesis
  - Interface Synthesis
  - Refinement of Specification
2. System Specification
Models | Architectures | Languages
Models, Architectures, Languages

- **Models**
  - State, Activity, Structure, Data, Heterogeneous
- **Architectures**
  - Function-Architecture, Platform-Based
- **Languages**
  - Hardware: VHDL / Verilog / SystemVerilog
  - Software: C / C++ / Java
  - System: SystemC / SLDL / SDL
  - Verification: PSL (Sugar, OVL)
Models & Architectures

- **Models** are conceptual views of the system’s functionality
- **Architectures** are abstract views of the system’s implementation
Hardware/Software Models

- State-Oriented Models
  - Finite-State Machines (FSM), Petri-Nets (PN), Hierarchical Concurrent FSM

- Activity-Oriented Models
  - Data Flow Graph, Flow-Chart

- Structure-Oriented Models
  - Block Diagram, RT netlist, Gate netlist

- Data-Oriented Models
  - Entity-Relationship Diagram, Jackson’s Diagram

- Heterogeneous Models
  - UML (OO), CDFG, PSM, Queuing Model, Programming Language Paradigm, Structure Chart
3. HW/SW Partitioning

Types | Approaches | Issues
Hardware/Software Partitioning

- Informal Definition
  - The process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or software

- Goal
  - To achieve a partition that will give us the required performance within the overall system requirements (in size, weight, power, cost, etc.)
Partitioning into hardware and software affects overall system cost and performance - automatic or iterative

Hardware implementation
- Provides higher performance via hardware speeds and parallel execution of operations
- Incurs additional expense of fabricating ASICs

Software implementation
- May run on high-performance processors at low cost (due to high-volume production)
- Incurs high cost of developing and maintaining (complex) software
Structural vs. Functional Partitioning

- **Structural**: Implement structure, then partition
  - Good for the hardware (size & pin) estimation.
  - Size/performance trade-offs are difficult.
  - Suffer for large possible number of objects.
  - Difficult for HW/SW trade-off.

- **Functional**: Partition function, then implement
  - Enables better size/performance trade-offs
  - Uses fewer objects, better for algorithms/humans
  - Permits hardware/software solutions
  - But, it’s harder than graph partitioning
Partitioning Approaches

○ Software-oriented partitioning
  ◦ Start with all functionality in software and move portions into hardware which are time-critical and can not be allocated to software

○ Hardware-oriented partitioning
  ◦ Start with all functionality in hardware and move portions into software implementation
Partitioning Approaches

- Determine which technology will be used for implementation of each design unit
- Criteria such as:
  - Execution time
  - Rate of use
  - Re-programmability
  - Re-use of existing components
  - Technology limitation
Binding Software to Hardware

- **Binding**: Assigning software to hardware components
- After parallel implementation of assigned modules, all design threads are joined for system integration
  - Early binding commits a design process to a certain course
  - Late binding, on the other hand, provides greater flexibility for last minute changes
- Specification-abstraction level: input definition
  - Executable languages becoming a requirement although natural languages common in practice.
  - Just indicating the language is insufficient
  - Abstraction-level indicates amount of design already done
    - e.g. task DFG, tasks, CDFG, FSMD

- Granularity: specification size in each object
  - Fine granularity yields more possible designs
  - Coarse granularity better for computation, designer interaction
    - e.g. tasks, procedures, statement blocks, statements

- Component allocation: types and numbers
  - e.g. ASICs, processors, memories, buses
3. Cosynthesis
Definition | Process | Categories
Definition of Cosynthesis

- Methodical approach to system implementations using automated synthesis-oriented techniques
- **Methodology** and **performance constraints** determine partitioning into hardware and software implementations
- The result is "optimal" system that benefits from analysis of hardware/software design trade-off analysis
○ Implementation of hardware and software components after partitioning
○ Constraints and optimization criteria similar to those for partitioning
○ Area and size traded-off against performance
○ Cost considerations
Cosynthesis Approach to System Implementation
Synthesis Flow

- HW synthesis of dedicated units
  - Based on research or commercial standard synthesis tools

- SW synthesis of dedicated units (processors)
  - Based on specialized compiling techniques

- Interface synthesis
  - Definition of HW/SW interface and synchronization
  - Drivers of peripheral devices
Hardware Synthesis

- Definition
  - The automatic design and implementation of hardware from a specification written in a hardware description language

- Goals/benefits
  - To quickly create and modify designs
  - To support a methodology that allows for multiple design alternative consideration
  - To remove from the designer the handling of the tedious details of VLSI design
  - To support the development of correct designs
Hardware Synthesis Categories

- **Algorithm synthesis**
  - Synthesis from design requirements to control-flow behaviour or abstract behavior
  - Largely a manual process

- **Register-transfer synthesis**
  - Also referred to as “high-level” or “behavioral” synthesis
  - Synthesis from abstract behavior, control-flow behavior, or register-transfer behavior (on one hand) to register-transfer structure (on the other)

- **Logic synthesis**
  - Synthesis from register-transfer structures or Boolean equations to gate-level logic (or physical implementations using a predefined cell or IC library)
Hardware Synthesis Process
Software Synthesis

- **Definition**
  - The automatic development of correct and efficient software from specifications and reusable components

- **Goals/benefits**
  - To increase software productivity
  - To lower development costs
  - To increase confidence that software implementation satisfies specification
  - To support the development of correct programs
Hardware Synthesis Categories

- **Language compilers**
  - ADA and C compilers
  - YACC - Yet-Another-Compiler compiler
  - Visual Basic
- **Domain-specific synthesis**
  - Application generators from software libraries
Why Software Synthesis

- Software development is becoming the major cost driver in fielding a system
- To significantly improve both the design cycle time and life-cycle cost of embedded systems, a new software design methodology, including automated code generation, is necessary
- Synthesis supports a correct-by-construction philosophy
- Techniques support software reuse
Why Software Synthesis (cont)

- More software → high complexity → need for automatic design (synthesis)
- Eliminate human and logical errors
- Relatively immature synthesis techniques for software
- Code optimizations
  - size
  - efficiency
- Automatic code generation
Software Synthesis Flow

Example of Embedded System with Time Petri-Net
4. Architecture Generation (System Integration)
- Starts with a set of interconnected hardware/software sub-systems (output of communication synthesis)
- Decomposed into two tasks:
  - Virtual prototyping
    - Produces a simulatable model of the system
  - Architecture mapping (prototyping)
    - Step produces an architecture that implements (or emulates) the initial specification
○ Each subsystem is mapped into a virtual processor, e.g. C for software, VHDL for hardware. The resulting virtual prototype allows a precise estimation of the resulting architecture.

○ The architecture mapping may be achieved using standard code generators to transform software parts into assembler code and hardware synthesis tools in order to translate hardware parts into ASICs or a virtual hardware processor (emulators).
Prototyping in Codesign

- Prototyping is the final step of co-design.
- Each virtual processor is mapped onto a real (or prototype) processor.
- Software modules are compiled for existing micro-processors or specific cores, and hardware processors are mapped onto ASICs or FPGAs.